MC9S12DP512
Device Guide
V01.25

Covers also

MC9S12DT512, MC9S12DJ512,
MC9S12A512

Original Release Date: 27 Nov 2001
Revised: 05 Jul 2005

Motorola, Inc

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.00	27 Nov 2001	11 Feb 2002		- Initial version based on DP256 V2.09.
V01.01	13 Mar 2002	13 Mar 2002		- Updated document formats. - Removed reference to SIM in overview. - Changed XCLKS to PE7 in signal description. - Removed "Oscillator start-up time from POR or STOP" from Oscillator Characterisitcs. - Changed VDD and VDDPLL to 2.35V. - Updated C_{INS}. - Updated I_{OL}/I_{OH} values. - Updated input capacitance. - Updated NVM timing characteristics.
V01.02	02 Apr 2002	02 Apr 2002		- Updated document reference (SPI, SCI).
V01.03	15 Apr 2002	15 Apr 2002		- Corrected values in device memory map (RAM start, flash protected sector sizes). - Updated document reference (SCI).
V01.04	06 Jun 2002	06 Jun 2002		- Changed all operating frequency references to 50MHz EXTAL and removed references to 80 pin LQFP.
V01.05	05 Jul 2002	05 Jul 2002		- Preface Table "Document References": Changed to full naming for each block. - Table "Interrupt Vector Locations", Column "Local Enable": Corrected several register and bit names. - Table "Signal Properties": Added column "Internal Pull Resistor". - Table "PLL Characteristics": Updated parameters K1 and f1 - Figure "Basic Pll functional diagram": Inserted XFC pin in diagram - Enhanced section "XFC Component Selection" - Added to Sections ATD, ECT and PWM: freeze mode = active BDM mode.

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.06	24 Jul 2002	24 Jul 2002		- Updated SPI electrical characteristics. - Updated Derivative Differences table. - Added ordering number example. - Added Detailed Register Map. - Changed Internal Pull Resistor column of signal table. - Added pull device description for MODC pin. - Corrected XCLKS figure titles. Moved table to section Modes of Operation. - Removed ’1/2’ from BDM in Figure Clock Connections. - Completely reworked section Modes of Operation. Added Chip Configuration Summary and Low Power Mode description. - Changed classification to C for internal pull currents inTable 5V I/O Characteristics. - Changed input leakage to 1uA for all pins. - Updated VREG section and layout recommendation. - Moved Power and Gound Connection Summary table to start of Power Supply Pins section. - Added ROMONE to pinout
V01.07	29 Jul 2002	05 Aug 2002		- Corrected mem map: ’MEBI map x of 3’ - Corrected mem map: KEYEN bits in FSEC. - Added section Printed Circuit Board Layout Proposal. - Corrected addresses in Reserved, CAN and EEP buffer map. - Updated NVM electricals.
V01.08	21 Aug 2002	21 Aug 2002		- Updated table ’Document References’ - Added section ’Oscillator (OSC) Block Description’
V01.09	24 Sep 2002	24 Sep 2002		- Section HCS12 Core Block Desciption: mentioned alternalte clock of BDM to be equivalent to oscillator clock - Corrected tables 0-1 and 0-2
V01.10	18 Oct 2002	18 Oct 2002		- Added derivatives to cover sheet. - Added part ID for 1L00M maskset. - Corrected in footnote of Table "PLL Characteristics": f_{OSC} = 4MHz.
V01.11	29 Oct 2002	29 Oct 2002		- Renamed Preface section to Derivative Differences and Document references. - Added A512 derivative. - Updated module set of DJ512 in Table 0-1. - Added details for derivatives without CAN and/or BDLC modules.
V01.12	03 Dec 2002	03 Dec 2002		- Corrected several entries in ’Detailed Memory Map’. - Removed footnote on input leakage current from table ’5V I/O Characteristics’.
V01.13	08 Jan 2003	08 Jan 2003		- Updated section ’Unsecuring the Microcontroller’. - Updated footnote 1 in table ’Operating Conditions’.
V01.14	23 Jan 2003	23 Jan 2003		- Renamed ROMONE pin to ROMCTL.
V01.15	28 Feb 2003	28 Feb 2003		- Corrected PE[1,0] pull specification in Signal Properties Summary Table.

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.16	31 Mar 2003	31 Mar 2003		- Corrections in App. A 'NVM, Flash and EEPROM': - Number of words per flash row = 64 - Replaced 'burst programming' with 'row programming' - Sector erase size = 1024 bytes - Corrected feature description ECT - Corrected min. bus freq. in table 'Operating Conditions'
V01.17	30 May 2003	30 May 2003		- Replaced references to HCS12 Core Guide with the individual HCS12 Block guides throughout document - Table 'Absolute Maximum Ratings' corrected footnote on clamp of TEST pin
V01.18	23 Jul 2003	23 Jul 2003		- Mentioned 'S12 LRAE' bootloader in Flash section - Document References: corrected S12 CPU document reference
V01.19	24 Jul 2003	24 Jul 2003		- Added part ID for 2L00M maskset.
V01.20	01 Sep 2003	01 Sep 2003		- Added part ID for 3L00M maskset. - Added cycle definition to 'CPU 12 Block Description'. - Diagram 'Clock Connections': Connected Bus Clock to HCS12 Core. - Corrected 'Background Debug Module' to 'HCS12 Breakpoint' at address $0028 - $002F in table 1-1. - Corrected 'Blank Check Time Flash' value in table 'NVM Timing Characteristics' - Added EXTAL pin VIH, VIL and EXTAL pin hysteresis value to 'Oscillator Characteristics'. Updated oscillator description and table note.
V01.21	08 Mar 2004	08 Mar 2004		- Added part ID for 4L00M maskset. - Corrected pin name KWP5 in device pinout.
V01.22	23 Aug 2004	23 Aug 2004		- Updated $V_{IH,EXTAL}$ and $V_{IL,EXTAL}$ in table 'Oscillator Characteristics' - Removed item 'Oscillator' from table 'Operating Conditions' as already covered in table 'Oscillator Characteristics'
V01.23	09 Feb 2005	09 Feb 2005		- Corrected Flash Row Programming Time in NVM Timing Characteristics
V01.24	01 Apr 2005	01 Apr 2005		- Changed T_{Javg} and added footnote to data retention time in NVM Reliability Characteristics
V01.25	05 Jul 2005	05 Jul 2005		- Updated NVM Reliability Characteristics

Table of Contents

Section 1 Introduction

Section 2 Signal Description

Section 3 System Clock Description

Section 4 Modes of Operation

Section 5 Resets and Interrupts

Section 6 HCS12 Core Block Description

Section 20 Port Integration Module (PIM) Block Description

Section 21 Voltage Regulator (VREG) Block Description

Section 22 Printed Circuit Board Layout Proposal

Appendix A Electrical Characteristics

Appendix B Package Information

List of Figures

List of Tables

Derivative Differences and Document References

Derivative Differences

Table 0-1 shows the availability of peripheral modules on the various derivatives. For details about the compatibility within the MC9S12D-Family refer also to engineering bulletin EB386.

Table 0-1 Derivative Differences[1]

Modules	MC9S12DP512	MC9S12DT512	MC9S12DJ512	MC9S12A512
# of CANs	5	3	2	0
CAN0	✓	✓	✓	—
CAN1	✓	✓	—	—
CAN2	✓	—	—	—
CAN3	✓	—	—	—
CAN4	✓	✓	✓	—
J1850/BDLC	✓	—	✓	—
Package	112 LQFP	112 LQFP	112 LQFP	112 LQFP
Package Code	PV	PV	PV	PV
Mask set	L00M	L00M	L00M	L00M
Temp Options	M, V, C	M, V, C	M, V, C	C
Notes	An errata exists contact Sales Office	An errata exists contact Sales Office	An errata exists contact Sales Office	An errata exists contact Sales Office

NOTES:
1. ✓: Available for this device, —: Not available for this device

The following figure provides an ordering number example for the MC9S12D-Family devices.

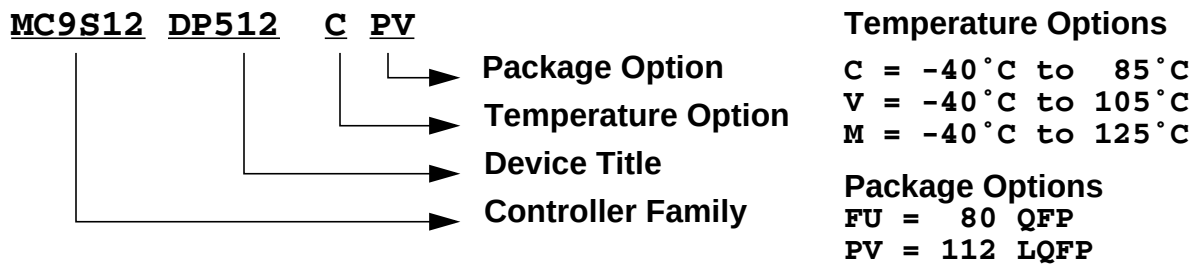


Figure 0-1 Order Part Number Example

The following items should be considered when using a derivative (**Table 0-1**):

- **Registers**
  - Do not write or read CAN0 registers (after reset: address range $0140 - $017F), if using a derivative without CAN0.
  - Do not write or read CAN1registers (after reset: address range $0180 - $01BF), if using a derivative without CAN1.
  - Do not write or read CAN2 registers (after reset: address range $01C0 - $01FF), if using a derivative without CAN2.
  - Do not write or read CAN3 registers (after reset: address range $0200 - $023F), if using a derivative without CAN3.
  - Do not write or read CAN4 registers (after reset: address range $0280 - $02BF), if using a derivative without CAN4.
  - Do not write or read BDLC registers (after reset: address range $00E8 - $00EF), if using a derivative without BDLC.
- **Interrupts**
  - Fill the four CAN0 interrupt vectors ($FFB0 - $FFB7) according to your coding policies for unused interrupts, if using a derivative without CAN0.
  - Fill the four CAN1 interrupt vectors ($FFA8 - $FFAF) according to your coding policies for unused interrupts, if using a derivative without CAN1.
  - Fill the four CAN2 interrupt vectors ($FFA0 - $FFA7) according to your coding policies for unused interrupts, if using a derivative without CAN2.
  - Fill the four CAN3 interrupt vectors ($FF98 - $FF9F) according to your coding policies for unused interrupts, if using a derivative without CAN3.
  - Fill the four CAN4 interrupt vectors ($FF90 - $FF97) according to your coding policies for unused interrupts, if using a derivative without CAN4.
  - Fill the BDLC interrupt vector ($FFC2, $FFC3) according to your coding policies for unused interrupts, if using a derivative without BDLC.
- **Ports**
  - The CAN0 pin functionality (TXCAN0, RXCAN0) is not available on port PJ7, PJ6, PM5, PM4, PM3, PM2, PM1 and PM0, if using a derivative without CAN0.
  - The CAN1 pin functionality (TXCAN1, RXCAN1) is not available on port PM3 and PM2, if using a derivative without CAN1.
  - The CAN2 pin functionality (TXCAN2, RXCAN2) is not available on port PM5 and PM4, if using a derivative without CAN2.
  - The CAN3 pin functionality (TXCAN3, RXCAN3) is not available on port PM7 and PM6, if using a derivative without CAN3.

- The CAN4 pin functionality (TXCAN4, RXCAN4) is not available on port PJ7, PJ6, PM7, PM6, PM5 and PM4, if using a derivative without CAN0.
- The BDLC pin functionality (TXB, RXB) is not available on port PM1 and PM0, if using a derivative without BDLC.
- Do not write MODRR1 and MODRR0 bits of Module Routing Register (PIM_9DP256 Block Guide), if using a derivative without CAN0.
- Do not write MODRR3 and MODRR2 bits of Module Routing Register (PIM_9DP256 Block Guide), if using a derivative without CAN4.

Document References

The Device Guide provides information about the MC9S12DP512 device made up of standard HCS12 blocks and the HCS12 processor core.

This document is part of the customer documentation. A complete set of device manuals also includes the individual Block Guides of the implemented modules. In an effort to reduce redundancy, all module specific information is located only in the respective Block Guide. If applicable, special implementation details of the module are given in the block description sections of this document.

See **Table 0-2** for names and versions of the referenced documents throughout the Device Guide.

Table 0-2 Document References

Block Guide	Version	Document Order Number
HCS12 CPU Reference Manual	V02	S12CPUV2/D
HCS12 Module Mapping Control (MMC) Block Guide	V04	S12MMCV4/D
HCS12 Multiplexed External Bus Interface (MEBI) Block Guide	V03	S12MEBIV3/D
HCS12 Interrupt (INT) Block Guide	V01	S12INTV1/D
HCS12 Background Debug (BDM) Block Guide	V04	S12BDMV4/D
HCS12 Breakpoint (BKP) Block Guide	V01	S12BKPV1/D
Clock and Reset Generator (CRG) Block Guide	V04	S12CRGV4/D
Enhanced Capture Timer 16 Bit 8 Channel (ECT_16B8C) Block Guide	V01	S12ECT16B8V1/D
Analog to Digital Converter 10 Bit 8 Channel (ATD_10B8C) Block Guide	V02	S12ATD10B8CV2/D
Inter IC Bus (IIC) Block Guide	V02	S12IICV2/D
Asynchronous Serial Interface (SCI) Block Guide	V02	S12SCIV2/D
Serial Peripheral Interface (SPI) Block Guide	V03	S12SPIV3/D
Pulse Width Modulator 8 Bit 8 Channel (PWM_8B8C) Block Guide	V01	S12PWM8B8CV1/D
512K Byte Flash (FTS512K4) Block Guide	V01	S12FTS512K4V1/D
4K Byte EEPROM (EETS4K) Block Guide	V02	S12EETS4KV2/D
Byte Level Data Link Controller -J1850 (BDLC) Block Guide	V01	S12BDLCV1/D
Motorola Scalable CAN (MSCAN) Block Guide	V02	S12MSCANV2/D
Voltage Regulator (VREG) Block Guide	V01	S12VREGV1/D
Port Integration Module (PIM_9DP256) Block Guide[1]	V03	S12DP256PIMV3/D
Oscillator (OSC) Block Guide	V02	S12OSCV2/D

NOTES:
1. Reused due to functional equivalence.

Section 1 Introduction

1.1 Overview

The MC9S12DP512 microcontroller unit (MCU) is a 16-bit device composed of standard on-chip peripherals including a 16-bit central processing unit (HCS12 CPU), 512K bytes of Flash EEPROM, 14K bytes of RAM, 4K bytes of EEPROM, two asynchronous serial communications interfaces (SCI), three serial peripheral interfaces (SPI), an 8-channel IC/OC enhanced capture timer, two 8-channel, 10-bit analog-to-digital converters (ADC), an 8-channel pulse-width modulator (PWM), a digital Byte Data Link Controller (BDLC), 29 discrete digital I/O channels (Port A, Port B, Port K and Port E), 20 discrete digital I/O lines with interrupt and wake up capability, five CAN 2.0 A, B software compatible modules (MSCAN12), and an Inter-IC Bus. The MC9S12DP512 has full 16-bit data paths throughout. However, the external bus can operate in an 8-bit narrow mode so single 8-bit wide memory can be interfaced for lower cost systems. The inclusion of a PLL circuit allows power consumption and performance to be adjusted to suit operational requirements.

1.2 Features

- HCS12 Core
  - 16-bit HCS12 CPU
    i. Upward compatible with M68HC11 instruction set
    ii. Interrupt stacking and programmer’s model identical to M68HC11
    iii. Instruction queue
    iv. Enhanced indexed addressing
  - MEBI (Multiplexed External Bus Interface)
  - MMC (Module Mapping Control)
  - INT (Interrupt control)
  - BKP (Breakpoints)
  - BDM (Background Debug Mode)
- CRG (Clock and Reset Generation)
  - Low current Colpitts oscillator or
  - Pierce oscillator
  - PLL
  - COP watchdog
  - Real Time Interrupt
  - Clock Monitor
- 8-bit and 4-bit ports with interrupt functionality

  - Digital filtering
  - Programmable rising or falling edge trigger
- Memory
  - 512K Flash EEPROM
  - 4K byte EEPROM
  - 14K byte RAM
- Two 8-channel Analog-to-Digital Converters
  - 10-bit resolution
  - External conversion trigger capability
- Five 1M bit per second, CAN 2.0 A, B software compatible modules
  - Five receive and three transmit buffers
  - Flexible identifier filter programmable as 2 x 32 bit, 4 x 16 bit or 8 x 8 bit
  - Four separate interrupt channels for Rx, Tx, error and wake-up
  - Low-pass filter wake-up function
  - Loop-back for self test operation
- Enhanced Capture Timer
  - 16-bit main counter with 7-bit prescaler
  - 8 programmable input capture or output compare channels
  - Four 8-bit or two 16-bit pulse accumulators
- 8 PWM channels
  - Programmable period and duty cycle
  - 8-bit 8-channel or 16-bit 4-channel
  - Separate control for each pulse width and duty cycle
  - Center-aligned or left-aligned outputs
  - Programmable clock select logic with a wide range of frequencies
  - Fast emergency shutdown input
  - Usable as interrupt inputs
- Serial interfaces
  - Two asynchronous Serial Communications Interfaces (SCI)
  - Three Synchronous Serial Peripheral Interface (SPI)
- Byte Data Link Controller (BDLC)
  - SAE J1850 Class B Data Communications Network Interface Compatible and ISO Compatible for Low-Speed (<125 Kbps) Serial Data Communications in Automotive Applications

- Inter-IC Bus (IIC)
  - Compatible with I^2C Bus standard
  - Multi-master operation
  - Software programmable for one of 256 different serial clock frequencies
- 112-Pin LQFP package
  - I/O lines with 5V input and drive capability
  - 5V A/D converter inputs
  - Operation at 50MHz equivalent to 25MHz Bus Speed over -40˚C <= T_A <= 125˚C
  - Development support
  - Single-wire background debug™ mode (BDM)
  - On-chip hardware breakpoints

1.3 Modes of Operation

User modes

- Normal and Emulation Operating Modes
  - Normal Single-Chip Mode
  - Normal Expanded Wide Mode
  - Normal Expanded Narrow Mode
  - Emulation Expanded Wide Mode
  - Emulation Expanded Narrow Mode
- Special Operating Modes
  - Special Single-Chip Mode with active Background Debug Mode
  - Special Test Mode (**Motorola use only**)
  - Special Peripheral Mode (**Motorola use only**)

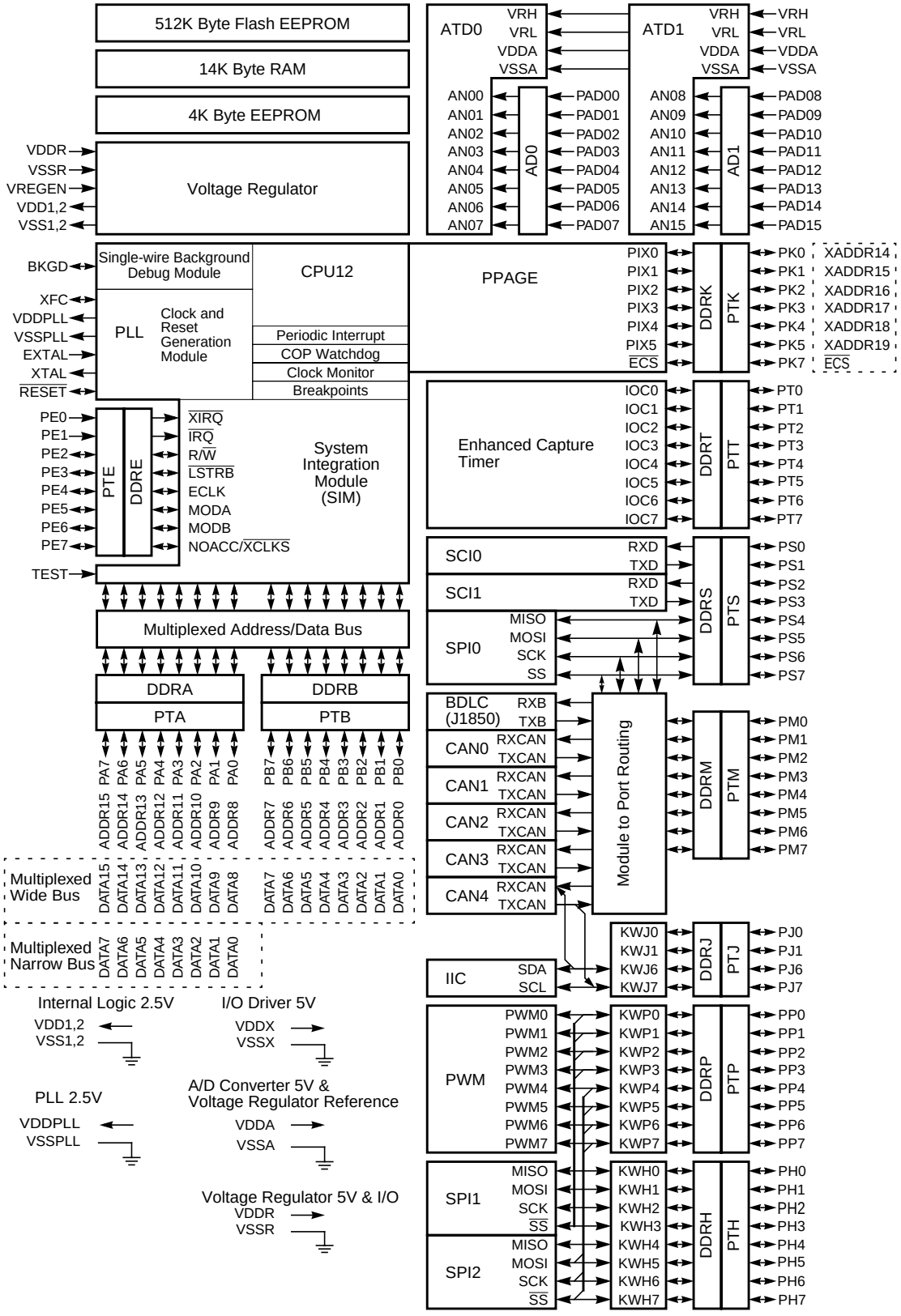
Low power modes

- Stop Mode
- Pseudo Stop Mode
- Wait Mode

1.4 Block Diagram

Figure 1-1 shows a block diagram of the MC9S12DP512 device.

Figure 1-1 MC9S12DP512 Block Diagram



1.5 Device Memory Map

Table 1-1 and **Figure 1-2** show the device memory map of the MC9S12DP512 after reset. Note that after reset the bottom 1k of the EEPROM ($0000 - $03FF) are hidden by the register space

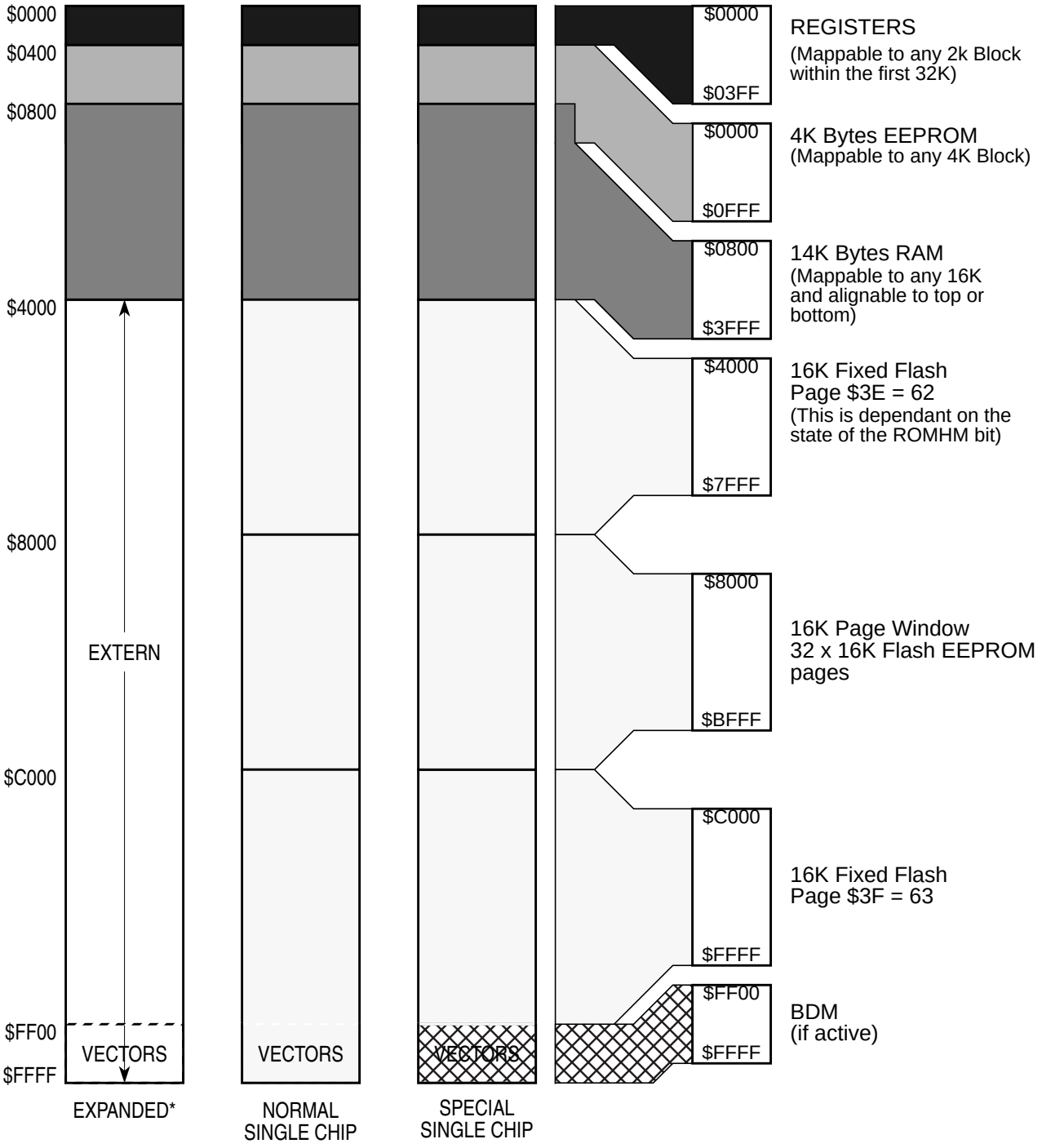
Table 1-1 Device Memory Map

Address	Module	Size (Bytes)
$0000 - $000F	HCS12 Multiplexed External Bus Interface	16
$0010 - $0014	HCS12 Module Mapping Control	5
$0015 - $0016	HCS12 Interrupt	2
$0017 - $0019	Reserved	3
$001A - $001B	Device ID register (PARTID)	2
$001C - $001D	HCS12 Module Mapping Control	2
$001E	HCS12 Multiplexed External Bus Interface	1
$001F	HCS12 Interrupt	1
$0020 - $0027	Reserved	8
$0028 - $002F	HCS12 Breakpoint	8
$0030 - $0031	HCS12 Module Mapping Control	2
$0032 - $0033	HCS12 Multiplexed External Bus Interface	2
$0034 - $003F	Clock and Reset Generator (PLL, RTI, COP)	12
$0040 - $007F	Enhanced Capture Timer 16-bit 8 channels	64
$0080 - $009F	Analog to Digital Converter 10-bit 8 channels (ATD0)	32
$00A0 - $00C7	Pulse Width Modulator 8-bit 8 channels (PWM)	40
$00C8 - $00CF	Serial Communications Interface 0 (SCI0)	8
$00D0 - $00D7	Serial Communications Interface 0 (SCI1)	8
$00D8 - $00DF	Serial Peripheral Interface (SPI0)	8
$00E0 - $00E7	Inter IC Bus	8
$00E8 - $00EF	Byte Data Link Controller (BDLC)	8
$00F0 - $00F7	Serial Peripheral Interface (SPI1)	8
$00F8 - $00FF	Serial Peripheral Interface (SPI2)	8
$0100- $010F	Flash Control Register	16
$0110 - $011B	EEPROM Control Register	12
$011C - $011F	Reserved	4
$0120 - $013F	Analog to Digital Converter 10-bit 8 channels (ATD1)	32
$0140 - $017F	Motorola Scalable Can (CAN0)	64
$0180 - $01BF	Motorola Scalable Can (CAN1)	64
$01C0 - $01FF	Motorola Scalable Can (CAN2)	64
$0200 - $023F	Motorola Scalable Can (CAN3)	64
$0240 - $027F	Port Integration Module (PIM)	64
$0280 - $02BF	Motorola Scalable Can (CAN4)	64
$02C0 - $03FF	Reserved	320
$0000 - $0FFF	EEPROM array	4096
$0800 - $3FFF	RAM array	14336
$4000 - $7FFF	Fixed Flash EEPROM array incl. 1K, 2K, 4K or 8K Protected Sector at start	16384

Table 1-1 Device Memory Map

Address	Module	Size (Bytes)
$8000 - $BFFF	Flash EEPROM Page Window	16384
$C000 - $FFFF	Fixed Flash EEPROM array incl. 2K, 4K, 8K or 16K Protected Sector at end and 256 bytes of Vector Space at $FF80 - $FFFF	16384

Figure 1-2 MC9S12DP512 Memory Map



* Assuming that a '0' was driven onto port K bit 7 during MCU is reset into normal expanded wide or narrow mode.

1.5.1 Detailed Register Map

$0000 - $000F MEBI map 1 of 3 (HCS12 Multiplexed External Bus Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0000	PORTA	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0001	PORTB	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0002	DDRA	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0003	DDRB	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0004 - $0007	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0008	PORTE	Read:	Bit 7	6	5	4	3	2	Bit 1	Bit 0
		Write:								
$0009	DDRE	Read:	Bit 7	6	5	4	3	Bit 2	0	0
		Write:								
$000A	PEAR	Read:	NOACCE	0	PIPOE	NECLK	LSTRE	RDWE	0	0
		Write:								
$000B	MODE	Read:	MODC	MODB	MODA	0	IVIS	0	EMK	EME
		Write:								
$000C	PUCR	Read:	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE
		Write:								
$000D	RDRIV	Read:	RDPK	0	0	RDPE	0	0	RDPB	RDPA
		Write:								
$000E	EBICTL	Read:	0	0	0	0	0	0	0	ESTR
		Write:								
$000F	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0010 - $0014 MMC map 1 of 4 (HCS12 Module Mapping Control)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0010	INITRM	Read:	RAM15	RAM14	RAM13	RAM12	RAM11	0	0	RAMHAL
		Write:								
$0011	INITRG	Read:	0	REG14	REG13	REG12	REG11	0	0	0
		Write:								
$0012	INITEE	Read:	EE15	EE14	EE13	EE12	EE11	0	0	EEON
		Write:								
$0013	MISC	Read:	0	0	0	0	EXSTR1	EXSTR0	ROMHM	ROMON
		Write:								
$0014	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0015 - $0016 INT map 1 of 2 (HCS12 Interrupt)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0015	ITCR	Read:	0	0	0	WRINT	ADR3	ADR2	ADR1	ADR0
		Write:								
$0016	ITEST	Read:	INTE	INTC	INTA	INT8	INT6	INT4	INT2	INT0
		Write:								

$0017 - $0019 Reserved

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0017-$0019	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$001A - $001B Device ID Register (Table 1-3)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$001A	PARTIDH	Read:	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8
		Write:								
$001B	PARTIDL	Read:	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
		Write:								

$001C - $001D MMC map 3 of 4 (HCS12 Module Mapping Control, Table 1-4)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$001C	MEMSIZ0	Read:	reg_sw0	0	eep_sw1	eep_sw0	0	ram_sw2	ram_sw1	ram_sw0
		Write:								
$001D	MEMSIZ1	Read:	rom_sw1	rom_sw0	0	0	0	0	pag_sw1	pag_sw0
		Write:								

$001E - $001E MEBI map 2 of 3 (HCS12 Multiplexed External Bus Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$001E	INTCR	Read:	IRQE	IRQEN	0	0	0	0	0	0
		Write:								

$001F - $001F INT map 2 of 2 (HCS12 Interrupt)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$001F	HPRIO	Read:	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0
		Write:								

$0020 - $0027 Reserved

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0020 - $0027	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0028 - $002F BKP (HCS12 Breakpoint)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0028	BKPCT0	Read:	BKEN	BKFULL	BKBDM	BKTAG	0	0	0	0
		Write:								
$0029	BKPCT1	Read:	BK0MBH	BK0MBL	BK1MBH	BK1MBL	BK0RWE	BK0RW	BK1RWE	BK1RW
		Write:								
$002A	BKP0X	Read:	0	0	BK0V5	BK0V4	BK0V3	BK0V2	BK0V1	BK0V0
		Write:								
$002B	BKP0H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$002C	BKP0L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$002D	BKP1X	Read:	0	0	BK1V5	BK1V4	BK1V3	BK1V2	BK1V1	BK1V0
		Write:								
$002E	BKP1H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$002F	BKP1L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$0030 - $0031 MMC map 4 of 4 (HCS12 Module Mapping Control)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0030	PPAGE	Read:	0	0	PIX5	PIX4	PIX3	PIX2	PIX1	PIX0
		Write:								
$0031	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0032 - $0033 MEBI map 3 of 3 (HCS12 Multiplexed External Bus Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0032	PORTK	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0033	DDRK	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$0034 - $003F CRG (Clock and Reset Generator)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0034	SYNR	Read:	0	0	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0
		Write:								
$0035	REFDV	Read:	0	0	0	0	REFDV3	REFDV2	REFDV1	REFDV0
		Write:								
$0036	CTFLG Test Only	Read:	TOUT7	TOUT6	TOUT5	TOUT4	TOUT3	TOUT2	TOUT1	TOUT0
		Write:								
$0037	CRGFLG	Read:	RTIF	PROF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM
		Write:								
$0038	CRGINT	Read:	RTIE	0	0	LOCKIE	0	0	SCMIE	0
		Write:								

$0034 - $003F CRG (Clock and Reset Generator)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0039	CLKSEL	Read:	PLLSEL	PSTP	SYSWAI	ROAWAI	PLLWAI	CWAI	RTIWAI	COPWAI
		Write:								
$003A	PLLCTL	Read:	CME	PLLON	AUTO	ACQ	0	PRE	PCE	SCME
		Write:								
$003B	RTICTL	Read:	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
		Write:								
$003C	COPCTL	Read:	WCOP	RSBCK	0	0	0	CR2	CR1	CR0
		Write:								
$003D	FORBYP Test Only	Read:	RTIBYP	COPBYP	0	PLLBYP	0	0	FCM	0
		Write:								
$003E	CTCTL Test Only	Read:	TCTL7	TCTL6	TCTL5	TCTL4	TCLT3	TCTL2	TCTL1	TCTL0
		Write:								
$003F	ARMCOP	Read:	0	0	0	0	0	0	0	0
		Write:	Bit 7	6	5	4	3	2	1	Bit 0

$0040 - $007F ECT (Enhanced Capture Timer 16 Bit 8 Channels)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0040	TIOS	Read:	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
		Write:								
$0041	CFORC	Read:	0	0	0	0	0	0	0	0
		Write:	FOC7	FOC6	FOC5	FOC4	FOC3	FOC2	FOC1	FOC0
$0042	OC7M	Read:	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
		Write:								
$0043	OC7D	Read:	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
		Write:								
$0044	TCNT (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0045	TCNT (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0046	TSCR1	Read:	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
		Write:								
$0047	TTOV	Read:	TOV7	TOV6	TOV5	TOV4	TOV3	TOV2	TOV1	TOV0
		Write:								
$0048	TCTL1	Read:	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
		Write:								
$0049	TCTL2	Read:	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
		Write:								
$004A	TCTL3	Read:	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
		Write:								
$004B	TCTL4	Read:	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
		Write:								
$004C	TIE	Read:	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
		Write:								
$004D	TSCR2	Read:	TOI	0	0	0	TCRE	PR2	PR1	PR0
		Write:								
$004E	TFLG1	Read:	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
		Write:								

$0040 - $007F ECT (Enhanced Capture Timer 16 Bit 8 Channels)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$004F	TFLG2	Read:	TOF	0	0	0	0	0	0	0
		Write:								
$0050	TC0 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0051	TC0 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0052	TC1 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0053	TC1 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0054	TC2 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0055	TC2 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0056	TC3 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0057	TC3 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0058	TC4 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0059	TC4 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$005A	TC5 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$005B	TC5 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$005C	TC6 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$005D	TC6 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$005E	TC7 (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$005F	TC7 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0060	PACTL	Read:	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
		Write:								
$0061	PAFLG	Read:	0	0	0	0	0	0	PAOVF	PAIF
		Write:								
$0062	PACN3 (hi)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0063	PACN2 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0064	PACN1 (hi)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0065	PACN0 (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0066	MCCTL	Read:	MCZI	MODMC	RDMCL	0	0	MCEN	MCPR1	MCPR0
		Write:				ICLAT	FLMC			
$0067	MCFLG	Read:	MCZF	0	0	0	POLF3	POLF2	POLF1	POLF0
		Write:								

$0040 - $007F ECT (Enhanced Capture Timer 16 Bit 8 Channels)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0068	ICPAR	Read:	0	0	0	0	PA3EN	PA2EN	PA1EN	PA0EN
		Write:								
$0069	DLYCT	Read:	0	0	0	0	0	0	DLY1	DLY0
		Write:								
$006A	ICOVW	Read:	NOVW7	NOVW6	NOVW5	NOVW4	NOVW3	NOVW2	NOVW1	NOVW0
		Write:								
$006B	ICSYS	Read:	SH37	SH26	SH15	SH04	TFMOD	PACMX	BUFEN	LATQ
		Write:								
$006C	Reserved	Read:								
		Write:								
$006D	TIMTST Test Only	Read:	0	0	0	0	0	0	TCBYP	0
		Write:								
$006E - $006F	Reserved	Read:								
		Write:								
$0070	PBCTL	Read:	0	PBEN	0	0	0	0	PBOVI	0
		Write:								
$0071	PBFLG	Read:	0	0	0	0	0	0	PBOVF	0
		Write:								
$0072	PA3H	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0073	PA2H	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0074	PA1H	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0075	PA0H	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0076	MCCNT (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0077	MCCNT (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0078	TC0H (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0079	TC0H (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$007A	TC1H (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$007B	TC1H (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$007C	TC2H (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$007D	TC2H (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$007E	TC3H (hi)	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$007F	TC3H (lo)	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$0080 - $009F ATD0 (Analog to Digital Converter 10 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0080	ATD0CTL0	Read:	0	0	0	0	0	0	0	0
		Write:								
$0081	ATD0CTL1	Read:	0	0	0	0	0	0	0	0
		Write:								
$0082	ATD0CTL2	Read:	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF
		Write:								
$0083	ATD0CTL3	Read:	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
		Write:								
$0084	ATD0CTL4	Read:	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
		Write:								
$0085	ATD0CTL5	Read:	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
		Write:								
$0086	ATD0STAT0	Read:	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0
		Write:								
$0087	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0088	ATD0TEST0	Read:	0	0	0	0	0	0	0	0
		Write:								
$0089	ATD0TEST1	Read:	0	0	0	0	0	0	0	SC
		Write:								
$008A	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$008B	ATD0STAT1	Read:	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
		Write:								
$008C	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$008D	ATD0DIEN	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$008E	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$008F	PORTAD0	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0090	ATD0DR0H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0091	ATD0DR0L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0092	ATD0DR1H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0093	ATD0DR1L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0094	ATD0DR2H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0095	ATD0DR2L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0096	ATD0DR3H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0097	ATD0DR3L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0098	ATD0DR4H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								

$0080 - $009F ATD0 (Analog to Digital Converter 10 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0099	ATD0DR4L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$009A	ATD0DR5H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$009B	ATD0DR5L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$009C	ATD0DR6H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$009D	ATD0DR6L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$009E	ATD0DR7H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$009F	ATD0DR7L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$00A0 - $00C7 PWM (Pulse Width Modulator 8 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00A0	PWME	Read:	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0
		Write:								
$00A1	PWMPOL	Read:	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0
		Write:								
$00A2	PWMCLK	Read:	PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0
		Write:								
$00A3	PWMPRCLK	Read:	0	PCKB2	PCKB1	PCKB0	0	PCKA2	PCKA1	PCKA0
		Write:								
$00A4	PWMCAE	Read:	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0
		Write:								
$00A5	PWMCTL	Read:	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
		Write:								
$00A6	PWMTST Test Only	Read:	0	0	0	0	0	0	0	0
		Write:								
$00A7	PWMPRSC	Read:	0	0	0	0	0	0	0	0
		Write:								
$00A8	PWMSCLA	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$00A9	PWMSCLB	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$00AA	PWMSCNTA	Read:	0	0	0	0	0	0	0	0
		Write:								
$00AB	PWMSCNTB	Read:	0	0	0	0	0	0	0	0
		Write:								
$00AC	PWMCNT0	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00AD	PWMCNT1	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00AE	PWMCNT2	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0

$00A0 - $00C7 PWM (Pulse Width Modulator 8 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00AF	PWMCNT3	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00B0	PWMCNT4	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00B1	PWMCNT5	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00B2	PWMCNT6	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00B3	PWMCNT7	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:	0	0	0	0	0	0	0	0
$00B4	PWMPER0	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00B5	PWMPER1	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00B6	PWMPER2	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00B7	PWMPER3	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00B8	PWMPER4	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00B9	PWMPER5	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BA	PWMPER6	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BB	PWMPER7	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BC	PWMDTY0	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BD	PWMDTY1	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BE	PWMDTY2	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00BF	PWMDTY3	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00C0	PWMDTY4	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00C1	PWMDTY5	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00C2	PWMDTY6	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00C3	PWMDTY7	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00C4	PWMSDN	Read:	PWMIF	PWMIE	PWM RSTRT	PWMLVL	0	PWM7IN	PWM7 INL	PWM7 ENA
		Write:								
$00C5 - $00C7	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$00C8 - $00CF SCI0 (Asynchronous Serial Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00C8	SCI0BDH	Read:	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
		Write:								
$00C9	SCI0BDL	Read:	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
		Write:								
$00CA	SC0CR1	Read:	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
		Write:								
$00CB	SCI0CR2	Read:	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
		Write:								
$00CC	SCI0SR1	Read:	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
		Write:								
$00CD	SC0SR2	Read:	0	0	0	0	0	BRK13	TXDIR	RAF
		Write:								
$00CE	SCI0DRH	Read:	R8	T8	0	0	0	0	0	0
		Write:								
$00CF	SCI0DRL	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0

$00D0 - $00D7 SCI1 (Asynchronous Serial Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00D0	SCI1BDH	Read:	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
		Write:								
$00D1	SCI1BDL	Read:	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
		Write:								
$00D2	SC1CR1	Read:	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
		Write:								
$00D3	SCI1CR2	Read:	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
		Write:								
$00D4	SCI1SR1	Read:	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
		Write:								
$00D5	SC1SR2	Read:	0	0	0	0	0	BRK13	TXDIR	RAF
		Write:								
$00D6	SCI1DRH	Read:	R8	T8	0	0	0	0	0	0
		Write:								
$00D7	SCI1DRL	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0

$00D8 - $00DF SPI0 (Serial Peripheral Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00D8	SPI0CR1	Read:	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
		Write:								
$00D9	SPI0CR2	Read:	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
		Write:								
$00DA	SPI0BR	Read:	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
		Write:								
$00DB	SPI0SR	Read:	SPIF	0	SPTEF	MODF	0	0	0	0
		Write:								

$00D8 - $00DF SPI0 (Serial Peripheral Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00DC	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$00DD	SPI0DR	Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
$00DE - $00DF	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$00E0 - $00E7 IIC (Inter IC Bus)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00E0	IBAD	Read: Write:	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	0
$00E1	IBFD	Read: Write:	IBC7	IBC6	IBC5	IBC4	IBC3	IBC2	IBC1	IBC0
$00E2	IBCR	Read:	IBEN	IBIE	MS/$\overline{SL}$	TX/$\overline{RX}$	TXAK	0	0	IBSWAI
		Write:						RSTA		
$00E3	IBSR	Read:	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
		Write:								
$00E4	IBDR	Read: Write:	D7	D6	D5	D4	D3	D2	D1	D 0
$00E5 - $00E7	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$00E8 - $00EF BDLC (Bytelevel Data Link Controller J1850)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00E8	DLCBCR1	Read:	IMSG	CLKS	0	0	0	0	IE	WCM
		Write:								
$00E9	DLCBSVR	Read:	0	0	I3	I2	I1	I0	0	0
		Write:								
$00EA	DLCBCR2	Read: Write:	SMRST	DLOOP	RX4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
$00EB	DLCBDR	Read: Write:	D7	D6	D5	D4	D3	D2	D1	D0
$00EC	DLCBARD	Read:	0	RXPOL	0	0	BO3	BO2	BO1	BO0
		Write:								
$00ED	DLCBRSR	Read:	0	0	R5	R4	R3	R2	R1	R0
		Write:								
$00EE	DLCSCR	Read:	0	0	0	BDLCE	0	0	0	0
		Write:								
$00EF	DLCBSTAT	Read:	0	0	0	0	0	0	0	IDLE
		Write:								

$00F0 - $00F7 SPI1 (Serial Peripheral Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00F0	SPI1CR1	Read:	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
		Write:								
$00F1	SPI1CR2	Read:	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
		Write:								
$00F2	SPI1BR	Read:	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
		Write:								
$00F3	SPI1SR	Read:	SPIF	0	SPTEF	MODF	0	0	0	0
		Write:								
$00F4	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$00F5	SPI1DR	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$00F6 - $00F7	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$00F8 - $00FF SPI2 (Serial Peripheral Interface)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$00F8	SPI2CR1	Read:	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
		Write:								
$00F9	SPI2CR2	Read:	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
		Write:								
$00FA	SPI2BR	Read:	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
		Write:								
$00FB	SPI2SR	Read:	SPIF	0	SPTEF	MODF	0	0	0	0
		Write:								
$00FC	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$00FD	SPI2DR	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$00FE - $00FF	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0100 - $010F Flash Control Register (fts512k4)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0100	FCLKDIV	Read:	FDIVLD	PRDIV8	FDIV5	FDIV4	FDIV3	FDIV2	FDIV1	FDIV0
		Write:								
$0101	FSEC	Read:	KEYEN1	KEYEN0	NV5	NV4	NV3	NV2	SEC1	SEC0
		Write:								
$0102	FTSTMOD	Read:	0	0	0	WRALL	0	0	0	0
		Write:								
$0103	FCNFG	Read:	CBEIE	CCIE	KEYACC	0	0	0	BKSEL1	BKSEL0
		Write:								
$0104	FPROT	Read:	FPOPEN	NV6	FPHDIS	FPHS1	FPHS0	FPLDIS	FPLS1	FPLS0
		Write:								
$0105	FSTAT	Read:	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
		Write:								

$0100 - $010F Flash Control Register (fts512k4)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0106	FCMD	Read:	0	CMDB6	CMDB5	0	0	CMDB2	0	CMDB0
		Write:								
$0107	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0108	FADDRHI	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0109	FADDRLO	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$010A	FDATAHI	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$010B	FDATALO	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$010C - $010F	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0110 - $011B EEPROM Control Register (eets4k)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0110	ECLKDIV	Read:	EDIVLD	PRDIV8	EDIV5	EDIV4	EDIV3	EDIV2	EDIV1	EDIV0
		Write:								
$0111 - $0112	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0113	ECNFG	Read:	CBEIE	CCIE	0	0	0	0	0	0
		Write:								
$0114	EPROT	Read:	EPOPEN	NV6	NV5	NV4	EPDIS	EP2	EP1	EP0
		Write:								
$0115	ESTAT	Read:	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
		Write:								
$0116	ECMD	Read:	0	CMDB6	CMDB5	0	0	CMDB2	0	CMDB0
		Write:								
$0117	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0118	EADDRHI	Read:	0	0	0	0	0	10	9	Bit 8
		Write:								
$0119	EADDRLO	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$011A	EDATAHI	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$011B	EDATALO	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$011C - $011F Reserved for RAM Control Register

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$011C - $011F	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

$0120 - $013F ATD1 (Analog to Digital Converter 10 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0120	ATD1CTL0	Read:	0	0	0	0	0	0	0	0
		Write:								
$0121	ATD1CTL1	Read:	0	0	0	0	0	0	0	0
		Write:								
$0122	ATD1CTL2	Read:	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIG	ASCIE	ASCIF
		Write:								
$0123	ATD1CTL3	Read:	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
		Write:								
$0124	ATD1CTL4	Read:	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
		Write:								
$0125	ATD1CTL5	Read:	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
		Write:								
$0126	ATD1STAT0	Read:	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0
		Write:								
$0127	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0128	ATD1TEST0	Read:	0	0	0	0	0	0	0	0
		Write:								
$0129	ATD1TEST1	Read:	0	0	0	0	0	0	0	SC
		Write:								
$012A	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$012B	ATD1STAT1	Read:	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
		Write:								
$012C	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$012D	ATD1DIEN	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$012E	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$012F	PORTAD1	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0130	ATD1DR0H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0131	ATD1DR0L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0132	ATD1DR1H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0133	ATD1DR1L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0134	ATD1DR2H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0135	ATD1DR2L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0136	ATD1DR3H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$0137	ATD1DR3L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$0138	ATD1DR4H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								

$0120 - $013F ATD1 (Analog to Digital Converter 10 Bit 8 Channel)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0139	ATD1DR4L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$013A	ATD1DR5H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$013B	ATD1DR5L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$013C	ATD1DR6H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$013D	ATD1DR6L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								
$013E	ATD1DR7H	Read:	Bit 15	14	13	12	11	10	9	Bit 8
		Write:								
$013F	ATD1DR7L	Read:	Bit 7	6	5	4	3	2	1	Bit 0
		Write:								

$0140 - $017F CAN0 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0140	CAN0CTL0	Read:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
		Write:								
$0141	CAN0CTL1	Read:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
		Write:								
$0142	CAN0BTR0	Read:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
		Write:								
$0143	CAN0BTR1	Read:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
		Write:								
$0144	CAN0RFLG	Read:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
		Write:								
$0145	CAN0RIER	Read:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
		Write:								
$0146	CAN0TFLG	Read:	0	0	0	0	0	TXE2	TXE1	TXE0
		Write:								
$0147	CAN0TIER	Read:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
		Write:								
$0148	CAN0TARQ	Read:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
		Write:								
$0149	CAN0TAAK	Read:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
		Write:								
$014A	CAN0TBSEL	Read:	0	0	0	0	0	TX2	TX1	TX0
		Write:								
$014B	CAN0IDAC	Read:	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
		Write:								
$014C - $014D	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$014E	CAN0RXERR	Read:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
		Write:								
$014F	CAN0TXERR	Read:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
		Write:								
$0150 - $0153	CAN0IDAR0 - CAN0IDAR3	Read:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		Write:								

$0140 - $017F CAN0 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0154 - $0157	CAN0IDMR0 - CAN0IDMR3	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0158 - $015B	CAN0IDAR4 - CAN0IDAR7	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$015C - $015F	CAN0IDMR4 - CAN0IDMR7	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0160 - $016F	CAN0RXFG	Read:	FOREGROUND RECEIVE BUFFER see **Table 1-2**							
		Write:								
$0170 - $017F	CAN0TXFG	Read: Write:	FOREGROUND TRANSMIT BUFFER see **Table 1-2**							

Table 1-2 Detailed MSCAN Foreground Receive and Transmit Buffer Layout

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Extended ID	Read:	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
$xxx0	Standard ID	Read:	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	CANxRIDR0	Write:								
	Extended ID	Read:	ID20	ID19	ID18	SRR=1	IDE=1	ID17	ID16	ID15
$xxx1	Standard ID	Read:	ID2	ID1	ID0	RTR	IDE=0			
	CANxRIDR1	Write:								
	Extended ID	Read:	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
$xxx2	Standard ID	Read:								
	CANxRIDR2	Write:								
	Extended ID	Read:	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
$xxx3	Standard ID	Read:								
	CANxRIDR3	Write:								
$xxx4 - $xxxB	CANxRDSR0 - CANxRDSR7	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
$xxxC	CANRxDLR	Read:					DLC3	DLC2	DLC1	DLC0
		Write:								
$xxxD	Reserved	Read: Write:								
$xxxE	CANxRTSRH	Read:	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
		Write:								
$xxxF	CANxRTSRL	Read:	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
		Write:								
$xx10	Extended ID CANxTIDR0	Read: Write:	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Standard ID	Read: Write:	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
$xx11	Extended ID CANxTIDR1	Read: Write:	ID20	ID19	ID18	SRR=1	IDE=1	ID17	ID16	ID15
	Standard ID	Read: Write:	ID2	ID1	ID0	RTR	IDE=0			
$xx12	Extended ID CANxTIDR2	Read: Write:	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Standard ID	Read: Write:								

Table 1-2 Detailed MSCAN Foreground Receive and Transmit Buffer Layout

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$xx13	Extended ID CANxTIDR3	Read: Write:	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Standard ID	Read: Write:								
$xx14 - $xx1B	CANxTDSR0 - CANxTDSR7	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
$xx1C	CANxTDLR	Read: Write:					DLC3	DLC2	DLC1	DLC0
$xx1D	CANxTTBPR	Read: Write:	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
$xx1E	CANxTTSRH	Read:	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
		Write:								
$xx1F	CANxTTSRL	Read:	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
		Write:								

$0180 - $01BF CAN1 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0180	CAN1CTL0	Read: Write:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
$0181	CAN1CTL1	Read: Write:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
$0182	CAN1BTR0	Read: Write:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
$0183	CAN1BTR1	Read: Write:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
$0184	CAN1RFLG	Read: Write:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
$0185	CAN1RIER	Read: Write:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
$0186	CAN1TFLG	Read: Write:	0	0	0	0	0	TXE2	TXE1	TXE0
$0187	CAN1TIER	Read: Write:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
$0188	CAN1TARQ	Read: Write:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
$0189	CAN1TAAK	Read: Write:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
$018A	CAN1TBSEL	Read: Write:	0	0	0	0	0	TX2	TX1	TX0
$018B	CAN1IDAC	Read: Write:	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
$018C - $018D	Reserved	Read: Write:	0	0	0	0	0	0	0	0
$018E	CAN1RXERR	Read:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
		Write:								
$018F	CAN1TXERR	Read:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
		Write:								
$0190 - $0193	CAN1IDAR0 - CAN1IDAR3	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

$0180 - $01BF CAN1 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0194 - $0197	CAN1IDMR0 - CAN1IDMR3	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0198 - $019B	CAN1IDAR4 - CAN1IDAR7	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$019C - $019F	CAN1IDMR4 - CAN1IDMR7	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$01A0 - $01AF	CAN1RXFG	Read:	FOREGROUND RECEIVE BUFFER see **Table 1-2**							
		Write:								
$01B0 - $01BF	CAN1TXFG	Read: Write:	FOREGROUND TRANSMIT BUFFER see **Table 1-2**							

$01C0 - $01FF CAN2 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$01C0	CAN2CTL0	Read:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
		Write:								
$01C1	CAN2CTL1	Read:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
		Write:								
$01C2	CAN2BTR0	Read: Write:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
$01C3	CAN2BTR1	Read: Write:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
$01C4	CAN2RFLG	Read:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
		Write:								
$01C5	CAN2RIER	Read: Write:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
$01C6	CAN2TFLG	Read:	0	0	0	0	0	TXE2	TXE1	TXE0
		Write:								
$01C7	CAN2TIER	Read:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
		Write:								
$01C8	CAN2TARQ	Read:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
		Write:								
$01C9	CAN2TAAK	Read:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
		Write:								
$01CA	CAN2TBSEL	Read:	0	0	0	0	0	TX2	TX1	TX0
		Write:								
$01CB	CAN2IDAC	Read:	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
		Write:								
$01CC - $01CD	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$01CE	CAN2RXERR	Read:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
		Write:								
$01CF	CAN2TXERR	Read:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
		Write:								
$01D0 - $01D3	CAN2IDAR0 - CAN2IDAR3	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$01D4 - $01D7	CAN2IDMR0 - CAN2IDMR3	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0

$01C0 - $01FF CAN2 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$01D8 - $01DB	CAN2IDAR4 - CAN2IDAR7	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$01DC - $01DF	CAN2IDMR4 - CAN2IDMR7	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$01E0 - $01EF	CAN2RXFG	Read: Write:	FOREGROUND RECEIVE BUFFER see **Table 1-2**							
$01F0 - $01FF	CAN2TXFG	Read: Write:	FOREGROUND TRANSMIT BUFFER see **Table 1-2**							

$0200 - $023F CAN3 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0200	CAN3CTL0	Read: Write:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
$0201	CAN3CTL1	Read: Write:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
$0202	CAN3BTR0	Read: Write:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
$0203	CAN3BTR1	Read: Write:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
$0204	CAN3RFLG	Read: Write:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
$0205	CAN3RIER	Read: Write:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
$0206	CAN3TFLG	Read: Write:	0	0	0	0	0	TXE2	TXE1	TXE0
$0207	CAN3TIER	Read: Write:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
$0208	CAN3TARQ	Read: Write:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
$0209	CAN3TAAK	Read: Write:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
$020A	CAN3TBSEL	Read: Write:	0	0	0	0	0	TX2	TX1	TX0
$020B	CAN3IDAC	Read: Write:	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
$020C - $020D	Reserved	Read: Write:	0	0	0	0	0	0	0	0
$020E	CAN3RXERR	Read: Write:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
$020F	CAN3TXERR	Read: Write:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
$0210 - $0213	CAN3IDAR0 - CAN3IDAR3	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$0214 - $0217	CAN3IDMR0 - CAN3IDMR3	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0218 - $021B	CAN3IDAR4 - CAN3IDAR7	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

$0200 - $023F CAN3 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$021C - $021F	CAN3IDMR4 - CAN3IDMR7	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0220 - $022F	CAN3RXFG	Read:	FOREGROUND RECEIVE BUFFER see **Table 1-2**							
		Write:								
$0230 - $023F	CAN3TXFG	Read: Write:	FOREGROUND TRANSMIT BUFFER see **Table 1-2**							

$0240 - $027F PIM (Port Integration Module PIM_9DP256)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0240	PTT	Read: Write:	PTT7	PTT6	PTT5	PTT4	PTT3	PTT2	PTT1	PTT0
$0241	PTIT	Read:	PTIT7	PTIT6	PTIT5	PTIT4	PTIT3	PTIT2	PTIT1	PTIT0
		Write:								
$0242	DDRT	Read: Write:	DDRT7	DDRT7	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0
$0243	RDRT	Read: Write:	RDRT7	RDRT6	RDRT5	RDRT4	RDRT3	RDRT2	RDRT1	RDRT0
$0244	PERT	Read: Write:	PERT7	PERT6	PERT5	PERT4	PERT3	PERT2	PERT1	PERT0
$0245	PPST	Read: Write:	PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
$0246 - $0247	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0248	PTS	Read: Write:	PTS7	PTS6	PTS5	PTS4	PTS3	PTS2	PTS1	PTS0
$0249	PTIS	Read:	PTIS7	PTIS6	PTIS5	PTIS4	PTIS3	PTIS2	PTIS1	PTIS0
		Write:								
$024A	DDRS	Read: Write:	DDRS7	DDRS7	DDRS5	DDRS4	DDRS3	DDRS2	DDRS1	DDRS0
$024B	RDRS	Read: Write:	RDRS7	RDRS6	RDRS5	RDRS4	RDRS3	RDRS2	RDRS1	RDRS0
$024C	PERS	Read: Write:	PERS7	PERS6	PERS5	PERS4	PERS3	PERS2	PERS1	PERS0
$024D	PPSS	Read: Write:	PPSS7	PPSS6	PPSS5	PPSS4	PPSS3	PPSS2	PPSS1	PPSS0
$024E	WOMS	Read: Write:	WOMS7	WOMS6	WOMS5	WOMS4	WOMS3	WOMS2	WOMS1	WOMS0
$024F	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$0250	PTM	Read: Write:	PTM7	PTM6	PTM5	PTM4	PTM3	PTM2	PTM1	PTM0
$0251	PTIM	Read:	PTIM7	PTIM6	PTIM5	PTIM4	PTIM3	PTIM2	PTIM1	PTIM0
		Write:								
$0252	DDRM	Read: Write:	DDRM7	DDRM7	DDRM5	DDRM4	DDRM3	DDRM2	DDRM1	DDRM0
$0253	RDRM	Read: Write:	RDRM7	RDRM6	RDRM5	RDRM4	RDRM3	RDRM2	RDRM1	RDRM0

$0240 - $027F PIM (Port Integration Module PIM_9DP256)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0254	PERM	Read:	PERM7	PERM6	PERM5	PERM4	PERM3	PERM2	PERM1	PERM0
		Write:								
$0255	PPSM	Read:	PPSM7	PPSM6	PPSM5	PPSM4	PPSM3	PPSM2	PPSM1	PPSM0
		Write:								
$0256	WOMM	Read:	WOMM7	WOMM6	WOMM5	WOMM4	WOMM3	WOMM2	WOMM1	WOMM0
		Write:								
$0257	MODRR	Read:	0	MODRR6	MODRR5	MODRR4	MODRR3	MODRR2	MODRR1	MODRR0
		Write:								
$0258	PTP	Read:	PTP7	PTP6	PTP5	PTP4	PTP3	PTP2	PTP1	PTP0
		Write:								
$0259	PTIP	Read:	PTIP7	PTIP6	PTIP5	PTIP4	PTIP3	PTIP2	PTIP1	PTIP0
		Write:								
$025A	DDRP	Read:	DDRP7	DDRP7	DDRP5	DDRP4	DDRP3	DDRP2	DDRP1	DDRP0
		Write:								
$025B	RDRP	Read:	RDRP7	RDRP6	RDRP5	RDRP4	RDRP3	RDRP2	RDRP1	RDRP0
		Write:								
$025C	PERP	Read:	PERP7	PERP6	PERP5	PERP4	PERP3	PERP2	PERP1	PERP0
		Write:								
$025D	PPSP	Read:	PPSP7	PPSP6	PPSP5	PPSP4	PPSP3	PPSP2	PPSP1	PPSS0
		Write:								
$025E	PIEP	Read:	PIEP7	PIEP6	PIEP5	PIEP4	PIEP3	PIEP2	PIEP1	PIEP0
		Write:								
$025F	PIFP	Read:	PIFP7	PIFP6	PIFP5	PIFP4	PIFP3	PIFP2	PIFP1	PIFP0
		Write:								
$0260	PTH	Read:	PTH7	PTH6	PTH5	PTH4	PTH3	PTH2	PTH1	PTH0
		Write:								
$0261	PTIH	Read:	PTIH7	PTIH6	PTIH5	PTIH4	PTIH3	PTIH2	PTIH1	PTIH0
		Write:								
$0262	DDRH	Read:	DDRH7	DDRH7	DDRH5	DDRH4	DDRH3	DDRH2	DDRH1	DDRH0
		Write:								
$0263	RDRH	Read:	RDRH7	RDRH6	RDRH5	RDRH4	RDRH3	RDRH2	RDRH1	RDRH0
		Write:								
$0264	PERH	Read:	PERH7	PERH6	PERH5	PERH4	PERH3	PERH2	PERH1	PERH0
		Write:								
$0265	PPSH	Read:	PPSH7	PPSH6	PPSH5	PPSH4	PPSH3	PPSH2	PPSH1	PPSH0
		Write:								
$0266	PIEH	Read:	PIEH7	PIEH6	PIEH5	PIEH4	PIEH3	PIEH2	PIEH1	PIEH0
		Write:								
$0267	PIFH	Read:	PIFH7	PIFH6	PIFH5	PIFH4	PIFH3	PIFH2	PIFH1	PIFH0
		Write:								
$0268	PTJ	Read:	PTJ7	PTJ6	0	0	0	0	PTJ1	PTJ0
		Write:								
$0269	PTIJ	Read:	PTIJ7	PTIJ6	0	0	0	0	PTIJ1	PTIJ0
		Write:								
$026A	DDRJ	Read:	DDRJ7	DDRJ7	0	0	0	0	DDRJ1	DDRJ0
		Write:								
$026B	RDRJ	Read:	RDRJ7	RDRJ6	0	0	0	0	RDRJ1	RDRJ0
		Write:								
$026C	PERJ	Read:	PERJ7	PERJ6	0	0	0	0	PERJ1	PERJ0
		Write:								

$0240 - $027F PIM (Port Integration Module PIM_9DP256)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$026D	PPSJ	Read:	PPSJ7	PPSJ6	0	0	0	0	PPSJ1	PPSJ0
		Write:								
$026E	PIEJ	Read:	PIEJ7	PIEJ6	0	0	0	0	PIEJ1	PIEJ0
		Write:								
$026F	PIFJ	Read:	PIFJ7	PIFJ6	0	0	0	0	PIFJ1	PIFJ0
		Write:								
$0270 - $027F	Reserved	Read:								

$0280 - $02BF CAN4 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0280	CAN4CTL0	Read:	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
		Write:								
$0281	CAN4CTL1	Read:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
		Write:								
$0282	CAN4BTR0	Read:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
		Write:								
$0283	CAN4BTR1	Read:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
		Write:								
$0284	CAN4RFLG	Read:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
		Write:								
$0285	CAN4RIER	Read:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
		Write:								
$0286	CAN4TFLG	Read:	0	0	0	0	0	TXE2	TXE1	TXE0
		Write:								
$0287	CAN4TIER	Read:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
		Write:								
$0288	CAN4TARQ	Read:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
		Write:								
$0289	CAN4TAAK	Read:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
		Write:								
$028A	CAN4TBSEL	Read:	0	0	0	0	0	TX2	TX1	TX0
		Write:								
$028B	CAN4IDAC	Read:	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
		Write:								
$028C - $028D	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$028E	CAN4RXERR	Read:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
		Write:								
$028F	CAN4TXERR	Read:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
		Write:								
$0290 - $0293	CAN4IDAR0 - CAN4IDAR3	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
$0294 - $0297	CAN4IDMR0 - CAN4IDMR3	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$0298 - $029B	CAN4IDAR4 - CAN4IDAR7	Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

$0280 - $02BF CAN4 (Motorola Scalable CAN - MSCAN)

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$029C - $029F	CAN4IDMR4 - CAN4IDMR7	Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
$02A0 - $02AF	CAN4RXFG	Read:	FOREGROUND RECEIVE BUFFER see **Table 1-2**							
		Write:								
$02B0 - $02BF	CAN4TXFG	Read: Write:	FOREGROUND TRANSMIT BUFFER see **Table 1-2**							

$02C0 - $03FF Reserved

Address	Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$02C0 - $03FF	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

1.6 Part ID Assignments

The part ID is located in two 8-bit registers PARTIDH and PARTIDL (addresses $001A and $001B after reset). The read-only value is a unique part ID for each revision of the chip. **Table 1-3** shows the assigned part ID number.

Table 1-3 Assigned Part ID Numbers

Device	Mask Set Number	Part ID[1]
MC9S12DP512	0L00M	$0400
MC9S12DP512	1L00M	$0401
MC9S12DP512	2L00M	$0402
MC9S12DP512	3L00M	$0403
MC9S12DP512	4L00M	$0404

NOTES:
1. The coding is as follows:
Bit 15 - 12: Major family identifier
Bit 11 - 8: Minor family identifier
Bit 7 - 4: Major mask set revision number including FAB transfers
Bit 3 - 0: Minor - non full - mask set revision

1.7 Memory Size Assignments

The device memory sizes are located in two 8-bit registers MEMSIZ0 and MEMSIZ1 (addresses $001C and $001D after reset). **Table 1-4** shows the read-only values of these registers. Refer to HCS12 Module Mapping Control (MMC) Block Guide for further details.

Table 1-4 Memory size registers

Register name	Value
MEMSIZ0	$26
MEMSIZ1	$82

Section 2 Signal Description

This section describes signals that connect off-chip. It includes a pinout diagram, a table of signal properties, and detailed discussion of signals. It is built from the signal description sections of the Block Guides of the individual IP blocks on the device.

2.1 Device Pinout

The MC9S12DP512 is available in a 112-pin low profile quad flat pack (LQFP). Most pins perform two or more functions, as described in the Signal Descriptions. **Figure 2-1** shows the pin assignments.

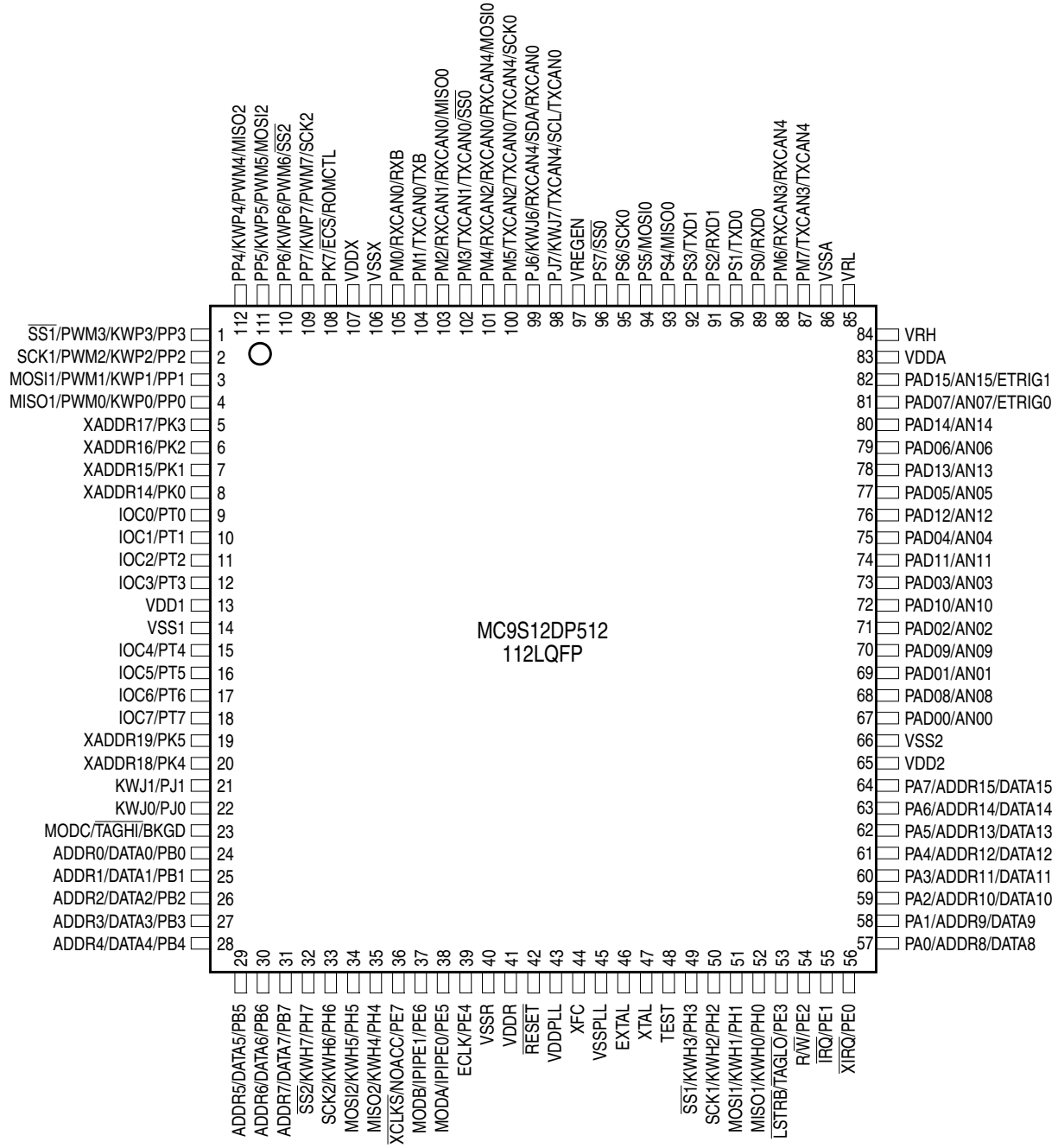


Figure 2-1 Pin Assignments in 112-pin LQFP

2.2 Signal Properties Summary

Table 2-1 summarizes the pin functionality.

Table 2-1 Signal Properties

Pin Name Funct. 1	Pin Name Funct. 2	Pin Name Funct. 3	Pin Name Funct. 4	Pin Name Funct. 5	Power Supply	Internal Pull Resistor		Description
						CTRL	Reset State	
EXTAL	—	—	—	—	VDDPLL	None	None	Oscillator Pins
XTAL	—	—	—	—				
$\overline{\text{RESET}}$	—	—	—	—	VDDR			External Reset
TEST	—	—	—	—	NA			Test Input
VREGEN	—	—	—	—	VDDX			Voltage Regulator Enable Input
XFC	—	—	—	—	VDDPLL			PLL Loop Filter
BKGD	$\overline{\text{TAGHI}}$	MODC	—	—	VDDR	Always Up	Up	Background Debug, Tag High, Mode Input
PAD15	AN15	ETRIG1	—	—	VDDA	None	None	Port AD Input, Analog Input AN7 of ATD1, External Trigger Input of ATD1
PAD[14:8]	AN[14:08]	—	—	—				Port AD Inputs, Analog Inputs AN[6:0] of ATD1
PAD07	AN07	ETRIG0	—	—				Port AD Input, Analog Input AN7 of ATD0, External Trigger Input of ATD0
PAD[06:00]	AN[06:00]	—	—	—				Port AD Inputs, Analog Inputs AN[6:0] of ATD0
PA[7:0]	ADDR[15:8]/ DATA[15:8]	—	—	—	VDDR	PUCR/ PUPAE	Disabled	Port A I/O, Multiplexed Address/Data
PB[7:0]	ADDR[7:0]/ DATA[7:0]	—	—	—		PUCR/ PUPBE		Port B I/O, Multiplexed Address/Data
PE7	NOACC	$\overline{\text{XCLKS}}$	—	—		PUCR/ PUPEE	Up	Port E I/O, Access, Clock Select
PE6	IPIPE1	MODB	—	—		While $\overline{\text{RESET}}$ pin is low: Down		Port E I/O, Pipe Status, Mode Input
PE5	IPIPE0	MODA	—	—		While $\overline{\text{RESET}}$ pin is low: Down		Port E I/O, Pipe Status, Mode Input
PE4	ECLK	—	—	—		PUCR/ PUPEE	Up	Port E I/O, Bus Clock Output
PE3	$\overline{\text{LSTRB}}$	$\overline{\text{TAGLO}}$	—	—				Port E I/O, Byte Strobe, Tag Low
PE2	R/$\overline{\text{W}}$	—	—	—				Port E I/O, R/$\overline{\text{W}}$ in expanded modes
PE1	$\overline{\text{IRQ}}$	—	—	—				Port E Input, Maskable Interrupt
PE0	$\overline{\text{XIRQ}}$	—	—	—				Port E Input, Non Maskable Interrupt

Pin Name Funct. 1	Pin Name Funct. 2	Pin Name Funct. 3	Pin Name Funct. 4	Pin Name Funct. 5	Power Supply	Internal Pull Resistor		Description
						CTRL	Reset State	
PH7	KWH7	$\overline{SS2}$	—	—	VDDR	PERH/ PPSH	Disabled	Port H I/O, Interrupt, $\overline{SS}$ of SPI2
PH6	KWH6	SCK2	—	—				Port H I/O, Interrupt, SCK of SPI2
PH5	KWH5	MOSI2	—	—				Port H I/O, Interrupt, MOSI of SPI2
PH4	KWH4	MISO2	—	—				Port H I/O, Interrupt, MISO of SPI2
PH3	KWH3	$\overline{SS1}$	—	—				Port H I/O, Interrupt, $\overline{SS}$ of SPI1
PH2	KWH2	SCK1	—	—				Port H I/O, Interrupt, SCK of SPI1
PH1	KWH1	MOSI1	—	—				Port H I/O, Interrupt, MOSI of SPI1
PH0	KWH0	MISO1	—	—				Port H I/O, Interrupt, MISO of SPI1
PJ7	KWJ7	TXCAN4	SCL	TXCAN0	VDDX	PERJ/ PPSJ	Up	Port J I/O, Interrupt, TX of CAN4, SCL of IIC, TX of CAN0
PJ6	KWJ6	RXCAN4	SDA	RXCAN0				Port J I/O, Interrupt, RX of CAN4, SDA of IIC, RX of CAN0
PJ[1:0]	KWJ[1:0]	—	—	—				Port J I/O, Interrupts
PK7	$\overline{ECS}$	ROMCTL	—	—	VDDX	PUCR/ PUPKE	Up	Port K I/O, Emulation Chip Select, ROM Control
PK[5:0]	XADDR [19:14]	—	—	—				Port K I/O, Extended Addresses
PM7	TXCAN3	TXCAN4	—	—	VDDX	PERM/ PPSM	Disabled	Port M I/O, TX of CAN3, TX of CAN4
PM6	RXCAN3	RXCAN4	—	—				Port M I/O, RX of CAN3, RX of CAN4
PM5	TXCAN2	TXCAN0	TXCAN4	SCK0				Port M I/O, TX of CAN2, CAN0, CAN4, SCK of SPI0
PM4	RXCAN2	RXCAN0	RXCAN4	MOSI0				Port M I/O, RX of CAN2, CAN0, CAN4, MOSI of SPI0
PM3	TXCAN1	TXCAN0	—	$\overline{SS0}$				Port M I/O, TX of CAN1, CAN0, $\overline{SS}$ of SPI0
PM2	RXCAN1	RXCAN0	—	MISO0				Port M I/O, RX of CAN1, CAN0, MISO of SPI0
PM1	TXCAN0	TXB	—	—				Port M I/O, TX of CAN0, RX of BDLC
PM0	RXCAN0	RXB	—	—				Port M I/O, RX of CAN0, RX of BDLC
PP7	KWP7	PWM7	SCK2	—	VDDX	PERP/ PPSP	Disabled	Port P I/O, Interrupt, Channel 7 of PWM, SCK of SPI2
PP6	KWP6	PWM6	$\overline{SS2}$	—				Port P I/O, Interrupt, Channel 6 of PWM, $\overline{SS}$ of SPI2
PP5	KWP5	PWM5	MOSI2	—				Port P I/O, Interrupt, Channel 5 of PWM, MOSI of SPI2
PP4	KWP4	PWM4	MISO2	—				Port P I/O, Interrupt, Channel 4 of PWM, MISO2 of SPI2
PP3	KWP3	PWM3	$\overline{SS1}$	—				Port P I/O, Interrupt, Channel 3 of PWM, $\overline{SS}$ of SPI1
PP2	KWP2	PWM2	SCK1	—				Port P I/O, Interrupt, Channel 2 of PWM, SCK of SPI1
PP1	KWP1	PWM1	MOSI1	—				Port P I/O, Interrupt, Channel 1 of PWM, MOSI of SPI1
PP0	KWP0	PWM0	MISO1	—				Port P I/O, Interrupt, Channel 0 of PWM, MISO2 of SPI1

Pin Name Funct. 1	Pin Name Funct. 2	Pin Name Funct. 3	Pin Name Funct. 4	Pin Name Funct. 5	Power Supply	Internal Pull Resistor		Description
						CTRL	Reset State	
PS7	$\overline{SS0}$	—	—	—	VDDX	PERS/ PPSS	Up	Port S I/O, $\overline{SS}$ of SPI0
PS6	SCK0	—	—	—				Port S I/O, SCK of SPI0
PS5	MOSI0	—	—	—				Port S I/O, MOSI of SPI0
PS4	MISO0	—	—	—				Port S I/O, MISO of SPI0
PS3	TXD1	—	—	—				Port S I/O, TXD of SCI1
PS2	RXD1	—	—	—				Port S I/O, RXD of SCI1
PS1	TXD0	—	—	—				Port S I/O, TXD of SCI0
PS0	RXD0	—	—	—				Port S I/O, RXD of SCI0
PT[7:0]	IOC[7:0]	—	—	—	VDDX	PERT/ PPST	Disabled	Port T I/O, Timer channels

2.3 Detailed Signal Descriptions

2.3.1 EXTAL, XTAL — Oscillator Pins

EXTAL and XTAL are the crystal driver and external clock pins. On reset all the device clocks are derived from the EXTAL input frequency. XTAL is the crystal output.

2.3.2 $\overline{\text{RESET}}$ — External Reset Pin

An active low bidirectional control signal, it acts as an input to initialize the MCU to a known start-up state, and an output when an internal MCU function causes a reset.

2.3.3 TEST — Test Pin

This input only pin is reserved for test.

NOTE: *The TEST pin must be tied to VSS in all applications.*

2.3.4 VREGEN — Voltage Regulator Enable Pin

This input only pin enables or disables the on-chip voltage regulator.

2.3.5 XFC — PLL Loop Filter Pin

PLL loop filter. Please ask your Motorola representative for the interactive application note to compute PLL loop filter elements. Any current leakage on this pin must be avoided.

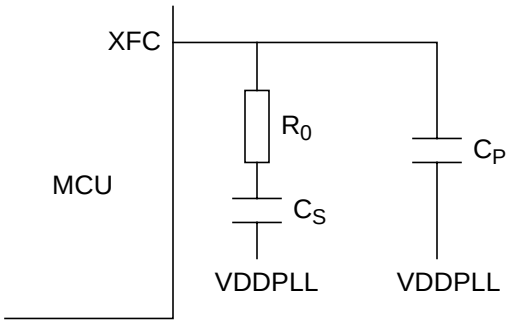


Figure 2-2 PLL Loop Filter Connections

2.3.6 BKGD / $\overline{\text{TAGHI}}$ / MODC — Background Debug, Tag High, and Mode Pin

The BKGD/$\overline{\text{TAGHI}}$/MODC pin is used as a pseudo-open-drain pin for the background debug communication. In MCU expanded modes of operation when instruction tagging is on, an input low on this pin during the falling edge of E-clock tags the high half of the instruction word being read into the instruction queue. It is used as a MCU operating mode select pin during reset. The state of this pin is latched to the MODC bit at the rising edge of $\overline{\text{RESET}}$. This pin has a permanently enabled pull-up device.

2.3.7 PAD15 / AN15 / ETRIG1 — Port AD Input Pin of ATD1

PAD15 is a general purpose input pin and analog input AN7 of the analog to digital converter ATD1. It can act as an external trigger input for the ATD1.

2.3.8 PAD[14:08] / AN[14:08] — Port AD Input Pins of ATD1

PAD14 - PAD08 are general purpose input pins and analog inputs AN[6:0] of the analog to digital converter ATD1.

2.3.9 PAD7 / AN07 / ETRIG0 — Port AD Input Pin of ATD0

PAD7 is a general purpose input pin and analog input AN7 of the analog to digital converter ATD0. It can act as an external trigger input for the ATD0.

2.3.10 PAD[06:00] / AN[06:00] — Port AD Input Pins of ATD0

PAD06 - PAD00 are general purpose input pins and analog inputs AN[6:0] of the analog to digital converter ATD0.

2.3.11 PA[7:0] / ADDR[15:8] / DATA[15:8] — Port A I/O Pins

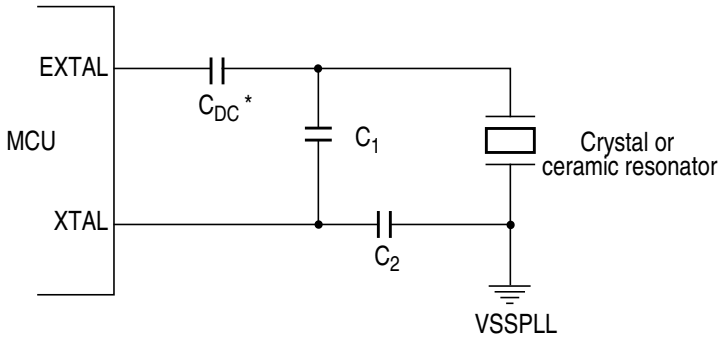
PA7-PA0 are general purpose input or output pins. In MCU expanded modes of operation, these pins are used for the multiplexed external address and data bus.

2.3.12 PB[7:0] / ADDR[7:0] / DATA[7:0] — Port B I/O Pins

PB7-PB0 are general purpose input or output pins. In MCU expanded modes of operation, these pins are used for the multiplexed external address and data bus.

2.3.13 PE7 / NOACC / $\overline{\text{XCLKS}}$ — Port E I/O Pin 7

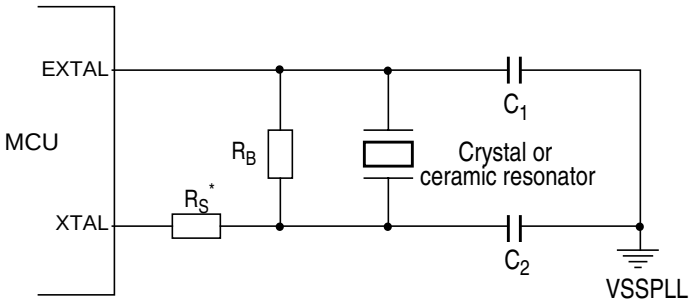
PE7 is a general purpose input or output pin. During MCU expanded modes of operation, the NOACC signal, when enabled, is used to indicate that the current bus cycle is an unused or “free” cycle. This signal will assert when the CPU is not using the bus.
The $\overline{\text{XCLKS}}$ is an input signal which controls whether a crystal in combination with the internal Colpitts (low power) oscillator is used or whether Pierce oscillator/external clock circuitry is used. The state of this pin is latched at the rising edge of $\overline{\text{RESET}}$. If the input is a logic low the EXTAL pin is configured for an external clock drive or a Pierce Oscillator. If input is a logic high a Colpitts oscillator circuit is configured on EXTAL and XTAL. Since this pin is an input with a pull-up device during reset, if the pin is left floating, the default configuration is a Colpitts oscillator circuit on EXTAL and XTAL.



* Due to the nature of a translated ground Colpitts oscillator a DC voltage bias is applied to the crystal

Please contact the crystal manufacturer for crystal DC bias conditions and recommended capacitor value C_{DC}.

Figure 2-3 Colpitts Oscillator Connections (PE7=1)



* Rs can be zero (shorted) when used with higher frequency crystals.
Refer to manufacturer's data.

Figure 2-4 Pierce Oscillator Connections (PE7=0)

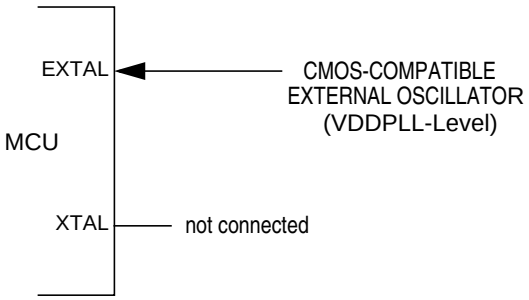


Figure 2-5 External Clock Connections (PE7=0)

2.3.14 PE6 / MODB / IPIPE1 — Port E I/O Pin 6

PE6 is a general purpose input or output pin. It is used as a MCU operating mode select pin during reset. The state of this pin is latched to the MODB bit at the rising edge of $\overline{\text{RESET}}$. This pin is shared with the instruction queue tracking signal IPIPE1. This pin is an input with a pull-down device which is only active when $\overline{\text{RESET}}$ is low.

2.3.15 PE5 / MODA / IPIPE0 — Port E I/O Pin 5

PE5 is a general purpose input or output pin. It is used as a MCU operating mode select pin during reset. The state of this pin is latched to the MODA bit at the rising edge of $\overline{\text{RESET}}$. This pin is shared with the instruction queue tracking signal IPIPE0. This pin is an input with a pull-down device which is only active when $\overline{\text{RESET}}$ is low.

2.3.16 PE4 / ECLK — Port E I/O Pin 4

PE4 is a general purpose input or output pin. It can be configured to drive the internal bus clock ECLK. ECLK can be used as a timing reference.

2.3.17 PE3 / $\overline{\text{LSTRB}}$ / $\overline{\text{TAGLO}}$ — Port E I/O Pin 3

PE3 is a general purpose input or output pin. In MCU expanded modes of operation, $\overline{\text{LSTRB}}$ can be used for the low-byte strobe function to indicate the type of bus access and when instruction tagging is on, $\overline{\text{TAGLO}}$ is used to tag the low half of the instruction word being read into the instruction queue.

2.3.18 PE2 / R/$\overline{\text{W}}$ — Port E I/O Pin 2

PE2 is a general purpose input or output pin. In MCU expanded modes of operations, this pin drives the read/write output signal for the external bus. It indicates the direction of data on the external bus.

2.3.19 PE1 / $\overline{\text{IRQ}}$ — Port E Input Pin 1

PE1 is a general purpose input pin and the maskable interrupt request input that provides a means of applying asynchronous interrupt requests. This will wake up the MCU from STOP or WAIT mode.

2.3.20 PE0 / $\overline{\text{XIRQ}}$ — Port E Input Pin 0

PE0 is a general purpose input pin and the non-maskable interrupt request input that provides a means of applying asynchronous interrupt requests. This will wake up the MCU from STOP or WAIT mode.

2.3.21 PH7 / KWH7 / $\overline{\text{SS2}}$ — Port H I/O Pin 7

PH7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as slave select pin $\overline{\text{SS}}$ of the Serial Peripheral Interface 2 (SPI2).

2.3.22 PH6 / KWH6 / SCK2 — Port H I/O Pin 6

PH6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 2 (SPI2).

2.3.23 PH5 / KWH5 / MOSI2 — Port H I/O Pin 5

PH5 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 2 (SPI2).

2.3.24 PH4 / KWH4 / MISO2 — Port H I/O Pin 2

PH4 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 2 (SPI2).

2.3.25 PH3 / KWH3 / $\overline{SS1}$ — Port H I/O Pin 3

PH3 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as slave select pin $\overline{SS}$ of the Serial Peripheral Interface 1 (SPI1).

2.3.26 PH2 / KWH2 / SCK1 — Port H I/O Pin 2

PH2 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 1 (SPI1).

2.3.27 PH1 / KWH1 / MOSI1 — Port H I/O Pin 1

PH1 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 1 (SPI1).

2.3.28 PH0 / KWH0 / MISO1 — Port H I/O Pin 0

PH0 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 1 (SPI1).

2.3.29 PJ7 / KWJ7 / TXCAN4 / SCL / TXCAN0 — PORT J I/O Pin 7

PJ7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as the transmit pin TXCAN for the Motorola Scalable Controller Area Network controller 0 or 4 (CAN0 or CAN4) or the serial clock pin SCL of the IIC module.

2.3.30 PJ6 / KWJ6 / RXCAN4 / SDA / RXCAN0 — PORT J I/O Pin 6

PJ6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as the receive pin RXCAN for the Motorola Scalable Controller Area Network controller 0 or 4 (CAN 0 or CAN4) or the serial data pin SDA of the IIC module.

2.3.31 PJ[1:0] / KWJ[1:0] — Port J I/O Pins [1:0]

PJ1 and PJ0 are general purpose input or output pins. They can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode.

2.3.32 PK7 / $\overline{ECS}$ / ROMCTL — Port K I/O Pin 7

PK7 is a general purpose input or output pin. During MCU expanded modes of operation, this pin is used as the emulation chip select output ($\overline{ECS}$). During MCU normal expanded modes of operation, this pin is

used to enable the Flash EEPROM memory in the memory map (ROMCTL). At the rising edge of $\overline{\text{RESET}}$, the state of this pin is latched to the ROMON bit.

2.3.33 PK[5:0] / XADDR[19:14] — Port K I/O Pins [5:0]

PK5-PK0 are general purpose input or output pins. In MCU expanded modes of operation, these pins provide the expanded address XADDR[19:14] for the external bus.

2.3.34 PM7 / TXCAN3 / TXCAN4 — Port M I/O Pin 7

PM7 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 3 or 4 (CAN3 or CAN4).

2.3.35 PM6 / RXCAN3 / RXCAN4 — Port M I/O Pin 6

PM6 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 3 or 4 (CAN3 or CAN4).

2.3.36 PM5 / TXCAN2 / TXCAN0 / TXCAN4 / SCK0 — Port M I/O Pin 5

PM5 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 2, 0 or 4 (CAN2, CAN0 or CAN4). It can be configured as the serial clock pin SCK of the Serial Peripheral Interface 0 (SPI0).

2.3.37 PM4 / RXCAN2 / RXCAN0 / RXCAN4/ MOSI0 — Port M I/O Pin 4

PM4 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 2, 0 or 4 (CAN2, CAN0 or CAN4). It can be configured as the master output (during master mode) or slave input pin (during slave mode) MOSI for the Serial Peripheral Interface 0 (SPI0).

2.3.38 PM3 / TXCAN1 / TXCAN0 / $\overline{\text{SS0}}$ — Port M I/O Pin 3

PM3 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controllers 1 or 0 (CAN1 or CAN0). It can be configured as the slave select pin $\overline{\text{SS}}$ of the Serial Peripheral Interface 0 (SPI0).

2.3.39 PM2 / RXCAN1 / RXCAN0 / MISO0 — Port M I/O Pin 2

PM2 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controllers 1 or 0 (CAN1 or CAN0). It can be configured as the master input (during master mode) or slave output pin (during slave mode) MISO for the Serial Peripheral Interface 0 (SPI0).

2.3.40 PM1 / TXCAN0 / TXB — Port M I/O Pin 1

PM1 is a general purpose input or output pin. It can be configured as the transmit pin TXCAN of the Motorola Scalable Controller Area Network controller 0 (CAN0). It can be configured as the transmit pin TXB of the BDLC.

2.3.41 PM0 / RXCAN0 / RXB — Port M I/O Pin 0

PM0 is a general purpose input or output pin. It can be configured as the receive pin RXCAN of the Motorola Scalable Controller Area Network controller 0 (CAN0). It can be configured as the receive pin RXB of the BDLC.

2.3.42 PP7 / KWP7 / PWM7 / SCK2 — Port P I/O Pin 7

PP7 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 7 output or an input for the PWM emergency shutdown. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 2 (SPI2).

2.3.43 PP6 / KWP6 / PWM6 / $\overline{SS2}$ — Port P I/O Pin 6

PP6 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 6 output. It can be configured as slave select pin $\overline{SS}$ of the Serial Peripheral Interface 2 (SPI2).

2.3.44 PP5 / KWP5 / PWM5 / MOSI2 — Port P I/O Pin 5

PP5 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 5 output. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 2 (SPI2).

2.3.45 PP4 / KWP4 / PWM4 / MISO2 — Port P I/O Pin 4

PP4 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 4 output. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 2 (SPI2).

2.3.46 PP3 / KWP3 / PWM3 / $\overline{SS1}$ — Port P I/O Pin 3

PP3 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 3 output. It can be configured as slave select pin $\overline{SS}$ of the Serial Peripheral Interface 1 (SPI1).

2.3.47 PP2 / KWP2 / PWM2 / SCK1 — Port P I/O Pin 2

PP2 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 2 output. It can be configured as serial clock pin SCK of the Serial Peripheral Interface 1 (SPI1).

2.3.48 PP1 / KWP1 / PWM1 / MOSI1 — Port P I/O Pin 1

PP1 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 1 output. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 1 (SPI1).

2.3.49 PP0 / KWP0 / PWM0 / MISO1 — Port P I/O Pin 0

PP0 is a general purpose input or output pin. It can be configured to generate an interrupt causing the MCU to exit STOP or WAIT mode. It can be configured as Pulse Width Modulator (PWM) channel 0 output. It can be configured as master input (during master mode) or slave output (during slave mode) pin MISO of the Serial Peripheral Interface 1 (SPI1).

2.3.50 PS7 / $\overline{SS0}$ — Port S I/O Pin 7

PS6 is a general purpose input or output pin. It can be configured as the slave select pin $\overline{SS}$ of the Serial Peripheral Interface 0 (SPI0).

2.3.51 PS6 / SCK0 — Port S I/O Pin 6

PS6 is a general purpose input or output pin. It can be configured as the serial clock pin SCK of the Serial Peripheral Interface 0 (SPI0).

2.3.52 PS5 / MOSI0 — Port S I/O Pin 5

PS5 is a general purpose input or output pin. It can be configured as master output (during master mode) or slave input pin (during slave mode) MOSI of the Serial Peripheral Interface 0 (SPI0).

2.3.53 PS4 / MISO0 — Port S I/O Pin 4

PS4 is a general purpose input or output pin. It can be configured as master input (during master mode) or slave output pin (during slave mode) MOSI of the Serial Peripheral Interface 0 (SPI0).

2.3.54 PS3 / TXD1 — Port S I/O Pin 3

PS3 is a general purpose input or output pin. It can be configured as the transmit pin TXD of Serial Communication Interface 1 (SCI1).

2.3.55 PS2 / RXD1 — Port S I/O Pin 2

PS2 is a general purpose input or output pin. It can be configured as the receive pin RXD of Serial Communication Interface 1 (SCI1).

2.3.56 PS1 / TXD0 — Port S I/O Pin 1

PS1 is a general purpose input or output pin. It can be configured as the transmit pin TXD of Serial Communication Interface 0 (SCI0).

2.3.57 PS0 / RXD0 — Port S I/O Pin 0

PS0 is a general purpose input or output pin. It can be configured as the receive pin RXD of Serial Communication Interface 0 (SCI0).

2.3.58 PT[7:0] / IOC[7:0] — Port T I/O Pins [7:0]

PT7-PT0 are general purpose input or output pins. They can be configured as input capture or output compare pins IOC7-IOC0 of the Enhanced Capture Timer (ECT).

2.4 Power Supply Pins

MC9S12DP512 power and ground pins are described below.

Table 2-2 MC9S12DP512 Power and Ground Connection Summary

Mnemonic	Pin Number 112-pin QFP	Nominal Voltage	Description
$V_{DD1, 2}$	13, 65	2.5 V	Internal power and ground generated by internal regulator
$V_{SS1, 2}$	14, 66	0V	
V_{DDR}	41	5.0 V	External power and ground, supply to pin drivers and internal voltage regulator.
V_{SSR}	40	0 V	
V_{DDX}	107	5.0 V	External power and ground, supply to pin drivers.
V_{SSX}	106	0 V	
V_{DDA}	83	5.0 V	Operating voltage and ground for the analog-to-digital converters and the reference for the internal voltage regulator, allows the supply voltage to the A/D to be bypassed independently.
V_{SSA}	86	0 V	
V_{RL}	85	0 V	Reference voltages for the analog-to-digital converter.
V_{RH}	84	5.0 V	
V_{DDPLL}	43	2.5 V	Provides operating voltage and ground for the Phased-Locked Loop. This allows the supply voltage to the PLL to be bypassed independently. Internal power and ground generated by internal regulator.
V_{SSPLL}	45	0 V	
VREGEN	97	5V	Internal Voltage Regulator enable/disable

NOTE: *All VSS pins must be connected together in the application.*

2.4.1 VDDX,VSSX — Power & Ground Pins for I/O Drivers

External power and ground for I/O drivers. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. Bypass requirements depend on how heavily the MCU pins are loaded.

2.4.2 VDDR, VSSR — Power & Ground Pins for I/O Drivers & Internal Voltage Regulator

External power and ground for I/O drivers and input to the internal voltage regulator. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. Bypass requirements depend on how heavily the MCU pins are loaded.

2.4.3 VDD1, VDD2, VSS1, VSS2 — Internal Logic Power Supply Pins

Power is supplied to the MCU through VDD and VSS. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. This 2.5V supply is derived from the internal voltage regulator. There is no static load on those pins allowed. The internal voltage regulator is turned off, if VREGEN is tied to ground.

NOTE: *No load allowed except for bypass capacitors.*

2.4.4 VDDA, VSSA — Power Supply Pins for ATD and VREG

VDDA, VSSA are the power supply and ground input pins for the voltage regulator and the analog to digital converter. It also provides the reference for the internal voltage regulator. This allows the supply voltage to the ATD and the reference voltage to be bypassed independently.

2.4.5 VRH, VRL — ATD Reference Voltage Input Pins

VRH and VRL are the reference voltage input pins for the analog to digital converter.

2.4.6 VDDPLL, VSSPLL — Power Supply Pins for PLL

Provides operating voltage and ground for the Oscillator and the Phased-Locked Loop. This allows the supply voltage to the Oscillator and PLL to be bypassed independently. This 2.5V voltage is generated by the internal voltage regulator.

NOTE: *No load allowed except for bypass capacitors.*

2.4.7 VREGEN — On Chip Voltage Regulator Enable

Enables the internal 5V to 2.5V voltage regulator. If this pin is tied low, VDD1,2 and VDDPLL must be supplied externally.

Section 3 System Clock Description

3.1 Overview

The Clock and Reset Generator provides the internal clock signals for the core and all peripheral modules. **Figure 3-1** shows the clock connections from the CRG to all modules.

Consult the CRG Block Guide and OSC Block Guide for details on clock generation.

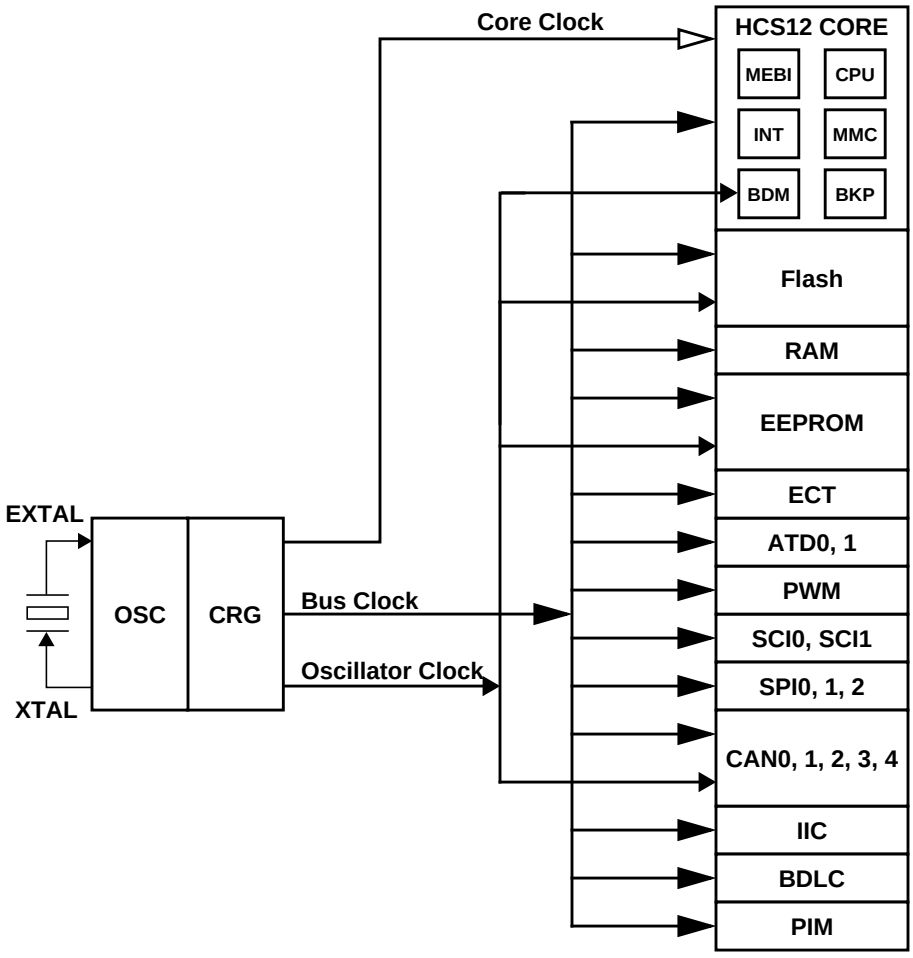


Figure 3-1 Clock Connections

Section 4 Modes of Operation

4.1 Overview

Eight possible modes determine the operating configuration of the MC9S12DP512. Each mode has an associated default memory map and external bus configuration controlled by a further pin.

Three low power modes exist for the device (**Section 4.4 Low Power Modes**).

4.2 Chip Configuration Summary

The operating mode out of reset is determined by the states of the MODC, MODB, and MODA pins during reset (**Table 4-1**). The MODC, MODB, and MODA bits in the MODE register show the current operating mode and provide limited mode switching during operation. The states of the MODC, MODB, and MODA pins are latched into these bits on the rising edge of the reset signal. The ROMCTL signal allows the setting of the ROMON bit in the MISC register thus controlling whether the internal Flash is visible in the memory map. ROMON = 1 means the Flash is visible in the memory map. The state of the ROMCTL pin is latched into the ROMON bit in the MISC register on the rising edge of the reset signal.

Table 4-1 Mode Selection

BKGD = MODC	PE6 = MODB	PE5 = MODA	PK7 = ROMCTL	ROMON Bit	Mode Description
0	0	0	X	1	Special Single Chip, BDM allowed and ACTIVE. BDM is allowed in all other modes but a serial command is required to make BDM active.
0	0	1	0	1	Emulation Expanded Narrow, BDM allowed
			1	0	
0	1	0	X	0	Special Test (Expanded Wide), BDM allowed
0	1	1	0	1	Emulation Expanded Wide, BDM allowed
			1	0	
1	0	0	X	1	Normal Single Chip, BDM allowed
1	0	1	0	0	Normal Expanded Narrow, BDM allowed
			1	1	
1	1	0	X	1	Peripheral; BDM allowed but bus operations would cause bus conflicts (must not be used)
1	1	1	0	0	Normal Expanded Wide, BDM allowed
			1	1	

For further explanation on the modes refer to the HCS12 Multiplexed External Bus Interface (MEBI) Block Guide.

Table 4-2 Clock Selection Based on PE7

PE7 = $\overline{\text{XCLKS}}$	Description
1	Colpitts Oscillator selected
0	Pierce Oscillator/external clock selected

Table 4-3 Voltage Regulator VREGEN

VREGEN	Description
1	Internal Voltage Regulator enabled
0	Internal Voltage Regulator disabled, VDD1,2 and VDDPLL must be supplied externally with 2.5V

4.3 Security

The device will make available a security feature preventing the unauthorized read and write of the memory contents. This feature allows:

- Protection of the contents of FLASH,
- Protection of the contents of EEPROM,
- Operation in single-chip mode,
- Operation from external memory with internal FLASH and EEPROM disabled.

The user must be reminded that part of the security must lie with the user's code. An extreme example would be user's code that dumps the contents of the internal program. This code would defeat the purpose of security. At the same time the user may also wish to put a back door in the user's program. An example of this is the user downloads a key through the SCI which allows access to a programming routine that updates parameters stored in EEPROM.

4.3.1 Securing the Microcontroller

Once the user has programmed the FLASH and EEPROM (if desired), the part can be secured by programming the security bits located in the FLASH module. These non-volatile bits will keep the part secured through resetting the part and through powering down the part.

The security byte resides in a portion of the Flash array.

Check the Flash Block Guide for more details on the security configuration.

4.3.2 Operation of the Secured Microcontroller

4.3.2.1 Normal Single Chip Mode

This will be the most common usage of the secured part. Everything will appear the same as if the part was not secured with the exception of BDM operation. The BDM operation will be blocked.

4.3.2.2 Executing from External Memory

The user may wish to execute from external space with a secured microcontroller. This is accomplished by resetting directly into expanded mode. The internal FLASH and EEPROM will be disabled. BDM operations will be blocked.

4.3.3 Unsecuring the Microcontroller

In order to unsecure the microcontroller, the internal FLASH and EEPROM must be erased. This can be done through an external program in expanded mode or via a sequence of BDM commands. Unsecuring is also possible via the Backdoor Key Access. Refer to Flash Block Guide for details..

Once the user has erased the FLASH and EEPROM, the part can be reset into special single chip mode. This invokes a program that verifies the erasure of the internal FLASH and EEPROM. Once this program completes, the user can erase and program the FLASH security bits to the unsecured state. This is generally done through the BDM, but the user could also change to expanded mode (by writing the mode bits through the BDM) and jumping to an external program (again through BDM commands). Note that if the part goes through a reset before the security bits are reprogrammed to the unsecure state, the part will be secured again.

4.4 Low Power Modes

The microcontroller features three main low power modes. Consult the respective Block Guide for information on the module behavior in Stop, Pseudo Stop, and Wait Mode. An important source of information about the clock system is the Clock and Reset Generator Block Guide (CRG).

4.4.1 Stop

Executing the CPU STOP instruction stops all clocks and the oscillator thus putting the chip in fully static mode. Wake up from this mode can be done via reset or external interrupts.

4.4.2 Pseudo Stop

This mode is entered by executing the CPU STOP instruction. In this mode the oscillator is still running and the Real Time Interrupt (RTI) or Watchdog (COP) sub module can stay active. Other peripherals are turned off. This mode consumes more current than the full STOP mode, but the wake up time from this mode is significantly shorter.

4.4.3 Wait

This mode is entered by executing the CPU WAI instruction. In this mode the CPU will not execute instructions. The internal CPU signals (address and databus) will be fully static. All peripherals stay active. For further power consumption the peripherals can individually turn off their local clocks.

4.4.4 Run

Although this is not a low power mode, unused peripheral modules should not be enabled in order to save power.

Section 5 Resets and Interrupts

5.1 Overview

Consult the Exception Processing section of the CPU12 Reference Manual for information on resets and interrupts.

5.2 Vectors

5.2.1 Vector Table

Table 5-1 lists interrupt sources and vectors in default order of priority.

Table 5-1 Interrupt Vector Locations

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
$FFFE, $FFFF	Reset	None	None	–
$FFFC, $FFFD	Clock Monitor fail reset	None	PLLCTL (CME, SCME)	–
$FFFA, $FFFB	COP failure reset	None	COP rate select	–
$FFF8, $FFF9	Unimplemented instruction trap	None	None	–
$FFF6, $FFF7	SWI	None	None	–
$FFF4, $FFF5	XIRQ	X-Bit	None	–
$FFF2, $FFF3	IRQ	I-Bit	IRQCR (IRQEN)	$F2
$FFF0, $FFF1	Real Time Interrupt	I-Bit	CRGINT (RTIE)	$F0
$FFEE, $FFEF	Enhanced Capture Timer channel 0	I-Bit	TIE (C0I)	$EE
$FFEC, $FFED	Enhanced Capture Timer channel 1	I-Bit	TIE (C1I)	$EC
$FFEA, $FFEB	Enhanced Capture Timer channel 2	I-Bit	TIE (C2I)	$EA
$FFE8, $FFE9	Enhanced Capture Timer channel 3	I-Bit	TIE (C3I)	$E8
$FFE6, $FFE7	Enhanced Capture Timer channel 4	I-Bit	TIE (C4I)	$E6
$FFE4, $FFE5	Enhanced Capture Timer channel 5	I-Bit	TIE (C5I)	$E4
$FFE2, $FFE3	Enhanced Capture Timer channel 6	I-Bit	TIE (C6I)	$E2
$FFE0, $FFE1	Enhanced Capture Timer channel 7	I-Bit	TIE (C7I)	$E0
$FFDE, $FFDF	Enhanced Capture Timer overflow	I-Bit	TSRC2 (TOI)	$DE
$FFDC, $FFDD	Pulse accumulator A overflow	I-Bit	PACTL (PAOVI)	$DC
$FFDA, $FFDB	Pulse accumulator input edge	I-Bit	PACTL (PAI)	$DA
$FFD8, $FFD9	SPI0	I-Bit	SPICR1 (SPIE, SPTIE)	$D8
$FFD6, $FFD7	SCI0	I-Bit	SCICR2 (TIE, TCIE, RIE, ILIE)	$D6
$FFD4, $FFD5	SCI1	I-Bit	SCICR2 (TIE, TCIE, RIE, ILIE)	$D4
$FFD2, $FFD3	ATD0	I-Bit	ATDCTL2 (ASCIE)	$D2
$FFD0, $FFD1	ATD1	I-Bit	ATDCTL2 (ASCIE)	$D0
$FFCE, $FFCF	Port J	I-Bit	PIEJ (PIEJ7, PIEJ6, PIEJ1, PIEJ0)	$CE
$FFCC, $FFCD	Port H	I-Bit	PIEH (PIEH7-0)	$CC

$FFCA, $FFCB	Modulus Down Counter underflow	I-Bit	MCCTL (MCZI)	$CA
$FFC8, $FFC9	Pulse Accumulator B Overflow	I-Bit	PBCTL (PBOVI)	$C8
$FFC6, $FFC7	CRG PLL lock	I-Bit	CRGINT (LOCKIE)	$C6
$FFC4, $FFC5	CRG Self Clock Mode	I-Bit	CRGINT (SCMIE)	$C4
$FFC2, $FFC3	BDLC	I-Bit	DLCBCR1 (IE)	$C2
$FFC0, $FFC1	IIC Bus	I-Bit	IBCR (IBIE)	$C0
$FFBE, $FFBF	SPI1	I-Bit	SPICR1 (SPIE, SPTIE)	$BE
$FFBC, $FFBD	SPI2	I-Bit	SPICR1 (SPIE, SPTIE)	$BC
$FFBA, $FFBB	EEPROM	I-Bit	ECNFG (CCIE, CBEIE)	$BA
$FFB8, $FFB9	FLASH	I-Bit	FCNFG (CCIE, CBEIE)	$B8
$FFB6, $FFB7	CAN0 wake-up	I-Bit	CANRIER (WUPIE)	$B6
$FFB4, $FFB5	CAN0 errors	I-Bit	CANRIER (CSCIE, OVRIE)	$B4
$FFB2, $FFB3	CAN0 receive	I-Bit	CANRIER (RXFIE)	$B2
$FFB0, $FFB1	CAN0 transmit	I-Bit	CANTIER (TXEIE2-TXEIE0)	$B0
$FFAE, $FFAF	CAN1 wake-up	I-Bit	CANRIER (WUPIE)	$AE
$FFAC, $FFAD	CAN1 errors	I-Bit	CANRIER (CSCIE, OVRIE)	$AC
$FFAA, $FFAB	CAN1 receive	I-Bit	CANRIER (RXFIE)	$AA
$FFA8, $FFA9	CAN1 transmit	I-Bit	CANTIER (TXEIE2-TXEIE0)	$A8
$FFA6, $FFA7	CAN2 wake-up	I-Bit	CANRIER (WUPIE)	$A6
$FFA4, $FFA5	CAN2 errors	I-Bit	CANRIER (CSCIE, OVRIE)	$A4
$FFA2, $FFA3	CAN2 receive	I-Bit	CANRIER (RXFIE)	$A2
$FFA0, $FFA1	CAN2 transmit	I-Bit	CANTIER (TXEIE2-TXEIE0)	$A0
$FF9E, $FF9F	CAN3 wake-up	I-Bit	CANRIER (WUPIE)	$9E
$FF9C, $FF9D	CAN3 errors	I-Bit	CANRIER (CSCIE, OVRIE)	$9C
$FF9A, $FF9B	CAN3 receive	I-Bit	CANRIER (RXFIE)	$9A
$FF98, $FF99	CAN3 transmit	I-Bit	CANTIER (TXEIE2-TXEIE0)	$98
$FF96, $FF97	CAN4 wake-up	I-Bit	CANRIER (WUPIE)	$96
$FF94, $FF95	CAN4 errors	I-Bit	CANRIER (CSCIE, OVRIE)	$94
$FF92, $FF93	CAN4 receive	I-Bit	CANRIER (RXFIE)	$92
$FF90, $FF91	CAN4 transmit	I-Bit	CANTIER (TXEIE2-TXEIE0)	$90
$FF8E, $FF8F	Port P Interrupt	I-Bit	PIEP (PIEP7-0)	$8E
$FF8C, $FF8D	PWM Emergency Shutdown	I-Bit	PWMSDN (PWMIE)	$8C
$FF80 to $FF8B	Reserved			

5.3 Effects of Reset

When a reset occurs, MCU registers and control bits are changed to known start-up states. Refer to the respective module Block Guides for register reset states.

5.3.1 I/O pins

Refer to the HCS12 Multiplexed External Bus Interface (MEBI) Block Guide for mode dependent pin configuration of port A, B, E and K out of reset.

Refer to the PIM Block Guide for reset configurations of all peripheral module ports.

5.3.2 Memory

Refer to **Table 1-1** for locations of the memories depending on the operating mode after reset.

The RAM array is not automatically initialized out of reset.

Section 6 HCS12 Core Block Description

6.1 CPU12 Block Description

Consult the HCS12 CPU Reference Manual for information on the CPU.

6.1.1 Device-specific information

When the HCS12 CPU Reference Manual refers to *cycles* this is equivalent to *Bus Clock periods*.
So *1 cycle* is equivalent to *1 Bus Clock period*.

6.2 HCS12 Module Mapping Control (MMC) Block Description

Consult the MMC Block Guide for information on the HCS12 Module Mapping Control module.

6.2.1 Device-specific information

- INITEE
  - Reset state: $01
  - Bits EE11-EE15 are "Write once in Normal and Emulation modes and write anytime in Special modes".
- PPAGE
  - Reset state: $00
  - Register is "Write anytime in all modes"

6.3 HCS12 Multiplexed External Bus Interface (MEBI) Block Description

Consult the MEBI Block Guide for information on HCS12 Multiplexed External Bus Interface module.

6.3.1 Device-specific information

- PUCR
  - Reset state: $90

6.4 HCS12 Interrupt (INT) Block Description

Consult the INT Block Guide for information on the HCS12 Interrupt module.

6.5 HCS12 Background Debug (BDM) Block Description

Consult the BDM Block Guide for information on the HCS12 Background Debug module.

6.5.1 Device-specific information

When the BDM Block Guide refers to *alternate clock* this is equivalent to *Oscillator Clock*.

6.6 HCS12 Breakpoint (BKP) Block Description

Consult the BKP Block Guide for information on the HCS12 Breakpoint module.

Section 7 Clock and Reset Generator (CRG) Block Description

Consult the CRG Block Guide for information about the Clock and Reset Generator module.

7.1 Device-specific information

The Low Voltage Reset feature of the CRG is not available on this device.

Section 8 Oscillator (OSC) Block Description

8.1 Device-specific information

The $\overline{\text{XCLKS}}$ input signal is active low (see **2.3.13 PE7 / NOACC / XCLKS — Port E I/O Pin 7**).

Section 9 Enhanced Capture Timer (ECT) Block Description

Consult the ECT_16B8C Block Guide for information about the Enhanced Capture Timer module.

When the ECT_16B8C Block Guide refers to *freeze mode* this is equivalent to *active BDM mode*.

Section 10 Analog to Digital Converter (ATD) Block Description

There are two Analog to Digital Converters (ATD1 and ATD0) implemented on the MC9S12DP512. Consult the ATD_10B8C Block Guide for information about each Analog to Digital Converter module.

When the ATD_10B8C Block Guide refers to *freeze mode* this is equivalent to *active BDM mode*.

Section 11 Inter-IC Bus (IIC) Block Description

Consult the IIC Block Guide for information about the Inter-IC Bus module.

Section 12 Serial Communications Interface (SCI) Block Description

There are two Serial Communications Interfaces (SCI1 and SCI0) implemented on the MC9S12DP512 device.

Consult the SCI Block Guide for information about each Serial Communications Interface module.

Section 13 Serial Peripheral Interface (SPI) Block Description

There are three Serial Peripheral Interfaces (SPI2, SPI1 and SPI0) implemented on MC9S12DP512.

Consult the SPI Block Guide for information about each Serial Peripheral Interface module.

Section 14 J1850 (BDLC) Block Description

Consult the BDLC Block Guide for information about the J1850 module.

Section 15 Pulse Width Modulator (PWM) Block Description

Consult the PWM_8B6C Block Guide for information about the Pulse Width Modulator module.

When the PWM_8B8C Block Guide refers to *freeze mode* this is equivalent to *active BDM mode*.

Section 16 Flash EEPROM 512K Block Description

Consult the FTS512K4 Block Guide for information about the flash module.

The "S12 LRAE" is a generic Load RAM and Execute (LRAE) program which will be programmed into the flash memory of this device during manufacture. This LRAE program will provide greater programming flexibility to the end users by allowing the device to be programmed directly using CAN or SCI after it is assembled on the PCB. Use of the LRAE program is at the discretion of the end user and, if not required, it must simply be erased prior to flash programming. For more details of the S12 LRAE and its implementation, please see the S12 LREA Application Note (AN2546/D).

It is planned that most HC9S12 devices manufactured after Q1 of 2004 will be shipped with the S12 LRAE programmed in the Flash. Exact details of the changeover (i.e. blank to programmed) for each product will be communicated in advance via GPCN and will be traceable by the customer via datecode marking on the device.

Please contact Motorola SPS Sales if you have any additional questions.

Section 17 EEPROM 4K Block Description

Consult the EETS4K Block Guide for information about the EEPROM module.

Section 18 RAM Block Description

This module supports single-cycle misaligned word accesses.

Section 19 MSCAN Block Description

There are five MSCAN modules (CAN4, CAN3, CAN2, CAN1 and CAN0) implemented on the MC9S12DP512.

Consult the MSCAN Block Guide for information about the Motorola Scalable CAN Module.

Section 20 Port Integration Module (PIM) Block Description

Consult the functionally equivalent PIM_9DP256 Block Guide for information about the Port Integration Module.

Section 21 Voltage Regulator (VREG) Block Description

Consult the VREG Block Guide for information about the dual output linear voltage regulator.

Section 22 Printed Circuit Board Layout Proposal

Table 22-1 Suggested External Component Values

Component	Purpose	Type	Value
C1	VDD1 filter cap	ceramic X7R	100 … 220nF
C2	VDD2 filter cap	ceramic X7R	100 … 220nF
C3	VDDA filter cap	ceramic X7R	100nF
C4	VDDR filter cap	X7R/tantalum	>= 100nF
C5	VDDPLL filter cap	ceramic X7R	100nF
C6	VDDX filter cap	X7R/tantalum	>= 100nF
C7	OSC load cap		
C8	OSC load cap		
C9 / C_S	PLL loop filter cap	See PLL specification chapter	
C10 / C_P	PLL loop filter cap		
C11 / C_{DC}	DC cutoff cap	Colpitts mode only, if recommended by quartz manufacturer	
R1 / R	PLL loop filter res	See PLL Specification chapter	
R2 / R_B		Pierce mode only	
R3 / R_S			
Q1	Quartz		

The PCB must be carefully laid out to ensure proper operation of the voltage regulator as well as of the MCU itself. The following rules must be observed:

- Every supply pair must be decoupled by a ceramic capacitor connected as near as possible to the corresponding pins (C1 – C6).
- Central point of the ground star should be the VSSR pin.
- Use low ohmic low inductance connections between VSS1, VSS2 and VSSR.
- VSSPLL must be directly connected to VSSR.
- Keep traces of VSSPLL, EXTAL and XTAL as short as possible and occupied board area for C7, C8, C11 and Q1 as small as possible.
- Do not place other signals or supplies underneath area occupied by C7, C8, C10 and Q1 and the connection area to the MCU.
- Central power input should be fed in at the VDDA/VSSA pins.

Figure 22-1 Recommended PCB Layout for 112LQFP Colpitts Oscillator

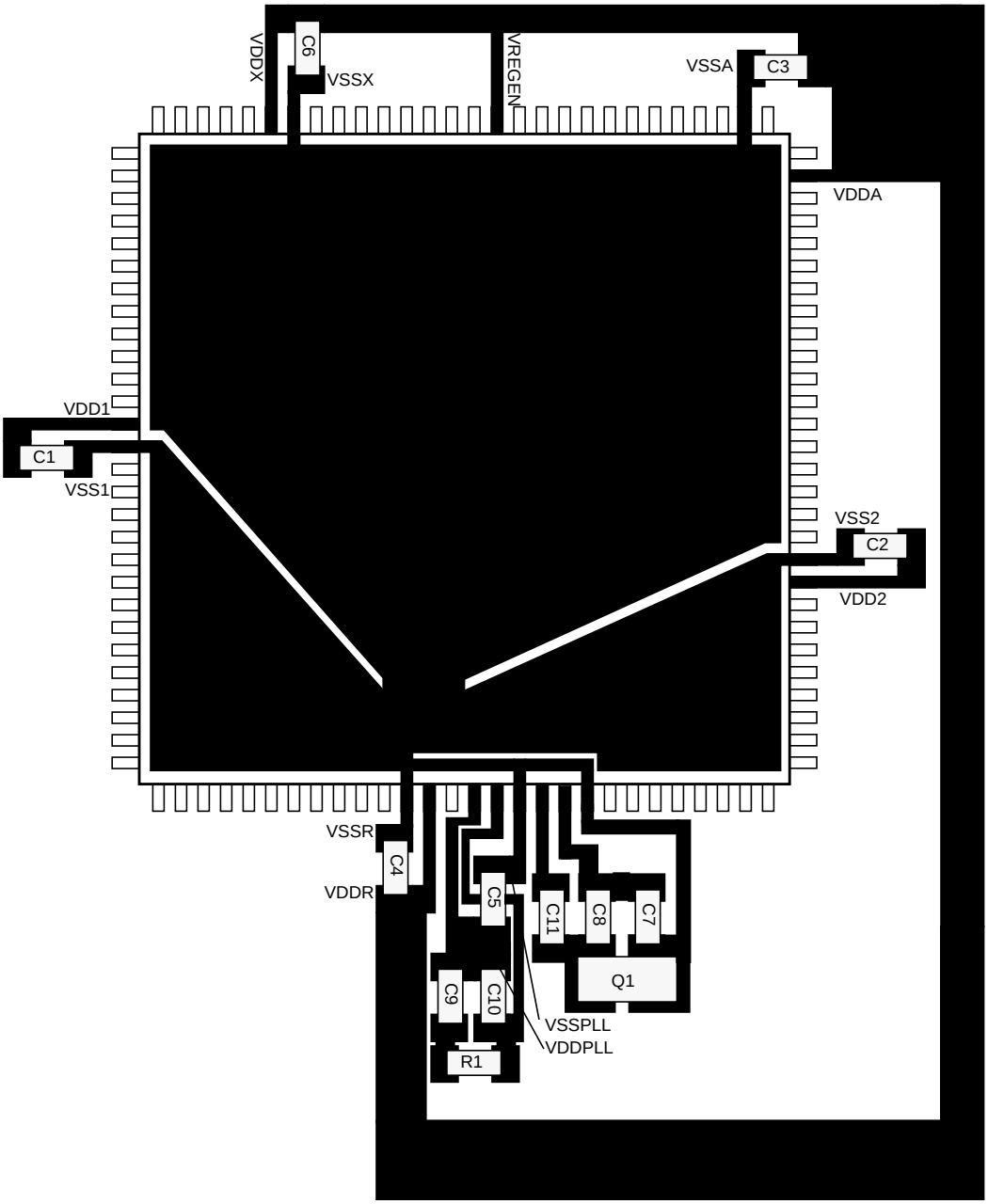
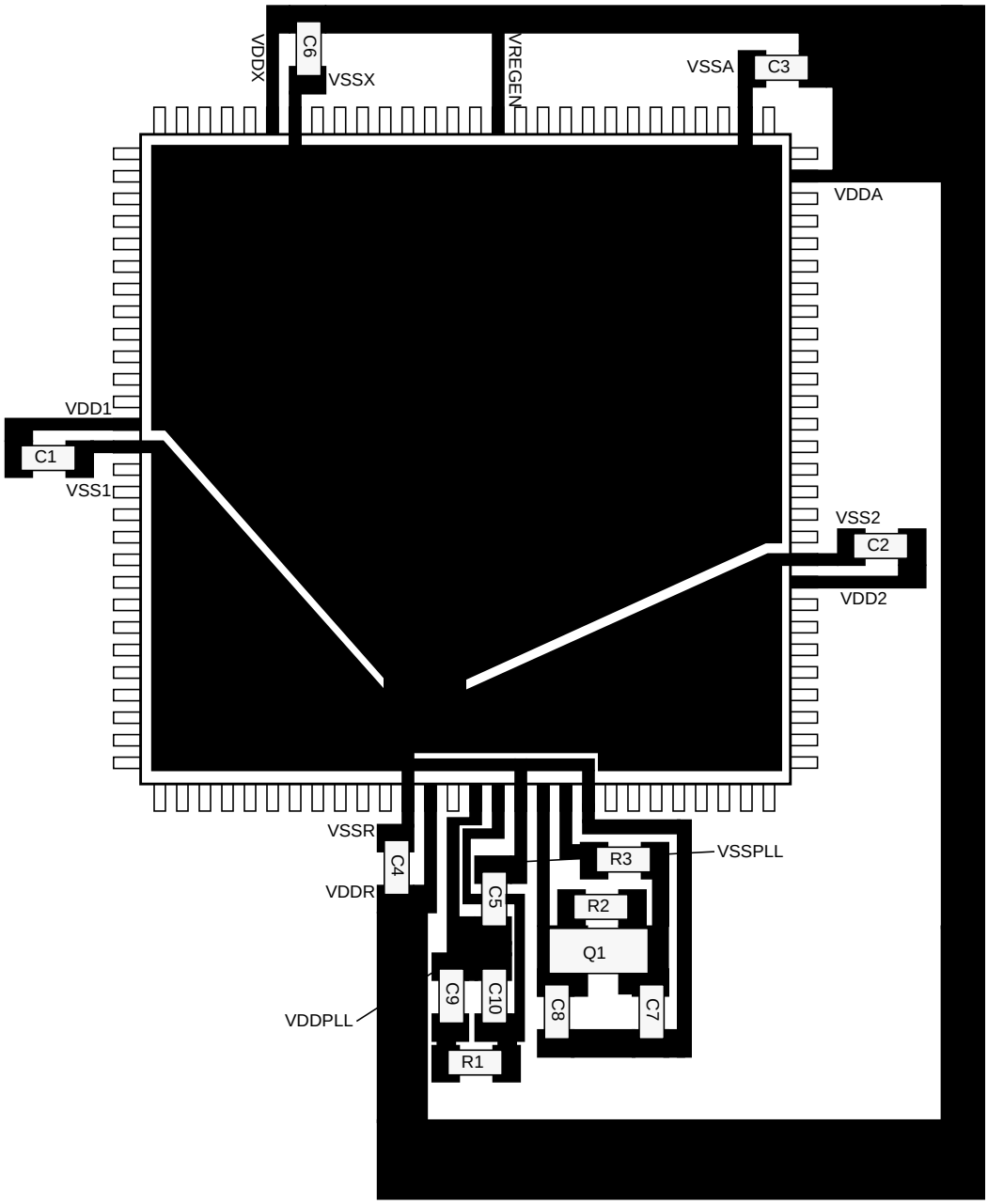


Figure 22-2 Recommended PCB Layout for 112LQFP Pierce Oscillator



Appendix A Electrical Characteristics

A.1 General

NOTE: *The electrical characteristics given in this section are preliminary and should be used as a guide only. Values cannot be guaranteed by Motorola and are subject to change without notice.*

This supplement contains the most accurate electrical information for the MC9S12DP512 microcontroller available at the time of publication. The information should be considered **PRELIMINARY** and is subject to change.

This introduction is intended to give an overview on several common topics like power supply, current injection etc.

A.1.1 Parameter Classification

The electrical parameters shown in this supplement are guaranteed by various methods. To give the customer a better understanding the following classification is used and the parameters are tagged accordingly in the tables where appropriate.

NOTE: *This classification is shown in the column labeled "C" in the parameter tables where appropriate.*

P:

Those parameters are guaranteed during production testing on each individual device.

C:

Those parameters are achieved by the design characterization by measuring a statistically relevant sample size across process variations.

T:

Those parameters are achieved by design characterization on a small sample size from typical devices under typical conditions unless otherwise noted. All values shown in the typical column are within this category.

D:

Those parameters are derived mainly from simulations.

A.1.2 Power Supply

The MC9S12DP512 utilizes several pins to supply power to the I/O ports, A/D converter, oscillator, PLL and internal logic.

The VDDA, VSSA pair supplies the A/D converter and the resistor ladder of the internal voltage regulator.

The VDDX, VSSX, VDDR and VSSR pairs supply the I/O pins, VDDR supplies also the internal voltage regulator.

VDD1, VSS1, VDD2 and VSS2 are the supply pins for the digital logic, VDDPLL, VSSPLL supply the oscillator and the PLL.

VSS1 and VSS2 are internally connected by metal.

VDDA, VDDX, VDDR as well as VSSA, VSSX, VSSR are connected by anti-parallel diodes for ESD protection.

NOTE: *In the following context VDD5 is used for either VDDA, VDDR and VDDX; VSS5 is used for either VSSA, VSSR and VSSX unless otherwise noted.*
IDD5 denotes the sum of the currents flowing into the VDDA, VDDX and VDDR pins.
VDD is used for VDD1, VDD2 and VDDPLL, VSS is used for VSS1, VSS2 and VSSPLL.
IDD is used for the sum of the currents flowing into VDD1 and VDD2.

A.1.3 Pins

There are four groups of functional pins.

A.1.3.1 5V I/O pins

Those I/O pins have a nominal level of 5V. This class of pins is comprised of all port I/O pins, the analog inputs, BKGD and the RESET pins.The internal structure of all those pins is identical, however some of the functionality may be disabled. E.g. for the analog inputs the output drivers, pull-up and pull-down resistors are disabled permanently.

A.1.3.2 Analog Reference

This group is made up by the VRH and VRL pins.

A.1.3.3 Oscillator

The pins XFC, EXTAL, XTAL dedicated to the oscillator have a nominal 2.5V level. They are supplied by VDDPLL.

A.1.3.4 TEST

This pin is used for production testing only.

A.1.3.5 VREGEN

This pin is used to enable the on chip voltage regulator.

A.1.4 Current Injection

Power supply must maintain regulation within operating V_{DD5} or V_{DD} range during instantaneous and operating maximum current conditions. If positive injection current ($V_{in} > V_{DD5}$) is greater than I_{DD5}, the injection current may flow out of VDD5 and could result in external power supply going out of regulation. Ensure external VDD5 load will shunt current greater than maximum injection current. This will be the greatest risk when the MCU is not consuming power; e.g. if no system clock is present, or if clock rate is very low which would reduce overall power consumption.

A.1.5 Absolute Maximum Ratings

Absolute maximum ratings are stress ratings only. A functional operation under or outside those maxima is not guaranteed. Stress beyond those limits may affect the reliability or cause permanent damage of the device.

This device contains circuitry protecting against damage due to high static voltage or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either V_{SS5} or V_{DD5}).

Table A-1 Absolute Maximum Ratings[1]

Num	Rating	Symbol	Min	Max	Unit
1	I/O, Regulator and Analog Supply Voltage	V_{DD5}	-0.3	6.0	V
2	Digital Logic Supply Voltage [2]	V_{DD}	-0.3	3.0	V
3	PLL Supply Voltage [(2)]	V_{DDPLL}	-0.3	3.0	V
4	Voltage difference VDDX to VDDR and VDDA	Δ_{VDDX}	-0.3	0.3	V
5	Voltage difference VSSX to VSSR and VSSA	Δ_{VSSX}	-0.3	0.3	V
6	Digital I/O Input Voltage	V_{IN}	-0.3	6.0	V
7	Analog Reference	$V_{RH,}$ V_{RL}	-0.3	6.0	V
8	XFC, EXTAL, XTAL inputs	V_{ILV}	-0.3	3.0	V
9	TEST input	V_{TEST}	-0.3	10.0	V
10	Instantaneous Maximum Current Single pin limit for all digital I/O pins [3]	I_D	-25	+25	mA
11	Instantaneous Maximum Current Single pin limit for XFC, EXTAL, XTAL[4]	I_{DL}	-25	+25	mA
12	Instantaneous Maximum Current Single pin limit for TEST [5]	I_{DT}	-0.25	0	mA
13	Storage Temperature Range	T_{stg}	– 65	155	°C

NOTES:
1. Beyond absolute maximum ratings device might be damaged.

2. The device contains an internal voltage regulator to generate the logic and PLL supply out of the I/O supply. The absolute maximum ratings apply when the device is powered from an external source.
3. All digital I/O pins are internally clamped to V_{SSX} and V_{DDX}, V_{SSR} and V_{DDR} or V_{SSA} and V_{DDA}.
4. Those pins are internally clamped to V_{SSPLL} and V_{DDPLL}.
5. This pin is clamped low to V_{SSX}, but not clamped high. This pin must be tied low in applications.

A.1.6 ESD Protection and Latch-up Immunity

All ESD testing is in conformity with CDF-AEC-Q100 Stress test qualification for Automotive Grade Integrated Circuits. During the device qualification ESD stresses were performed for the Human Body Model (HBM), the Machine Model (MM) and the Charge Device Model.

A device will be defined as a failure if after exposure to ESD pulses the device no longer meets the device specification. Complete DC parametric and functional testing is performed per the applicable device specification at room temperature followed by hot temperature, unless specified otherwise in the device specification.

Table A-2 ESD and Latch-up Test Conditions

Model	Description	Symbol	Value	Unit
Human Body	Series Resistance	R1	1500	Ohm
	Storage Capacitance	C	100	pF
	Number of Pulse per pin positive negative	-	- 3 3	
Machine	Series Resistance	R1	0	Ohm
	Storage Capacitance	C	200	pF
	Number of Pulse per pin positive negative	-	- 3 3	
Latch-up	Minimum input voltage limit		-2.5	V
	Maximum input voltage limit		7.5	V

Table A-3 ESD and Latch-up Protection Characteristics

Num	C	Rating	Symbol	Min	Max	Unit
1	C	Human Body Model (HBM)	V_{HBM}	2000	-	V
2	C	Machine Model (MM)	V_{MM}	200	-	V
3	C	Charge Device Model (CDM)	V_{CDM}	500	-	V
4	C	Latch-up Current at T_A = 125°C positive negative	I_{LAT}	+100 -100	-	mA
5	C	Latch-up Current at T_A = 27°C positive negative	I_{LAT}	+200 -200	-	mA

A.1.7 Operating Conditions

This chapter describes the operating conditions of the device. Unless otherwise noted those conditions apply to all the following data.

NOTE: *Please refer to the temperature rating of the device (C, V, M) with regards to the ambient temperature T_A and the junction temperature T_J. For power dissipation calculations refer to* ***Section A.1.8 Power Dissipation and Thermal Characteristics****.*

Table A-4 Operating Conditions

Rating	Symbol	Min	Typ	Max	Unit
I/O, Regulator and Analog Supply Voltage	V_{DD5}	4.5	5	5.25	V
Digital Logic Supply Voltage [1]	V_{DD}	2.35	2.5	2.75	V
PLL Supply Voltage (1)	V_{DDPLL}	2.35	2.5	2.75	V
Voltage Difference VDDX to VDDR and VDDA	Δ_{VDDX}	-0.1	0	0.1	V
Voltage Difference VSSX to VSSR and VSSA	Δ_{VSSX}	-0.1	0	0.1	V
Bus Frequency (MC9S12DP512C, V, M)	f_{bus}	0.25 [2]	-	25	MHz
MC9S12DP512**C** Operating Junction Temperature Range	T_J	-40	-	100	°C
Operating Ambient Temperature Range [3]	T_A	-40	27	85	°C
MC9S12DP512**V** Operating Junction Temperature Range	T_J	-40	-	120	°C
Operating Ambient Temperature Range (3)	T_A	-40	27	105	°C
MC9S12DP512**M** Operating Junction Temperature Range	T_J	-40	-	140	°C
Operating Ambient Temperature Range (3)	T_A	-40	27	125	°C

NOTES:
1. The device contains an internal voltage regulator to generate the logic and PLL supply out of the I/O supply. The given operating range applies when this regulator is disabled and the device is powered from an external source.
2. Some blocks e.g. ATD (conversion) and NVMs (program/erase) require higher bus frequencies for proper operation.
3. Please refer to **Section A.1.8 Power Dissipation and Thermal Characteristics** for more details about the relation between ambient temperature T_A and device junction temperature T_J.

A.1.8 Power Dissipation and Thermal Characteristics

Power dissipation and thermal characteristics are closely related. The user must assure that the maximum operating junction temperature is not exceeded. The average chip-junction temperature (T_J) in °C can be obtained from:

$$T_J = T_A + (P_D \bullet \Theta_{JA})$$

T_J = Junction Temperature, [°C]

T_A = Ambient Temperature, [°C]

P_D = Total Chip Power Dissipation, [W]

Θ_{JA} = Package Thermal Resistance, [°C/W]

The total power dissipation can be calculated from:

$$P_D = P_{INT} + P_{IO}$$

P_{INT} = Chip Internal Power Dissipation, [W]

Two cases with internal voltage regulator enabled and disabled must be considered:

1. Internal Voltage Regulator disabled

$$P_{INT} = I_{DD} \cdot V_{DD} + I_{DDPLL} \cdot V_{DDPLL} + I_{DDA} \cdot V_{DDA}$$

$$P_{IO} = \sum_i R_{DSON} \cdot I_{IO_i}^2$$

P_{IO} is the sum of all output currents on I/O ports associated with VDDX and VDDR.

For R_{DSON} is valid:

$$R_{DSON} = \frac{V_{OL}}{I_{OL}}; \text{for outputs driven low}$$

respectively

$$R_{DSON} = \frac{V_{DD5} - V_{OH}}{I_{OH}}; \text{for outputs driven high}$$

2. Internal voltage regulator enabled

$$P_{INT} = I_{DDR} \cdot V_{DDR} + I_{DDA} \cdot V_{DDA}$$

I_{DDR} is the current shown in **Table A-7** and not the overall current flowing into VDDR, which additionally contains the current flowing into the external loads with output high.

$$P_{IO} = \sum_i R_{DSON} \cdot I_{IO_i}^2$$

P_{IO} is the sum of all output currents on I/O ports associated with VDDX and VDDR.

Table A-5 Thermal Package Characteristics[1]

Num	C	Rating	Symbol	Min	Typ	Max	Unit
1	T	Thermal Resistance LQFP112, single sided PCB[2]	θ_{JA}	-	-	54	$^{o}C/W$
2	T	Thermal Resistance LQFP112, double sided PCB with 2 internal planes[3]	θ_{JA}	-	-	41	$^{o}C/W$

NOTES:
1. The values for thermal resistance are achieved by package simulations
2. PC Board according to EIA/JEDEC Standard 51-2
3. PC Board according to EIA/JEDEC Standard 51-7

A.1.9 I/O Characteristics

This section describes the characteristics of all 5V I/O pins. All parameters are not always applicable, e.g. not all pins feature pull up/down resistances.

Table A-6 5V I/O Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	Input High Voltage	V_{IH}	0.65*V_{DD5}	-	-	V
	T	Input High Voltage	V_{IH}	-	-	VDD5 + 0.3	V
2	P	Input Low Voltage	V_{IL}	-	-	0.35*V_{DD5}	V
	T	Input Low Voltage	V_{IL}	VSS5 - 0.3	-	-	V
3	C	Input Hysteresis	V_{HYS}	-	250	-	mV
4	P	Input Leakage Current (pins in high impedance input mode) V_{in} = V_{DD5} or V_{SS5}	I_{in}	–1	-	1	µA
5	P	Output High Voltage (pins in output mode) Partial Drive I_{OH} = –2mA Full Drive I_{OH} = –10mA	V_{OH}	V_{DD5} – 0.8	-	-	V
6	P	Output Low Voltage (pins in output mode) Partial Drive I_{OL} = +2mA Full Drive I_{OL} = +10mA	V_{OL}	-	-	0.8	V
7	P	Internal Pull Up Device Current, tested at V_{IL} Max.	I_{PUL}	-	-	-130	µA
8	C	Internal Pull Up Device Current, tested at V_{IH} Min.	I_{PUH}	-10	-	-	µA
9	P	Internal Pull Down Device Current, tested at V_{IH} Min.	I_{PDH}	-	-	130	µA
10	C	Internal Pull Down Device Current, tested at V_{IL} Max.	I_{PDL}	10	-	-	µA
11	D	Input Capacitance	C_{in}	-	6	-	pF
12	T	Injection current[1] Single Pin limit Total Device Limit. Sum of all injected currents	 I_{ICS} I_{ICP}	 -2.5 -25	-	 2.5 25	mA
13	P	Port H, J, P Interrupt Input Pulse filtered[2]	t_{PIGN}	-	-	3	µs
14	P	Port H, J, P Interrupt Input Pulse passed(2)	t_{PVAL}	10	-	-	µs

NOTES:
1. Refer to **Section A.1.4 Current Injection**, for more details
2. Parameter only applies in STOP or Pseudo STOP mode.

A.1.10 Supply Currents

This section describes the current consumption characteristics of the device as well as the conditions for the measurements.

A.1.10.1 Measurement Conditions

All measurements are without output loads. Unless otherwise noted the currents are measured in single chip mode, internal voltage regulator enabled and at 25MHz bus frequency using a 4MHz oscillator in Colpitts mode. Production testing is performed using a square wave signal at the EXTAL input.

A.1.10.2 Additional Remarks

In expanded modes the currents flowing in the system are highly dependent on the load at the address, data and control signals as well as on the duty cycle of those signals. No generally applicable numbers can be given. A very good estimate is to take the single chip currents and add the currents due to the external loads.

Table A-7 Supply Current Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	Run supply currents Single Chip, Internal regulator enabled	I_{DD5}	-	-	65	mA
2	P P	Wait Supply current All modules enabled, PLL on only RTI enabled (1)	I_{DDW}	-	-	40 5	mA
3	C P C C P C P C P	Pseudo Stop Current (RTI and COP disabled) 1, 2 -40°C 27°C 70°C 85°C "C" Temp Option 100°C 105°C "V" Temp Option 120°C 125°C "M" Temp Option 140°C	I_{DDPS}	-	370 400 450 550 600 650 800 850 1200	500 1600 2100 5000	µA
4	C C C C C C C	Pseudo Stop Current (RTI and COP enabled) (1), (2) -40°C 27°C 70°C 85°C 105°C 125°C 140°C	I_{DDPS}	-	570 600 650 750 850 1200 1500	-	µA
5	C P C C P C P C P	Stop Current (2) -40°C 27°C 70°C 85°C "C" Temp Option 100°C 105°C "V" Temp Option 120°C 125°C "M" Temp Option 140°C	I_{DDS}	-	12 25 100 130 160 200 350 400 600	100 1200 1700 5000	µA

NOTES:
1. PLL off
2. At those low power dissipation levels $T_J = T_A$ can be assumed

A.2 ATD Characteristics

This section describes the characteristics of the analog to digital converter.

A.2.1 ATD Operating Characteristics

The **Table A-8** shows conditions under which the ATD operates.

The following constraints exist to obtain full-scale, full range results:
$V_{SSA} \le V_{RL} \le V_{IN} \le V_{RH} \le V_{DDA}$. This constraint exists since the sample buffer amplifier can not drive beyond the power supply levels that it ties to. If the input level goes outside of this range it will effectively be clipped.

Table A-8 ATD Operating Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	D	Reference Potential Low High	 V_{RL} V_{RH}	 V_{SSA} $V_{DDA}/2$	-	 $V_{DDA}/2$ V_{DDA}	 V V
2	C	Differential Reference Voltage[1]	$V_{RH}-V_{RL}$	4.50	5.00	5.25	V
3	D	ATD Clock Frequency	f_{ATDCLK}	0.5	-	2.0	MHz
4	D	ATD 10-Bit Conversion Period Clock Cycles[2] Conv, Time at 2.0MHz ATD Clock f_{ATDCLK}	 N_{CONV10} T_{CONV10}	 14 7	-	 28 14	 Cycles µs
5	D	ATD 8-Bit Conversion Period Clock Cycles(2) Conv, Time at 2.0MHz ATD Clock f_{ATDCLK}	 N_{CONV8} T_{CONV8}	 12 6	-	 26 13	 Cycles µs
6	D	Recovery Time (V_{DDA}=5.0 Volts)	t_{REC}	-	-	20	µs
7	P	Reference Supply current 2 ATD blocks on	I_{REF}	-	-	0.750	mA
8	P	Reference Supply current 1 ATD block on	I_{REF}	-	-	0.375	mA

NOTES:
1. Full accuracy is not guaranteed when differential voltage is less than 4.50V
2. The minimum time assumes a final sample period of 2 ATD clocks cycles while the maximum time assumes a final sample period of 16 ATD clocks.

A.2.2 Factors influencing accuracy

Three factors - source resistance, source capacitance and current injection - have an influence on the accuracy of the ATD.

A.2.2.1 Source Resistance

Due to the input pin leakage current as specified in **Table A-6** in conjunction with the source resistance there will be a voltage drop from the signal source to the ATD input. The maximum source resistance R_S

specifies results in an error of less than 1/2 LSB (2.5mV) at the maximum leakage current. If device or operating conditions are less than worst case or leakage-induced error is acceptable, larger values of source resistance is allowed.

A.2.2.2 Source Capacitance

When sampling an additional internal capacitor is switched to the input. This can cause a voltage drop due to charge sharing with the external and the pin capacitance. For a maximum sampling error of the input voltage ≤ 1LSB, then the external filter capacitor, $C_f \geq 1024 * (C_{INS} - C_{INN})$.

A.2.2.3 Current Injection

There are two cases to consider.

1. A current is injected into the channel being converted. The channel being stressed has conversion values of $3FF ($FF in 8-bit mode) for analog inputs greater than V_{RH} and $000 for values less than V_{RL} unless the current is higher than specified as disruptive condition.
2. Current is injected into pins in the neighborhood of the channel being converted. A portion of this current is picked up by the channel (coupling ratio K), This additional current impacts the accuracy of the conversion depending on the source resistance.
   The additional input voltage error on the converted channel can be calculated as $V_{ERR} = K * R_S * I_{INJ}$, with I_{INJ} being the sum of the currents injected into the two pins adjacent to the converted channel.

Table A-9 ATD Electrical Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	C	Max input Source Resistance	R_S	-	-	1	KΩ
2	T	Total Input Capacitance Non Sampling Sampling	 C_{INN} C_{INS}	-	-	 10 22	pF
3	C	Disruptive Analog Input Current	I_{NA}	-2.5	-	2.5	mA
4	C	Coupling Ratio positive current injection	K_p	-	-	10^{-4}	A/A
5	C	Coupling Ratio negative current injection	K_n	-	-	10^{-2}	A/A

A.2.3 ATD accuracy

Table A-10 specifies the ATD conversion performance excluding any errors due to current injection, input capacitance and source resistance.

Table A-10 ATD Conversion Performance

Conditions are shown in **Table A-4** unless otherwise noted $V_{REF} = V_{RH} - V_{RL}$ = 5.12V. Resulting to one 8 bit count = 20mV and one 10 bit count = 5mV f_{ATDCLK} = 2.0MHz							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	10-Bit Resolution	LSB	-	5	-	mV
2	P	10-Bit Differential Nonlinearity	DNL	–1	-	1	Counts
3	P	10-Bit Integral Nonlinearity	INL	–2.5	±1.5	2.5	Counts
4	P	10-Bit Absolute Error[1]	AE	-3	±2.0	3	Counts
5	P	8-Bit Resolution	LSB	-	20	-	mV
6	P	8-Bit Differential Nonlinearity	DNL	–0.5	-	0.5	Counts
7	P	8-Bit Integral Nonlinearity	INL	–1.0	±0.5	1.0	Counts
8	P	8-Bit Absolute Error(1)	AE	-1.5	±1.0	1.5	Counts

NOTES:
1. These values include the quantization error which is inherently 1/2 count for any A/D converter.

For the following definitions see also **Figure A-1**.

Differential Non-Linearity (DNL) is defined as the difference between two adjacent switching steps.

$$DNL(i) = \frac{V_i - V_{i-1}}{1LSB} - 1$$

The Integral Non-Linearity (INL) is defined as the sum of all DNLs:

$$INL(n) = \sum_{i=1}^{n} DNL(i) = \frac{V_n - V_0}{1LSB} - n$$

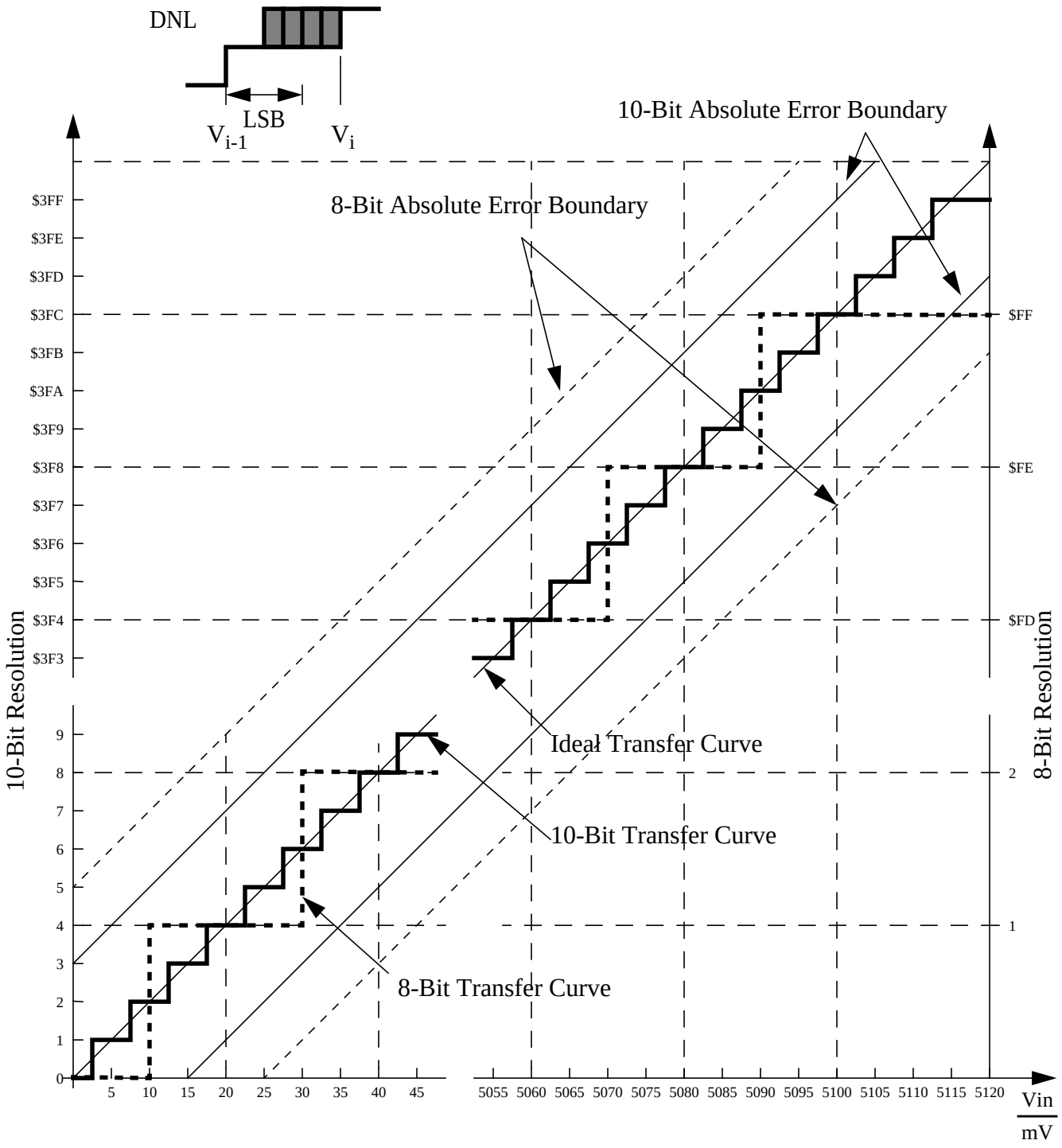


Figure A-1 ATD Accuracy Definitions

NOTE: ***Figure A-1*** *shows only definitions, for specification values refer to* ***Table A-10****.*

A.3 NVM, Flash and EEPROM

NOTE: *Unless otherwise noted the abbreviation NVM (Non Volatile Memory) is used for both Flash and EEPROM.*

A.3.1 NVM timing

The time base for all NVM program or erase operations is derived from the oscillator. A minimum oscillator frequency f_{NVMOSC} is required for performing program or erase operations. The NVM modules do not have any means to monitor the frequency and will not prevent program or erase operation at frequencies above or below the specified minimum. Attempting to program or erase the NVM modules at a lower frequency a full program or erase transition is not assured.

The Flash and EEPROM program and erase operations are timed using a clock derived from the oscillator using the FCLKDIV and ECLKDIV registers respectively. The frequency of this clock must be set within the limits specified as f_{NVMOP}.

The minimum program and erase times shown in **Table A-11** are calculated for maximum f_{NVMOP} and maximum f_{bus}. The maximum times are calculated for minimum f_{NVMOP} and a f_{bus} of 2MHz.

A.3.1.1 Single Word Programming

The programming time for single word programming is dependant on the bus frequency as a well as on the frequency f_{NVMOP} and can be calculated according to the following formula.

$$t_{swpgm} = 9 \cdot \frac{1}{f_{NVMOP}} + 25 \cdot \frac{1}{f_{bus}}$$

A.3.1.2 Row Programming

This applies only to the Flash where up to 64 words in a row can be programmed consecutively by keeping the command pipeline filled. The time to program a consecutive word can be calculated as:

$$t_{bwpgm} = 4 \cdot \frac{1}{f_{NVMOP}} + 9 \cdot \frac{1}{f_{bus}}$$

The time to program a whole row is:

$$t_{brpgm} = t_{swpgm} + 63 \cdot t_{bwpgm}$$

Row programming is more than 2 times faster than single word programming.

A.3.1.3 Sector Erase

Erasing a 1024 byte Flash sector or a 4 byte EEPROM sector takes:

$$t_{era} \approx 4000 \cdot \frac{1}{f_{NVMOP}}$$

The setup time can be ignored for this operation.

A.3.1.4 Mass Erase

Erasing a NVM block takes:

$$t_{mass} \approx 20000 \cdot \frac{1}{f_{NVMOP}}$$

The setup time can be ignored for this operation.

A.3.1.5 Blank Check

The time it takes to perform a blank check on the Flash or EEPROM is dependant on the location of the first non-blank word starting at relative address zero. It takes one bus cycle per word to verify plus a setup of the command.

$$t_{check} \approx location \cdot t_{cyc} + 10 \cdot t_{cyc}$$

Table A-11 NVM Timing Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	D	External Oscillator Clock	f_{NVMOSC}	0.5	-	50 [1]	MHz
2	D	Bus frequency for Programming or Erase Operations	f_{NVMBUS}	1	-	-	MHz
3	D	Operating Frequency	f_{NVMOP}	150	-	200	kHz
4	P	Single Word Programming Time	t_{swpgm}	46 [2]	-	74.5 [3]	µs
5	D	Flash Row Programming consecutive word [4]	t_{bwpgm}	20.4 (2)	-	31 (3)	µs
6	D	Flash Row Programming Time for 64 Words (4)	t_{brpgm}	1331.2 (2)	-	2027.5 (3)	µs
7	P	Sector Erase Time	t_{era}	20 [5]	-	26.7 (3)	ms
8	P	Mass Erase Time	t_{mass}	100 (5)	-	133 (3)	ms
9	D	Blank Check Time Flash per block	t_{check}	11 [6]	-	65546 [7]	t_{cyc}
10	D	Blank Check Time EEPROM per block	t_{check}	11 (6)	-	2058 (7)	t_{cyc}

NOTES:
1. Restrictions for oscillator in crystal mode apply!
2. Minimum Programming times are achieved under maximum NVM operating frequency f_{NVMOP} and maximum bus frequency f_{bus}.
3. Maximum Erase and Programming times are achieved under particular combinations of f_{NVMOP} and bus frequency f_{bus}. Refer to formulae in Sections **Section A.3.1.1 Single Word Programming- Section A.3.1.4 Mass Erase**for guidance.
4. Row Programming operations are not applicable to EEPROM
5. Minimum Erase times are achieved under maximum NVM operating frequency f_{NVMOP}.
6. Minimum time, if first word in the array is not blank
7. Maximum time to complete check on an erased block

A.3.2 NVM Reliability

The reliability of the NVM blocks is guaranteed by stress test during qualification, constant process monitors and burn-in to screen early life failures.

The program/erase cycle count on the sector is incremented every time a sector or mass erase event is executed

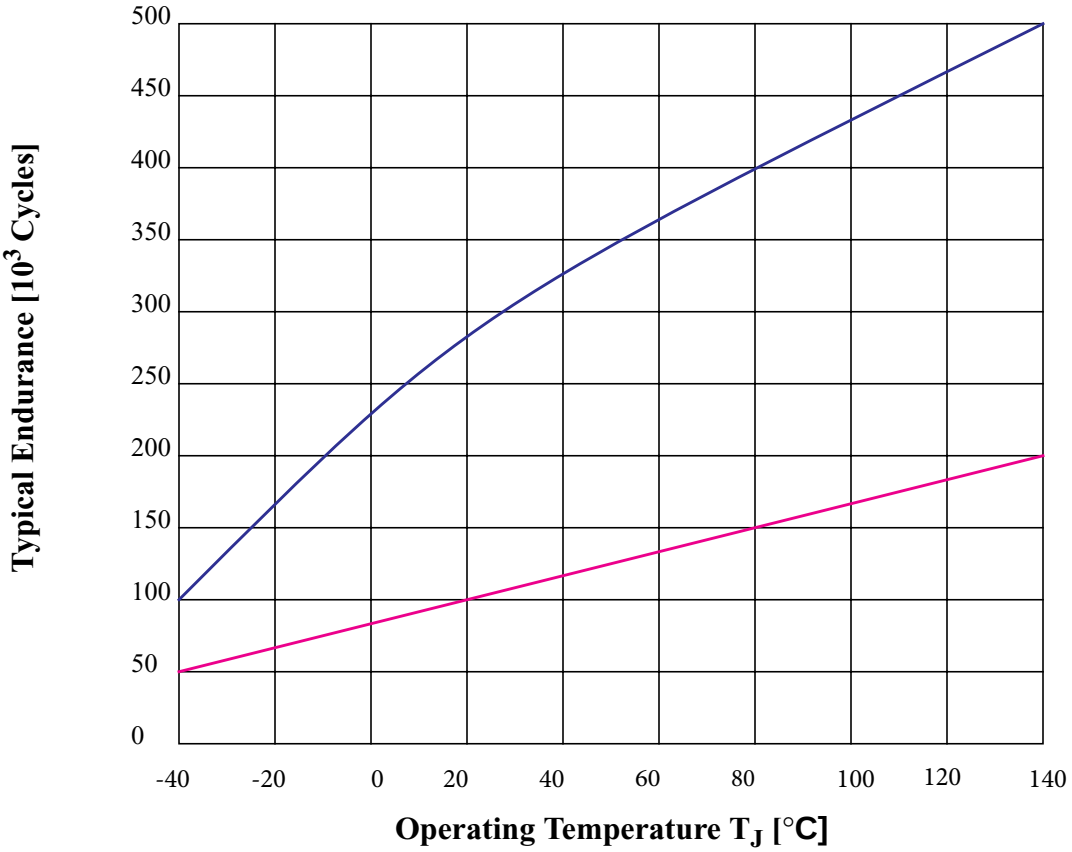
Table A-12 NVM Reliability Characteristics[1]

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
Flash Reliability Characteristics							
1	C	Data retention after 10,000 program/erase cycles at an average junction temperature of $T_{Javg} \leq 85°C$	t_{FLRET}	15	100[2]	—	Years
2	C	Data retention with <100 program/erase cycles at an average junction temperature $T_{Javg} \leq 85°C$		20	100[2]	—	
3	C	Number of program/erase cycles ($-40°C \leq T_J \leq 0°C$)	n_{FL}	10,000	—	—	Cycles
4	C	Number of program/erase cycles ($0°C \leq T_J \leq 140°C$)		10,000	100,000[3]	—	
EEPROM Reliability Characteristics							
5	C	Data retention after up to 100,000 program/erase cycles at an average junction temperature of $T_{Javg} \leq 85°C$	t_{EEPRET}	15	100[2]	—	Years
6	C	Data retention with <100 program/erase cycles at an average junction temperature $T_{Javg} \leq 85°C$		20	100[2]	—	
7	C	Number of program/erase cycles ($-40°C \leq T_J \leq 0°C$)	n_{EEP}	10,000	—	—	Cycles
8	C	Number of program/erase cycles ($0°C < T_J \leq 140°C$)		100,000	300,000[3]	—	

NOTES:

1. T_{Javg} will not exeed 85°C considering a typical temperature profile over the lifetime of a consumer, industrial or automotive application.
2. Typical data retention values are based on intrinsic capability of the technology measured at high temperature and de-rated to 25°C using the Arrhenius equation. For additional information on how Freescale defines Typical Data Retention, please refer to Engineering Bulletin EB618.
3. Spec table quotes typical endurance evaluated at 25°C for this product family, typical endurance at various temperature can be estimated using the graph below. For additional information on how Freescale defines Typical Endurance, please refer to Engineering Bulletin EB619.

Figure A-2 Typical Endurance vs Temperature



A.4 Voltage Regulator

The on-chip voltage regulator is intended to supply the internal logic and oscillator circuits. No external DC load is allowed.

Table A-13 Voltage Regulator Recommended Load Capacitances

Rating	Symbol	Min	Typ	Max	Unit
Load Capacitance on VDD1, 2	C_{LVDD}	-	220	-	nF
Load Capacitance on VDDPLL	$C_{LVDDfcPLL}$	-	220	-	nF

A.5 Reset, Oscillator and PLL

This section summarizes the electrical characteristics of the various startup scenarios for Oscillator and Phase-Locked-Loop (PLL).

A.5.1 Startup

Table A-14 summarizes several startup characteristics explained in this section. Detailed description of the startup behavior can be found in the Clock and Reset Generator (CRG) Block Guide.

Table A-14 Startup Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	T	POR release level	V_{PORR}	-	-	2.07	V
2	T	POR assert level	V_{PORA}	0.97	-	-	V
3	D	Reset input pulse width, minimum input time	PW_{RSTL}	2	-	-	t_{osc}
4	D	Startup from Reset	n_{RST}	192	-	196	n_{osc}
5	D	Interrupt pulse width, $\overline{IRQ}$ edge-sensitive mode	PW_{IRQ}	20	-	-	ns
6	D	Wait recovery startup time	t_{WRS}	-	-	14	t_{cyc}

A.5.1.1 POR

The release level V_{PORR} and the assert level V_{PORA} are derived from the V_{DD} Supply. They are also valid if the device is powered externally. After releasing the POR reset the oscillator and the clock quality check are started. If after a time t_{CQOUT} no valid oscillation is detected, the MCU will start using the internal self clock. The fastest startup time possible is given by n_{uposc}.

A.5.1.2 SRAM Data Retention

Provided an appropriate external reset signal is applied to the MCU, preventing the CPU from executing code when VDD5 is out of specification limits, the SRAM contents integrity is guaranteed if after the reset the PORF bit in the CRG Flags Register has not been set.

A.5.1.3 External Reset

When external reset is asserted for a time greater than PW_{RSTL} the CRG module generates an internal reset, and the CPU starts fetching the reset vector without doing a clock quality check, if there was an oscillation before reset.

A.5.1.4 Stop Recovery

Out of STOP the controller can be woken up by an external interrupt. A clock quality check as after POR is performed before releasing the clocks to the system.

A.5.1.5 Pseudo Stop and Wait Recovery

The recovery from Pseudo STOP and Wait are essentially the same since the oscillator was not stopped in both modes. The controller can be woken up by internal or external interrupts. After t_{wrs} the CPU starts fetching the interrupt vector.

A.5.2 Oscillator

The device features an internal Colpitts and Pierce oscillator. The selection of Colpitts oscillator or Pierce oscillator/external clock depends on the $\overline{XCLKS}$ signal which is sampled during reset. Pierce oscillator/external clock mode allows the input of a square wave. Before asserting the oscillator to the internal system clocks the quality of the oscillation is checked for each start from either power-on, STOP or oscillator fail. t_{CQOUT} specifies the maximum time before switching to the internal self clock mode after POR or STOP if a proper oscillation is not detected. The quality check also determines the minimum oscillator start-up time t_{UPOSC} . The device also features a clock monitor. A Clock Monitor Failure is asserted if the frequency of the incoming clock signal is below the Assert Frequency f_{CMFA}.

Table A-15 Oscillator Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1a	C	Crystal oscillator range (Colpitts)	f_{OSC}	0.5	-	16	MHz
1b	C	Crystal oscillator range (Pierce) [1]	f_{OSC}	0.5	-	40	MHz
2	P	Startup Current	i_{OSC}	100	-	-	µA
3	C	Oscillator start-up time (Colpitts)	t_{UPOSC}	-	8[2]	100[3]	ms
4	D	Clock Quality check time-out	t_{CQOUT}	0.45	-	2.5	s
5	P	Clock Monitor Failure Assert Frequency	f_{CMFA}	50	100	200	KHz
6	P	External square wave input frequency [4]	f_{EXT}	0.5	-	50	MHz
7	D	External square wave pulse width low [4]	t_{EXTL}	9.5	-	-	ns
8	D	External square wave pulse width high [4]	t_{EXTH}	9.5	-	-	ns
9	D	External square wave rise time [4]	t_{EXTR}	-	-	1	ns
10	D	External square wave fall time [4]	t_{EXTF}	-	-	1	ns
11	D	Input Capacitance (EXTAL, XTAL pins)	C_{IN}	-	7	-	pF
12	C	DC Operating Bias in Colpitts Configuration on EXTAL Pin	V_{DCBIAS}	-	1.1	-	V
13	P	EXTAL Pin Input High Voltage [4]	$V_{IH,EXTAL}$	0.75*V_{DDPLL}	-	-	V
	T	EXTAL Pin Input High Voltage [4]	$V_{IH,EXTAL}$	-	-	V_{DDPLL} + 0.3	V
14	P	EXTAL Pin Input Low Voltage [4]	$V_{IL,EXTAL}$	-	-	0.25*V_{SSPLL}	V
	T	EXTAL Pin Input Low Voltage [4]	$V_{IL,EXTAL}$	V_{SSPLL} - 0.3	-	-	V
15	C	EXTAL Pin Input Hysteresis [4]	$V_{HYS,EXTAL}$	-	250	-	mV

NOTES:
1. Depending on the crystal a damping series resistor might be necessary
2. f_{osc} = 4MHz, C = 22pF.
3. Maximum value is for extreme cases using high Q, low frequency crystals
4. Only valid if Pierce oscillator/external clock mode is selected

A.5.3 Phase Locked Loop

The oscillator provides the reference clock for the PLL. The PLL´s Voltage Controlled Oscillator (VCO) is also the system clock source in self clock mode.

A.5.3.1 XFC Component Selection

This section describes the selection of the XFC components to achieve a good filter characteristics.

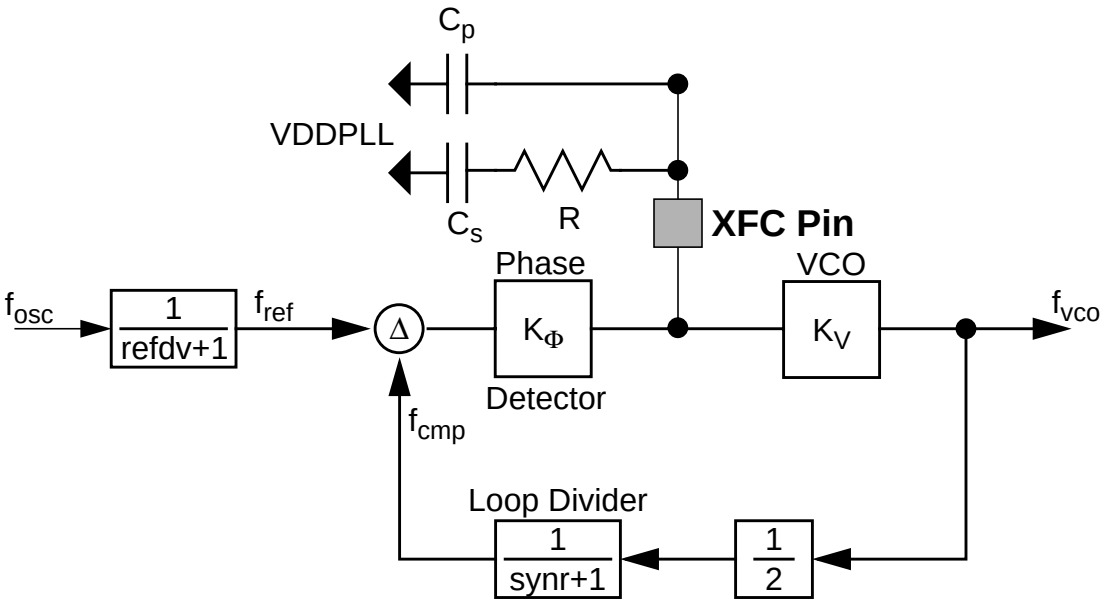


Figure A-3 Basic PLL functional diagram

The following procedure can be used to calculate the resistance and capacitance values using typical values for K_1, f_1 and i_{ch} from **Table A-16**.

The grey boxes show the calculation for f_{VCO} = 50MHz and f_{ref} = 1MHz. E.g., these frequencies are used for f_{OSC} = 4MHz and a 25MHz bus clock.

The VCO Gain at the desired VCO frequency is approximated by:

$$K_V = K_1 \cdot e^{\frac{(f_1 - f_{vco})}{K_1 \cdot 1V}} = -100 \cdot e^{\frac{(60-50)}{-100}} = -90.48\text{MHz/V}$$

The phase detector relationship is given by:

$$K_\Phi = -|i_{ch}| \cdot K_V \qquad = 316.7\text{Hz}/\Omega$$

i_{ch} is the current in tracking mode.

The loop bandwidth f_C should be chosen to fulfill the Gardner's stability criteria by at least a factor of 10, typical values are 50. $\zeta = 0.9$ ensures a good transient response.

$$f_C < \frac{2 \cdot \zeta \cdot f_{ref}}{\pi \cdot \left(\zeta + \sqrt{1 + \zeta^2}\right)} \cdot \frac{1}{10} \rightarrow f_C < \frac{f_{ref}}{4 \cdot 10}; (\zeta = 0.9) \qquad f_C < 25\text{kHz}$$

And finally the frequency relationship is defined as

$$n = \frac{f_{VCO}}{f_{ref}} = 2 \cdot (\text{synr} + 1) \qquad = 50$$

With the above values the resistance can be calculated. The example is shown for a loop bandwidth f_C=10kHz:

$$R = \frac{2 \cdot \pi \cdot n \cdot f_C}{K_\Phi} \qquad = 2*\pi*50*10\text{kHz}/(316.7\text{Hz}/\Omega)=9.9\text{k}\Omega=\sim 10\text{k}\Omega$$

The capacitance C_s can now be calculated as:

$$C_s = \frac{2 \cdot \zeta^2}{\pi \cdot f_C \cdot R} \approx \frac{0.516}{f_C \cdot R}; (\zeta = 0.9) \qquad = 5.19\text{nF} =\sim 4.7\text{nF}$$

The capacitance C_p should be chosen in the range of:

$$C_s/20 \le C_p \le C_s/10 \qquad C_p = 470\text{pF}$$

A.5.3.2 Jitter Information

The basic functionality of the PLL is shown in **Figure A-3**. With each transition of the clock f_{cmp}, the deviation from the reference clock f_{ref} is measured and input voltage to the VCO is adjusted accordingly.The adjustment is done continuously with no abrupt changes in the clock output frequency. Noise, voltage, temperature and other factors cause slight variations in the control loop resulting in a clock jitter. This jitter affects the real minimum and maximum clock periods as illustrated in **Figure A-4**.

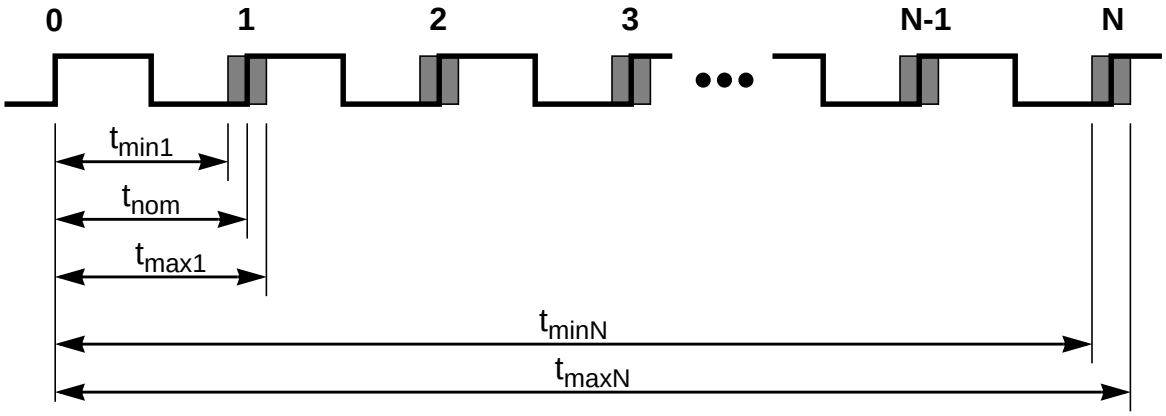


Figure A-4 Jitter Definitions

The relative deviation of t_{nom} is at its maximum for one clock period, and decreases towards zero for larger number of clock periods (N).

Defining the jitter as:

$$J(N) = \max\left(\left|1 - \frac{t_{max}(N)}{N \cdot t_{nom}}\right|, \left|1 - \frac{t_{min}(N)}{N \cdot t_{nom}}\right|\right)$$

For N < 100, the following equation is a good fit for the maximum jitter:

$$J(N) = \frac{j_1}{\sqrt{N}} + j_2$$

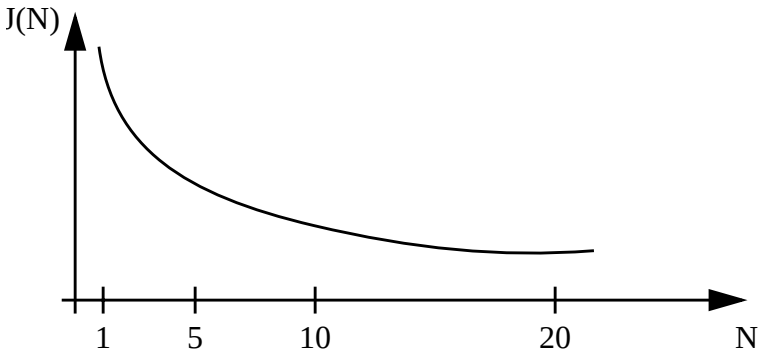


Figure A-5 Maximum bus clock jitter approximation

This is very important to notice with respect to timers, serial modules where a pre-scaler will eliminate the effect of the jitter to a large extent.

Table A-16 PLL Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	Self Clock Mode frequency	f_{SCM}	1	-	5.5	MHz
2	D	VCO locking range	f_{VCO}	8	-	50	MHz
3	D	Lock Detector transition from Acquisition to Tracking mode	$\|\Delta_{trk}\|$	3	-	4	%[1]
4	D	Lock Detection	$\|\Delta_{Lock}\|$	0	-	1.5	%(1)
5	D	Un-Lock Detection	$\|\Delta_{unl}\|$	0.5	-	2.5	%(1)
6	D	Lock Detector transition from Tracking to Acquisition mode	$\|\Delta_{unt}\|$	6	-	8	%(1)
7	C	PLLON Total Stabilization delay (Auto Mode) [2]	t_{stab}	-	0.5	-	ms
8	D	PLLON Acquisition mode stabilization delay (2)	t_{acq}	-	0.3	-	ms
9	D	PLLON Tracking mode stabilization delay (2)	t_{al}	-	0.2	-	ms
10	D	Fitting parameter VCO loop gain	K_1	-	-100	-	MHz/V
11	D	Fitting parameter VCO loop frequency	f_1	-	60	-	MHz
12	D	Charge pump current acquisition mode	$\| i_{ch} \|$	-	38.5	-	µA
13	D	Charge pump current tracking mode	$\| i_{ch} \|$	-	3.5	-	µA
14	C	Jitter fit parameter 1(2)	j_1	-	-	1.1	%
15	C	Jitter fit parameter 2(2)	j_2	-	-	0.13	%

NOTES:
1. % deviation from target frequency
2. f_{OSC} = 4MHz, f_{BUS} = 25MHz equivalent f_{VCO} = 50MHz: REFDV = #$03, SYNR = #$018, Cs = 4.7nF, Cp = 470pF, Rs = 10KΩ.

A.6 MSCAN

Table A-17 MSCAN Wake-up Pulse Characteristics

Conditions are shown in **Table A-4** unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	MSCAN Wake-up dominant pulse filtered	t_{WUP}	-	-	2	µs
2	P	MSCAN Wake-up dominant pulse pass	t_{WUP}	5	-	-	µs

A.7 SPI

This section provides electrical parametrics and ratings for the SPI.

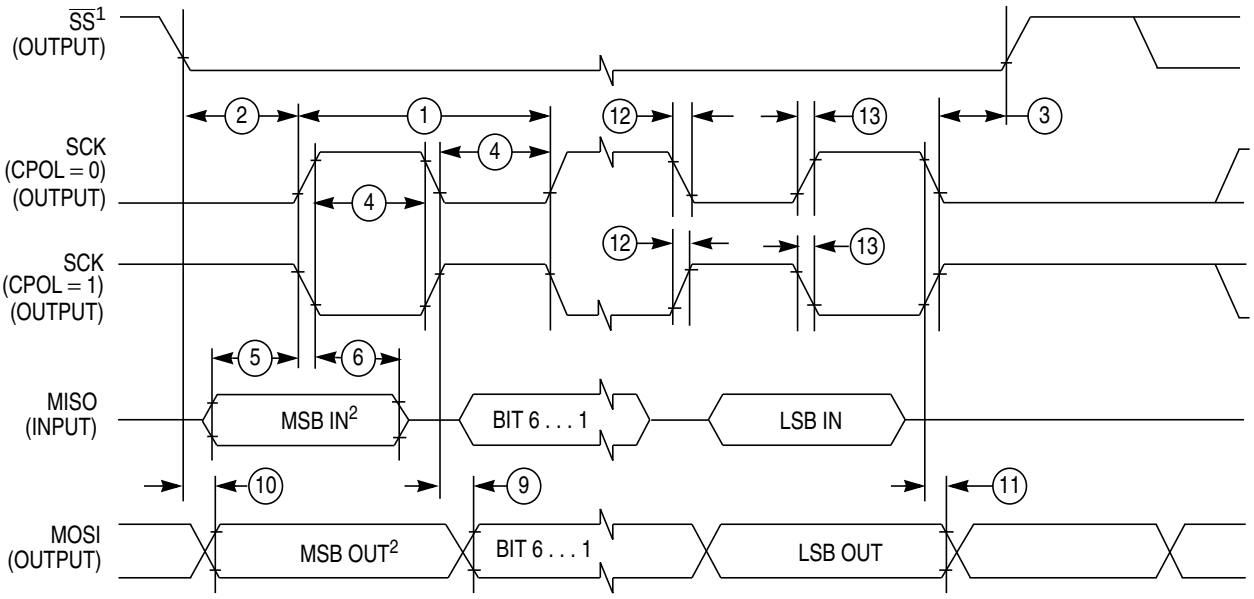
In **Table A-18** the measurement conditions are listed.

Table A-18 Measurement Conditions

Description	Value	Unit
Drive mode	full drive mode	—
Load capacitance C_{LOAD}, on all outputs	50	pF
Thresholds for delay measurement points	(20% / 80%) VDDX	V

A.7.1 Master Mode

In **Figure A-6** the timing diagram for master mode with transmission format CPHA=0 is depicted.

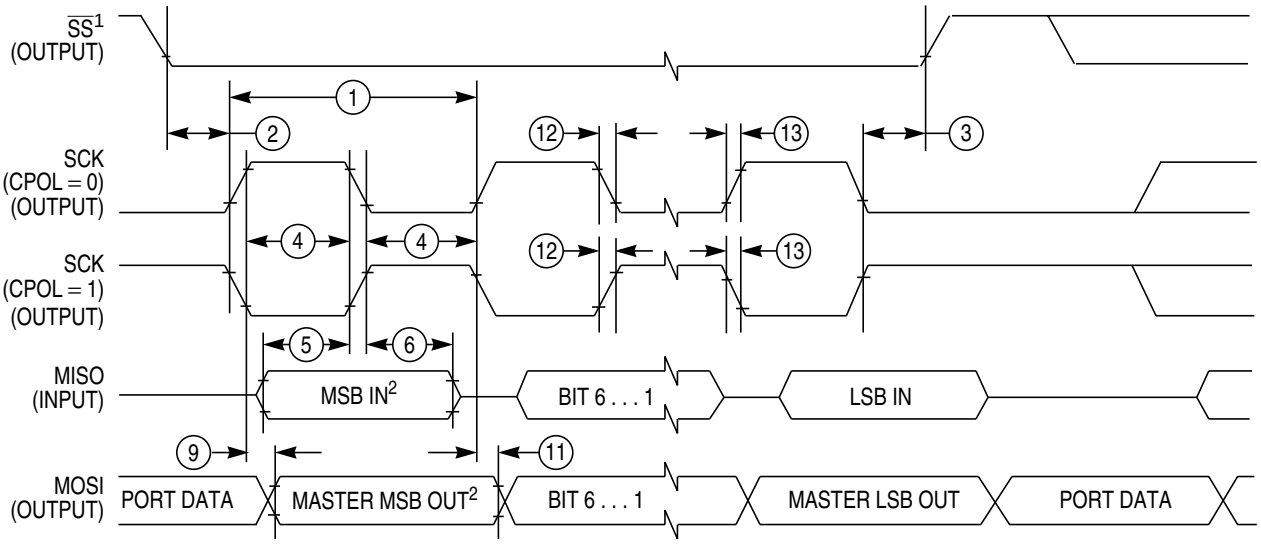


1.if configured as an output.

2. LSBF = 0. For LSBF = 1, bit order is LSB, bit 1, ..., bit 6, MSB.

Figure A-6 SPI Master Timing (CPHA=0)

In **Figure A-7** the timing diagram for master mode with transmission format CPHA=1 is depicted.



1.If configured as output

2. LSBF = 0. For LSBF = 1, bit order is LSB, bit 1, ..., bit 6, MSB.

Figure A-7 SPI Master Timing (CPHA=1)

In **Table A-19** the timing characteristics for master mode are listed.

Table A-19 SPI Master Mode Timing Characteristics

Num	Characteristic	Symbol	Min	Typ	Max	Unit
1	SCK Frequency	f_{sck}	1/2048	—	1/2	f_{bus}
1	SCK Period	t_{sck}	2	—	2048	t_{bus}
2	Enable Lead Time	t_{lead}	—	1/2	—	t_{sck}
3	Enable Lag Time	t_{lag}	—	1/2	—	t_{sck}
4	Clock (SCK) High or Low Time	t_{wsck}	—	1/2	—	t_{sck}
5	Data Setup Time (Inputs)	t_{su}	8	—	—	ns
6	Data Hold Time (Inputs)	t_{hi}	8	—	—	ns
9	Data Valid after SCK Edge	t_{vsck}	—	—	30	ns
10	Data Valid after $\overline{SS}$ fall (CPHA=0)	t_{vss}	—	—	15	ns
11	Data Hold Time (Outputs)	t_{ho}	20	—	—	ns
12	Rise and Fall Time Inputs	t_{rfi}	—	—	8	ns
13	Rise and Fall Time Outputs	t_{rfo}	—	—	8	ns

A.7.2 Slave Mode

In **Figure A-8** the timing diagram for slave mode with transmission format CPHA=0 is depicted.

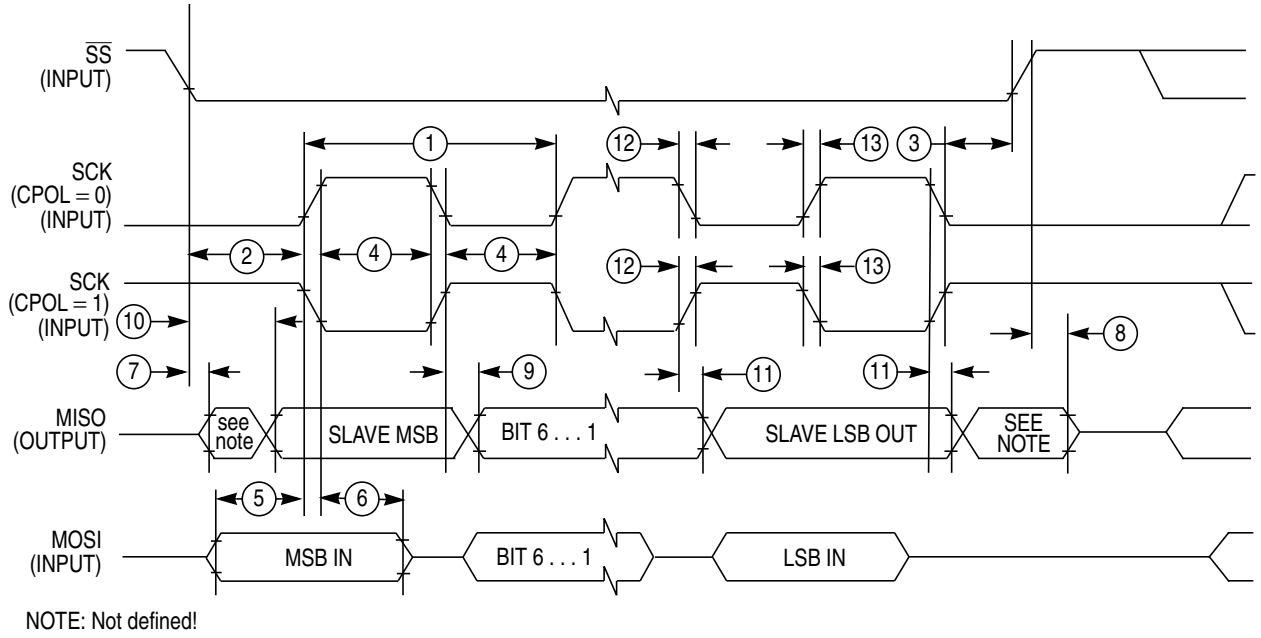


Figure A-8 SPI Slave Timing (CPHA=0)

In **Figure A-9** the timing diagram for slave mode with transmission format CPHA=1 is depicted.

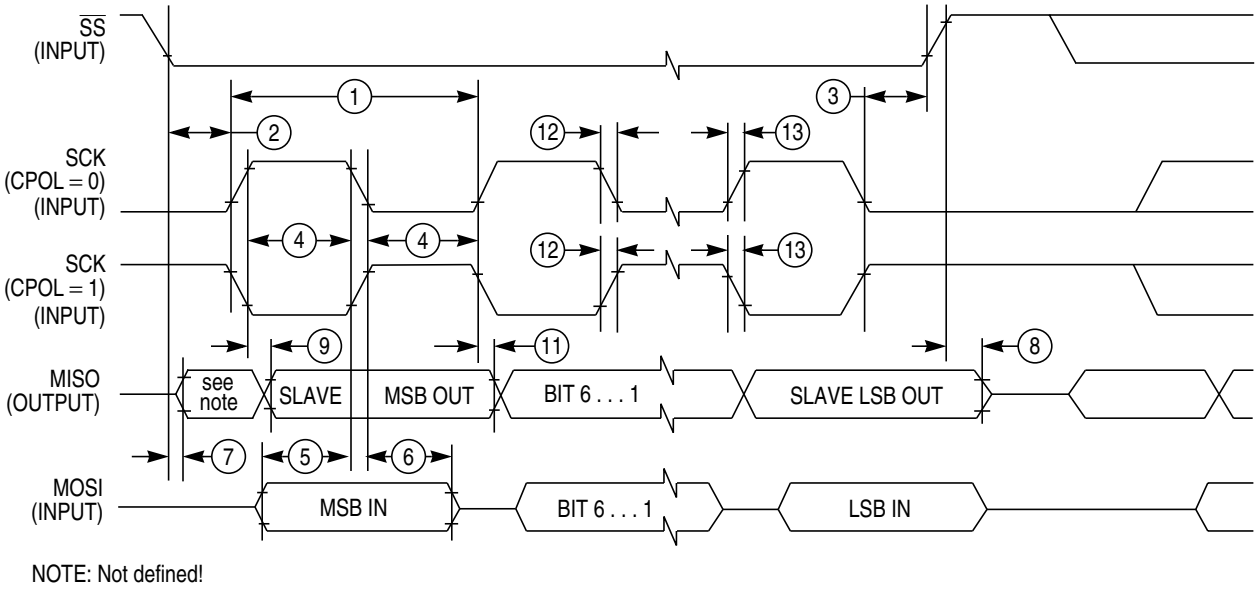


Figure A-9 SPI Slave Timing (CPHA=1)

In **Table A-20** the timing characteristics for slave mode are listed.

Table A-20 SPI Slave Mode Timing Characteristics

Num	Characteristic	Symbol	Min	Typ	Max	Unit
1	SCK Frequency	f_{sck}	DC	—	1/4	f_{bus}
1	SCK Period	t_{sck}	4	—	∞	t_{bus}
2	Enable Lead Time	t_{lead}	4	—	—	t_{bus}
3	Enable Lag Time	t_{lag}	4	—	—	t_{bus}
4	Clock (SCK) High or Low Time	t_{wsck}	4	—	—	t_{bus}
5	Data Setup Time (Inputs)	t_{su}	8	—	—	ns
6	Data Hold Time (Inputs)	t_{hi}	8	—	—	ns
7	Slave Access Time (time to data active)	t_a	—	—	20	ns
8	Slave MISO Disable Time	t_{dis}	—	—	22	ns
9	Data Valid after SCK Edge	t_{vsck}	—	—	30 + t_{bus} [1]	ns
10	Data Valid after $\overline{SS}$ fall	t_{vss}	—	—	30 + t_{bus} [1]	ns
11	Data Hold Time (Outputs)	t_{ho}	20	—	—	ns
12	Rise and Fall Time Inputs	t_{rfi}	—	—	8	ns
13	Rise and Fall Time Outputs	t_{rfo}	—	—	8	ns

NOTES:
1. t_{bus} added due to internal synchronization delay

A.8 External Bus Timing

A timing diagram of the external multiplexed-bus is illustrated in **Figure A-10** with the actual timing values shown on table **Table A-21**. All major bus signals are included in the diagram. While both a data write and data read cycle are shown, only one or the other would occur on a particular bus cycle.

A.8.1 General Muxed Bus Timing

The expanded bus timings are highly dependent on the load conditions. The timing parameters shown assume a balanced load across all outputs.

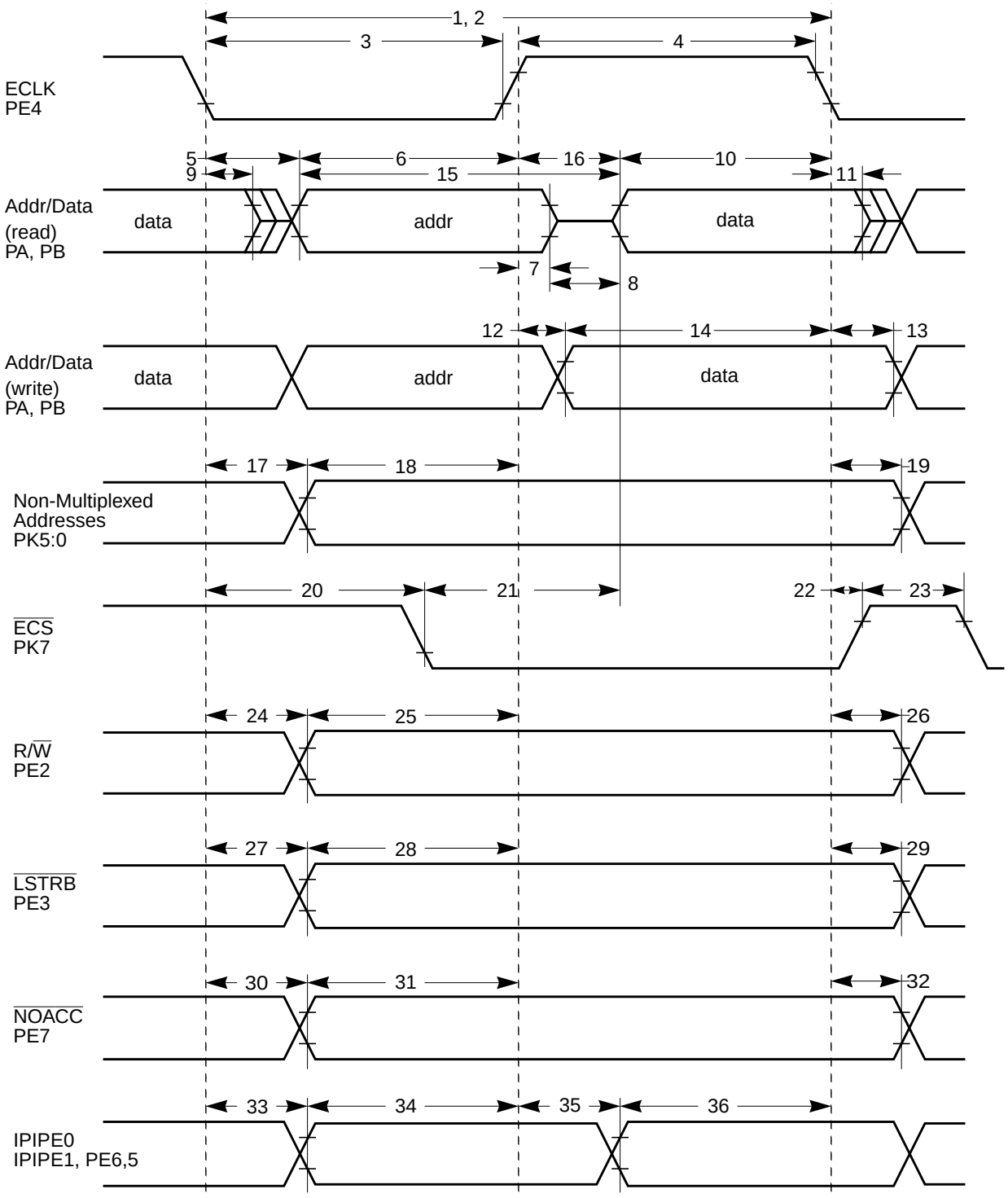


Figure A-10 General External Bus Timing

Table A-21 Expanded Bus Timing Characteristics

Conditions are shown in **Table A-4** unless otherwise noted, C_{LOAD} = 50pF							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
1	P	Frequency of operation (E-clock)	f_o	0	-	25.0	MHz
2	P	Cycle time	t_{cyc}	40	-	-	ns
3	D	Pulse width, E low	PW_{EL}	19	-	-	ns
4	D	Pulse width, E high[1]	PW_{EH}	19	-	-	ns
5	D	Address delay time	t_{AD}	-	-	8	ns
6	D	Address valid time to E rise (PW_{EL}–t_{AD})	t_{AV}	11	-	-	ns
7	D	Muxed address hold time	t_{MAH}	2	-	-	ns
8	D	Address hold to data valid	t_{AHDS}	7	-	-	ns
9	D	Data hold to address	t_{DHA}	2	-	-	ns
10	D	Read data setup time	t_{DSR}	13	-	-	ns
11	D	Read data hold time	t_{DHR}	0	-	-	ns
12	D	Write data delay time	t_{DDW}	-	-	7	ns
13	D	Write data hold time	t_{DHW}	2	-	-	ns
14	D	Write data setup time(1) (PW_{EH}–t_{DDW})	t_{DSW}	12	-	-	ns
15	D	Address access time(1) (t_{cyc}–t_{AD}–t_{DSR})	t_{ACCA}	19	-	-	ns
16	D	E high access time(1) (PW_{EH}–t_{DSR})	t_{ACCE}	6	-	-	ns
17	D	Non-multiplexed address delay time	t_{NAD}	-	-	6	ns
18	D	Non-muxed address valid to E rise (PW_{EL}–t_{NAD})	t_{NAV}	13	-	-	ns
19	D	Non-multiplexed address hold time	t_{NAH}	2	-	-	ns
20	D	Chip select delay time	t_{CSD}	-	-	16	ns
21	D	Chip select access time(1) (t_{cyc}–t_{CSD}–t_{DSR})	t_{ACCS}	11	-	-	ns
22	D	Chip select hold time	t_{CSH}	2	-	-	ns
23	D	Chip select negated time	t_{CSN}	8	-	-	ns
24	D	Read/write delay time	t_{RWD}	-	-	7	ns
25	D	Read/write valid time to E rise (PW_{EL}–t_{RWD})	t_{RWV}	14	-	-	ns
26	D	Read/write hold time	t_{RWH}	2	-	-	ns
27	D	Low strobe delay time	t_{LSD}	-	-	7	ns
28	D	Low strobe valid time to E rise (PW_{EL}–t_{LSD})	t_{LSV}	14	-	-	ns
29	D	Low strobe hold time	t_{LSH}	2	-	-	ns
30	D	NOACC strobe delay time	t_{NOD}	-	-	7	ns
31	D	NOACC valid time to E rise (PW_{EL}–t_{NOD})	t_{NOV}	14	-	-	ns

Table A-21 Expanded Bus Timing Characteristics

Conditions are shown in **Table A-4** unless otherwise noted, C_{LOAD} = 50pF							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
32	D	NOACC hold time	t_{NOH}	2	-	-	ns
33	D	IPIPE[1:0] delay time	t_{P0D}	2	-	7	ns
34	D	IPIPE[1:0] valid time to E rise (PW_{EL}–t_{P0D})	t_{P0V}	11	-	-	ns
35	D	IPIPE[1:0] delay time(1) (PW_{EH}-t_{P1V})	t_{P1D}	2	-	7	ns
36	D	IPIPE[1:0] valid time to E fall	t_{P1V}	11	-	-	ns

NOTES:

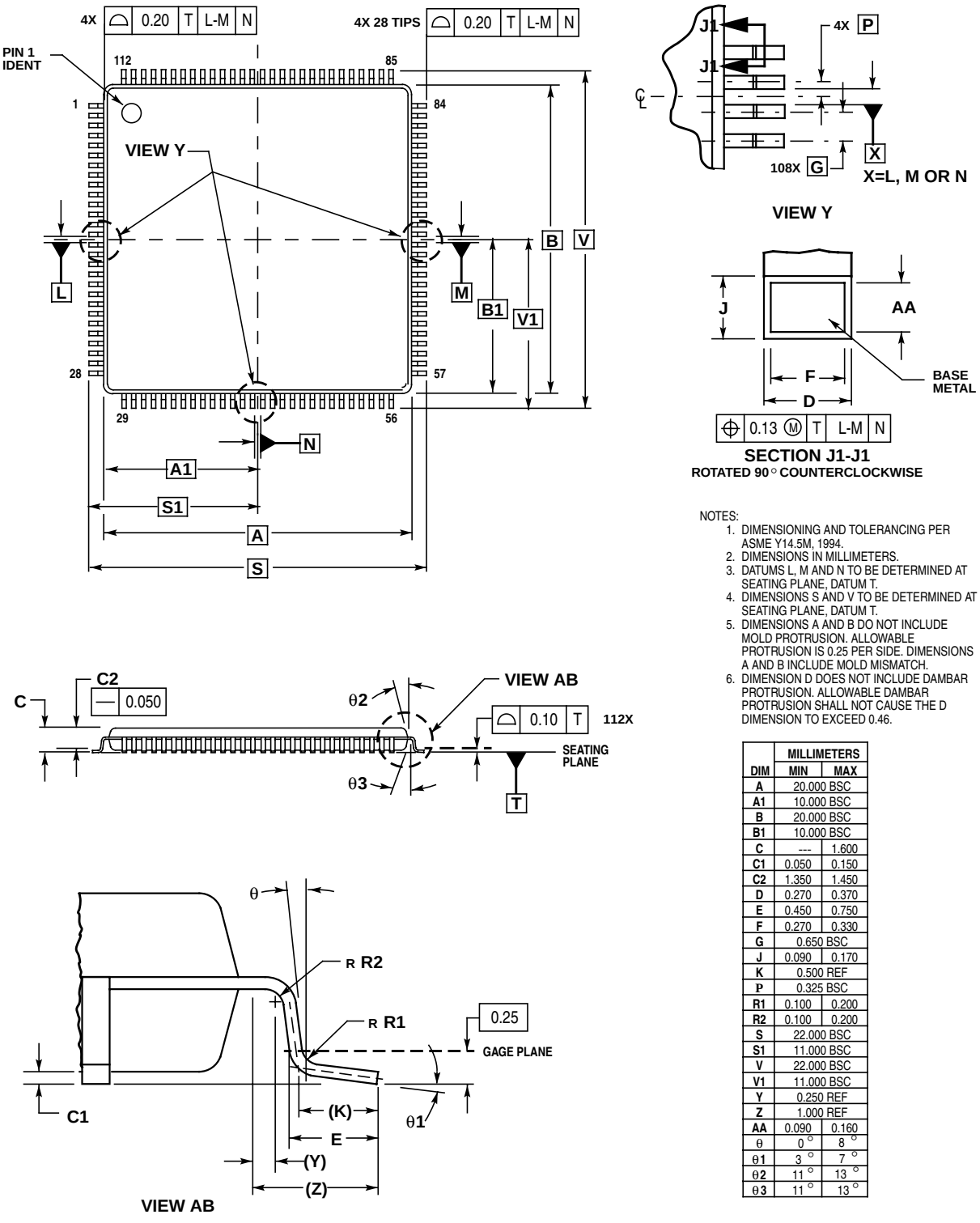
1. Affected by clock stretch: add N x t_{cyc} where N=0,1,2 or 3, depending on the number of clock stretches.

Appendix B Package Information

B.1 General

This section provides the physical dimensions of the MC9S12DP512 packages.

B.2 112-pin LQFP package



NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.
2. DIMENSIONS IN MILLIMETERS.
3. DATUMS L, M AND N TO BE DETERMINED AT SEATING PLANE, DATUM T.
4. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE, DATUM T.
5. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 PER SIDE. DIMENSIONS A AND B INCLUDE MOLD MISMATCH.
6. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL NOT CAUSE THE D DIMENSION TO EXCEED 0.46.

	MILLIMETERS	
DIM	MIN	MAX
A	20.000 BSC	
A1	10.000 BSC	
B	20.000 BSC	
B1	10.000 BSC	
C	---	1.600
C1	0.050	0.150
C2	1.350	1.450
D	0.270	0.370
E	0.450	0.750
F	0.270	0.330
G	0.650 BSC	
J	0.090	0.170
K	0.500 REF	
P	0.325 BSC	
R1	0.100	0.200
R2	0.100	0.200
S	22.000 BSC	
S1	11.000 BSC	
V	22.000 BSC	
V1	11.000 BSC	
Y	0.250 REF	
Z	1.000 REF	
AA	0.090	0.160
θ	0°	8°
θ1	3°	7°
θ2	11°	13°
θ3	11°	13°

Figure B-1 112-pin LQFP mechanical dimensions (case no. 987)

User Guide End Sheet

FINAL PAGE OF
124
PAGES

ATD_10B8C
Block User Guide
V02.12

Original Release Date: 27 OCT 2000
Revised: 28 June 2005

Motorola Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
00.00	27-10-2000	-		Initial SRS2 release.
01.00	06-06-2001	-		Updated the description of ATDDIEN and PORTAD1 register.
01.10	16-06-2001	-		Made SRS2 Compliant
V02.00	20 June 2001	20 June 2001		Reworked whole document to make it more user friendly
V02.01	26 July 2001	-		Added document names Variable definitions and names have been hidden
V02.02	5 Sept 2001	5 Sept 2001		Corrected sampling phase description, other minor corrections
V02.03	8 Nov 2001	8 Nov 2001		Corrected AWAI bit description
V02.04	16 Jan 2002	16 Jan 2002		Syntax corrections
V02.05	8 Mar 2002	8 Mar 2002		Removed document number from all pages except cover sheet
V02.06	11 Apr 2002	11 Apr 2002		Documented special channel conversion in ATDTEST1 register
V02.07	22 Apr 2002	22 Apr 2002		Corrected Table "Available Result Data Formats"
V02.08	16 Aug 2002	16 Aug 2002		FIFOR flag: corrected clearing mechanism B)
V02.09	23 Aug 2002	23 Aug 2002		Detailed AWAI Bit description. Functional Description: Detailed and corrected Low power modes Table "Available Result Data Formats": Re-corrected
V02.10	21 Feb 2003	21 Feb 2003		Formal corrections on ATDTEST0/1 and ATDDRHx/ATDDRLx register descriptions
V02.11	24 Mar 2005	24 Mar 2005		Corrected PAD7-0 port description
V02.12	28 June 2005	28 June 2005		Enhanced FIFO bit description

Table 0-1 Revision History

Table of Contents

Section 1 Introduction

Section 2 Signal Description

Section 3 Memory Map and Register Definition

Section 4 Functional Description

Section 5 Resets

Section 6 Interrupts

List of Figures

.

List of Tables

.

Section 1 Introduction

1.1 Overview

The ATD_10B8C is an 8-channel, 10-bit, multiplexed input successive approximation analog-to-digital converter. Refer to device electrical specifications for ATD accuracy.

The block is designed to be upwards compatible with the 68HC11 standard 8-bit A/D converter. In addition, there are new operating modes that are unique to the HC12 design.

1.2 Features

- 8/10 Bit Resolution.
- 7 μsec, 10-Bit Single Conversion Time.
- Sample Buffer Amplifier.
- Programmable Sample Time.
- Left/Right Justified, Signed/Unsigned Result Data.
- External Trigger Control.
- Conversion Completion Interrupt Generation.
- Analog Input Multiplexer for 8 Analog Input Channels.
- Analog/Digital Input Pin Multiplexing.
- 1 to 8 Conversion Sequence Lengths.
- Continuous Conversion Mode.
- Multiple Channel Scans.

1.3 Modes of Operation

1.3.1 Conversion modes

There is software programmable selection between performing **single** or **continuous conversion** on a **single channel** or **multiple channels**.

1.3.2 MCU Operating Modes

- **Stop Mode**
  Entering Stop Mode causes all clocks to halt and thus the system is placed in a minimum power standby mode. This aborts any conversion sequence in progress. During recovery from Stop Mode, there must be a minimum delay for the Stop Recovery Time t_{SR} before initiating a new ATD conversion sequence.

- **Wait Mode**
  Entering Wait Mode the ATD conversion either continues or aborts for low power depending on the logical value of the AWAIT bit.
- **Freeze Mode**
  In Freeze Mode the ATD_10B8C will behave according to the logical values of the FRZ1 and FRZ0 bits. This is useful for debugging and emulation.

1.4 Block Diagram

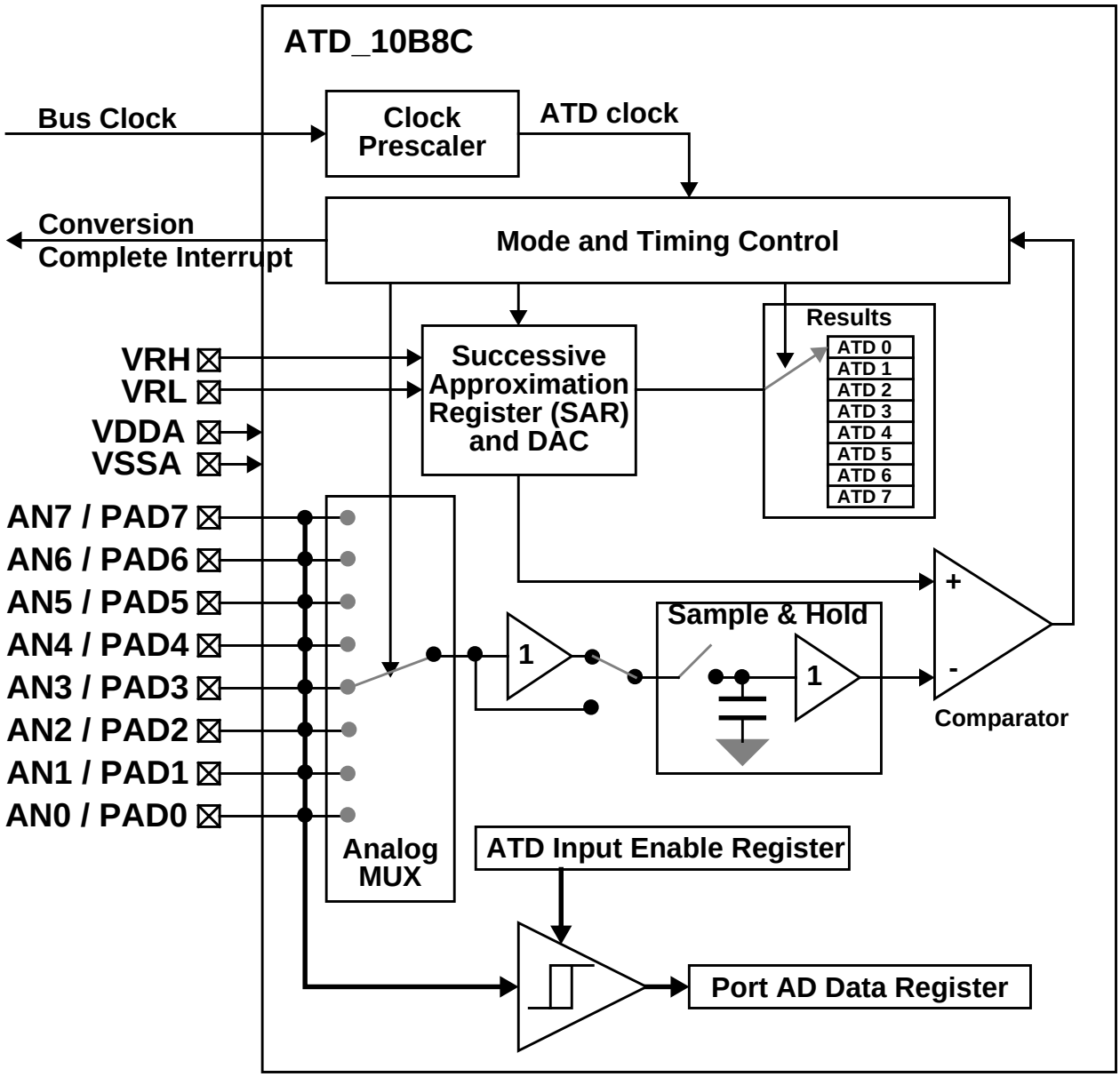


Figure 1-1 ATD_10B8C Block Diagram

Section 2 Signal Description

2.1 Overview

The ATD_10B8C has a total of 12 external pins.

2.2 Detailed Signal Descriptions

2.2.1 AN7 / ETRIG / PAD7

This pin serves as the analog input Channel 7. It can be configured to provide an external trigger for the ATD conversion. It can be configured as digital port pin.

2.2.2 AN6 / PAD6

This pin serves as the analog input Channel 6. It can be configured as digital port pin.

2.2.3 AN5 / PAD5

This pin serves as the analog input Channel 5. It can be configured as digital port pin.

2.2.4 AN4 / PAD4

This pin serves as the analog input Channel 4. It can be configured as digital port pin.

2.2.5 AN3 / PAD3

This pin serves as the analog input Channel 3. It can be configured as digital port pin.

2.2.6 AN2 / PAD2

This pin serves as the analog input Channel 2. It can be configured as digital port pin.

2.2.7 AN1 / PAD1

This pin serves as the analog input Channel 1. It can be configured as digital port pin.

2.2.8 AN0 / PAD0

This pin serves as the analog input Channel 0. It can be configured as digital port pin.

2.2.9 VRH, VRL

VRH is the high reference voltage and VRL is the low reference voltage for ATD conversion.

2.2.10 VDDA, VSSA

These pins are the power supplies for the analog circuitry of the ATD_10B8C block.

Section 3 Memory Map and Register Definition

3.1 Overview

This section provides a detailed description of all registers accessible in the ATD_10B8C.

3.2 Module Memory Map

Table 3-1 gives an overview on all ATD_10B8C registers.

Table 3-1 Module Memory Map

Address Offset	Use	Access
$_00	ATD Control Register 0 (ATDCTL0)[1]	R
$_01	ATD Control Register 1 (ATDCTL1)[2]	R
$_02	ATD Control Register 2 (ATDCTL2)	R/W
$_03	ATD Control Register 3 (ATDCTL3)	R/W
$_04	ATD Control Register 4 (ATDCTL4)	R/W
$_05	ATD Control Register 5 (ATDCTL5)	R/W
$_06	ATD Status Register 0 (ATDSTAT0)	R/W
$_07	Unimplemented	
$_08	ATD Test Register 0 (ATDTEST0)[3]	R
$_09	ATD Test Register 1 (ATDTEST1)	R/W
$_0A	Unimplemented	
$_0B	ATD Status Register 1 (ATDSTAT1)	R
$_0C	Unimplemented	
$_0D	ATD Input Enable Register (ATDDIEN)	R/W
$_0E	Unimplemented	
$_0F	Port Data Register (PORTAD)	R
$_10, $_11	ATD Result Register 0 (ATDDR0H, ATDDR0L)	R/W
$_12, $_13	ATD Result Register 1 (ATDDR1H, ATDDR1L)	R/W
$_14, $_15	ATD Result Register 2 (ATDDR2H, ATDDR2L)	R/W
$_16, $_17	ATD Result Register 3 (ATDDR3H, ATDDR3L)	R/W
$_18, $_19	ATD Result Register 4 (ATDDR4H, ATDDR4L)	R/W
$_1A, $_1B	ATD Result Register 5 (ATDDR5H, ATDDR5L)	R/W
$_1C, $_1D	ATD Result Register 6 (ATDDR6H, ATDDR6L)	R/W
$_1E, $_1F	ATD Result Register 7 (ATDDR7H, ATDDR7L)	R/W

NOTES:
1. ATDCTL0 is intended for factory test purposes only.
2. ATDCTL1 is intended for factory test purposes only.
3. ATDTEST0 is intended for factory test purposes only.

NOTE: *Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.*

3.3 Register Descriptions

This section describes in address order all the ATD_10B8C registers and their individual bits.

3.3.1 Reserved Register (ATDCTL0)

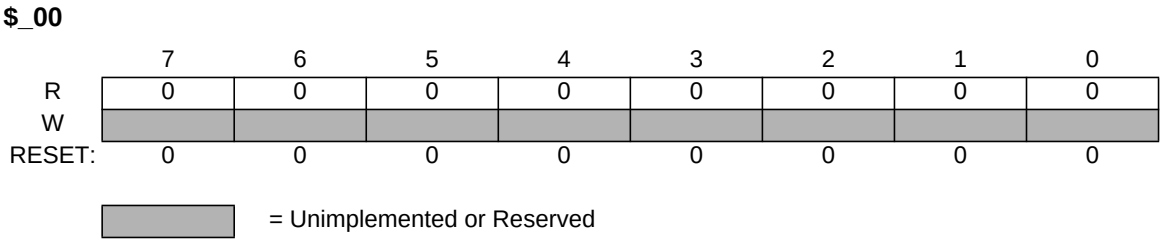


Figure 3-1 Reserved Register (ATDCTL0)

Read: always read $00 in normal modes

Write: unimplemented in normal modes

3.3.2 Reserved Register (ATDCTL1)

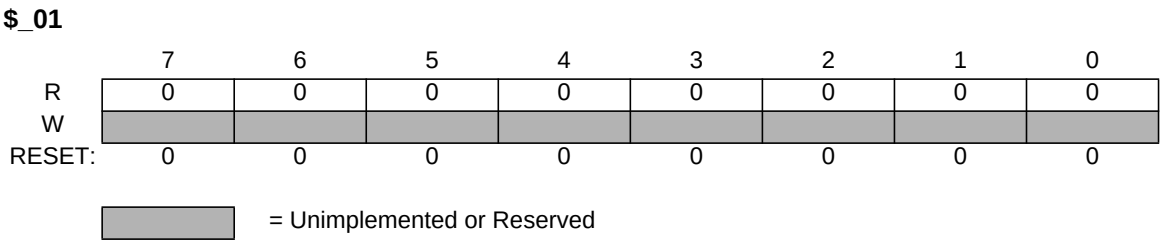


Figure 3-2 Reserved Register (ATDCTL1)

Read: always read $00 in normal modes

Write: unimplemented in normal modes

NOTE: *Writing to this registers when in special modes can alter functionality.*

3.3.3 ATD Control Register 2 (ATDCTL2)

This register controls power down, interrupt and external trigger. Writes to this register will abort current conversion sequence but will not start a new sequence.

$_02

	7	6	5	4	3	2	1	0
R	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 ATD Control Register 2 (ATDCTL2)

Read: anytime

Write: anytime

ADPU — ATD Power Up

This bit provides on/off control over the ATD_10B8C block allowing reduced MCU power consumption. Because analog electronic is turned off when powered down, the ATD requires a recovery time period after ADPU bit is enabled.

1 = Normal ATD functionality
0 = Power down ATD

AFFC — ATD Fast Flag Clear All

1 = Changes all ATD conversion complete flags to a fast clear sequence. Any access to a result register will cause the associate CCF flag to clear automatically.
0 = ATD flag clearing operates normally (read the status register ATDSTAT1 before reading the result register to clear the associate CCF flag).

AWAI — ATD Power Down in Wait Mode

When entering Wait Mode this bit provides on/off control over the ATD_10B8C block allowing reduced MCU power. Because analog electronic is turned off when powered down, the ATD requires a recovery time period after exit from Wait mode.

1 = Halt conversion and power down ATD during Wait mode
After exiting Wait mode with an interrupt conversion will resume. But due to the recovery time the result of this conversion should be ignored.
0 = ATD continues to run in Wait mode

ETRIGLE — External Trigger Level/Edge Control

This bit controls the sensitivity of the external trigger signal. See **Table 3-2** for details.

ETRIGP — External Trigger Polarity

This bit controls the polarity of the external trigger signal. See **Table 3-2** for details.

Table 3-2 External Trigger Configurations

ETRIGLE	ETRIGP	External Trigger Sensitivity
0	0	falling edge
0	1	rising edge
1	0	low level
1	1	high level

ETRIGE — External Trigger Mode Enable

This bit enables the external trigger on ATD channel 7. The external trigger allows to synchronize sample and ATD conversions processes with external events.

1 = Enable external trigger
0 = Disable external trigger

NOTE: *The conversion results for the external trigger ATD channel 7 have no meaning while external trigger mode is enabled.*

ASCIE — ATD Sequence Complete Interrupt Enable

1 = ATD Interrupt will be requested whenever ASCIF=1 is set.
0 = ATD Sequence Complete interrupt requests are disabled.

ASCIF — ATD Sequence Complete Interrupt Flag

If ASCIE=1 the ASCIF flag equals the SCF flag (see **3.3.7**), else ASCIF reads zero. Writes have no effect.

1 = ATD sequence complete interrupt pending
0 = No ATD interrupt occurred

3.3.4 ATD Control Register 3 (ATDCTL3)

This register controls the conversion sequence length, FIFO for results registers and behavior in Freeze Mode. Writes to this register will abort current conversion sequence but will not start a new sequence.

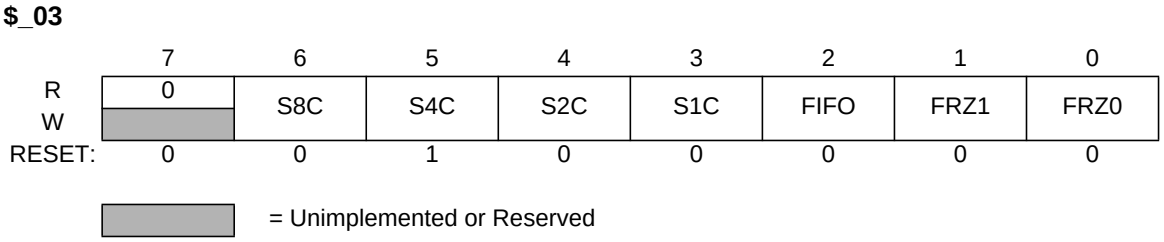


Figure 3-4 ATD Control Register 3 (ATDCTL3)

Read: anytime

Write: anytime

S8C, S4C, S2C, S1C — Conversion Sequence Length

These bits control the number of conversions per sequence. **Table 3-3** shows all combinations. At reset, S4C is set to 1 (sequence length is 4). This is to maintain software continuity to HC12 family.

Table 3-3 Conversion Sequence Length Coding

S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

FIFO — Result Register FIFO Mode

If this bit is zero (non-FIFO mode), the A/D conversion results map into the result registers based on the conversion sequence; the result of the first conversion appears in the first result register, the second result in the second result register, and so on.

If this bit is one (FIFO mode) the conversion counter is not reset at the beginning or ending of a conversion sequence; sequential conversion results are placed in consecutive result registers. In a continuously scanning conversion sequence, the result register counter will wrap around when it reaches the end of the result register file. The conversion counter value (CC2-0 in ATDSTAT0) can be used to determine where in the result register file, the current conversion result will be placed. Aborting a conversion or starting a new conversion by write to an ATDCTL register (ATDCTL5-0) clears the conversion counter even if FIFO=1. So the first result of a new conversion sequence, started by writing to ATDCTL5, will always be place in the first result register (ATDDDR0). Intended usage of FIFO mode is continuos conversion (SCAN=1) or triggered conversion (ETRIG=1).

Which result registers hold valid data can be tracked using the conversion complete flags. Fast flag clear mode may or may not be useful in a particular application to track valid data.

1 = Conversion results are placed in consecutive result registers (wrap around at end).

0 = Conversion results are placed in the corresponding result register up to the selected sequence length.

FRZ1, FRZ0 — Background Debug Freeze Enable

When debugging an application, it is useful in many cases to have the ATD pause when a breakpoint (Freeze Mode) is encountered. These 2 bits determine how the ATD will respond to a breakpoint as shown in **Table 3-4**. Leakage onto the storage node and comparator reference capacitors may compromise the accuracy of an immediately frozen conversion depending on the length of the freeze period.

Table 3-4 ATD Behavior in Freeze Mode (breakpoint)

FRZ1	FRZ0	Behavior in Freeze mode
0	0	Continue conversion
0	1	Reserved
1	0	Finish current conversion, then freeze
1	1	Freeze Immediately

3.3.5 ATD Control Register 4 (ATDCTL4)

This register selects the conversion clock frequency, the length of the second phase of the sample time and the resolution of the A/D conversion (i.e.: 8-bits or 10-bits). Writes to this register will abort current conversion sequence but will not start a new sequence.

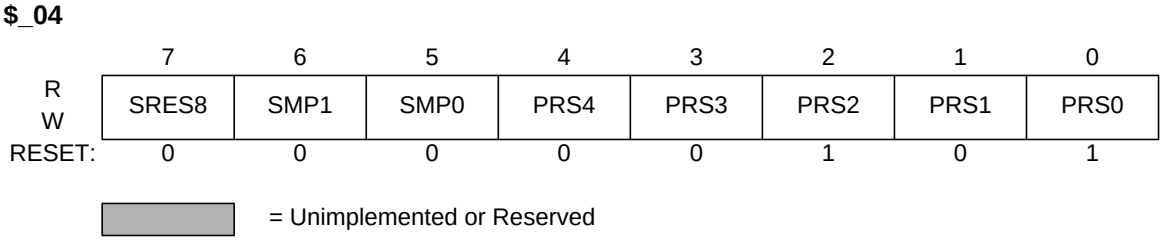


Figure 3-5 ATD Control Register 4 (ATDCTL4)

Read: anytime

Write: anytime

SRES8 — A/D Resolution Select

This bit selects the resolution of A/D conversion results as either 8 or 10 bits. The A/D converter has an accuracy of 10 bits; however, if low resolution is required, the conversion can be speeded up by selecting 8-bit resolution.

1 = 8 bit resolution
0 = 10 bit resolution

SMP1, SMP0 — Sample Time Select

These two bits select the length of the second phase of the sample time in units of ATD conversion clock cycles. Note that the ATD conversion clock period is itself a function of the prescaler value (bits PRS4-0). The sample time consists of two phases. The first phase is two ATD conversion clock cycles long and transfers the sample quickly (via the buffer amplifier) onto the A/D machine's storage node. The second phase attaches the external analog signal directly to the storage node for final charging and high accuracy. **Table 3-5** lists the lengths available for the second sample phase.

Table 3-5 Sample Time Select

SMP1	SMP0	Length of 2nd phase of sample time
0	0	2 A/D conversion clock periods
0	1	4 A/D conversion clock periods
1	0	8 A/D conversion clock periods
1	1	16 A/D conversion clock periods

PRS4, PRS3, PRS2, PRS1, PRS0 — ATD Clock Prescaler

These 5 bits are the binary value prescaler value PRS. The ATD conversion clock frequency is calculated as follows:

$$ATDclock = \frac{[BusClock]}{[PRS + 1]} \times 0.5$$

Note that the maximum ATD conversion clock frequency is half the Bus Clock. The default (after reset) prescaler value is 5 which results in a default ATD conversion clock frequency that is Bus Clock divided by 12. **Table 3-6** illustrates the divide-by operation and the appropriate range of the Bus Clock.

Table 3-6 Clock Prescaler Values

Prescale Value	Total Divisor Value	Max. Bus Clock[1]	Min. Bus Clock[2]
00000	divide by 2	4 MHz	1 MHz
00001	divide by 4	8 MHz	2 MHz
00010	divide by 6	12 MHz	3 MHz
00011	divide by 8	16 MHz	4 MHz
00100	divide by 10	20 MHz	5 MHz
00101	divide by 12	24 MHz	6 MHz
00110	divide by 14	28 MHz	7 MHz
00111	divide by 16	32 MHz	8 MHz
01000	divide by 18	36 MHz	9 MHz
01001	divide by 20	40 MHz	10 MHz
01010	divide by 22	44 MHz	11 MHz
01011	divide by 24	48 MHz	12 MHz
01100	divide by 26	52 MHz	13 MHz
01101	divide by 28	56 MHz	14 MHz
01110	divide by 30	60 MHz	15 MHz
01111	divide by 32	64 MHz	16 MHz
10000	divide by 34	68 MHz	17 MHz
10001	divide by 36	72 MHz	18 MHz
10010	divide by 38	76 MHz	19 MHz
10011	divide by 40	80 MHz	20 MHz
10100	divide by 42	84 MHz	21 MHz
10101	divide by 44	88 MHz	22 MHz
10110	divide by 46	92 MHz	23 MHz
10111	divide by 48	96 MHz	24 MHz
11000	divide by 50	100 MHz	25 MHz
11001	divide by 52	104 MHz	26 MHz
11010	divide by 54	108 MHz	27 MHz
11011	divide by 56	112 MHz	28 MHz
11100	divide by 58	116 MHz	29 MHz
11101	divide by 60	120 MHz	30 MHz
11110	divide by 62	124 MHz	31 MHz
11111	divide by 64	128 MHz	32 MHz

NOTE:

1. Maximum ATD conversion clock frequency is 2MHz. The maximum allowed Bus Clock frequency is shown in this column.
2. Minimum ATD conversion clock frequency is 500KHz. The minimum allowed Bus Clock frequency is shown in this column.

3.3.6 ATD Control Register 5 (ATDCTL5)

This register selects the type of conversion sequence and the analog input channels sampled. Writes to this register will abort current conversion sequence and start a new conversion sequence.

$_05

	7	6	5	4	3	2	1	0
R	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 ATD Control Register 5 (ATDCTL5)

Read: anytime

Write: anytime

DJM — Result Register Data Justification

This bit controls justification of conversion data in the result registers. See **3.3.13 ATD Conversion Result Registers (ATDDRHx/ATDDRLx)** for details.

1 = Right justified data in the result registers
0 = Left justified data in the result registers

DSGN — Result Register Data Signed or Unsigned Representation

This bit selects between signed and unsigned conversion data representation in the result registers. Signed data is represented as 2’s complement. Signed data is not available in right justification. See **3.3.13 ATD Conversion Result Registers (ATDDRHx/ATDDRLx)** for details.

1 = Signed data representation in the result registers
0 = Unsigned data representation in the result registers

Table 3-7 summarizes the result data formats available and how they are set up using the control bits.

Table 3-8 illustrates the difference between the signed and unsigned, left justified output codes for an input signal range between 0 and 5.12 Volts.

Table 3-7 Available Result Data Formats

SRES8	DJM	DSGN	Result Data Formats Description and Bus Bit Mapping
1	0	0	8-bit / left justified / unsigned - bits 8-15
1	0	1	8-bit / left justified / signed - bits 8-15
1	1	X	8-bit / right justified / unsigned - bits 0-7
0	0	0	10-bit / left justified / unsigned - bits 6-15
0	0	1	10-bit / left justified / signed - bits 6-15
0	1	X	10-bit / right justified / unsigned - bits 0-9

Table 3-8 Left Justified, Signed and Unsigned ATD Output Codes.

Input Signal Vrl = 0 Volts Vrh = 5.12 Volts	Signed 8-Bit Codes	Unsigned 8-Bit Codes	Signed 10-Bit Codes	Unsigned 10-Bit Codes
5.120 Volts	7F	FF	7FC0	FFC0
5.100	7F	FF	7F00	FF00
5.080	7E	FE	7E00	FE00
2.580	01	81	0100	8100
2.560	00	80	0000	8000
2.540	FF	7F	FF00	7F00
0.020	81	01	8100	0100
0.000	80	00	8000	0000

SCAN — Continuous Conversion Sequence Mode

This bit selects whether conversion sequences are performed continuously or only once.

1 = Continuous conversion sequences (scan mode)

0 = Single conversion sequence

MULT — Multi-Channel Sample Mode

When MULT is 0, the ATD sequence controller samples only from the specified analog input channel for an entire conversion sequence. The analog channel is selected by channel selection code (control bits CC/CB/CA located in ATDCTL5). When MULT is 1, the ATD sequence controller samples across channels. The number of channels sampled is determined by the sequence length value (S8C, S4C, S2C, S1C). The first analog channel examined is determined by channel selection code (CC, CB, CA control bits); subsequent channels sampled in the sequence are determined by incrementing the channel selection code.

1 = Sample across several channels

0 = Sample only one channel

CC, CB, CA — Analog Input Channel Select Code

These bits select the analog input channel(s) whose signals are sampled and converted to digital codes. **Table 3-9** lists the coding used to select the various analog input channels. In the case of single channel scans (MULT=0), this selection code specified the channel examined. In the case of multi-channel scans (MULT=1), this selection code represents the first channel to be examined in the conversion sequence. Subsequent channels are determined by incrementing channel selection code; selection codes that reach the maximum value wrap around to the minimum value.

Table 3-9 Analog Input Channel Select Coding

CC	CB	CA	Analog Input Channel
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3

Table 3-9 Analog Input Channel Select Coding

CC	CB	CA	Analog Input Channel
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

3.3.7 ATD Status Register 0 (ATDSTAT0)

This read-only register contains the Sequence Complete Flag, overrun flags for external trigger and FIFO mode, and the conversion counter.

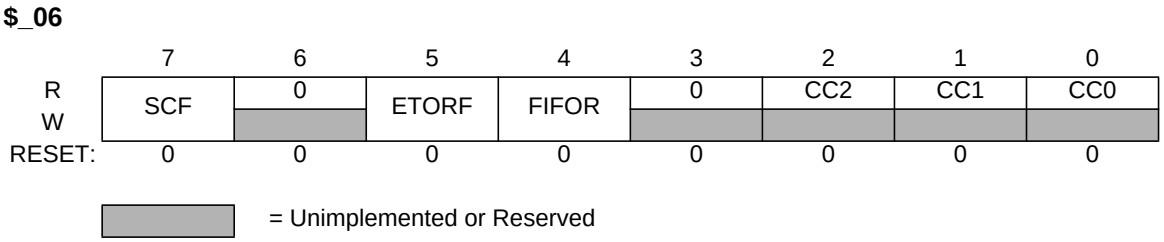


Figure 3-7 ATD Status Register 0 (ATDSTAT0)

Read: anytime

Write: anytime (No effect on (CC2, CC1, CC0))

SCF — Sequence Complete Flag

This flag is set upon completion of a conversion sequence. If conversion sequences are continuously performed (SCAN=1), the flag is set after each one is completed. This flag is cleared when one of the following occurs:
A) Write “1” to SCF
B) Write to ATDCTL5 (a new conversion sequence is started)
C) If AFFC=1 and read of a result register
1 = Conversion sequence has completed
0 = Conversion sequence not completed

ETORF — External Trigger Overrun Flag

While in edge trigger mode (ETRIGLE=0), if additional active edges are detected while a conversion sequence is in process the overrun flag is set. This flag is cleared when one of the following occurs:
A) Write “1” to ETORF
B) Write to ATDCTL2, ATDCTL3 or ATDCTL4 (a conversion sequence is aborted)
C) Write to ATDCTL5 (a new conversion sequence is started)
1 = External trigger over run error has occurred

0 = No External trigger over run error has occurred

FIFOR - FIFO Over Run Flag.

This bit indicates that a result register has been written to before its associated conversion complete flag (CCF) has been cleared. This flag is most useful when using the FIFO mode because the flag potentially indicates that result registers are out of sync with the input channels. However, it is also practical for non-FIFO modes, and indicates that a result register has been over written before it has been read (i.e. the old data has been lost). This flag is cleared when one of the following occurs:
A) Write “1” to FIFOR
B) Start a new conversion sequence (write to ATDCTL5 or external trigger)
1 = An over run condition exists
0 = No over run has occurred

CC2, CC1, CC0 — Conversion Counter

These 3 read-only bits are the binary value of the conversion counter. The conversion counter points to the result register that will receive the result of the current conversion. E.g. CC2=1, CC1=1, CC0=0 indicates that the result of the current conversion will be in ATD Result Register 6. If in non-FIFO mode (FIFO=0) the conversion counter is initialized to zero at the begin and end of the conversion sequence. If in FIFO mode (FIFO=1) the register counter is not initialized. The conversion counters wraps around when its maximum value is reached.
Aborting a conversion or starting a new conversion by write to an ATDCTL register (ATDCTL5-0) clears the conversion counter even if FIFO=1.

3.3.8 Reserved Register (ATDTEST0)

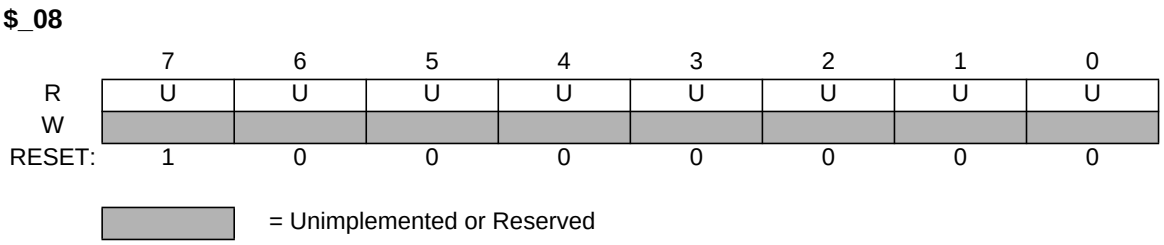


Figure 3-8 Reserved Register (ATDTEST0)

Read: anytime, returns unpredictable values

Write: anytime in special modes, unimplemented in normal modes

NOTE: *Writing to this registers when in special modes can alter functionality.*

3.3.9 ATD Test Register 1 (ATDTEST1)

This register contains the SC bit used to enable special channel conversions.

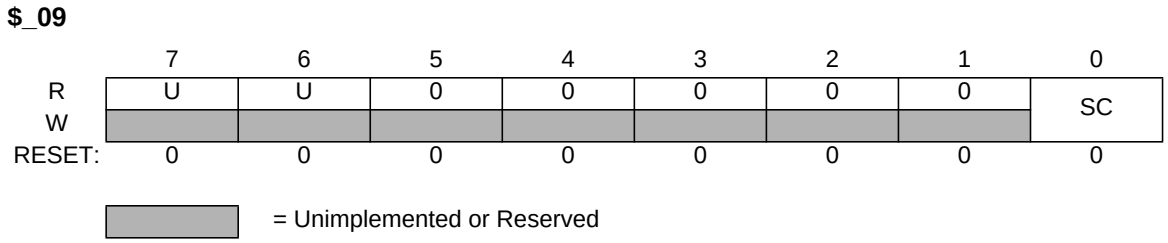


Figure 3-9 ATD Test Register 1 (ATDTEST1)

Read: anytime, returns unpredictable values for Bit7 and Bit6

Write: anytime

SC - Special Channel Conversion Bit

If this bit is set, then special channel conversion can be selected using CC, CB and CA of ATDCTL5. **Table 3-10** lists the coding.

1 = Special channel conversions enabled
0 = Special channel conversions disabled

NOTE: *Always write remaining bits of ATDTEST1 (Bit7 to Bit1) zero when writing SC bit. Not doing so might result in unpredictable ATD behavior.*

Table 3-10 Special Channel Select Coding

SC	CC	CB	CA	Analog Input Channel
1	0	X	X	Reserved
1	1	0	0	V_{RH}
1	1	0	1	V_{RL}
1	1	1	0	$(V_{RH}+V_{RL}) / 2$
1	1	1	1	Reserved

3.3.10 ATD Status Register 1 (ATDSTAT1)

This read-only register contains the Conversion Complete Flags.

$_0B

	7	6	5	4	3	2	1	0
R	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-10 ATD Status Register 1 (ATDSTAT1)

Read: anytime

Write: anytime, no effect

CCFx — Conversion Complete Flag x (x=7,6,5,4,3,2,1,0)

A conversion complete flag is set at the end of each conversion in a conversion sequence. The flags are associated with the conversion position in a sequence (and also the result register number). Therefore, CCF0 is set when the first conversion in a sequence is complete and the result is available in result register ATDDR0; CCF1 is set when the second conversion in a sequence is complete and the result is available in ATDDR1, and so forth. A flag CCFx (x=7,6,5,4,3,2,1,0) is cleared when one of the following occurs:

A) Write to ATDCTL5 (a new conversion sequence is started)

B) If AFFC=0 and read of ATDSTAT1 followed by read of result register ATDDRx

C) If AFFC=1 and read of result register ATDDRx

1 = Conversion number x has completed, result ready in ATDDRx

0 = Conversion number x not completed

3.3.11 ATD Input Enable Register (ATDDIEN)

$_0D

	7	6	5	4	3	2	1	0
R	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-11 ATD Input Enable Register (ATDDIEN)

Read: anytime

Write: anytime

IENx — ATD Digital Input Enable on channel x (x= 7, 6, 5, 4, 3, 2, 1, 0)

This bit controls the digital input buffer from the analog input pin (ANx) to PTADx data register.

1 = Enable digital input buffer to PTADx.

0 = Disable digital input buffer to PTADx

NOTE: *Setting this bit will enable the corresponding digital input buffer continuously. If this bit is set while simultaneously using it as an analog port, there is potentially increased power consumption because the digital input buffer maybe in the linear region.*

3.3.12 Port Data Register (PORTAD)

The digital port pins are shared with the analog A/D inputs AN7-0.

$_0F

	7	6	5	4	3	2	1	0
R	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
W								
RESET:	1	1	1	1	1	1	1	1
Pin Function	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0

= Unimplemented or Reserved

Figure 3-12 Port Data Register (PORTAD)

Read: anytime

Write: anytime, no effect

PTADx — A/D Channel x (ANx) Digital Input (x= 7,6,5,4,3,2,1,0)

If the digital input buffer on the ANx pin is enabled (IENx=1) read returns the logic level on ANx pin (signal potentials not meeting VIL or VIH specifications will have an indeterminate value)).

If the digital input buffers are disabled (IENx=0), read returns a “1”.

Reset sets all PORTAD bits to “1”.

3.3.13 ATD Conversion Result Registers (ATDDRHx/ATDDRLx)

The A/D conversion results are stored in 8 read-only result registers ATDDRHx/ATDDRLx. The result data is formatted in the result registers based on two criteria. First there is left and right justification; this selection is made using the DJM control bit in ATDCTL5. Second there is signed and unsigned data; this selection is made using the DSGN control bit in ATDCTL5. Signed data is stored in 2’s complement format and only exists in left justified format. Signed data selected for right justified format is ignored.

Read: anytime

Write: anytime, no effect in normal modes

3.3.13.1 Left Justified Result Data

$_10 = ATDDR0H, $_12 = ATDDR1H, $_14 = ATDDR2H, $_16 = ATDDR3H
$_18 = ATDDR4H, $_1A = ATDDR5H, $_1C = ATDDR6H, $_1E = ATDDR7H

	7	6	5	4	3	2	1	0	
R	BIT 9 MSB	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	10-bit data
W	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	8-bit data
RESET:	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 3-13 Left Justified, ATD Conversion Result Register, High Byte (ATDDRxH)

$_11 = ATDDR0L, $_13 = ATDDR1L, $_15 = ATDDR2L, $_17 = ATDDR3L
$_19 = ATDDR4L, $_1B = ATDDR5L, $_1D = ATDDR6L, $_1F = ATDDR7L

	7	6	5	4	3	2	1	0	
R	BIT 1	BIT 0	0	0	0	0	0	0	10-bit data
W	U	U	0	0	0	0	0	0	8-bit data
RESET:	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 3-14 Left Justified, ATD Conversion Result Register, Low Byte (ATDDRxL)

3.3.13.2 Right Justified Result Data

$_10 = ATDDR0H, $_12 = ATDDR1H, $_14 = ATDDR2H, $_16 = ATDDR3H
$_18 = ATDDR4H, $_1A = ATDDR5H, $_1C = ATDDR6H, $_1E = ATDDR7H

	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	BIT 9 MSB	BIT 8	10-bit data
W	0	0	0	0	0	0	0	0	8-bit data
RESET:	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 3-15 Right Justified, ATD Conversion Result Register, High Byte (ATDDRxH)

$_11 = ATDDR0L, $_13 = ATDDR1L, $_15 = ATDDR2L, $_17 = ATDDR3L
$_19 = ATDDR4L, $_1B = ATDDR5L, $_1D = ATDDR6L, $_1F = ATDDR7L

	7	6	5	4	3	2	1	0	
R	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	10-bit data
W	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	8-bit data
RESET:	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 3-16 Right Justified, ATD Conversion Result Register, Low Byte (ATDDRxL)

Section 4 Functional Description

4.1 General

The ATD_10B8C is structured in an analog and a digital sub-block.

4.2 Analog Sub-block

The analog sub-block contains all analog electronics required to perform a single conversion. Separate power supplies VDDA and VSSA allow to isolate noise of other MCU circuitry from the analog sub-block.

4.2.1 Sample and Hold Machine

The Sample and Hold (S/H) Machine accepts analog signals from the external surroundings and stores them as capacitor charge on a storage node.

The sample process uses a two stage approach. During the first stage, the sample amplifier is used to quickly charge the storage node.The second stage connects the input directly to the storage node to complete the sample for high accuracy.

When not sampling, the sample and hold machine disables its own clocks. The analog electronics still draw their quiescent current. The power down (ADPU) bit must be set to disable both the digital clocks and the analog power consumption.

The input analog signals are unipolar and must fall within the potential range of VSSA to VDDA.

4.2.2 Analog Input Multiplexer

The analog input multiplexer connects one of the 8 external analog input channels to the sample and hold machine.

4.2.3 Sample Buffer Amplifier

The sample amplifier is used to buffer the input analog signal so that the storage node can be quickly charged to the sample potential.

4.2.4 Analog-to-Digital (A/D) Machine

The A/D Machine performs analog to digital conversions. The resolution is program selectable at either 8 or 10 bits. The A/D machine uses a successive approximation architecture. It functions by comparing the stored analog sample potential with a series of digitally generated analog potentials. By following a binary search algorithm, the A/D machine locates the approximating potential that is nearest to the sampled potential.

When not converting the A/D machine disables its own clocks. The analog electronics still draws quiescent current. The power down (ADPU) bit must be set to disable both the digital clocks and the analog power consumption.

Only analog input signals within the potential range of V_{RL} to V_{RH} (A/D reference potentials) will result in a non-railed digital output codes.

4.3 Digital Sub-block

This subsection explains some of the digital features in more detail. See register descriptions for all details.

4.3.1 External Trigger Input (ETRIG)

The external trigger feature allows the user to synchronize ATD conversions to the external environment events rather than relying on software to signal the ATD module when ATD conversions are to take place. The input signal (ATD channel 7) is programmable to be edge or level sensitive with polarity control. **Table 4-1** gives a brief description of the different combinations of control bits and their affect on the external trigger function.

Table 4-1 External Trigger Control Bits

ETRIGLE	ETRIGP	ETRIGE	SCAN	Description
X	X	0	0	Ignores external trigger. Performs one conversion sequence and stops.
X	X	0	1	Ignores external trigger. Performs continuous conversion sequences.
0	0	1	X	Falling edge triggered. Performs one conversion sequence per trigger.
0	1	1	X	Rising edge triggered. Performs one conversion sequence per trigger.
1	0	1	X	Trigger active low. Performs continuous conversions while trigger is active.
1	1	1	X	Trigger active high. Performs continuous conversions while trigger is active.

During a conversion, if additional active edges are detected the overrun error flag ETORF is set.

In either level or edge triggered modes, the first conversion begins when the trigger is received. In both cases, the maximum latency time is one Bus Clock cycle plus any skew or delay introduced by the trigger circuitry.

NOTE: *The conversion results for the external trigger ATD channel 7 have no meaning while external trigger mode is enabled.*

Once ETRIGE is enabled, conversions cannot be started by a write to ATDCTL5, but rather must be triggered externally.

If the level mode is active and the external trigger both de-asserts and re-asserts itself during a conversion sequence, this does not constitute an overrun; therefore, the flag is not set. If the trigger is left asserted in level mode while a sequence is completing, another sequence will be triggered immediately.

4.3.2 General Purpose Digital Input Port Operation

The input channel pins can be multiplexed between analog and digital data. As analog inputs, they are multiplexed and sampled to supply signals to the A/D converter. As digital inputs, they supply external input data that can be accessed through the digital port register PORTAD (input-only).

The analog/digital multiplex operation is performed in the input pads. The input pad is always connected to the analog inputs of the ATD_10B8C. The input pad signal is buffered to the digital port registers. This buffer can be turned on or off with the ATDDIEN register. This is important so that the buffer does not draw excess current when analog potentials are presented at its input.

4.3.3 Low Power Modes

The ATD_10B8C can be configured for lower MCU power consumption in 3 different ways:

- Stop Mode: This halts A/D conversion. Exit from Stop mode will resume A/D conversion, But due to the recovery time the result of this conversion should be ignored.
- Wait Mode with AWAI=1: This halts A/D conversion. Exit from Wait mode will resume A/D conversion, but due to the recovery time the result of this conversion should be ignored.
- Writing ADPU=0 (Note that all ATD registers remain accessible.): This aborts any A/D conversion in progress.

Note that the reset value for the ADPU bit is zero. Therefore, when this module is reset, it is reset into the power down state.

Section 5 Resets

5.1 General

At reset the ATD_10B8C is in a power down state. The reset state of each individual bit is listed within the Register Description section (see **Section 3 Memory Map and Register Definition**) which details the registers and their bit-field.

Section 6 Interrupts

6.1 General

The interrupt requested by the ATD_10B8C is listed in **Table 6-1**. Refer to MCU specification for related vector address and priority.

Table 6-1 ATD_10B8C Interrupt Vectors

Interrupt Source	CCR Mask	Local Enable
Sequence Complete Interrupt	I bit	ASCIE in ATDCTL2

See register descriptions for further details.

User Guide End Sheet

FINAL PAGE OF
40
PAGES

BDLC
Block Guide
V01.03

Original Release Date:19 JAN 2001
Revised: July 19, 2001

Motorola, Inc

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.00	04/20/2001	01/19/2001		Original Release
V01.01	06/12/2001	06/13/2001		Corrected formal issues w/paragraph formats, cross references and master pages.
V01.02	06/13/2001	06/14/2001		Removed references to internal signals Moved initialization chap. in 'RESETS' to functional descrip Removed redundant references in the Interrupts section Updated the Interrupts table Changed some explanations to bullets
V01.03	07/19/2001			Document names have been added Names and Variable defenitions have been hidden

Table of Contents

Section 1 Introduction

Section 2 Signal Description

Section 3 Memory Map and Registers

Section 4 Functional Description

Section 5 Resets

Section 6 Interrupts

Appendix A Electrical Specifications

List of Tables

List of Figures

Section 1 Introduction

1.1 Overview

The BDLC module is a serial communication module which allows the user to send and receive messages across a Society of Automotive Engineers (SAE) J1850 serial communication network. The user's software handles each transmitted or received message on a byte-by-byte basis, while the BDLC performs all of the network access, arbitration, message framing and error detection duties.

It is recommended that the reader be familiar with the operation and requirements of the SAE J1850 protocol as described in the document "SAE Standard J1850 Class B Data Communications Network Interface" prior to proceeding with this specification.

The BDLC module is designed in a modular structure for use as an IP block. A general working knowledge of the IP Bus signals and bus control is assumed in the writing of this document. For details, refer to the SRS IP Bus specifications.

1.2 Features

Features of the BDLC module include the following:

- SAE J1850 Class B Data Communications Network Interface Compatible and ISO Compatible for Low-Speed (≤ 125 Kbps) Serial Data Communications in Automotive Applications
- 10.4 Kbps Variable Pulse Width (VPW) Bit Format
- Digital Noise Filter
- Digital Loopback Mode
- 4X Receive Mode, 41.6 Kbps, Supported
- Block Mode Receive and Transmit Supported
- Collision Detection
- Hardware Cyclical Redundancy Check (CRC) Generation and Checking
- Dedicated Register for Symbol Timing Adjustments
- IP Bus Interface
- In-Frame Response (IFR) Types 0, 1, 2, and 3 Supported
- Power-Saving Stop and Wait Modes with Automatic Wakeup on Network Activity
- Polling and CPU Interrupt Generation with Vector Lookup Available

1.3 Modes of Operation

- The BDLC module has 6 main modes of operation which interact with the power supplies, pins, and the rest of the MCU as shown below.

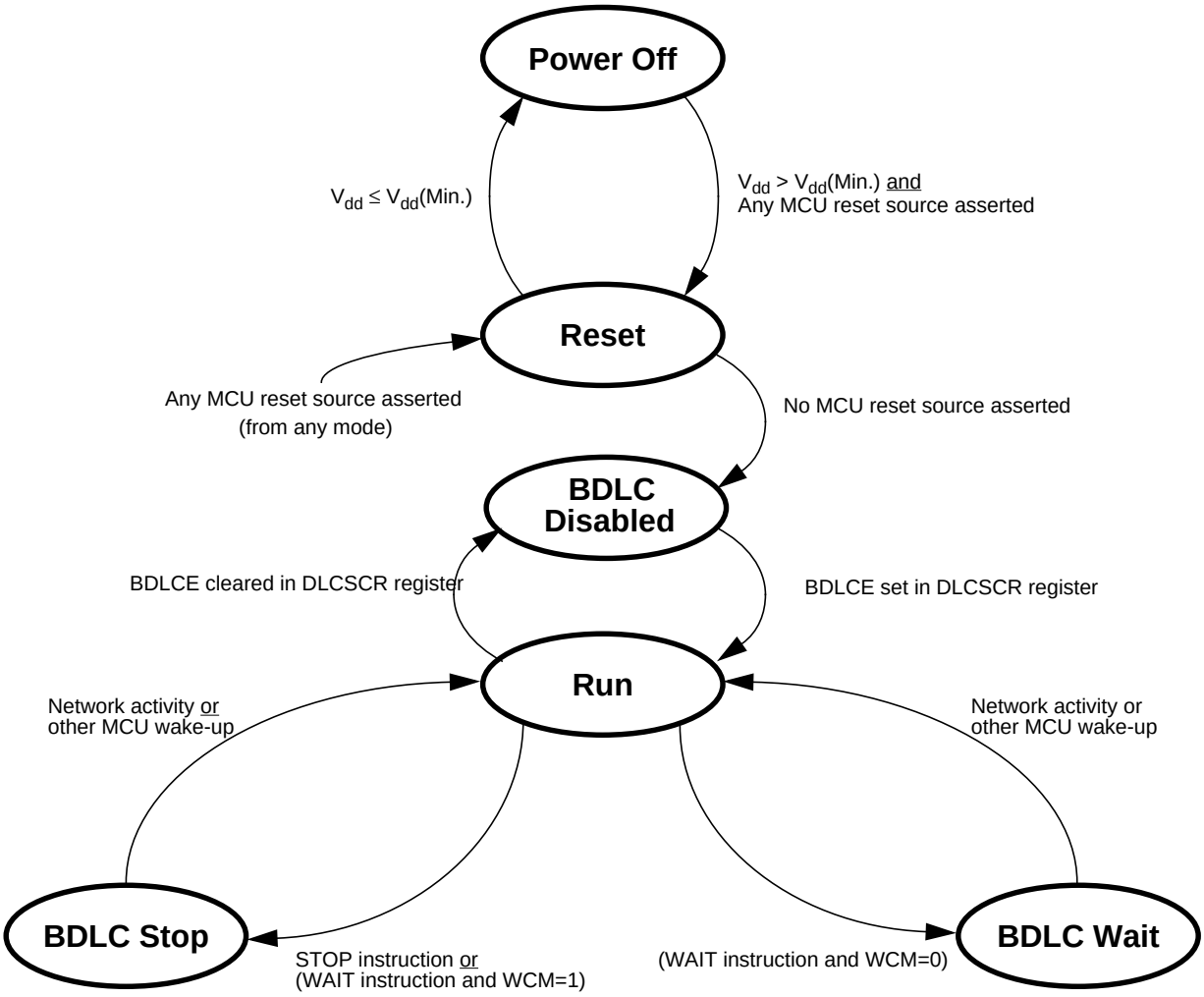


Figure 1-1 BDLC Operating Modes State Diagram

- **Power Off**

  This mode is entered from the Reset mode whenever the BDLC module supply voltage V_{dd} drops below its minimum specified value for the BDLC module to guarantee operation. The BDLC module will be placed in the Reset mode by a system Low Voltage Reset (LVR) before being powered down. In this mode, the pin input and output specifications are not guaranteed.

- **Reset**

  This mode is entered from the Power Off mode whenever the BDLC module supply voltage V_{dd} rises above its minimum specified value ($V_{dd(MIN)}$) and some MCU reset source is asserted. To prevent the BDLC from entering an unknown state, the internal MCU reset is asserted while

powering up the BDLC module. BDLC Reset mode is also entered from any other mode as soon as one of the MCU's possible reset sources (e.g. LVR, POR, COP watchdog, Reset pin etc.) is asserted.

In this mode, the internal BDLC module voltage references are operative, V_{dd} is supplied to the internal circuits, which are held in their reset state and the internal BDLC module system clock is running. Registers will assume their reset condition. Outputs are held in their programmed Reset state, inputs and network activity are ignored.

- **BDLC Disabled**

  This mode is entered from the Reset mode after all MCU reset sources are no longer asserted. It is entered from the Run mode whenever the BDLCE bit in the DLCSCR register is cleared.

  In this mode the mux interface clock (f_{bdlc}) is stopped to conserve power and allow the BDLC module to be configured for proper operation on the J1850 bus. The IP bus interface clocks are left running in this mode to allow access to all BDLC module registers for initialization.

- **Run**

  This mode is entered from the BDLC Disabled mode when the BDLCE bit in the DLCSCR register is set. It is entered from the BDLC Wait mode whenever activity is sensed on the J1850 bus or some other MCU source wakes the CPU out of Wait mode.

  It is entered from the BDLC Stop mode whenever network activity is sensed or some other MCU source wakes the CPU out of Stop mode. Messages will not be received properly until the clocks have stabilized and the CPU is also in the Run mode.

- **BDLC Wait (Core Specific)**

  This power conserving mode is automatically entered from the Run mode whenever the CPU executes a WAIT instruction and if the WCM bit in the DLCBCR1 register is previously cleared. In this mode, the BDLC module internal clocks continue to run. Any activity on the J1850 network will cause the BDLC module to exit BDLC Wait mode and generate an unmaskable interrupt of the CPU. This wakeup interrupt state is reflected in the DLCBSVR, encoded as the highest priority interrupt. This interrupt can be cleared by the CPU with a read of the DLCBSVR.

  – Wakeup from BDLC Wait with CPU in WAIT

    If the CPU executes the WAIT instruction and the BDLC module enters the WAIT mode (WCM = 0), the clocks to the BDLC module as well as the clocks in the MCU continue to run. Therefore, the message which wakes up the BDLC module from WAIT and the CPU from WAIT mode will also be received correctly by the BDLC module. This is because all of the required clocks continue to run in the BDLC module in WAIT mode.The wakeup behavior of the BDLC module applies regardless of whether the BDLC module is in normal or 4X mode when the WAIT instruction is executed.

- **BDLC Stop (Core Specific)**

  This power conserving mode is automatically entered from the Run mode whenever the CPU executes a STOP instruction, or if the CPU executes a WAIT instruction and the WCM bit in the DLCBCR1 register is previously set. In this mode, the BDLC internal clocks are stopped. Any activity on the network will cause the BDLC module to exit BDLC Stop mode and generate an

unmaskable interrupt of the CPU. This wakeup interrupt state is reflected in the DLCBSVR, encoded as the highest priority interrupt. This interrupt can be cleared by the CPU with a read of the DLCBSVR. Depending upon which low-power mode instruction the CPU executes to cause the BDLC module to enter BDLC Stop, the message which wakes up the BDLC module (and the CPU) may or may not be received. There are two different possibilities, both of which is described below. These descriptions apply regardless of whether the BDLC module is in normal or 4X mode when the STOP or WAIT instruction is executed.

– Wakeup from BDLC Stop with CPU in STOP

  When the CPU executes the STOP instruction, all clocks in the MCU, including clocks to the BDLC module, are turned off. Therefore, the message which wakes up the BDLC module and the CPU from STOP mode will not be received. This is due primarily to the amount of time required for the MCU's oscillator to stabilize before the clocks can be applied internally to the other MCU modules, including the BDLC module.

– Wakeup from BDLC Stop with CPU in WAIT

  If the CPU executes the WAIT instruction and the BDLC module enters the Stop mode (WCM = 1), the clocks to the BDLC module are turned off, but the clocks in the MCU continue to run. Therefore, the message which wakes up the BDLC module from Stop and the CPU from WAIT mode will be received correctly by the BDLC module. This is because very little time is required for the CPU to turn the clocks to the BDLC module back on once the wakeup interrupt occurs.

NOTE: *While the BDLC module will correctly receive a message which arrives when the BDLC module is in Stop mode or Wait mode and the MCU is in WAIT mode, if the user enters this mode while a message is being received, the data in the message will become corrupted. This is due to the steps required for the BDLC module to resume operation upon exiting Stop mode or Wait mode, and its subsequent resynchronization with the SAE J1850 bus.*

- **Digital Loopback**

  When a bus fault has been detected, the digital loopback mode is used to determine if the fault condition is caused by failure in the node's internal circuits or elsewhere in the network, including the node's analog physical interface. In this mode, the input to the digital filter is disconnected from the receive pin input (RXB). The input to the digital filter is then connected to the transmitter output to form the loopback connection. The transmit pin (TXB) is negated and will always drive a passive state onto the bus. Digital loopback mode is entered by setting the DLOOP bit in Section 3.3.3 BDLC Control Register 2 (DLCBCR2).

- **Normal and Emulation Mode Operation (Core Specific)**

  The BDLC module operates in the same manner in all Normal and Emulation Modes. All BDLC module registers can be read and written except those that are reserved, unimplemented, or write once. The user must be careful not to unintentionally write a register when using 16-bit writes in order to avoid unexpected BDLC module behavior.

- **Special Mode Operation (Core Specific)**

Some aspects of BDLC module operation can be modified in special test mode. This mode is reserved for internal use only.

- **Low Power Options (Core Specific)**

  The BDLC module can save power in Disabled, Wait, and Stop modes. A complete description of what the BDLC module does while in a low power mode can be found in **Section 1.3 Modes of Operation**.

1.4 Block Diagram

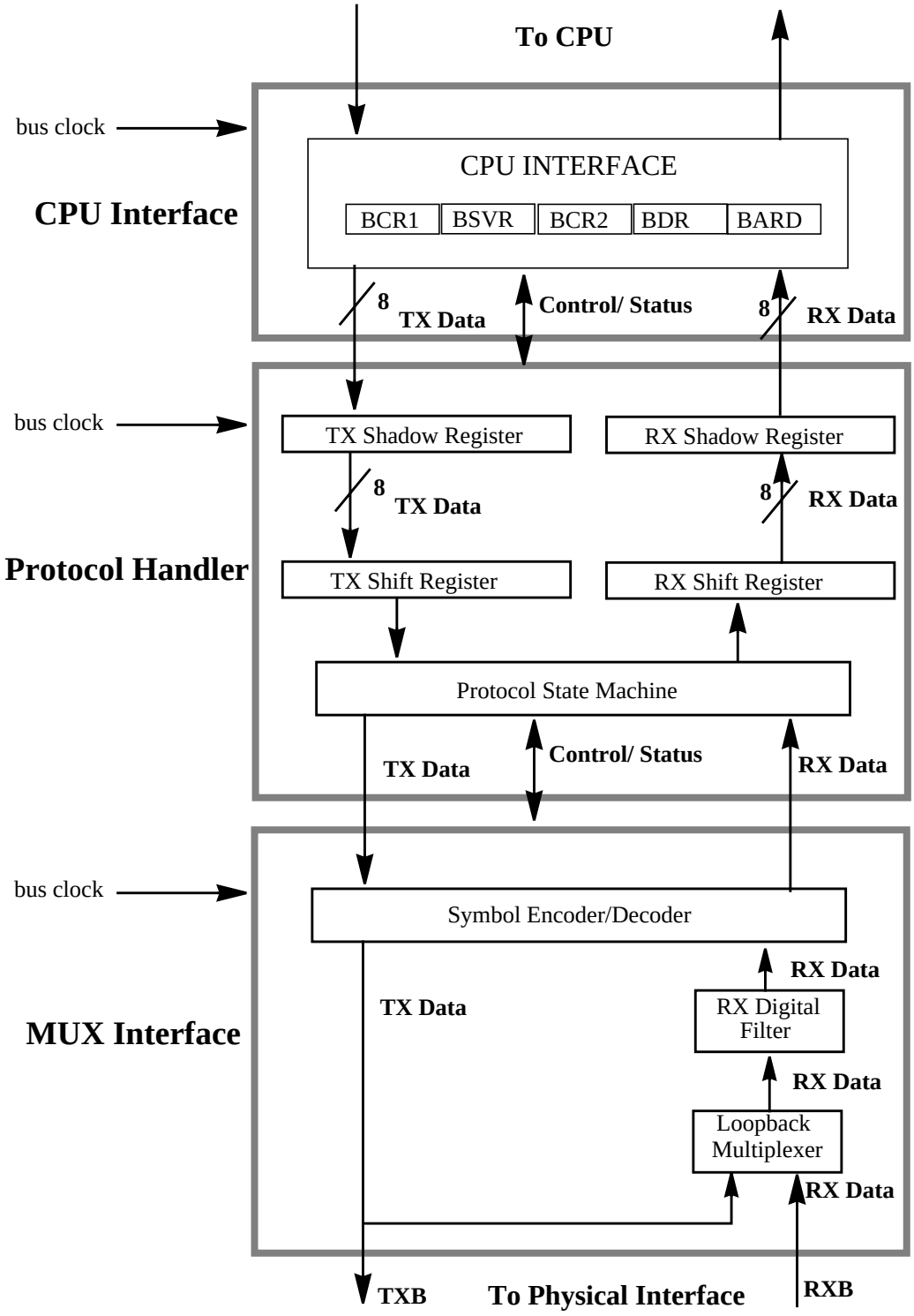


Figure 1-2 BDLC Block Diagram

Figure 1-2 shows the organization of the BDLC module. The Buffers provide storage for data received and data to be transmitted onto the J1850 bus. The Protocol Handler is responsible for the encoding and decoding of data bits and special message symbols during transmission and reception. The MUX Interface provides the link between the BDLC digital section and the analog Physical Interface. The wave shaping, driving and digitizing of data is performed by the Physical Interface.

NOTE: *The Physical Interface is not implemented in the BDLC module and must be provided externally.*

The main functional blocks of the BDLC module are explained in greater detail in the following sections.

Use of the BDLC module in message networking fully implements the "SAE Standard J1850 Class B Data Communication Network Interface" specification.

Section 2 Signal Description

2.1 Overview

The BDLC module has a total of 2 external pins.

2.2 Detailed Signal Descriptions

2.2.1 TXB - BDLC Transmit Pin

The TXB pin serves as the transmit output channel for the BDLC module.

2.2.2 RXB - BDLC Receive Pin

The RXB pin serves as the receive input channel for the BDLC module.

Section 3 Memory Map and Registers

3.1 Overview

This section provides a detailed description of all memory and registers accessible to the end user.

3.2 Module Memory Map

Table 3-1 Module Memory Map

Address	Use	Access
Base + $_00	BDLC Control Register 1 (DLCBCR1)	R/W
Base + $_01	BDLC State Vector Register (DLCBSVR)	R/W
Base + $_02	BDLC Control Register 2 (DLCBCR2)	R/W
Base + $_03	BDLC Data Register (DLCBDR)	R/W
Base + $_04	BDLC Analog RoundTrip Delay Register (DLCBARD)	R/W
Base + $_05	BDLC Rate Select Register (DLCBRSR)	R/W
Base + $_06	BDLC Control Register (DLCSCR)	R/W
Base + $_07	BDLC Status Register (DLCBSTAT)	R/W

3.3 Register Descriptions

3.3.1 BDLC Control Register 1 (DLCBCR1)

This register is used to configure and control the BDLC module.

Register Offset: $_00

	7	6	5	4	3	2	1	0
R	IMSG	CLKS	0	0	0	0	IE	WCM
W								
RESET:	1	1	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-1 BDLC Control Register 1

READ: any time

WRITE: IMSG, IE, and WCM any time.

CLKS write once in normal and emulation modes.

CLKS bit has modified functionality in special test mode.

Writes to unimplemented bits 5-2 are ignored.

IMSG — Ignore Message (Bit 7)

This bit allows the CPU to ignore messages by disabling updates of the DLCBSVR register until a new Start of Frame (SOF) or a BREAK symbol is detected. BDLC module transmitter and receiver operation are unaffected by the state of the IMSG bit.

1 = Disable DLCBSVR Updates. When set, all BDLC interrupt sources (exceptions are described below) will be prevented from updating DLCBSVR status bits. Setting IMSG does not clear pending interrupt flags, the behavior of which will still be as described in Section BDLC State Vector Register (DLCBSVR). If this bit is set while the BDLC is receiving or transmitting a message, state vector register updates will be inhibited for the rest of the message.

0 = Enable DLCBSVR Updates. This bit is automatically cleared by the reception of a SOF symbol or a BREAK symbol. It will then allow updates of the state vector register to occur.

There are two situations in which interrupts will not be masked by the IMSG bit: when a wakeup interrupt occurs; and when a receiver error occurs which causes a byte pending transmission to be flushed from the transmit shadow register. See Section 3.3.4 BDLC Data Register (DLCBDR) for a description of the conditions which cause a pending transmission to be flushed.

CLKS — Clock Select (Bit 6)

The nominal BDLC operating frequency (mux interface clock frequency - f_{bdlc}) **must** always be 1.048576 MHz or 1 MHz in order for J1850 bus communications to take place properly. The CLKS register bit is provided to allow the user to indicate to the BDLC module which frequency (1.048576 MHz or 1 MHz) is used so that each symbol time can be automatically adjusted.

The CLKS bit is a write once bit. All writes to this bit will be ignored after the first one.

1 = Binary frequency (1.048576 MHz) is used for f_{bdlc}.

0 = Integer frequency (1 MHz) is used. for f_{bdlc}

Section 4.1.3 J1850 VPW Valid/Invalid Bits & Symbols on page 41 describes the transmitter and receiver VPW symbol timing for integer and binary frequencies.

IE — Interrupt Enable (Bit 1)

This bit determines whether the BDLC module will generate CPU interrupt requests. It does not affect CPU interrupt requests when exiting the BDLC module Stop or Wait modes. Interrupt requests will be maintained until all of the interrupt request sources are cleared, by performing the specified actions upon the BDLC module's registers. Interrupts that were pending at the time that this bit is cleared may be lost.

1 = Enable interrupt requests from BDLC module

0 = Disable interrupt requests from BDLC module

If the programmer does not wish to use the interrupt capability of the BDLC module, the BDLC State Vector Register (DLCBSVR) can be polled periodically by the programmer to determine BDLC module states. Refer to Section 3.3.2 BDLC State Vector Register (DLCBSVR) on page 23 for a description of DLCBSVR register and how to clear interrupt requests.

WCM — Wait Clock Mode (Bit 0) *(Provided CPU has Low Power Mode Options)*

This bit determines how the BDLC module responds when the CPU enters WAIT mode. As described in Section 1.3 Modes of Operation on page 11, the BDLC module can respond by either entering BDLC_STOP mode, where all internal clocks are stopped, or entering BDLC_WAIT mode where internal clocks are allowed to run.

1 = Stop BDLC internal clocks during CPU wait mode (BDLC_STOP)
0 = Run BDLC internal clocks during CPU wait mode (BDLC_WAIT)

3.3.2 BDLC State Vector Register (DLCBSVR)

This register is provided to substantially decrease the CPU overhead associated with servicing interrupts while under operation of a MUX protocol. It provides a index offset that is directly related to the BDLC module's current state, which can be used with a user supplied jump table to rapidly enter an interrupt service routine. This eliminates the need for the user to maintain a duplicate state machine in software.

Register Offset: $_01

	7	6	5	4	3	2	1	0
R	0	0	I3	I2	I1	I0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 BDLC State Vector Register

READ: any time

WRITE: ignored

I[3:0] — Interrupt State Vector (Bits 5- 2)

These bits indicate the source of the interrupt request that is currently pending.

Table 3-2 Interrupt Summary

BSVR	I3	I2	I1	I0	Interrupt Source	Priority
$00	0	0	0	$\overline{0}$	No Interrupts Pending	0 (Lowest)
$04	0	0	0	1	Received EOF	1
$08	0	0	1	0	Received IFR byte	2
$0C	0	0	1	1	Rx data register full	3
$10	0	1	0	0	Tx data register empty	4
$14	0	1	0	1	Loss of arbitration	5
$18	0	1	1	0	CRC error	6
$1C	0	1	1	1	Symbol invalid or out of range	7
$20	1	0	0	0	Wakeup	8 (Highest)

The state encoding of the interrupt sources mean that only one interrupt source is dealt with at a time. Once the highest priority interrupt source is dealt with, if another interrupt event of a lower priority has also occurred, the value corresponding to that interrupt source appears in the BSVR. This continues until all BDLC interrupt sources have been dealt with and all bits in the BSVR are cleared.

- Wakeup

  The BDLC has two different power-conserving modes, stop and wait. Wakeup from these modes is described below.

- Wakeup from BDLC Wait with CPU in Wait

  If the CPU executes a WAIT instruction and the BDLC enters the BDLC wait mode, the clocks to the BDLC as well as the clocks in the MCU continue to run. The message which generates a Wake-up interrupt of the BDLC and the CPU will be received correctly.

- Wakeup from BDLC Stop with CPU in Wait

  If the CPU executes a WAIT instruction and the BDLC enters the BDLC stop mode, the clocks to the BDLC are turned off, but the clocks in the MCU continue to run. The message which generates a Wake-up interrupt of the BDLC and the CPU will be received correctly. To ensure this, the EOF following the last message appearing on the bus must be received; otherwise, the message will not be received correctly.

- Wakeup from BDLC Stop with CPU in Stop

  If the CPU executes a STOP all clocks to the BDLC as well as the clocks in the MCU are turned off including clocks to the BDLC. The message which generates a Wake-up interrupt of the BDLC and the CPU will not be received correctly.

- Symbol Invalid or Out of Range

- CRC Error

  The Cyclical Redundancy Check Byte is used by the receiver(s) of each message to determine if any errors have occurred during the transmission of the message. If the message is not error free, the CRC error status is shown in the BSVR.

- Loss of Arbitration

  The Loss of Arbitration status is entered when a loss of arbitration occurs while the BDLC is transmitting onto the bus.

- Tx Data Register Empty

  The Tx Data Register Empty (TDRE) Byte is used to tell when data has been unloaded from the BDLC Data Register (BDR).

- Rx Data Register Full

  The Rx Data Register Full (RDRF) Byte is used to tell when data has been loaded in the BDLC Data Register (BDR).

- Received IFR Byte

  The BDLC can transmit and receive all four types of in-frame responses. As each byte of an IFR is received, the BSVR indicates this by setting this state.

- Received EOF

  When a 280us passive period on the bus is received, it signifies an EOF. Whenever this occurs, the EOF flag is set.

- No Interrupts Pending

  This interrupt cannot generate an interrupt of the CPU.

3.3.3 BDLC Control Register 2 (DLCBCR2)

This register controls transmitter operations of the BDLC module.

Register Offset: $_02

	7	6	5	4	3	2	1	0
R W	SMRST	DLOOP	RX4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
RESET:	0	1	0	0	0	0	0	0

Figure 3-3 BDLC Control Register 2

READ: any time

WRITE: any time

SMRST — State Machine Reset (Bit 7)

The programmer can use this bit to reset the BDLC state machines to an initial state after the user put the off-chip analog transceiver in loop back mode.

1 = Setting SMRST arms the state machine reset generation logic. Setting SMRST does not affect BDLC module behavior in any way.

0 = Clearing SMRST after it has been set will cause the generation of a state machine reset. After SMRST is cleared, the BDLC requires the bus to be idle for a minimum of an End of Frame symbol (EOF) time before allowing the reception of a message. The BDLC requires the bus to be idle for a minimum of an Inter-Frame Separator symbol (IFS) time before allowing any message to be transmitted.

DLOOP — Digital Loopback Mode (Bit 6)

This bit determines the source to which the input of the digital filter is connected and can be used to isolate bus fault conditions. If a fault condition has been detected on the bus, this control bit allows the programmer to disconnect the digital filter from input from the receive pin (RXB) and connect it to the transmit output to the pin (TXB). In this configuration, data sent from the transmit buffer should be reflected back into the receive buffer. If no faults exist in the digital block, the fault is in the physical interface block or elsewhere on the J1850 bus.

1 = When set, digital filter input is connected to the transmitter output. The BDLC module is now in Digital Loopback Mode of operation. The transmit pin (TXB) is driven low and not driven by the transmitter output.

0 = When cleared, digital filter input is connected to receive pin (RXB) and the transmitter output is connected to the transmit pin (TXB). The BDLC module is taken out of Digital Loopback Mode and can now drive and receive from the J1850 bus normally. After writing DLOOP to zero, the BDLC module requires the bus to be idle for a minimum of an End of Frame symbol time before allowing a reception of a message. The BDLC module requires the bus to be idle for a minimum of an Inter-Frame Separator symbol time before allowing any message to be transmitted.

NOTE: *The DLOOP bit is a fault condition aid and should never be altered after the DLCBDR is loaded for transmission. Changing DLOOP during a transmission may cause corrupted data to be transmitted onto the J1850 network.*

RX4XE — Receive 4X Enable (Bit 5)

This bit determines if the BDLC operates at normal transmit and receive speed (10.4 kbps) or receive only at 41.6 kbps. This feature is useful for fast download of data into a J1850 node for diagnostic or factory programming of the node.

1 = When set, the BDLC module is put in 4X (41.6 kbps) receive only operation.

0 = When cleared, the BDLC module transmits and receives at 10.4 kbps. Reception of a BREAK symbol automatically clears this bit and sets the symbol invalid or out of range flag (DLCBSVR = $1C).

The effect of 4X receive operation on receive symbol timing boundaries is described in Transmit and Receive Symbol Timing Specifications.The RX4XE bit is not affected by entry or exit from BDLC stop or wait modes.

NBFS — Normalization Bit Format Select (Bit 4)

This bit controls the format of the Normalization Bit (NB). SAE J1850 strongly encourages the use of an active long: ‘0’ for In-Frame Responses containing CRC and active short, ‘1’ for In-Frame Responses without CRC.

1 = NB that is received or transmitted is a ‘0’ when the response part of an In-Frame Response (IFR) ends with a CRC byte. NB that is received or transmitted is a ‘1’ when the response part of an In-Frame Response (IFR) does not end with a CRC byte.

0 = NB that is received or transmitted is a ‘1’ when the response part of an In-Frame Response (IFR) ends with a CRC byte. NB that is received or transmitted is a ‘0’ when the response part of an In-Frame Response (IFR) does not end with a CRC byte.

TEOD — Transmit End of Data (Bit 3)

This bit is set by the programmer to indicate the end of a message being sent by the BDLC. It will append an 8-bit CRC after completing transmission of the current byte in the Tx Shift Register followed by the EOD symbol. If the transmit shadow register (refer to Rx & Tx Shadow Registers for a description of the transmit shadow register) is full when TEOD is set, the CRC byte and EOD will be transmitted after the current byte in the Tx Shift Register and the byte in the Tx Shadow Register have been transmitted. Once TEOD is set, the transmit data register empty flag (TDRE) in the BDLC

State Vector Register (DLCBSVR) is cleared to allow lower priority interrupts to occur. This bit is also used to end an IFR. Bits TSIFR, TMIFR1, and TMIFR0 determine whether a CRC byte is appended before EOD transmission for IFRs.

1 = Transmit EOD symbol.

0 = The TEOD bit will be automatically cleared after the first CRC bit is sent, or if an error or loss of arbitration is detected on the bus. When TEOD is used to end an IFR transmission, TEOD is cleared when the BDLC receives back a valid EOD symbol, or an error condition or loss of arbitration occurs.

TSIFR, TMIFR1, TMIFR0 — Transmit In-Frame Response Control (Bits 2-0)

These three bits control the type of In-Frame Response being sent. The programmer should not set more than one of these control bits to a one at any given time. However, if more than one of these three control bits are set to one, the priority encoding logic will force the internal register bits to a known value as shown in the following table. But, when these bits are read, they will be the same as written earlier. For instance, if “011” is written to TSIFR, TMIFR1, TMIFR0, then internally, they’ll be encoded as “010”. However, when these bits are later read back, it’ll still be “011”.

Table 3-3 Transmit In-Frame Response Control Bit Priority Encoding

WRITE			READ			ACTUAL (internal register)		
TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0
0	0	0	0	0	0	0	0	0
1	X	X	1	X	X	1	0	0
0	1	X	0	1	X	0	1	0
0	0	1	0	0	1	0	0	1

The BDLC supports the In-frame Response (IFR) feature of J1850 by setting these bits correctly. The four types of J1850 IFR are shown in Figure 3-4. The purpose of the in-frame response modes is to allow single or multiple nodes to acknowledge receipt of the data by responding to a received message after they have seen the EOD symbol. For VPW modulation, the first bit of the IFR is always passive; therefore, an active normalization bit must be generated by the responder and sent prior to its ID/address byte. When there are multiple responders on the J1850 bus, only one normalization bit is sent which assists all other transmitting nodes to sync their responses.

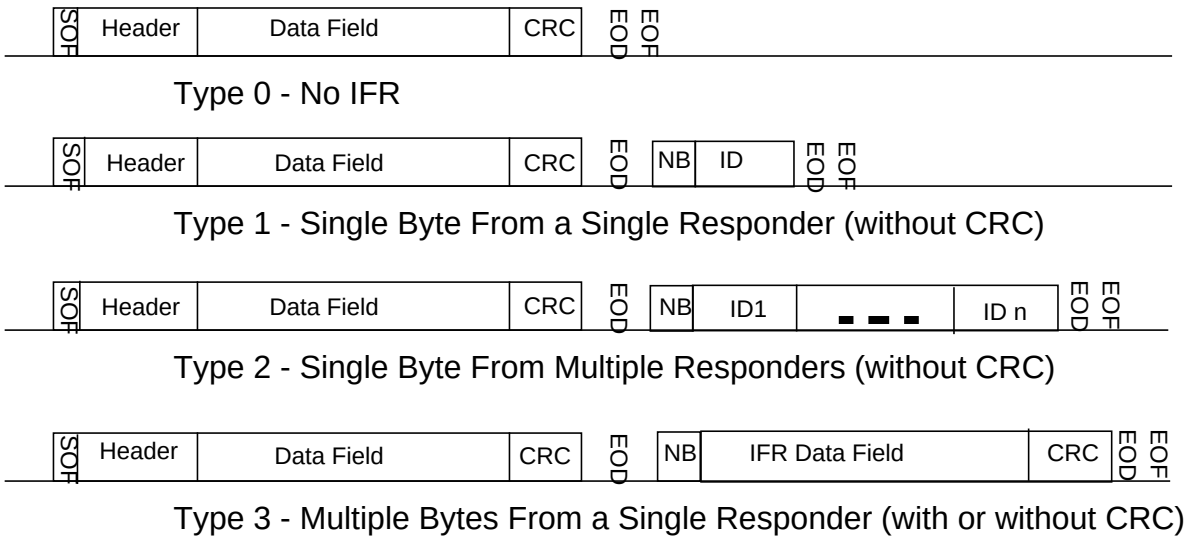


Figure 3-4 Types of In-Frame Response

TSIFR — Transmit Single Byte IFR with no CRC (Type 1 or 2)

This bit is used to request the BDLC to transmit the byte in the BDLC Data Register (DLCBDR) as a single byte IFR with no CRC. Typically, the byte transmitted is a unique identifier or address of the transmitting (responding) node.

1 = If this bit is set prior to a valid EOD being received with no CRC error, once the EOD symbol has been received the BDLC module will attempt to transmit the appropriate normalization bit followed by the byte in the DLCBDR.

0 = The TSIFR bit will be automatically cleared once the EOD following one or more IFR bytes has been received or an error is detected on the bus.

The user must set the TSIFR bit before the EOF following the main part of the message frame is received, or no IFR transmit attempts will be made for the current message. If another node transmits an IFR to this message, the user must set the TSIFR bit before the normalization bit is received or no IFR transmit attempts will be made for the message. If another node does transmit a successful IFR or a reception error occurs, the TSIFR bit will be cleared. If not, the IFR will be transmitted after the EOD of the next received message.

If a loss of arbitration occurs when the BDLC module attempts transmission, after the IFR byte winning arbitration completes transmission, the BDLC module will again attempt to transmit the byte in the DLCBDR (with no normalization bit). The BDLC module will continue transmission attempts until an error is detected on the bus, or TEOD is set by the CPU, or the BDLC transmission is successful.

NOTE: *Setting the TEOD bit before transmission of the IFR byte will direct the BDLC to make only one attempt at transmitting the byte.*

If loss of arbitration occurs in the last bit of the IFR byte, two additional ‘1’ bits **will not** be sent out because the BDLC will attempt to retransmit the byte in the transmit shift register after the IFR byte winning arbitration completes transmission.

TMIFR1 — Transmit Multiple Byte IFR with CRC (Type 3)

This bit requests the BDLC module to transmit the byte in the BDLC Data Register (DLCBDR) as the first byte of a multiple byte IFR with CRC or as a single byte IFR with CRC.Response IFR bytes are still subject to J1850 message length maximums.

1 = If this bit is set prior to a valid EOD being received with no CRC error, once the EOD symbol has been received, the BDLC module will attempt to transmit the appropriate normalization bit followed by IFR bytes.The programmer should set TEOD after the last IFR byte has been written into DLCBDR register. After TEOD has been set and the last IFR byte has been transmitted, the CRC byte is transmitted.

0 = The TMIFR1 bit will be automatically cleared once the BDLC module has successfully transmitted the CRC byte and EOD symbol, by the detection of an error on the multiplex bus, a transmitter underrun, or loss of arbitration.

After the byte in the DLCBDR has been loaded into the transmit shift register, the TDRE flag will be set in the DLCBSVR register, similar to the main message transmit sequence. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated.The programmer should then load the next byte of the IFR into the DLCBDR for transmission. When the last byte of the IFR has been loaded into the DLCBDR, the programmer should set the TEOD bit in the BDLC control register 2 (DLCBCR2). This will instruct the BDLC module to transmit a CRC byte once the byte in the DLCBDR is transmitted, and then transmit an EOD symbol, indicating the end of the IFR portion of the message frame.

However, if the programmer wishes to transmit a single byte followed by a CRC byte, the programmer should load the byte into the DLCBDR and then set the TMIFR1 bit before the EOD symbol has been received. Once the TDRE flag is set and interrupt occurs (if enabled), the programmer should then set the TEOD bit in DLCBCR2. This will result in the byte in the DLCBDR being the only byte transmitted before the IFR CRC byte.

The user must set the TMIFR1 bit before the EOF following the main part of the message frame is received, or no IFR transmit attempts will be made for the current message. If another node transmits an IFR to this message, the user must set the TMIFR1 bit before the normalization bit is received or no IFR transmit attempts will be made for the message. If another node does transmit a successful IFR or a reception error occurs, the TMIFR1 bit will be cleared. If not, the IFR will be transmitted after the EOD of the next received message.

If a transmitter underrun error occurs during transmission (caused by the programmer not writing another byte to the DLCBDR following the TDRE flag being set) the BDLC module will automatically disable the transmitter after the byte currently in the shifter plus two extra 1-bits have been transmitted. The receiver will pick this up as an framing error and relay it in the State Vector Register as an invalid symbol error. The TMIFR1 bit will also be cleared.

If a loss of arbitration occurs when the BDLC module is transmitting a multiple byte IFR with CRC, the BDLC module will go to the loss of arbitration state, set the appropriate flag and cease transmission. The TMIFR1 bit will be cleared and no attempt will be made to retransmit the byte in the DLCBDR. If loss of arbitration occurs in the last bit of the IFR byte, two additional one bits (a passive long followed by an active short) **will** be sent out.

NOTE: *The extra logic 1s are an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise on the J1850 bus from corrupting a message.*

TMIFR0 — Transmit Multiple Byte IFR with no CRC (Type 3)

This bit is used to request the BDLC module to transmit the byte in the BDLC Data Register (DLCBDR) as the first byte of a multiple byte IFR without CRC. Response IFR bytes are still subject to J1850 message length maximums.

1 = If this bit is set prior to a valid EOD being received with no CRC error, once the EOD symbol has been received the BDLC module will attempt to transmit the appropriate normalization bit followed by IFR bytes. The programmer should set TEOD after the last IFR byte has been written into DLCBDR register. After TEOD has been set, the last IFR byte to be transmitted will be the last byte which was written into the DLCBDR register.

0 = The TMIFR0 bit will be automatically cleared once the BDLC module has successfully transmitted the EOD symbol, by the detection of an error on the multiplex bus, a transmitter underrun, or loss of arbitration.

After the byte in the DLCBDR has been loaded into the transmit shift register, the TDRE flag will be set in the DLCBSVR register, similar to the main message transmit sequence. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. The programmer should then load the next byte of the IFR into the DLCBDR for transmission. When the last byte of the IFR has been loaded into the DLCBDR, the programmer should set the TEOD bit in the DLCBCR2 register. This will instruct the BDLC to transmit an EOD symbol, indicating the end of the IFR portion of the message frame. The BDLC module will not append a CRC.

However, if the programmer wishes to transmit a single byte, the programmer should load the byte into the DLCBDR and then set the TMIFR0 bit before the EOD symbol has been received. Once the TDRE flag is set and interrupt occurs (if enabled), the programmer should then set the TEOD bit in DLCBCR2. This will result in the byte in the DLCBDR being the only byte transmitted.

The user must set the TMIFR0 bit before the EOF following the main part of the message frame is received, or no IFR transmit attempts will be made for the current message. If another node transmits an IFR to this message, the user must set the TMIFR0 bit before the normalization bit is received or no IFR transmit attempts will be made for the message. If another node does transmit a successful IFR or a reception error occurs, the TMIFR0 bit will be cleared. If not, the IFR will be transmitted after the EOD of the next received message.

If a transmitter underrun error occurs during transmission (caused by the programmer not writing another byte to the DLCBDR following the TDRE flag being set) the BDLC module will automatically disable the transmitter after the byte currently in the shifter plus two extra 1-bits have been transmitted. The receiver will pick this up as an framing error and relay it in the State Vector Register as an invalid symbol error. The TMIFR0 bit will also be cleared.

If a loss of arbitration occurs when the BDLC module is transmitting a multiple byte IFR without CRC, the BDLC module will go to the loss of arbitration state, set the appropriate flag and cease transmission. The TMIFR0 bit will be cleared and no attempt will be made to retransmit the byte in the DLCBDR. If loss of arbitration occurs in the last bit of the IFR byte, two additional one bits (a passive long followed by an active short) **will** be sent out.

NOTE: *The extra logic 1s are an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise on the J1850 bus from corrupting a message.*

3.3.4 BDLC Data Register (DLCBDR)

This register is used to pass the data to be transmitted to the J1850 bus from the CPU to the BDLC module. It is also used to pass data received from the J1850 bus to the CPU.

Register Offset: $_03

	7	6	5	4	3	2	1	0
R W	D7	D6	D5	D4	D3	D2	D1	D0
RESET:	0	0	0	0	0	0	0	0

Figure 3-5 BDLC Data Register

READ: any time

WRITE: any time

D7:D0 — Receive/Transmit Data (Bits 7 - 0)

While transmitting, each data byte (after the first one) should be written only after a “Tx Data Register Empty” (TDRE) interrupt has occurred, or the DLCBSVR register has been polled indicating this condition.

Data read from this register will be the last data byte received from the J1850 bus. This received data should only be read after a “Rx Data Register Full” (RDRF) or “Received IFR byte” (RXIFR) interrupt has occurred or the DLCBSVR register has been polled indicating either of these two conditions.

The DLCBDR register is double buffered via a transmit shadow register and a receive shadow register. After the byte in the transmit shift register has been transmitted, the byte currently stored in the transmit shadow register is loaded into the transmit shift register. Once the transmit shift register has shifted the first bit out, the TDRE flag is set, and the shadow register is ready to accept the next byte of data.

The receive shadow register works similarly. Once a complete byte has been received, the receive shift register stores the newly received byte into the receive shadow register. The RDRF flag (or RXIFR flag if the received byte is part of an IFR) is set to indicate that a new byte of data has been received. The programmer has one BDLC module byte reception time to read the shadow register and clear the RDRF or RXIFR flag before the shadow register is overwritten by the newly received byte.

If the user writes the first byte of a message to be transmitted to the DLCBDR and then determines that a different message should be transmitted, the user can write a new byte to the DLCBDR up until the transmission begins.This new byte will replace the original byte in the DLCBDR.

From the time a byte is written to the DLCBDR until it is transferred to the transmit shift register, the transmit shadow register is considered full and the byte pending transmission. If one of the IFR transmission control bits (TSIFR, TMIFR1, or TMIFR0 in DLCBCR2) is also set, the byte is pending transmission as an IFR. A byte pending transmission will be flushed from the transmit shadow register and the transmission canceled if one of the following occurs: a loss of arbitration or transmitter error on the byte currently being transmitted; a symbol error, framing error, bus fault, or BREAK symbol is received. If the byte pending transmission is an IFR byte, the reception of a message with a CRC error will also cause the byte in the transmit shadow register to be flushed.

To abort an in-progress transmission, the programmer should simply stop loading more data into the BDR. This will cause a transmitter underrun error and the BDLC module will automatically disable the transmitter on the next non-byte boundary. This means that the earliest a transmission can be halted is after at least one byte (plus two extra 1-bits) has been transmitted. The receiver will pick this up as an error and relay it in the State Vector Register as an invalid symbol error.

3.3.5 BDLC Analog Round Trip Delay Register (DLCBARD)

This register is used to program the BDLC module so that it compensates for the round trip delays of different external transceivers. Also the polarity of the receive pin (RXB) is set in this register.

Register Offset: $_04

	7	6	5	4	3	2	1	0
R	0	RXPOL	0	0	BO3	BO2	BO1	BO0
W								
RESET:	0	1	0	0	0	1	1	1

= Unimplemented

Figure 3-6 BDLC Analog Round Trip Delay Register

READ: any time

WRITE: write once in normal and emulation modes.

Register functionality modified in special test mode.

Writes to unimplemented bits 7, 5, 4 are ignored.

RXPOL — Receive Pin Polarity (Bit 6)

The Receive pin Polarity bit is used to select the polarity of incoming signal on the receive pin. Some external analog transceiver inverts the receive signal from the J1850 bus before feeding back to the digital receive pin.

1 = Select normal/true polarity; true non-inverted signal from J1850 bus, i.e., the external transceiver does not invert the receive signal.
0 = Select inverted polarity, where external transceiver inverts the receive signal.

BO3-BO0 — BDLC Analog Roundtrip Delay Offset Field (Bits 3-0)

BO[3:0] adjust the transmitted symbol timings to account for the differing roundtrip delays found in different SAE J1850 analog transceivers. The allowable delay range is from 9 ms to 24 ms, with a nominal target of 16 ms (reset value). Refer to Table 3-4 for the BO[3:0] values corresponding to the expected transceiver delays and the resultant transmitter timing adjustment (in mux interface clock periods (t_{bdlc})). Refer to the analog transceiver device specification for the expected roundtrip delay through both the transmitter and the receiver. The sum of these two delays makes up the total roundtrip delay value.

Table 3-4 BARD Values vs. Transceiver Delay and Transmitter Timing Adjustment

BARD Offset Bits (BO3,BO2,BO1,BO0)	Corresponding Expected Transceiver's delays (µs)	Transmitter Symbol Timing Adjustment (t_{bdlc})[1]
0000	9	9
0001	10	10
0010	11	11
0011	12	12
0100	13	13
0101	14	14
0110	15	15
0111	16	16
1000	17	17
1001	18	18
1010	19	19
1011	20	20
1100	21	21
1101	22	22
1110	23	23
1111	24	24

NOTE:
1. The transmitter symbol timing adjustment is the same for binary and integer bus frequencies.

3.3.6 BDLC Rate Select Register (DLCBRSR)

This register determines the divider prescaler value for the mux interface clock (f_{bdlc}).

Register Offset: $_05

	7	6	5	4	3	2	1	0
R	0	0	R5	R4	R3	R2	R1	R0
W	Unimplemented							
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-7 BDLC Rate Select Register

READ: any time

WRITE: write once in normal and emulation modes.

Register functionality modified in special test mode.

Writes to unimplemented bits 7, 6 are ignored.

NOTE: *After writing to the divide rate register, the divide counter will start counting ONLY after the next access to the BDLC register space. E.g. write the module enable bit after writing to the divide rate register.*

R5-R0 — Rate Select (Bits 5-0)

These bits determine the amount by which the frequency of the system clock signal is divided to generate the MUX Interface clock (f_{bdlc}) which defines the basic timing resolution of the MUX Interface. The value programmed into these bits is dependent on the chosen system clock frequency. See Table 3-5 and Table 3-6 for example rate selects for different bus frequencies. All divisor values from divide by 1 to divide by 64 are possible, but are not shown in the tables.

NOTE: *Although the maximum divider is 64, a divider which will generate a 1 MHz or 1.048576 MHz f_{bdlc} must be selected in order for J1850 communications to occur.*

Table 3-5 BDLC Rate Selection for Binary Frequencies [CLKS = 1]

IP bus clock frequency	R[5:0]	division	f_{bdlc}
f_{CLOCK}=1.048576 MHz	$00	1	1.048576 MHz
f_{CLOCK}=2.09715 MHz	$01	2	1.048576 MHz
f_{CLOCK}=3.14573 MHz	$02	3	1.048576 MHz
f_{CLOCK}=4.19430 MHz	$03	4	1.048576 MHz
f_{CLOCK}=8.38861 MHz	$07	8	1.048576 MHz
f_{CLOCK}=10.48576 MHz	$09	10	1.048576 MHz
f_{CLOCK}=67.10886 MHz	$3F	64	1.048576 MHz

Table 3-6 BDLC Rate Selection for Integer Frequencies [CLKS = 0]

IP bus clock frequency	R[5:0]	division	f_{bdlc}
f_{CLOCK}=1.00000 MHz	$00	1	1.000000 MHz
f_{CLOCK}=2.00000 MHz	$01	2	1.000000 MHz
f_{CLOCK}=3.00000 MHz	$02	3	1.000000 MHz
f_{CLOCK}=4.00000 MHz	$03	4	1.000000 MHz
f_{CLOCK}=8.00000 MHz	$07	8	1.000000 MHz
f_{CLOCK}=10.00000 MHz	$09	10	1.000000 MHz
f_{CLOCK}=64.00000 MHz	$3F	64	1.000000 MHz

3.3.7 BDLC Control Register (DLCSCR)

The following register enables the BLDC module.

Register Offset: $_06

	7	6	5	4	3	2	1	0
R	0	0	0	BDLCE	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-8 BDLC Control Register

READ: any time

WRITE: any time

BDLCE — BDLC Enable (Bit 4)

This bit serves as a mux interface clock (f_{bdlc}) enable/disable for power savings.

1 = The mux interface clock (f_{bdlc}) and BDLC module are enabled to allow J1850 communications to take place.

0 = The mux interface clock (f_{bdlc}) is disabled, shutting down the BDLC module for power saving. Bus clocks are still running allowing registers to be accessed.

3.3.8 BDLC Status Register (DLCBSTAT)

This register Indicates the status of the BLDC module.

Register Offset: $_07

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	IDLE
W	Unimplemented					Reserved	Unimplemented	
RESET:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-9 BDLC Status Register

READ: any time

WRITE: ignored in normal and emulation modes

Register functionality is modified in special test mode.

IDLE Idle (Bit 0)

This bit indicates when the BDLC module is idle.

1 = BDLC module has received IFS and no data is being transmitted or received.

0 = BDLC module is either transmitting or receiving data.

NOTE: *BDLC module is only idle after receiving IFS. The IDLE bit is 0 during reset since the BDLC module needs to wait for an IFS before becoming idle. Noise on the bus will be filtered and the IDLE bit will remain unchanged.*

Section 4 Functional Description

4.1 General

The BDLC module is a serial communication module which allows the user to send and receive messages across a Society of Automotive Engineers (SAE) J1850 serial communication network. The user's software handles each transmitted or received message on a byte-by-byte basis, while the BDLC performs all of the network access, arbitration, message framing and error detection duties.

4.1.1 J1850 Frame Format

As noted above and in **Section 1.2 Features on page 11**, the BDLC module communicates across an SAE J1850 network. As such, all messages transmitted on the J1850 bus are structured using the format below. The following sections describe this format and it's meanings.

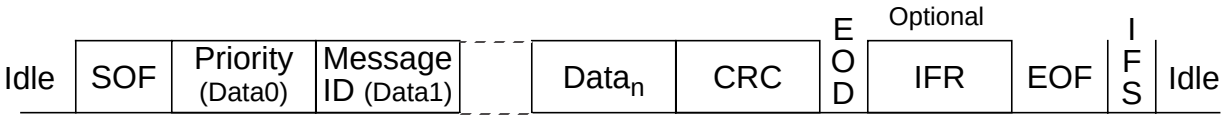


Figure 4-1 J1850 Bus Message Format (VPW)

SAE J1850 states that each message has a maximum length of 101 bit times or 12 bytes (excluding SOF, EOD, NB and EOF).

- SOF - Start of Frame Symbol

  All messages transmitted onto the J1850 bus must begin with an long active SOF symbol. This indicates to any listeners on the J1850 bus the start of a new message transmission. The SOF symbol is not used in the CRC calculation.

- Data - In Message Data Bytes

  The data bytes contained in the message include the message priority/type, message I.D. byte, and any actual data being transmitted to the receiving node. See SAE J1850 - Class B Data Communications Network Interface, for more information about 1 and 3 Byte Headers.

  Messages transmitted by the BDLC module onto the J1850 bus must contain at least one data byte, and therefore can be as short as one data byte and one CRC byte. Each data byte in the message is 8 bits in length, transmitted MSB to LSB.

- CRC - Cyclical Redundancy Check Byte

This byte is used by the receiver(s) of each message to determine if any errors have occurred during the transmission of the message. The BDLC calculates the CRC byte and appends it onto any messages transmitted onto the J1850 bus, and also performs CRC detection on any messages it receives from the J1850 bus.

CRC generation uses the divisor polynomial $X^8+X^4+X^3+X^2+1$. The remainder polynomial is initially set to all ones, and then each byte in the message after the SOF symbol is serially processed through the CRC generation circuitry. The one's complement of the remainder then becomes the 8-bit CRC byte, which is appended to the message after the data bytes, in MSB to LSB order.

When receiving a message, the BDLC uses the same divisor polynomial. All data bytes, excluding the SOF and EOD symbols, but including the CRC byte, are used to check the CRC. If the message is error free, the remainder polynomial will equal $X^7+X^6+X^2$ ($C4), regardless of the data contained in the message. If the calculated CRC does not equal $C4, the BDLC will recognize this as a CRC error and set the CRC error flag in the BSVR register.

- EOD - End of Data Symbol

  The EOD symbol is a long passive period on the J1850 bus used to signify to any recipients of a message that the transmission by the originator has completed. No flag is set upon reception of the EOD symbol.

- IFR - In Frame Response Bytes

  The IFR section of the J1850 message format is optional. Users desiring further definition of in-frame response should review the "SAE J1850 Class B Data Communications Network Interface" specification.

- EOF - End of Frame Symbol

  This symbol is a passive period on the J1850 bus, longer than an EOD symbol, which signifies the end of a message. Since an EOF symbol is longer than an EOD symbol, if no response is transmitted after an EOD symbol, it becomes an EOF, and the message is assumed to be completed. The EOF flag is set upon receiving the EOF symbol.

- IFS - Inter-Frame Separation Symbol

  The IFS symbol is a passive period on the J1850 bus which allows proper synchronization between nodes during continuous message transmission. The IFS symbol is transmitted by a node following the completion of the EOF period.

  When the last byte of a message has been transmitted onto the J1850 bus, and the EOF symbol time has expired, all nodes must then wait for the IFS symbol time to expire before transmitting an SOF, marking the beginning of another message.

  However, if the BDLC module is waiting for the IFS period to expire before beginning a transmission and a rising edge is detected before the IFS time has expired, it will internally synchronize to that edge.

  A rising edge may occur during the IFS period because of varying clock tolerances and loading of the J1850 bus, causing different nodes to observe the completion of the IFS period at different times. Receivers must synchronize to any SOF occurring during an IFS period to allow for individual clock tolerances.

- Break

  If the BDLC module is transmitting at the time a BREAK is detected, it treats the BREAK as if a transmission error had occurred, and halts transmission.The BDLC module cannot transmit a BREAK symbol. If while receiving a message the BDLC module detects a BREAK symbol, it treats the BREAK as a reception error and sets the invalid symbol flag. If while receiving a message in 4X mode, the BDLC module detects a BREAK symbol, it treats the BREAK as a reception error, sets BSVR register to $1C, and exits 4X mode.The RX4XE bit in BCR2 is automatically cleared upon reception of the BREAK symbol.

- Idle Bus

  An idle condition exists on the bus during any passive period after expiration of the IFS period. Any node sensing an idle bus condition can begin transmission immediately.

4.1.2 J1850 VPW Symbols

Variable Pulse Width modulation (VPW) is an encoding technique in which each bit is defined by the time between successive transitions, and by the level of the bus between transitions, active or passive. Active and passive bits are used alternately. This encoding technique is used to reduced the number of bus transitions for a given bit rate. See **Section 1.2 Features on page 11.**

The symbol values shown below are nominal values. Refer to the electrical specification for a more complete description of symbol values. Each logic one or logic zero contains a single transition, and can be at either the active or passive level and one of two lengths, either 64µs or 128µs (T_{NOM} at 10.4kbps baud rate), depending upon the encoding of the previous bit. The SOF, EOD, EOF and IFS symbols will always be encoded at an assigned level and length. See Figure 4-2.

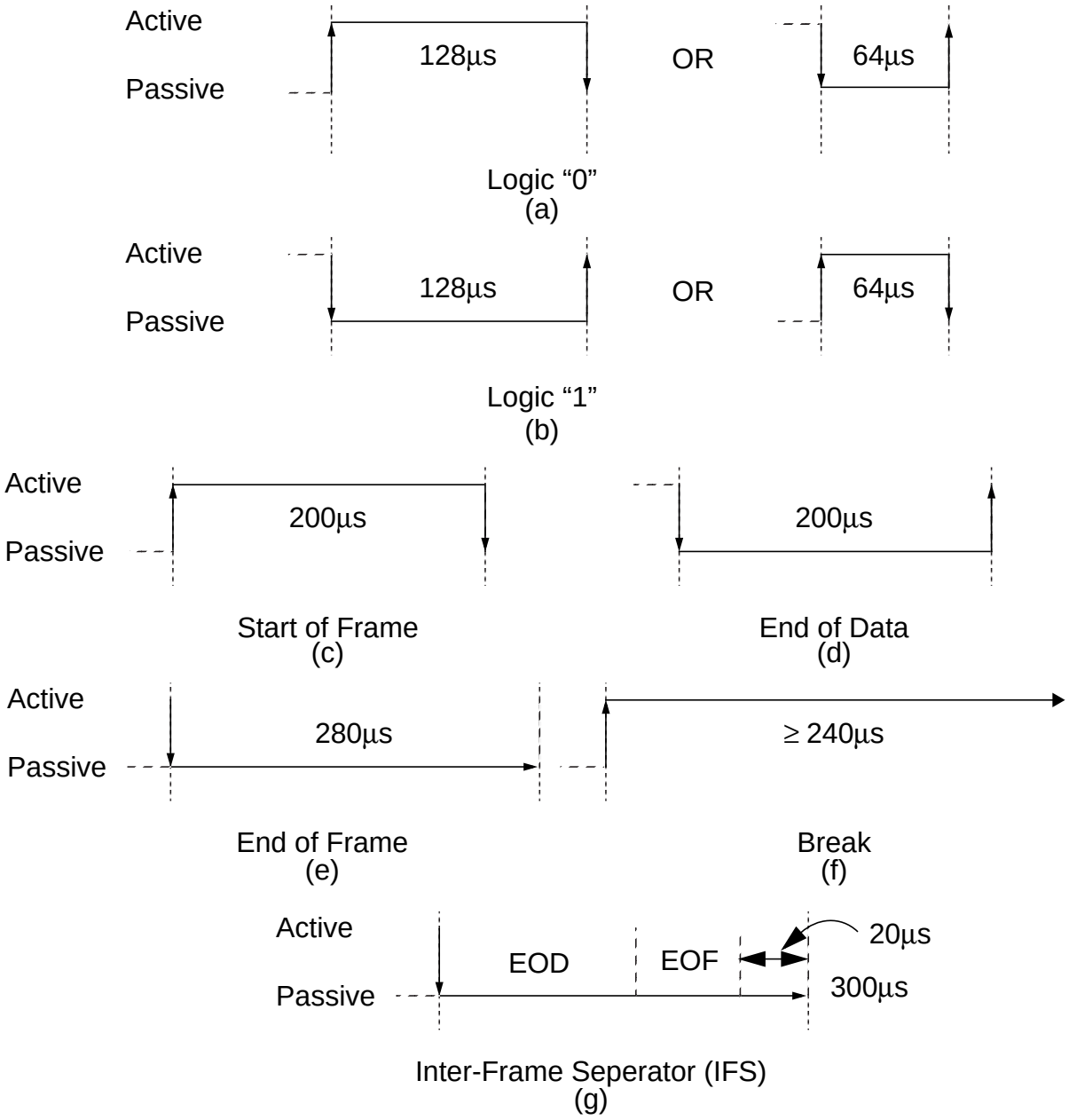


Figure 4-2 J1850 VPW Symbols

Each message will begin with an SOF symbol, an active symbol, and therefore each data byte (including the CRC byte) will begin with a passive bit, regardless of whether it is a logic one or a logic zero. All VPW bit lengths stated in the following descriptions are typical values at a 10.4kbps bit rate.

- Logic “0”

A logic zero is defined as either an active to passive transition followed by a passive period 64µs in length, or a passive to active transition followed by an active period 128µs in length (**Figure 4-2**(a)).

- Logic “1”

  A logic one is defined as either an active to passive transition followed by a passive period 128µs in length, or a passive to active transition followed by an active period 64µs in length (**Figure 4-2**(b)).

- NB - Normalization Bit

  The NB symbol has the same property as a logic “1” or a logic “0”.It is only used in IFR message responses. This bit is defined as an active bit.

- SOF - Start of Frame Symbol

  The SOF symbol is defined as passive to active transition followed by an active period 200µs in length (**Figure 4-2**(c)). This allows the data bytes which follow the SOF symbol to begin with a passive bit, regardless of whether it is a logic one or a logic zero.

- EOD - End of Data Symbol

  The EOD symbol is defined as an active to passive transition followed by a passive period 200µs in length (**Figure 4-2**(d)).

- EOF - End of Frame Symbol

  The EOF symbol is defined as an active to passive transition followed by a passive period 280µs in length (**Figure 4-2**(e)). If there is no IFR byte transmitted after an EOD symbol is transmitted, after another 80µs the EOD becomes an EOF, indicating the completion of the message.

- IFS - Inter-Frame Separation Symbol

  The IFS symbol is defined as a passive period 300µs in length. The IFS symbol contains no transition, since when used it always follows an EOF symbol.(**Figure 4-2**(g))

- BREAK - Break Signal

  The BREAK signal is defined as a passive to active transition followed by an active period of at least 240µs (**Figure 4-2**(f)).

- IDLE

  An IDLE is defined as a passive period greater than 300µs in length.

4.1.3 J1850 VPW Valid/Invalid Bits & Symbols

The timing tolerances for receiving data bits and symbols from the J1850 bus have been defined to allow for variations in oscillator frequencies. In many cases the maximum time allowed to define a data bit or symbol is equal to the minimum time allowed to define another data bit or symbol.

Since the minimum resolution of the BDLC module for determining what symbol is being received is equal to a single period of the MUX Interface clock, (t_{bdlc}). i.e. the receiver symbol timing boundaries are subject to an uncertainty of 1 t_{bdlc} due to sampling considerations.

This clock resolution of 1 t_{bdlc} allows the BDLC module to properly differentiate between the different bits and symbols, without reducing the valid window for receiving bits and symbols from transmitters onto the J1850 bus having varying oscillator frequencies.

- Transmit and Receive Symbol Timing Specifications

  Tables 4-1 through 4-6 contain the SAE J1850 transmit and receive symbol timing specifications for the BDLC module. The units used in these tables are mux interface clock periods (t_{bdlc}). The mux interface clock is a divided down version of the bus clock input to the module (see Section 3.3.6 BDLC Rate Select Register (DLCBRSR)). The mux interface clock drives the transmit and receive counters which control symbol generation and identification. The symbol timing in effect during J1850 operations is dependent the state of two control bits: the CLKS bit DLCBCR1, which indicates whether the bus clock is an integer frequency or a binary frequency; the RX4XE bit in DLCBCR2, which is used to select 4X receiver operation.

Tables 4-1 and 4-3 indicate the transmit and receive timing for integer bus frequencies (CLKS = 0) and 4X receive operation disabled (RX4XE = 0). It is assumed that for integer bus frequencies the divided down mux interface clock frequency will be 1MHz (t_{bdlc} = 1 ms).

Tables 4-2 and 4-4 indicated the transmit and receive timing for binary bus frequencies (CLKS = 1) and 4X receive operation disabled (RX4XE = 0). It is assumed that for binary bus frequencies the divided down mux interface clock frequency will be 1.048576 MHz (t_{bdlc} = 0.953674 ms). The symbol timing values are adjusted to compensate for the shortening of the mux interface clock period.

Tables 4-5 and 4-6 show how the receive symbol timing values are adjusted when 4X receive operation is enabled (RX4XE = 1) for both integer bus frequencies (CLKS = 0) and binary bus frequencies (CLKS = 1), respectively.

The values specified in the tables are for the symbols appearing on the SAE J1850 bus. These values assume the BDLC module is communicating on the SAE J1850 bus using an external analog transceiver, and that the BDLC module analog roundtrip delay value programed into the DLCBARD register is the appropriate value for the transceiver being used. If these conditions are not met, the symbol timings being measured on the SAE J1850 bus will be significantly affected. For a detailed description of how symbol timings are measured on the SAE J1850 bus, refer to the appropriate SAE documents.

Table 4-1 BDLC Transmitter VPW Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{tvp1}	62	64	66	t_{bdlc}
2	Passive Logic 1	T_{tvp2}	126	128	130	t_{bdlc}
3	Active Logic 0	T_{tva1}	126	128	130	t_{bdlc}
4	Active Logic 1	T_{tva2}	62	64	66	t_{bdlc}
5	Start of Frame (SOF)	T_{tva3}	198	200	202	t_{bdlc}
6	End of Data (EOD)[1]	T_{tvp3}	162	164	166	t_{bdlc}
7	End of Frame (EOF)[1]	T_{tv4}	238	240	242	t_{bdlc}
8	Inter-Frame Separator (IFS)[1]	T_{tv5}	298	300	302	t_{bdlc}

Table 4-1 BDLC Transmitter VPW Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
NOTE: 1. The transmitter timing for this symbol depends upon the minimum detection time of the symbol by the receiver.						

Table 4-2 BDLC Transmitter VPW Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{tvp1}	65	67	69	t_{bdlc}
2	Passive Logic 1	T_{tvp2}	132	134	136	t_{bdlc}
3	Active Logic 0	T_{tva1}	132	134	136	t_{bdlc}
4	Active Logic 1	T_{tva2}	65	67	69	t_{bdlc}
5	Start of Frame (SOF)	T_{tva3}	208	210	212	t_{bdlc}
6	End of Data (EOD)[1]	T_{tvp3}	170	172	174	t_{bdlc}
7	End of Frame (EOF)[1]	T_{tv4}	250	252	254	t_{bdlc}
8	Inter-Frame Separator (IFS)[1]	T_{tv5}	313	315	317	t_{bdlc}
NOTE: 1. The transmitter timing for this symbol depends upon the minimum detection time of the symbol by the receiver.						

Table 4-3 BDLC Receiver VPW Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	32	64	95	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	96	128	163	t_{bdlc}
3	Active Logic 0	T_{rva1}	96	128	163	t_{bdlc}
4	Active Logic 1	T_{rva2}	32	64	95	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	164	200	239	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	164	200	239	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	240	280	299	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	300	---	---	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	240	---	---	t_{bdlc}

NOTE:
The receiver symbol timing boundaries are subject to an uncertainty of 1 t_{bdlc} due to sampling considerations.

Table 4-4 BDLC Receiver VPW Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	34	67	100	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	101	134	171	t_{bdlc}
3	Active Logic 0	T_{rva1}	101	134	171	t_{bdlc}
4	Active Logic 1	T_{rva2}	34	67	100	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	172	210	251	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	172	210	251	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	252	293	314	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	315	---	---	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	252	---	---	t_{bdlc}

NOTE:
The receiver symbol timing boundaries are subject to an uncertainty of 1 t_{bdlc} due to sampling considerations.

Table 4-5 BDLC Receiver VPW 4X Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	8	16	23	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	24	32	40	t_{bdlc}
3	Active Logic 0	T_{rva1}	24	32	40	t_{bdlc}
4	Active Logic 1	T_{rva2}	8	16	23	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	41	50	59	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	41	50	59	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	60	70	74	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	75	---	---	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	60	---	---	t_{bdlc}

NOTE:
The receiver symbol timing boundaries are subject to an uncertainty of 1 t_{bdlc} due to sampling considerations.

Table 4-6 BDLC Receiver VPW 4X Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	9	17	25	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	26	34	42	t_{bdlc}
3	Active Logic 0	T_{rva1}	26	34	42	t_{bdlc}

Table 4-6 BDLC Receiver VPW 4X Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
4	Active Logic 1	T_{rva2}	9	17	25	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	43	53	62	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	43	53	62	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	63	74	78	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	79	---	---	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	63	---	---	t_{bdlc}

NOTE:
The receiver symbol timing boundaries are subject to an uncertainty of 1 t_{bdlc} due to sampling considerations.

The min and max symbol limits shown in the following sections (Invalid Passive Bit - Valid BREAK Symbol) and figures (Figure 4-3 - Figure 4-6) refer to the values listed in Tables 4-1 through 4-6.

- Invalid Passive Bit

  If the passive to active transition beginning the next data bit or symbol occurs between the active to passive transition beginning the current data bit or symbol and $T_{rvp1(Min)}$, the current bit would be invalid. See **Figure 4-3**(1).

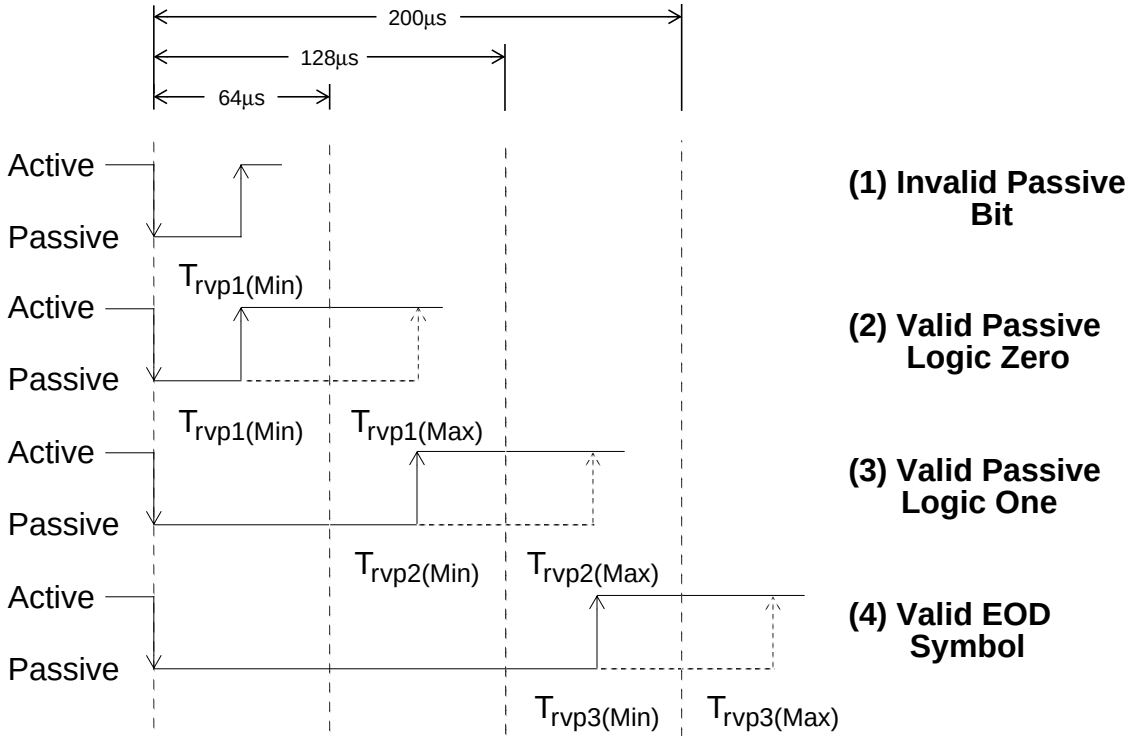


Figure 4-3 J1850 VPW Passive Symbols

- Valid Passive Logic Zero

  If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp1(Min)}$ and $T_{rvp1(Max)}$, the current bit would be considered a logic zero. See **Figure 4-3**(2).

- Valid Passive Logic One

  If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp2(Min)}$ and $T_{rvp2(Max)}$, the current bit would be considered a logic one. See **Figure 4-3**(3).

- Valid EOD Symbol

  If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp3(Min)}$ and $T_{rvp3(Max)}$, the current symbol would be considered a valid EOD symbol. See **Figure 4-3**(4).

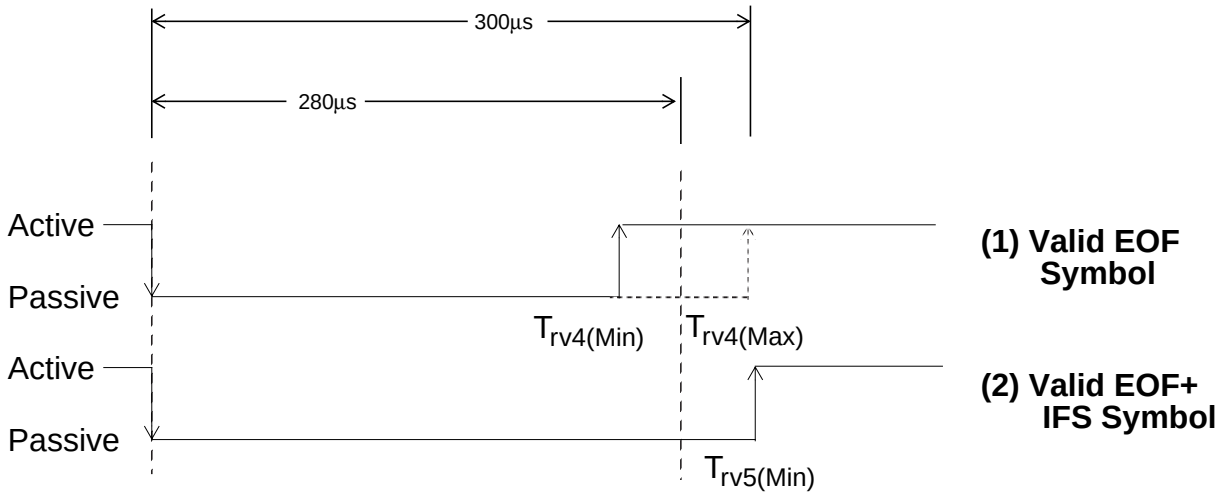


Figure 4-4 J1850 VPW EOF and IFS Symbols

- Valid EOF & IFS Symbol

  In **Figure 4-4**(1), if the passive to active transition beginning the SOF symbol of the next message occurs between $T_{rv4(Min)}$ and $T_{rv4(Max)}$, the current symbol will be considered a valid EOF symbol.

  If the passive to active transition beginning the SOF symbol of the next message occurs after $T_{rv5(Min)}$, the current symbol will be considered a valid EOF symbol followed by a valid IFS symbol. See **Figure 4-4**(2). All nodes must wait until a valid IFS symbol time has expired before beginning transmission. However, due to variations in clock frequencies and bus loading, some nodes may recognize a valid IFS symbol before others, and immediately begin transmitting. Therefore, anytime a node waiting to transmit detects a passive to active transition once a valid EOF has been detected, it should immediately begin transmission, initiating the arbitration process.

- Idle Bus

  If the passive to active transition beginning the SOF symbol of the next message does not occur before $T_{tv5(Min)}$**,** the bus is considered to be idle, and any node wishing to transmit a message may do so immediately.

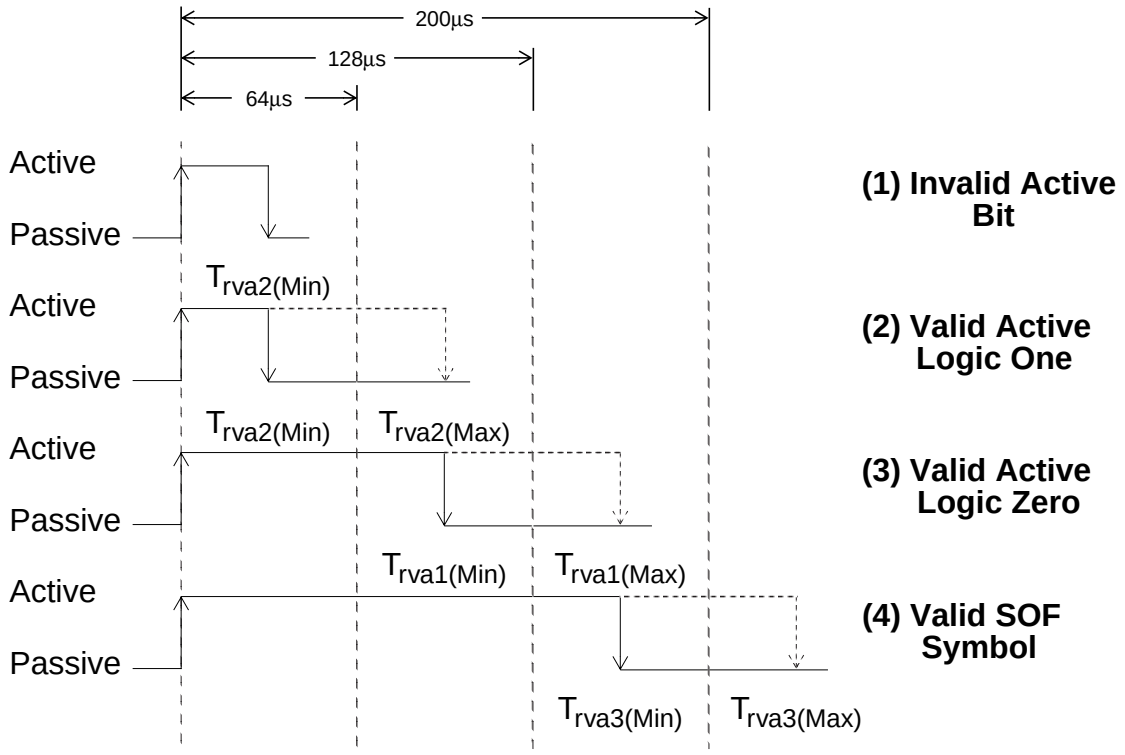


Figure 4-5 J1850 VPW Active Symbols

- Invalid Active Bit

  If the active to passive transition beginning the next data bit or symbol occurs between the passive to active transition beginning the current data bit or symbol and $T_{rva2(Min)}$, the current bit would be invalid. See **Figure 4-5**(1).

- Valid Active Logic One

  If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva2(Min)}$ and $T_{rva2(Max)}$, the current bit would be considered a logic one. See **Figure 4-5**(2).

- Valid Active Logic Zero

  If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva1(Min)}$ and $T_{rva1(Max)}$, the current bit would be considered a logic zero. See **Figure 4-5**(3).

- Valid SOF Symbol

  If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva3(Min)}$ and $T_{rva3(Max)}$, the current symbol would be considered a valid SOF symbol. See **Figure 4-5**(4).

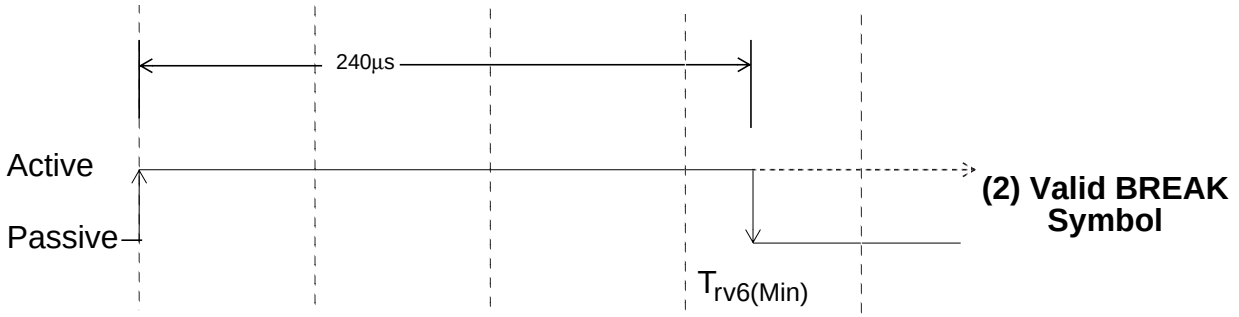


Figure 4-6 J1850 VPW BREAK Symbol

- Valid BREAK Symbol

  If the next active to passive transition does not occur until after $T_{rv6(Min)}$, the current symbol will be considered a valid BREAK symbol. A BREAK symbol should be followed by a SOF symbol beginning the next message to be transmitted onto the J1850 bus. See **Figure 4-6**.

- Message Arbitration

  Message arbitration on the J1850 bus is accomplished in a non-destructive manner, allowing the message with the highest priority to be transmitted, while any transmitters which lose arbitration simply stop transmitting and wait for an idle bus to begin transmitting again.

  If the BDLC module wishes to transmit onto the J1850 bus, but detects that another message is in progress, it automatically waits until the bus is idle. However, if multiple nodes begin to transmit in the same synchronization window, message arbitration will occur beginning with the first bit after the SOF symbol and continue with each bit thereafter.

  The VPW symbols and J1850 bus electrical characteristics are carefully chosen so that a logic zero (active or passive type) will always dominate over a logic one (active or passive type) simultaneously transmitted. Hence logic zeroes are said to be ‘dominant’ and logic ones are said to be ‘recessive’.

  Whenever a node transmits a recessive bit and detects a dominant bit, it loses arbitration, and immediately stops transmitting. This is known as ‘bitwise arbitration’.The loss of arbitration flag (in DLCBSVR) is set when arbitration is lost. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

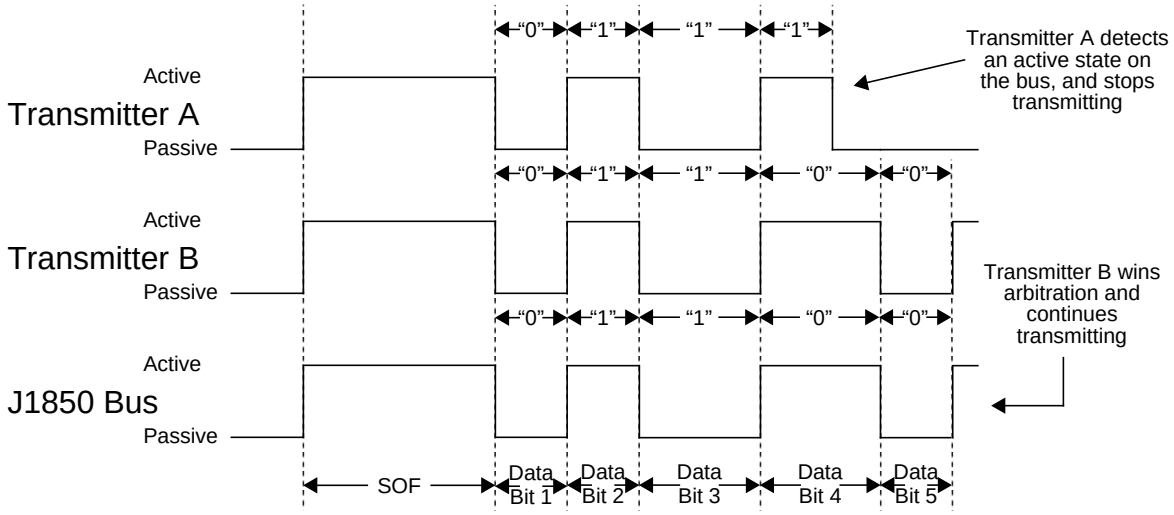


Figure 4-7 J1850 VPW Bitwise Arbitrations

During arbitration, or even throughout the transmitting message, when an opposite bit is detected, transmission is immediately stopped unless it occurs on the 8th bit of a byte. In this case the BDLC module will automatically append up to two extra 1 bits and then stop transmitting. These two extra bits will be arbitrated normally and thus will not interfere with another message. The second 1 bit will not be sent if the first loses arbitration. If the BDLC module has lost arbitration to another valid message then the two extra ones will not corrupt the current message. However, if the BDLC module has lost arbitration due to noise on the bus, then the two extra ones will ensure that the current message will be detected and ignored as a noise-corrupted message.

Since a "0" dominates a "1", the message with the lowest value will have the highest priority, and will always win arbitration, i.e. a message with priority 000 will win arbitration over a message with priority 011. This method of arbitration will work no matter how many bits of priority encoding are contained in the message.

4.1.4 J1850 Bus Errors

The BDLC module detects several types of transmit and receive errors which can occur during the transmission of a message onto the J1850 bus.

- Transmission Error

  If the BDLC module is transmitting a message and the message received contains a symbol error, a framing error, a bus fault, a BREAK symbol, or a logic '1' symbol when a logic "0" is being transmitted, this constitutes a transmission error. Receiving a logic '0' symbol when transmitting a logic '1' is considered a loss of arbitration condition (See Message Arbitration) and not a transmission error. When a transmission error is detected, the BDLC module will immediately cease transmitting. Further transmission or reception will be disabled until a valid EOF symbol is

detected on the J1850 bus. The error condition is reflected by setting the symbol invalid or out of range flag in the DLCBSVR register. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

- CRC Error

  A cyclical redundancy check (CRC) error is detected when the data bytes and CRC byte of a received message are processed, and the CRC calculation result is not equal to $C4.The CRC code should detect any single and 2 bit errors, as well as all 8 bit burst errors, and almost all other types of errors. The CRC error flag (in DLCBSVR) is set when a CRC error is detected. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

- Symbol Error

  A symbol error is detected when an abnormal (invalid) symbol is detected in a message being received from the J1850 bus. See sections Invalid Passive Bit and Invalid Active Bit which define invalid symbols.The symbol invalid or out of range flag (in DLCBSVR) is set when a symbol error is detected. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

- Framing Error

  A framing error is detected when a received symbol occurs in an inappropriate location in the message frame. The following situations result in framing errors:

  – An active logic “0” or logic “1” received as the first symbol of the frame.

  – An SOF symbol received in any location other than the first symbol of a frame. Erroneous locations include: Within the data portion of a message or IFR; Immediately following the EOD in a message or IFR.

  – An EOD symbol received on a non-byte boundary in a message or IFR.

  – An active logic “0” or logic “1” received immediately following the EOD at the end of an IFR.

  The symbol invalid or out of range flag (in DLCBSVR) is set when a framing error is detected. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

- Bus Fault

  If a bus fault occurs, the response of the BDLC module will depend upon the type of bus fault.

  If the bus is shorted to V_{DD}, the BDLC module will wait for the bus to fall to a passive state before it will attempt to transmit a message. As long as the short remains, the BDLC will never attempt to transmit a message onto the J1850 bus.

  If the bus is shorted to ground, the BDLC module will see an idle bus, begin to transmit the message, and then detect a transmission error, since the short to ground would not allow the bus to be driven to the active (dominant) state. The BDLC module will wait for assertion of the receive pin for (64 - analog round trip delay) t_{bdlc} cycles, after assertion of the transmit pin, before detecting the error. If the transmission is an IFR, the BDLC module will wait for (280 - analog round trip delay) t_{bdlc} cycles before detecting an error. The “analog round trip delay” is determined by the value stored in the DLCBARD register. The BDLC module will set the symbol invalid or out of range flag (in

DLCBSVR), abort that transmission and wait for the next CPU command to transmit. In this case, the transmitter does not have to wait for an EOF symbol to be received to be enabled. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

In any case, if the bus fault is temporary, as soon as the fault is cleared, the BDLC module will resume normal operation. If the bus fault is permanent, it may result in permanent loss of communication on the J1850 bus.

- BREAK - Break

  Any BDLC transmitting at the time a BREAK is detected will treat the BREAK as if a transmission error had occurred, and halt transmission.

  If while receiving a message the BDLC module detects a BREAK symbol, it will treat the BREAK as a reception error.

  If a BREAK symbol is received while the BDLC module is transmitting or receiving, the symbol invalid or out of range flag (in DLCBSVR) is set. Further transmission/reception will be disabled until the J1850 bus returns to the passive state and a valid EOF symbol is detected on the J1850 bus. If the interrupt enable bit (IE in DLCBCR1) is set, an interrupt request from the BDLC module is generated. Reading the DLCBSVR register will clear this flag.

  The BDLC module cannot transmit a BREAK symbol. It can only receive a BREAK symbol from the J1850 bus.

- Bus Error Summary

  The possible J1850 bus errors and the actions taken by the BDLC module are summarized in Table 4-7.

Table 4-7 BDLC module J1850 Error Summary

Error Condition	BDLC Module Function
Transmission Error	BDLC module will immediately cease transmitting. Further transmission and reception will be disabled until a valid EOF symbol is detected. The symbol invalid or out of range flag will be set and interrupt generated if enabled.
Cyclical Redundancy Check (CRC) Error	CRC error flag set and interrupt generated if enabled.
Symbol Error	The symbol invalid or out of range flag will be set and interrupt generated if enabled. Transmission and reception will be disabled until a valid EOF symbol is detected.
Framing Error	The symbol invalid or out of range flag will be set and interrupt generated if enabled. Transmission and reception will be disabled until a valid EOF symbol is detected.

Table 4-7 BDLC module J1850 Error Summary

Error Condition	BDLC Module Function
Bus short to V_{DD}.	The BDLC module will not transmit until short is corrected and a valid EOF is detected. Depending upon when short occurs and is corrected, this error condition may set the symbol invalid or out of range, crc error, or loss of arbitration flags.
Bus short to GND.	Short will be seen as an idle bus by BDLC module. If a transmission attempt is made before short is corrected, the symbol invalid or out of range flag will be set and interrupt generated if enabled. Another transmission can be initiated as soon as short is corrected.
BREAK symbol reception	If doing so, the BDLC module will immediately cease transmitting. Symbol invalid or out of range flag set and interrupt generated if enabled.Transmission and reception will be disabled until a valid EOF symbol is detected.

4.2 Mux Interface

The MUX Interface is responsible for bit encoding/decoding and digital noise filtering between the Protocol Handler and the Physical Interface. Refer to **Figure 1-2 BDLC Block Diagram on page 16**.

4.2.1 Mux Interface - Rx Digital Filter

The Receiver section of the BDLC module includes a digital low pass filter to remove narrow noise pulses from the incoming message. An outline of the digital filter is shown in Figure 4-8.

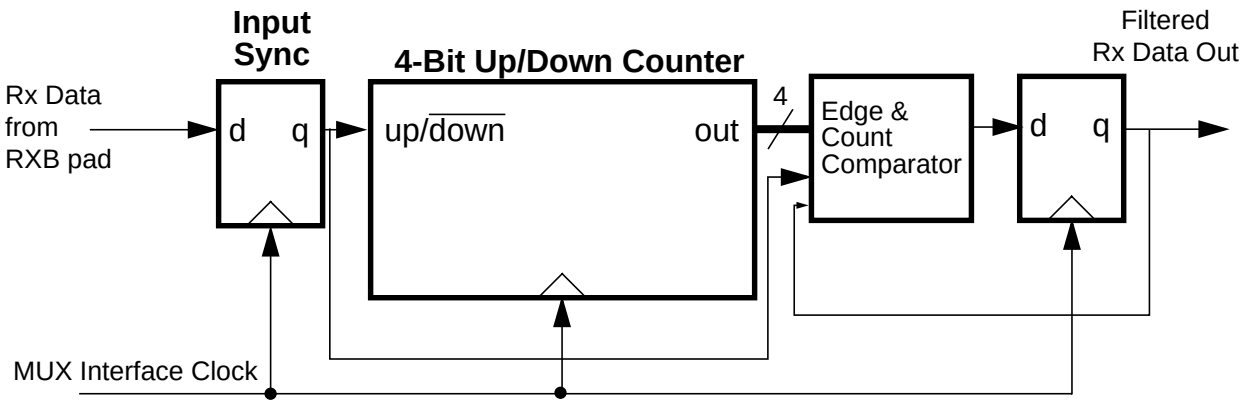


Figure 4-8 BDLC Module Rx Digital Filter Block Diagram

- Operation

  The clock for the digital filter is provided by the MUX Interface clock.At each positive edge of the clock signal, the current state of the Receiver input signal from the RXB pad is sampled.The RXB signal state is used to determine whether the counter should increment or decrement at the next positive edge of the clock signal.

  The counter will increment if the input data sample is high but decrement if the input sample is low.The counter will thus progress up towards ‘15’ if, on average, the RXB signal remains high or progress down towards ‘0’ if, on average, the RXB signal remains low.

  When the counter eventually reaches the value ‘15’, the digital filter decides that the condition of the RXB signal is at a stable logic level one and the Data Latch is set, causing the Filtered Rx Data signal to become a logic level one. Furthermore, the counter is prevented from overflowing and can only be decremented from this state.

  Alternatively, should the counter eventually reach the value ‘0’, the digital filter decides that the condition of the RXB signal is at a stable logic level zero and the Data Latch is reset, causing the Filtered Rx Data signal to become a logic level zero. Furthermore, the counter is prevented from underflowing and can only be incremented from this state.

  The Data Latch will retain its value until the counter next reaches the opposite end point, signifying a definite transition of the RXB signal.

- Performance

  The performance of the digital filter is best described in the time domain rather than the frequency domain.

If the signal on the RXB signal transitions, then there will be a delay before that transition appears at the Filtered Rx Data output signal. This delay will be between 15 and 16 clock periods, depending on where the transition occurs with respect to the sampling points. This ‘filter delay’ must be taken into account when performing message arbitration.

For example, if the frequency of the MUX Interface clock (f_{bdlc}) is 1.0486MHz, then the period (t_{bdlc}) is 954ns and the maximum filter delay in the absence of noise will be 15.259us.

The effect of random noise on the RXB signal depends on the characteristics of the noise itself. Narrow noise pulses on the RXB signal will be completely ignored if they are shorter than the filter delay. This provides a degree of low pass filtering.

If noise occurs during a symbol transition, the detection of that transition may be delayed by an amount equal to the length of the noise burst. This is just a reflection of the uncertainty of where the transition is truly occurring within the noise.

Noise pulses that are wider than the filter delay, but narrower than the shortest allowable symbol length will be detected by the next stage of the BDLC module’s receiver as an invalid symbol.

Noise pulses that are longer than the shortest allowable symbol length will normally be detected as an invalid symbol or as invalid data when the frame’s CRC is checked.

4.3 Protocol Handler

The Protocol Handler is responsible for framing, collision detection, arbitration, CRC generation/checking, and error detection. The Protocol Handler conforms to SAE J1850 - Class B Data Communications Network Interface. Refer to **Figure 1-2 BDLC Block Diagram on page 16**

4.3.1 Protocol Architecture

The Protocol Handler contains the State Machine, Rx Shadow Register, Tx Shadow Register, Rx Shift Register, Tx Shift Register, and Loopback Multiplexer as shown in Figure 4-9 BDLC Protocol Handler Outline. Each block will now be described in more detail.

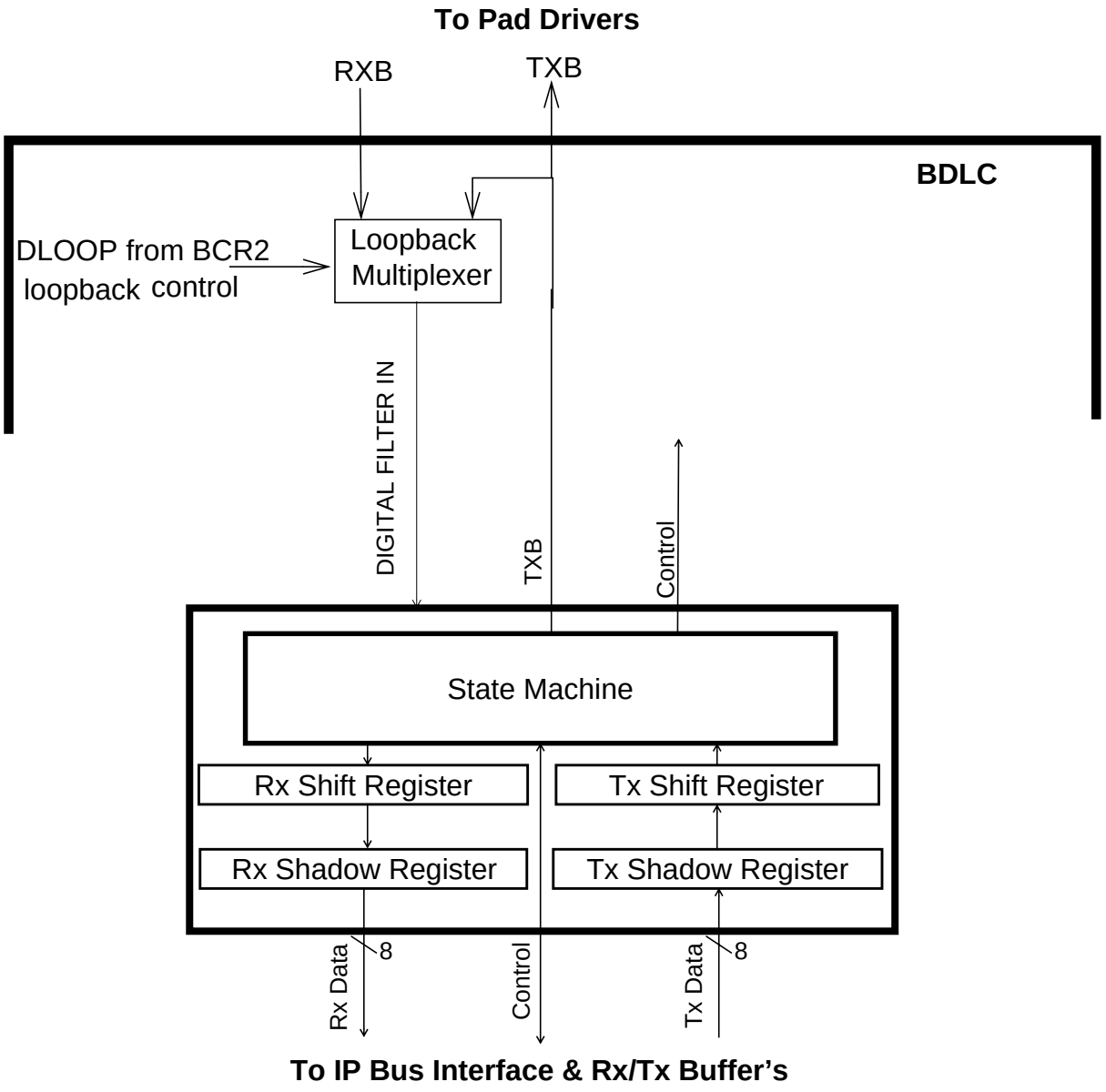


Figure 4-9 BDLC Protocol Handler Outline

- Rx & Tx Shift Registers

  The Rx Shift Register gathers received serial data bits from the J1850 bus and makes them available in parallel form to the Rx Shadow Register. The Tx Shift Register takes data, in parallel form, from the Tx Shadow Register and presents it serially to the State Machine so that it can be transmitted onto the J1850 bus.

- Rx & Tx Shadow Registers

Immediately after the Rx Shift Register has completed shifting in a byte of data, this data is transferred to the Rx Shadow Register and RDRF or RXIFR is set and interrupt is generated if the interrupt enable bit (IE) in BCR1 is set. After the transfer takes place, this new data byte in the Rx Shadow Register is available to the CPU, and the Rx Shift Register is ready to shift in the next byte of data. Data in Rx Shadow Register must be retrieved by the CPU before it is overwritten by new data from the Rx Shift Register.

Once the Tx Shift Register has completed its shifting operation for the current byte, the data byte in the Tx Shadow Register is loaded into the Tx Shift Register. After this transfer takes place, the Tx Shadow Register is ready to accept new data from the CPU.

- Digital Loopback Multiplexer

  The Digital Loopback Multiplexer connects the input of the receive digital filter (See Figure 4-9) to either the transmit signal out to the pad (TXB) or the receive signal from the pad (RXB), depending on the state of the DLOOP bit in DLCBCR2 register.

- State Machine

  All of the functions associated with performing the protocol are executed or controlled by the State Machine. The State Machine is responsible for framing, collision detection, arbitration, CRC generation/checking, and error detection. The following sections describe the BDLC module's actions in a variety of situations.

- 4X Mode

  The BDLC module can exist on the same J1850 bus as modules which use a special 4X (41.6 kbps) mode of J1850 VPW operation. The BDLC module cannot transmit in 4X mode, but can receive messages in 4X mode, if the RX4X bit is set in BCR2 register. If the RX4X bit is not set in the BCR2 register, any 4X message on the J1850 bus is treated as noise by the BDLC module and is ignored. Likewise, 4X messages transmitted on the SAE J1850 bus when the BDLC module is in normal mode will be interpreted as noise on the network by the BDLC module.

- Receiving a Message in Block Mode

  Although not a part of the SAE J1850 protocol, the BDLC module does allow for a special "Block Mode" of operation of the receiver. As far as the BDLC module is concerned, a Block Mode message is simply a long J1850 frame that contains an indefinite number of data bytes. All of the other features of the frame remain the same, including the SOF, CRC, and EOD symbols.

  Another node wishing to send a Block Mode transmission must first inform all other nodes on the network that this is about to happen. This is usually accomplished by sending a special predefined message.

- Transmitting a Message in Block Mode

  A Block mode message is transmitted inherently by simply loading the bytes one by one into the BDR register until the message is complete. The programmer should wait until the TDRE flag is set prior to writing a new byte of data into the BDR register. The BDLC module does not contain any predefined maximum J1850 message length requirement.

4.4 Transmitting A Message

The design of the BDLC module enables the user to easily handle message reception and message transmission separately. This can greatly simplify the communication software, as all received messages can be handled virtually the same, regardless of their origin.

This chapter will therefore describe only the steps necessary for transmitting a message, and will not address the resulting reception of that message by the BDLC module. Message reception is described in Section 4.5 Receiving A Message. Later sections will deal with transmitting and receiving In-Frame Responses on the SAE J1850 bus.

4.4.1 BDLC Transmission Control Bits

There is only one BDLC module control bit which is used when transmitting a message onto the SAE J1850 bus. This bit, the Transmit End of Data (TEOD) bit, is set by the user to indicate to the BDLC module that the last byte of that part of the message frame has been loaded into the DLCBDR. The TEOD bit, located in DLCBCR2, is also used when transmitting an In-Frame Response (IFR), but that usage is described in Section 4.6 Transmitting An In-Frame Response (IFR) on page 67. Setting the TEOD bit indicates to the BDLC module that the last byte written to the BDLC Data Register is the final byte to be transmitted, and that following this byte a CRC byte and EOD symbol should be transmitted automatically. Setting the TEOD bit will also inhibit any further TDRE interrupts until TEOD is cleared. The TEOD bit will be cleared on the rising edge of the first bit of the transmitted CRC byte, or if an error or loss of arbitration is detected on the bus.

- BDLC Data Register

  The BDLC Data Register is a double-buffered register which is used for handling the transmitted and received message bytes.Bytes to be transmitted onto the SAE J1850 bus are written to the DLCBDR, and bytes received from the bus by the BDLC module are read from the DLCBDR. Since this register is double buffered, bytes written into it cannot be read by the CPU. If this is attempted, the byte which is read will be the last byte placed in the DLCBDR by the BDLC module, not the last byte written to the DLCBDR by the CPU. For an illustration of the DLCBDR, refer to Section 3.3.4 BDLC Data Register (DLCBDR) on page 31.

- Transmitting a Message with the BDLC

  To transmit a message using the BDLC module, the user just writes the first byte of the message to be transmitted into the DLCBDR, initiating the transmission process. When the TDRE status appears in the DLCBSVR, the user writes the next byte into the DLCBDR. Once all of the bytes have been loaded into the DLCBDR, the user sets the TEOD bit, and the BDLC module completes the message transmission. What follows is an overview of the basic steps required to transmit a message onto an SAE J1850 network using the BDLC module. For an illustration of this sequence, refer to Figure 4-10 Basic BDLC Transmit Flowchart on page 62.

NOTE: *Due to the byte-level architecture of the BDLC module, the 12-byte limit on message length as defined in SAE J1850 must be enforced by the user's software. The number of bytes in a message (transmitted or received) has no meaning to the BDLC module.*

– Step 1: Write the First Byte into the DLCBDR

To initiate a message transmission, the CPU simply loads the first byte of the message to be transmitted into the DLCBDR. The BDLC module will then perform the necessary bus acquisition duties to determine when the message transmission can begin.

Once the BDLC module determines that the SAE J1850 bus is free, a Start of Frame (SOF) symbol will be transmitted, followed by the byte written to the DLCBDR. Once the BDLC module readies this byte for transmission, the DLCBSVR will reflect that the next byte can be written to the DLCBDR (TDRE interrupt).

NOTE: *If the user writes the first byte of a message to be transmitted to the DLCBDR and then determines that a different message should be transmitted, the user can write a new byte to the DLCBDR up until the transmission begins. This new byte will replace the original byte in the DLCBDR.*

– Step 2: When TDRE is Indicated, Write the Next Byte into the DLCBDR

When a TDRE state is reflected in the BSVR, the CPU writes the next byte to be transmitted into the BDR. This step is repeated until the last byte to be transmitted is written to the DLCBDR.

NOTE: *Due to the design and operation of the BDLC module, when transmitting a message the user may write two, or possibly even three of the bytes to be transmitted into the DLCBDR before the first RDRF interrupt occurs. For this reason, the user should never use receive interrupts to control the sequencing of bytes to be transmitted.*

– Step 3: Write the Last Byte to the DLCBDR and Set TEOD

Once the user has written the last byte to be transmitted into the DLCBDR, the user then sets the TEOD bit in DLCBCR2. When the TEOD bit is set, once the byte written to the DLCBDR is transmitted onto the bus, the BDLC module will begin transmitting the 8-bit CRC byte, as specified in SAE J1850. Following the CRC byte, the BDLC module will transmit an EOD symbol onto the SAE J1850 bus, indicating that this part of the message has been completed. If no IFR bytes are transmitted following the EOD, an EOF will be recognized and the message will be complete.

Setting the TEOD bit is the last step the CPU needs to take to complete the message transmission, and no further transmission-related interrupts will occur. Once the message has been completely received by the BDLC module, an EOF interrupt will be generated. However, this is technically a receive function which can be handled by the message reception routine.

NOTE: *While the TEOD bit is typically set immediately following the write of the last byte to the BDR, it is also acceptable to wait until a TDRE interrupt is generated before setting the TEOD bit. While the example flowchart in Figure 4-10 shows the TEOD bit being set after the write to the BDR, either method is correct. If a TDRE interrupt is pending, it will be cleared when the TEOD bit is set.*

4.4.2 Transmitting Exceptions

While this is the basic transmit flow, at times the message transmit process will be interrupted. This can be due to a loss of arbitration to a higher priority message or due to an error being detected on the network. For the transmit routine, either of these events can be dealt with in a similar manner.

- Loss of Arbitration

  If a loss of arbitration (LOA) occurs while the BDLC module is transmitting onto the SAE J1850 bus, the BDLC module will immediately stop transmitting, and a LOA status will be reflected in the DLCBSVR. If the loss of arbitration has occurred on a byte boundary, an RDRF interrupt may also be pending once the LOA interrupt is cleared.

  When a loss of arbitration occurs, the J1850 message handling software should immediately switch into the receive mode. If the TEOD bit was set, it will be cleared automatically. **If another attempt is to be made to transmit the same message, the user must start the transmit sequence over from the beginning of the message.**

- Error Detection

  Similar to a loss of arbitration, if any error (except a CRC error) is detected on the SAE J1850 bus during a transmission, the BDLC module will stop transmitting immediately. The byte which was being transmitted will be discarded, and the “Symbol Invalid or Out of Range” status will be reflected in the DLCBSVR. As with the loss of arbitration, if the TEOD bit was set, it will be cleared automatically, and any attempt to transmit the same message will have to start from the beginning.

  If a CRC error occurs following a transmission, this will also be reflected in the DLCBSVR. However, since the CRC error is really a receive error based on the received CRC byte, at this point all bytes of the message will have been transmitted. It is therefore up to the user’s software to determine if another attempt should be made to transmit the message in which the error occurred.

- Transmitter Underrun

  A transmitter underrun can occur when a TDRE interrupt is not serviced in a timely fashion. If the last byte loaded into the DLCBDR is completely transmitted onto the network before the next byte is loaded into the BDR, a transmitter underrun will occur. If this does happen, the BDLC module will transmit two additional logic ones to ensure that the partial message which was transmitted onto the bus does not end on a byte boundary. This will be followed by an EOD and EOF symbol. The only indication to the CPU that an underrun occurred is the Symbol Invalid or Out of Range error which will be indicated in the DLCBSVR. As with the other errors, it is up to the user’s software to determine if another transmission attempt should be made.

- In-Frame Response to a Transmitted Message

  If an In-Frame Response (IFR) is received following the transmission of a message, the status indicating that an IFR byte has been received will be indicated in the DLCBSVR before an EOF is indicated. Refer to Section 4.7 Receiving An In-Frame Response (IFR) on page 78 for a description of how to handle the reception of IFR bytes.

4.4.3 Aborting a Transmission

The BDLC module does not have a mechanism designed specifically for aborting a transmission. Since the module transmits each message on a byte-by-byte basis, there is little need to implement an abort mechanism. If the user has loaded a byte into the DLCBDR to initiate a message transmission and decides to send a different message, the byte in the DLCBDR can be replaced, right up to the point that the message transmission begins.

If the user has loaded a byte into the DLCBDR and then decides not to send any message at all, the user can let the byte transmit, and when the TDRE interrupt occurs let the transmitter underrun. This will cause two extra logic ones followed by an EOF to be transmitted. While this method may require a small amount of bus bandwidth, the need to do this should be very rare. Replacing the byte originally written to the BDR with $FF will also increase the probability of the transmitter losing arbitration if another node begins transmitting at the same time, also reducing the bus bandwidth needed.

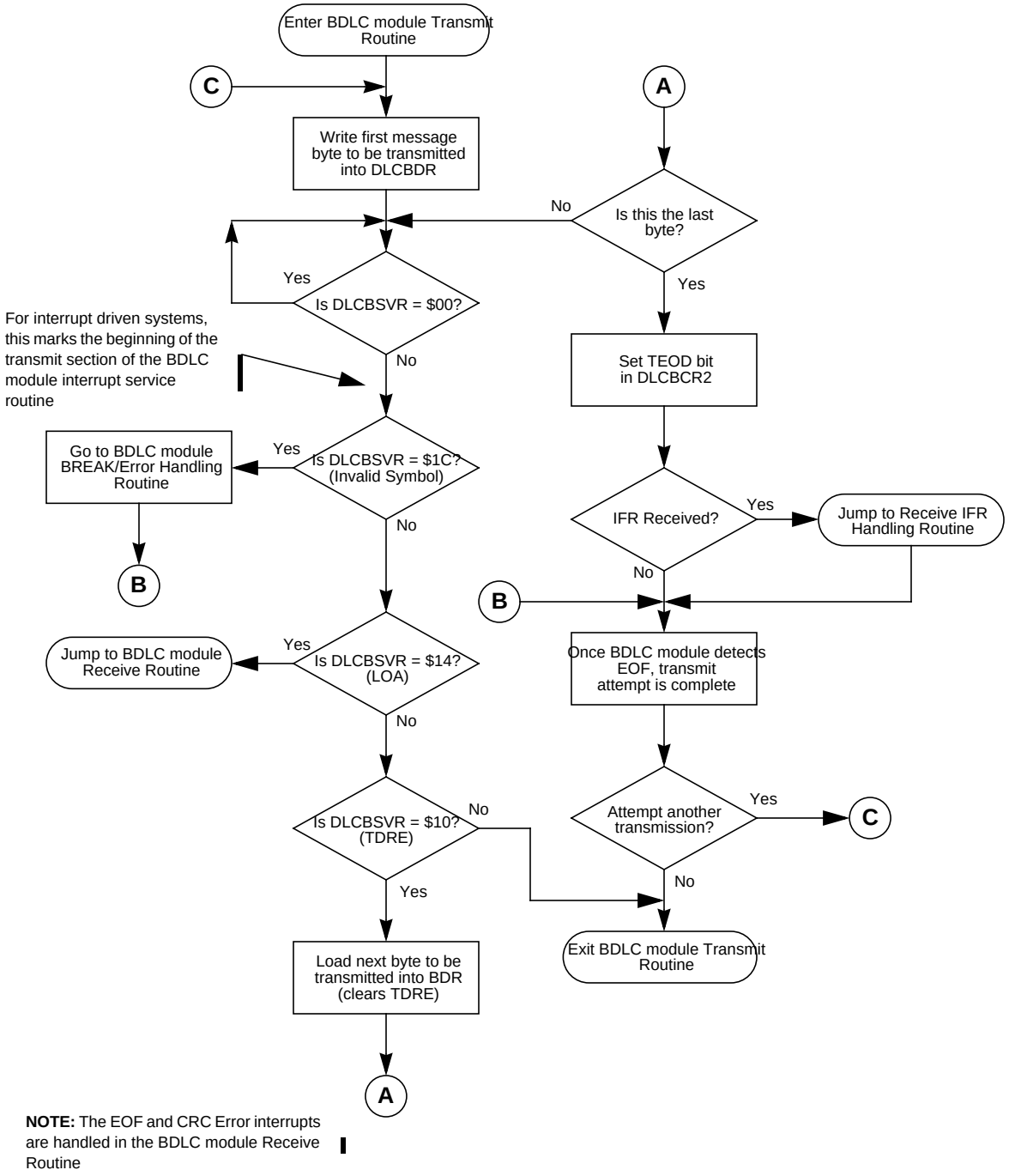


Figure 4-10 Basic BDLC Transmit Flowchart

4.5 Receiving A Message

The design of the BDLC module makes it especially easy to use for receiving messages off of the SAE J1850 bus. When the first byte of a message comes in, the DLCBSVR will indicate to the CPU that a byte

has been received. As each successive byte is received, that will in turn be reflected in the DLCBSVR. When the message is complete and the EOF has been detected on the bus, the DLCBSVR will reflect this, indicating that the message is complete.

The basic steps required for receiving a message from the SAE J1850 bus are outlined below. For more information on receiving IFR bytes, refer to Section 4.7 Receiving An In-Frame Response (IFR) on page 78.

4.5.1 BDLC Reception Control Bits

The only control bit which is used for message reception, the IMSG bit, is actually used to prevent message reception. When the IMSG bit is set BDLC module interrupts of the CPU are inhibited until the next SOF symbol is received. This allows the BDLC module to ignore the remainder of a message once the CPU has determined that it is of no interest. This helps reduce the amount of CPU overhead used to service messages received from the SAE J1850 network, since otherwise the BDLC module would require attention from the CPU for each byte broadcast on the network. The IMSG bit is cleared when the BDLC module receives an SOF symbol, or it can also be cleared by the CPU.

NOTE: *While the IMSG bit can be used to prevent the CPU from having to service the BDLC module for every byte transmitted on the SAE J1850 bus,* ***the IMSG bit should never be used to ignore the BDLC module's own transmission****. Because setting the IMSG bit prevents all DLCBSVR bits from being updated until an SOF is received, the user would not receive any further transmit-related interrupts until another SOF was received, making it very difficult for the CPU to complete the transmission correctly.*

4.5.2 Receiving a Message with the BDLC module

Receiving a message using the BDLC module is extremely straight-forward. As each byte of a message is received and placed into the DLCBDR, the BDLC module will indicate this to the CPU with an Rx Data Register Full (RDRF) status in the DLCBSVR. When an EOF symbol is received, indicating to the CPU that the message is complete, this too will be reflected in the DLCBSVR.

Outlined below are the basic steps to be followed for receiving a message from the SAE J1850 bus with the BDLC module. For an illustration of this sequence, refer to Basic BDLC Receive Flowchart on page 66.

- Step 1: When RDRF Interrupt Occurs, Retrieve Data Byte

  When the first byte of a message following a valid SOF symbol is received that byte is placed in the DLCBDR, and an RDRF state is reflected in the DLCBSVR. No indication of the SOF reception is made, since the end of the previous message is marked by an EOF indication. The first RDRF state following this EOF indication should allow the user to determine when a new message begins.

  The RDRF interrupt is cleared when the received byte is read from the DLCBDR. Once this is done, no further CPU intervention is necessary until the next byte is received, and this step is repeated.

  All bytes of the message, including the CRC byte, will be placed into the DLCBDR as they are received for the CPU to retrieve.

- Step 2: When an EOF is Received, the Message is Complete

  Once all bytes (including the CRC byte) have been received from the bus, the bus will be idle for a time period equal to an EOD symbol. Once the EOD symbol is received, the BDLC module will verify that the CRC byte is correct. If the CRC byte is not correct, this will be reflected in the DLCBSVR.

  If no In-Frame Response bytes are transmitted following the EOD symbol, the EOD will transition into an EOF symbol. When the EOF is received it will be reflected in the DLCBSVR, indicating to the user that the message is complete. If IFR bytes do follow the first EOD symbol, once they are complete another EOD will be transmitted, followed by an EOF.

  Once the EOF state is reflected in the DLCBSVR, this indicates to the user that the message is complete, and that when another byte is received it is the first byte of a new message.

4.5.3 Filtering Received Messages

No message filtering hardware is included on the BDLC module, so all message filtering functions must be performed in software. Because the BDLC module handles each message on a byte-by-byte basis, message filtering can be done as each byte is received, rather than after the entire message is complete. This enables the CPU to decide while a message is still in progress whether or not that message is of any interest.

At any point during a message, if the CPU determines that the message is of no interest the IMSG bit can be set. Setting the IMSG bit commands the BDLC module not to update the DLCBSVR until the next valid SOF is received. This prevents the CPU from having to service the BDLC module for each byte of every message sent over the network.

4.5.4 Receiving Exceptions

As with a message transmission, this basic message reception flow can be interrupted if errors are detected by the BDLC module. This can occur if an incorrect CRC is detected or if an invalid or out of range symbol appears on the SAE J1850 bus. A problem can also arise if the CPU fails to service the DLCBDR in a timely manner during a message reception.

- Receiver Overrun

  Once a message byte has been received, the CPU must service the DLCBDR before the next byte is received, or the first byte will be lost. If the DLCBDR is not serviced quickly enough, the next byte received will be written over the previous byte in the DLCBDR. No receiver overrun indication is made to the CPU. If the CPU fails to service the BDLC module during the reception of an entire message, the byte remaining in the DLCBDR will be last byte received (usually a CRC byte).

  Once a receiver overrun occurs, there is no way for the CPU to recover the lost byte(s), so the entire message should be discarded. To prevent receiver overrun, the user should ensure that a BDLC RDRF interrupt will be serviced before the next byte can be received. When polling the DLCBSVR, the user should select a polling interval which will provide timely monitoring of the BDLC module.

- CRC Error

If a CRC error is detected during a message reception, this will be reflected in the BSVR once an EOD time is recognized by the BDLC module. Since all bytes of the message will have been received when this error is detected, it is up to the user to ensure that all the received message bytes are discarded.

- Invalid or Out of Range Symbol

  If an invalid or out of range symbol, a framing error or a BREAK symbol is detected on the SAE J1850 bus during the reception of a message, the BDLC module will immediately stop receiving the message and discard any partially received byte. The “Symbol Invalid or Out of Range” status will immediately be reflected in the DLCBSVR. Following this the BDLC module will wait until the bus has been idle for a time period equal to an EOF symbol before receiving another message. As with the CRC error, the user should discard any partially received message if this occurs.

- In-Frame Response to a Received Message

  As mentioned above, if one or more IFR bytes are received following the reception of a message, the status indicating the reception of the IFR byte(s) will be indicated in the DLCBSVR before the EOF is indicated. Refer to Receiving An In-Frame Response (IFR) on page 78 for a description of how to deal with the reception of IFR bytes.

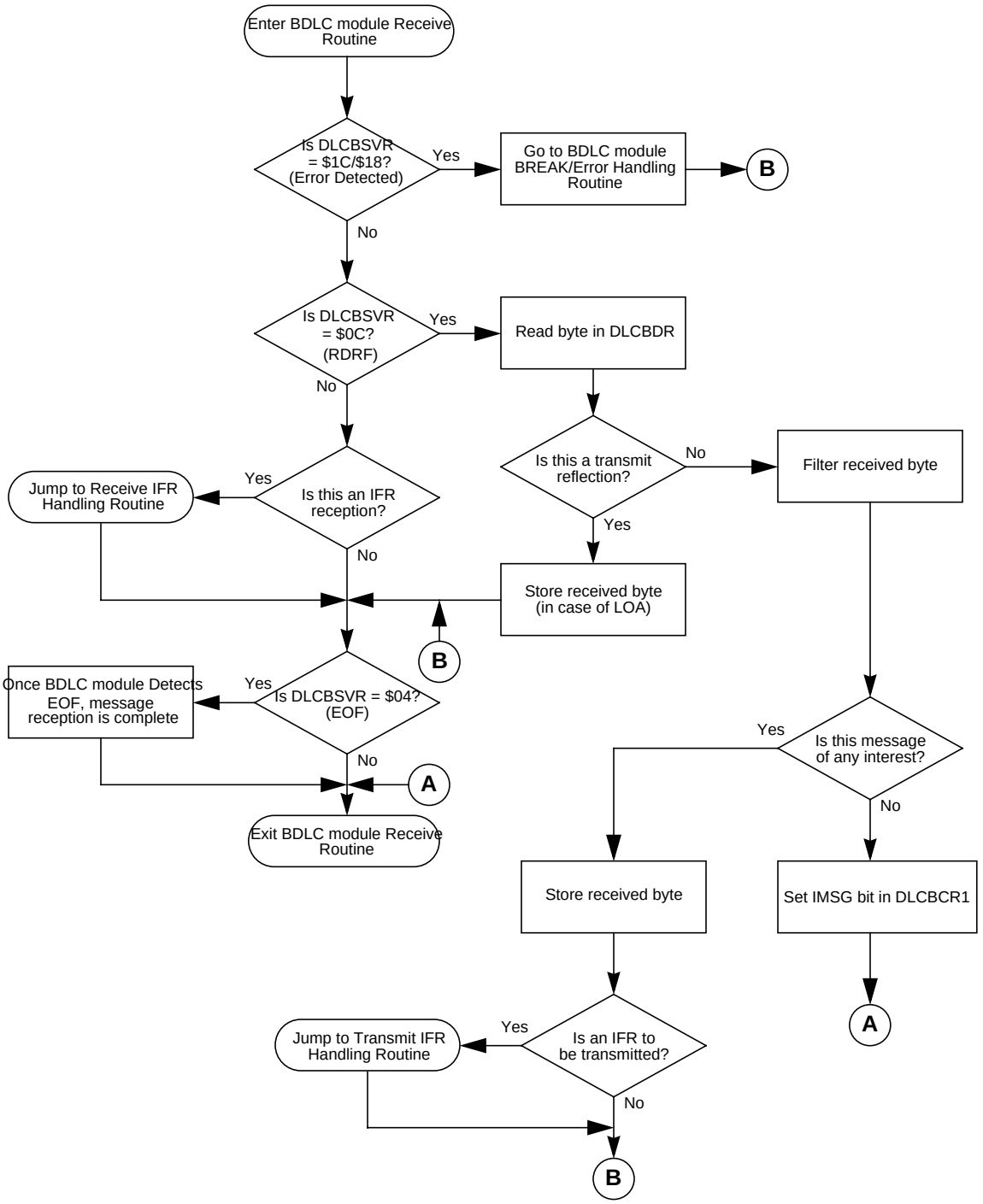


Figure 4-11 Basic BDLC Receive Flowchart

4.6 Transmitting An In-Frame Response (IFR)

The BDLC module can be used to transmit all four types of In-Frame Response (IFR) which are defined in SAE J1850. A very brief definition of each IFR type is given below. For a more detailed description of each, refer the SAE J1850 document.

The explanation regarding IFR support by the BDLC module which assumes the user is familiar with the use of IFRs as defined in SAE J1850, and understands the message header bit encoding and normalization bit formats which are used with the different types of IFRs. For more information on this, refer to the SAE J1850 document.

4.6.1 IFR Types Supported by the BDLC module

SAE J1850 defines four distinct types of IFR. The first (and most basic) IFR is Type 0, or no IFR. IFR types 1, 2 & 3 are each made up of one or more bytes and, depending upon the type used, may be followed by a CRC byte. The BDLC module is designed to allow the user to transmit and receive all types of SAE J1850 IFRs, but only the network framing/error checking/bus acquisition duties are performed by the BDLC module. The user is responsible for determining the type of IFR to be transmitted, the number of retries to be made (if allowed), and the allowable number of bytes to be transmitted.

- IFR Type 0: No Response

  The most basic type of IFR is no IFR. The Type 0 IFR, as defined in SAE J1850, is no response. The EOD and EOF symbols follow directly after the CRC byte at the end of the message frame being transmitted. This type of IFR is, of course, inherently supported by the BDLC module with no additional user intervention required.

- IFR Type 1: Single Byte from a Single Responder

  SAE J1850 defines the Type 1 IFR as a single byte from a single receiver. This type of IFR is used to acknowledge to the transmitter that the message frame was transmitted successfully on the network, and that at least one receiver received it correctly. A Type 1 IFR generally consists of the physical node ID of the receiver responding to the message, with no CRC byte appended. This type of response is used for Broadcast-type messages, where there may be several intended receivers for a message but the transmitter only wants to know that at least one node received it. In this case, all receivers will begin transmitting their node ID following the EOD. Since all nodes on an SAE J1850 network have a unique node ID, if multiple nodes begin transmitting their node ID simultaneously, arbitration takes place. The node with the highest priority (lowest value) ID wins this arbitration process, and that node’s ID makes up the IFR. No retries are attempted by the nodes which lose arbitration during a Type 1 IFR transmission.

  A Type 1 IFR can also be used as a response to a physically addressed message, where the only intended receiver is the one which responds. In this case, no arbitration would take place during the IFR transmission, but the resulting IFR would still consist of a single byte.

- IFR Type 2: Single Byte from Multiple Responders

The Type 2 IFR, as defined in SAE J1850, is a series of single bytes, each transmitted by a different responder. This IFR type not only acknowledges to the transmitter that the message was transmitted successfully, but also reveals which receivers actually received the message. As with the Type 1 IFR, no CRC byte is appended to the end of a Type 2 IFR.

This IFR type is typically used with Function-type messages, where the original transmitter may need to know which nodes actually received the message. The basic difference between this type of IFR and the Type 1 IFR is that the nodes which lose arbitration while attempting to transmit their node ID during a Type 2 IFR wait until the byte which wins arbitration is transmitted and then again attempt to transmit their node ID onto the bus. The result is a series of node IDs, one from each receiver of the original message.

- IFR Type 3: Multiple Bytes from a Single Responder

  The last type of IFR defined by SAE J1850 is the Type 3 IFR. This IFR type consists of one or more bytes from a single responder. This type of IFR is used to return data to the original transmitter within the original message frame. This type of IFR may or may not have a CRC byte appended to it.

  The Type 3 IFR is typically used with Function Read-type or Function Query-type messages, where the original transmitter is requesting data from the intended receiver. The node requesting the data transmits the initial portion of the message, and the intended receiver responds by transmitting the desired data in an IFR. In most cases, the original message requiring a Type 3 IFR is addressed to one particular node, so no arbitration should take place during the IFR portion of the message.

4.6.2 BDLC IFR Transmit Control Bits

The BDLC module has three bits which are used to control the transmission of an In-Frame Response. These bits, all located in DLCBCR2, are TSIFR, TMIFR1 and TMIFR0. Each is used in conjunction with the TEOD bit to transmit one of three IFR types defined in SAE J1850. What follows is a brief description of each bit.

Because each of the bits used for transmitting an IFR with the BDLC module is used to transmit a particular type of IFR, only one bit should be set by the CPU at a time. However, should more than one of these bits get set at one time, a priority encoding scheme is used to determine which type of IFR is sent. This scheme prevents unpredictable operation caused by conflicting signals to the BDLC module. Table 4-8 illustrates which IFR bit will actually be acted upon by the BDLC module should multiple IFR bits get set at the same time.

NOTE: *As with transmitted messages, IFRs transmitted by the BDLC module will also be received by the BDLC module. For a description of how IFR bytes received by the BDLC module should be handled, refer to Section 4.7 Receiving An In-Frame Response (IFR) on page 78.*

Table 4-8 IFR Control Bit Priority Encoding

READ/WRITE			ACTUAL		
TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0
0	0	0	0	0	0
1	X	X	1	0	0
0	1	X	0	1	0
0	0	1	0	0	1

4.6.3 Transmit Single Byte IFR

The Transmit Single Byte IFR (TSIFR) bit in DLCBCR2 is used to transmit Type 1 and Type 2 IFRs onto the SAE J1850 bus. If this bit is set after a byte is loaded into the BDR, the BDLC module will attempt to send that byte, preceded by the appropriate Normalization Bit, as a single byte IFR without a CRC. If arbitration is lost, the BDLC module will automatically attempt to transmit the byte again (without a Normalization Bit) as soon as the byte winning arbitration completes transmission. Attempts to transmit the byte will continue until either the byte is successfully transmitted, the TEOD bit is set by the user or an error is detected on the bus.

The user must set the TSIFR bit before the EOD following the main part of the message frame is received, or no IFR transmit attempts will be made for the current message. If another node does transmit an IFR to this message or a reception error occurs, the TSIFR bit will be cleared. If not, the IFR will be transmitted after the EOD of the next received message.

The TSIFR bit will be automatically cleared once the EOD following one or more IFR bytes has been received or an error is detected on the bus.

4.6.4 Transmit Multi-Byte IFR 1

The Transmit Multi-Byte IFR 1 (TMIFR1) bit is used to transmit an SAE J1850 Type 3 IFR with a CRC byte appended. If this bit is set after the user has loaded the first byte of a multi-byte IFR into the DLCBDR, the BDLC module will begin transmitting that byte, preceded by the appropriate Normalization Bit, onto the SAE J1850 bus. Once this happens a TDRE interrupt will occur, indicating to the user that the next IFR byte should be loaded into the DLCBDR. When the last byte to be transmitted is written to the DLCBDR, the user sets the TEOD bit. This will cause a CRC byte and an EOD symbol to be transmitted following the last IFR byte.

As with the TSIFR bit, the TMIFR1 bit must be set before the EOD symbol is received, or it will remain cleared and no IFR transmit attempt will be made. The TMIFR1 bit will be cleared once the CRC byte and EOD are transmitted, if an error is detected on the bus, if a loss of arbitration occurs during the IFR transmission or if a transmitter underrun occurs when the user fails to service the TDRE interrupt in a timely manner. If a loss of arbitration occurs while the Type 3 IFR is being transmitted, transmission will halt immediately and the loss of arbitration will be indicated in the DLCBSVR.

4.6.5 Transmit Multi-Byte IFR 0

The Transmit Multi-Byte IFR 0 (TMIFR0) bit is used to transmit an SAE J1850 Type 3 IFR without a CRC byte appended. If this bit is set after the user has loaded the first byte of a multi-byte IFR into the DLCBDR, the BDLC module will begin transmitting that byte, preceded by the appropriate Normalization Bit, onto the SAE J1850 bus. Once this happens a TDRE interrupt will occur, indicating to the user that the next IFR byte should be loaded into the DLCBDR. When the last byte to be transmitted is written to the DLCBDR, the user sets the TEOD bit. This will cause an EOD symbol to be transmitted following the last IFR byte.

As with the TSIFR and TMIFR1 bits, the TMIFR0 bit must be set before the EOD symbol is received, or it will remain cleared and no IFR transmit attempt will be made. The TMIFR0 bit will be cleared once the CRC byte and EOD are transmitted, if an error is detected on the bus, if a loss of arbitration occurs during the IFR transmission or if a transmitter underrun occurs when the user fails to service the TDRE interrupt in a timely manner. If a loss of arbitration occurs while the Type 3 IFR is being transmitted, transmission will halt immediately and the loss of arbitration will be indicated in the DLCBSVR.

NOTE: *The TMIFR0 bit should not be used to transmit a Type 1 IFR. If a loss of arbitration occurs on the last bit of a byte being transmitted using the TMIFR0 bit, two extra logic ones will be transmitted to ensure that the IFR will not end on a byte boundary. This can cause an error in a Type 1 IFR.*

4.6.6 Transmitting An IFR with the BDLC module

While the design of the BDLC module makes the transmission of each type of IFR similar, the steps necessary for sending each will be discussed. Again, a discussion of the bytes making up any particular IFR is not within the scope of this document. For a more detailed description of the use of IFRs on an SAE J1850 network, refer to the SAE J1850 document.

- Transmitting a Type 1 IFR

  To transmit a Type 1 IFR, the user loads the byte to be transmitted into the DLCBDR and sets both the TSIFR bit and the TEOD bit. This will direct the BDLC module to attempt transmitting the byte written to the DLCBDR one time, preceded by the appropriate Normalization Bit. If the transmission is not successful, the byte will be discarded and no further transmission attempts will be made. For an illustration of the steps described below, refer to Section 4-12 Transmitting A Type 1 IFR on page 72.

  – Step 1: Load the IFR Byte into the BDR

    The user begins initiation of a Type 1 IFR by loading the desired IFR byte into the DLCBDR. If a byte has already been written into the DLCBDR for transmission as a new message, the user can simply write the IFR byte to the DLCBDR, replacing the previously written byte. This must be done before the first EOD symbol is received.

  – Step 2: Set the TSIFR and TEOD Bits

The final step in transmitting a Type 1 IFR with the BDLC module is to set the TSIFR and TEOD bits in DLCBCR2. Setting both bits will direct the BDLC module to make one attempt at transmitting the byte in the DLCBDR as an IFR. If the byte is transmitted successfully, or if an error or loss of arbitration occurs, TEOD and TSIFR will be cleared and no further transmit attempts will be made.

- Transmitting a Type 2 IFR

  To transmit a Type 2 IFR, the user loads the byte to be transmitted into the DLCBDR and sets the TSIFR bit. Once this is done, the BDLC module will attempt to transmit the byte in the DLCBDR as a single byte IFR, preceded by the appropriate Normalization Bit. If the first BDLC module loses arbitration on the first attempt, it will make repeated attempts to transmit this byte until it is successful, an error occurs or the user sets the TEOD bit.

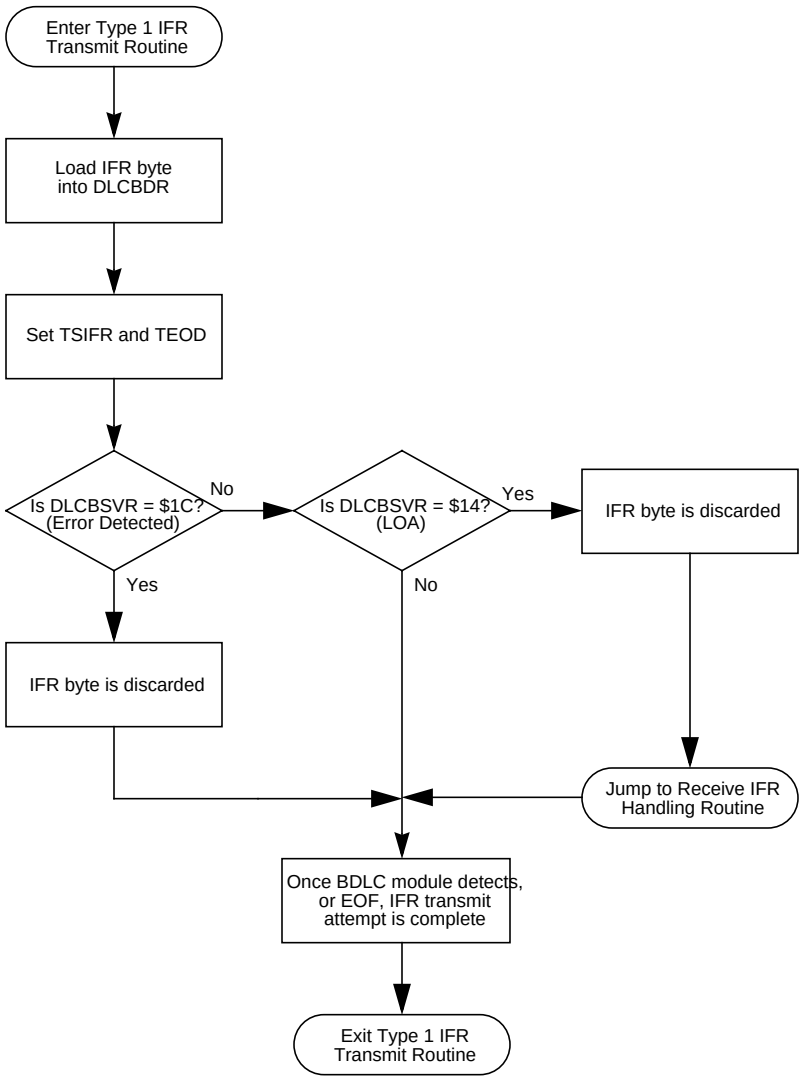


Figure 4-12 Transmitting A Type 1 IFR

– Step 1: Load the IFR Byte into the BDR

  As with the Type 1 IFR, the user begins initiation of a Type 2 IFR by loading the desired IFR byte into the DLCBDR. If a byte has already been written into the DLCBDR for transmission as a new message, the user can simply write the IFR byte to the DLCBDR, replacing the previously written byte. This must be done before the first EOD symbol is received.

– Step 2: Set the TSIFR Bit

The second step necessary for transmitting a Type 2 IFR is to set the TSIFR bit in DLCBCR2. Setting this bit will direct the BDLC module to attempt to transmit the byte in the DLCBDR as an IFR until it is successful. If the byte is transmitted successfully, or if an error or loss of arbitration occurs, TSIFR will be cleared and no further transmit attempts will be made.

– Step 3: If Necessary, Set the TEOD Bit

The third step in transmitting a Type 2 IFR is only necessary if the user wishes to halt the transmission attempts. This may be necessary if the BDLC module's attempt to transmit the byte loaded into the DLCBDR continually loses arbitration, and the overall message length approaches the 12-byte limit as defined in SAE J1850.

If it becomes necessary to halt the IFR transmission attempts, the user simply sets the TEOD bit in BCR2. If the BDLC module is between transmission attempts, it will make one more attempt to transmit the IFR byte. If it is transmitting the byte when TEOD is set, the BDLC module will continue the transmission until it is successful or it loses arbitration to another transmitter. At this point it will then discard the byte and make no more transmit attempts.

NOTE: *When transmitting a Type 2 IFR, the user should monitor the number of IFR bytes received to ensure that the overall message length does not exceed the 12-byte limit for the length of SAE J1850 messages. The user should set the TEOD bit when the 11th byte is received, which will prevent the 12-byte limit from being exceeded.*

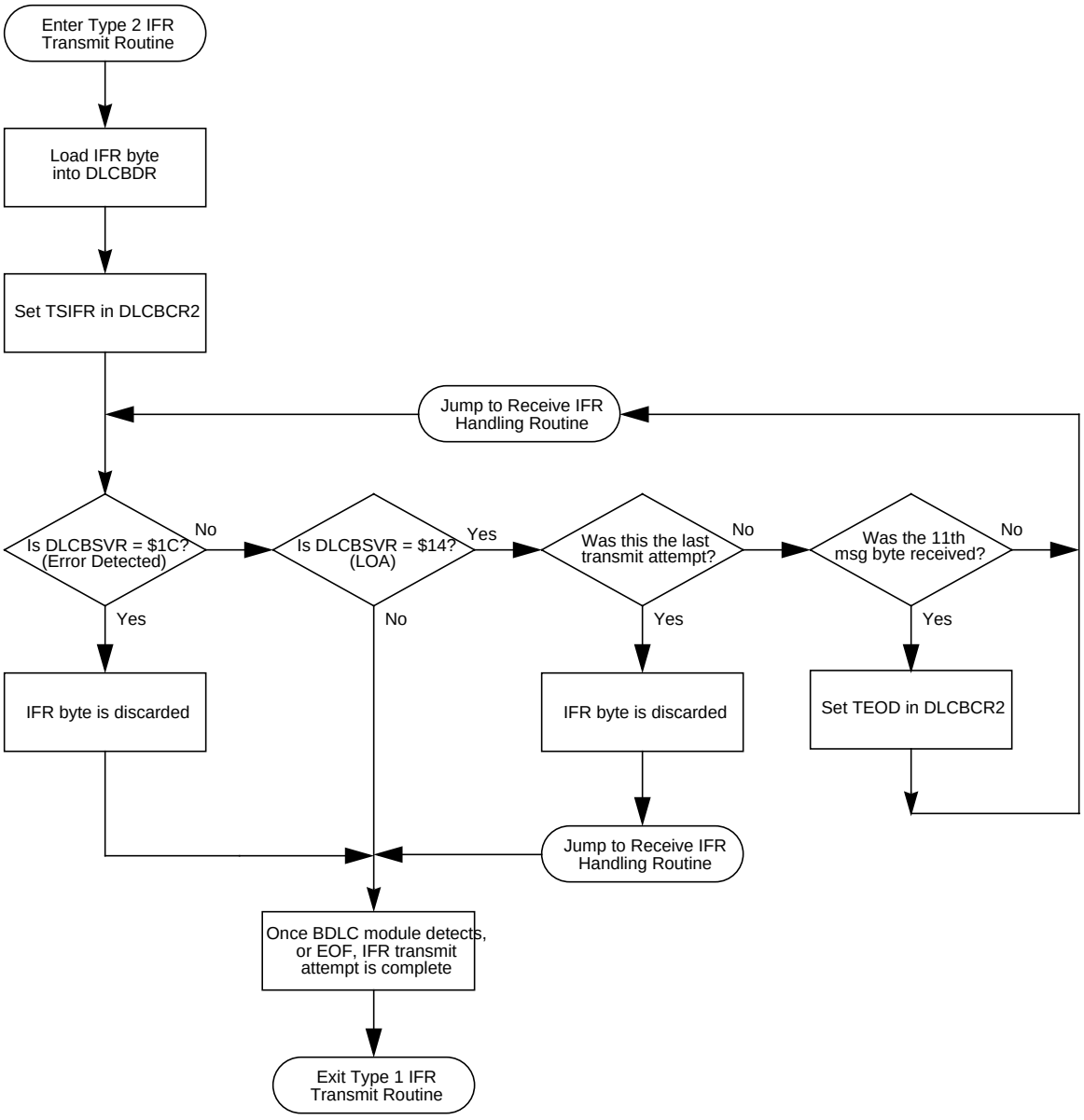


Figure 4-13 Transmitting A Type 2 IFR

- Transmitting a Type 3 IFR

Transmitting a Type 3 IFR, with or without a CRC byte, is done in a fashion similar to transmitting a message frame. The user loads the first byte to be transmitted into the DLCBDR and then sets the appropriate TMIFR bit, depending upon whether a CRC byte is desired. When the last byte is written to the BDR, the TEOD bit is set, and a CRC byte (if desired) and an EOD are then transmitted. Because the two versions of the Type 3 IFR are transmitted identically, the description which follows will discuss both. For an illustration of the Type 3 IFR transmit sequence, refer to Figure 4-14 Transmitting A Type 3 IFR on page 77.

– Step 1: Load the First IFR Byte into the DLCBDR

  The user begins initiation of a Type 3 IFR, as with each of the other IFR types, by loading the desired IFR byte into the DLCBDR. If a byte has already been written into the DLCBDR for transmission as a new message, the user can simply write the first IFR byte to the DLCBDR, replacing the previously written byte. This must be done before the first EOD symbol is received.

– Step 2: Set the TMIFR Bit

  The second step necessary for transmitting a Type 3 IFR is to set the desired TMIFR bit in DLCBCR2, depending upon whether or not a CRC is desired. As previously described in Section 4.6.2 BDLC IFR Transmit Control Bits on page 68, the TMIFR1 bit should be set if the user requires a CRC byte to be appended following the last byte of the Type 3 IFR, and TMIFR0 if no CRC byte is required.

  Setting the TMIFR1 or TMIFR0 bit will direct the BDLC module to transmit the byte in the BDR as the first byte of a single or multi-byte IFR preceded by the appropriate Normalization Bit. Once this has occurred, the DLCBSVR will reflect that the next byte of the IFR can be written to the DLCBDR (TDRE interrupt).

NOTE: *The user must set the TMIFR1 or TMIFR0 bit before the EOD following the main part of the message frame is received, or no IFR transmit attempts will be made for the current message. If another node does transmit an IFR to this message or a reception error occurs, the TMIFR1 or TMIFR0 bit will be cleared. If not, the IFR will be transmitted after the EOD of the next received message.*

– Step 3: When TDRE is Indicated, Write the Next IFR Byte into the DLCBDR

  When a TDRE state is reflected in the DLCBSVR, the CPU writes the next IFR byte to be transmitted into the DLCBDR, clearing the TDRE interrupt. This step is repeated until the last IFR byte to be transmitted is written to the DLCBDR.

NOTE: *As when transmitting a message, when transmitting a Type 3 IFR the user may write two, or possibly even three of the bytes to be transmitted into the DLCBDR before the first RxIFR interrupt occurs. For this reason, the user should never use receive IFR byte interrupts to control the sequencing of IFR bytes to be transmitted.*

– Step 4: Write the Last IFR Byte into the DLCBDR and Set TEOD

  Once the last IFR byte to be transmitted is written to the DLCBDR, the CPU then sets the TEOD bit in DLCBCR2. Once the TEOD bit is set, after the last IFR byte written to the DLCBDR is transmitted onto the bus, if the TMIFR1 bit has been set the BDLC module will begin

transmitting the CRC byte, followed by an EOD. If the TMIFR0 bit has been set, the last IFR byte will immediately be followed by the transmission of an EOD. Following the EOD, and EOF will be recognized and the message will be complete.

If at any time during the transmission of a Type 3 IFR a loss of arbitration occurs, the TMIFR bit which is set and the TEOD bit (if set) will be cleared, any IFR byte being transmitted will be discarded and the loss of arbitration state will be reflected in the DLCBSVR. Likewise, if an error is detected during the transmission of a Type 3 IFR the IFR control bits will be cleared, the byte being transmitted will be discarded and the DLCBSVR will reflect the detected error.

NOTE: *If the Type 3 IFR being transmitted is made up of a single byte, the appropriate TMIFR bit and the TEOD bit can be set at the same time. The BDLC module will then treat that byte as both the first and last IFR byte to be sent.*

4.6.7 Transmitting IFR Exceptions

This basic IFR transmitting flow can be interrupted for the same reasons as a normal message transmission. The IFR transmit process can be adversely affected due to a loss of arbitration, an Invalid or Out of Range Symbol, or due to a transmitter underrun caused by the CPU failing to service a TDRE interrupt in a timely fashion. For a description of how these exceptions can affect the IFR transmit process, refer to Section 4.4.2 Transmitting Exceptions on page 60.

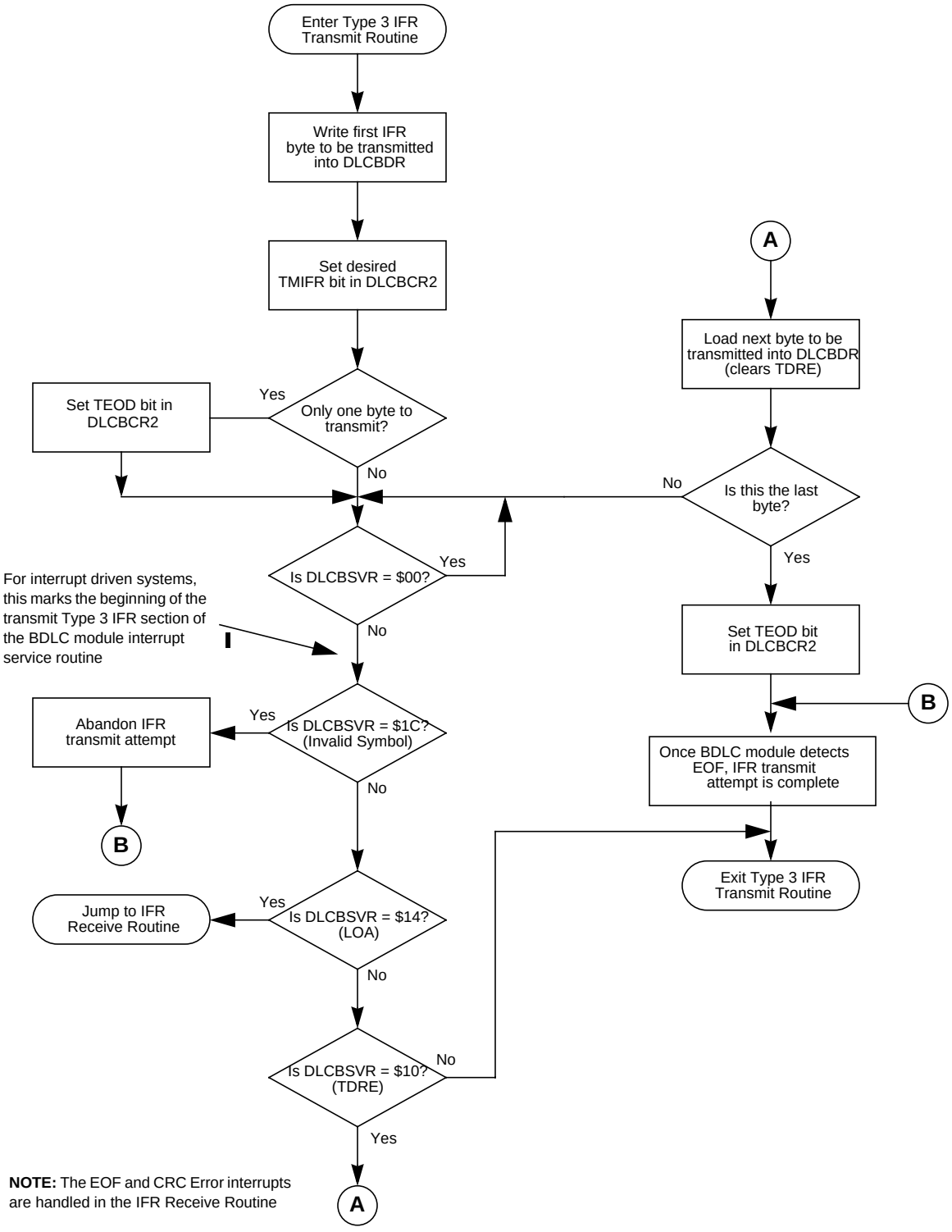


Figure 4-14 Transmitting A Type 3 IFR

4.7 Receiving An In-Frame Response (IFR)

Receiving an In-Frame Response with the BDLC module is very similar to receiving a message frame. As each byte of an IFR is received, the DLCBSVR will indicate this to the CPU. An EOF indication in the DLCBSVR indicates that the IFR (and message) is complete. Also, the IMSG bit can also be used to command the BDLC module to mask any further network activity from the CPU, including IFR bytes being received, until the next valid SOF is received.

NOTE: *As with a message transmission, the IMSG bit should never be used to ignore the BDLC module's own IFR transmissions. This is again due to the DLCBSVR bits being inhibited from updating until IMSG is cleared, preventing the CPU from detecting any IFR-related state changes which may be of interest.*

4.7.1 Receiving an IFR with the BDLC module

Receiving an IFR from the SAE J1850 bus requires the same procedure that receiving a message does, except that as each byte is received the Received IFR Byte (RxIFR) state is indicated in the DLCBSVR. All other actions are the same. For an illustration of the steps described below, refer to Figure 4-15 Receiving An IFR With the BDLC module on page 79.

- Step 1: When RxIFR Interrupt Occurs, Retrieve IFR Byte

  When the first byte of an IFR following a valid EOD symbol is received that byte is placed in the DLCBDR, and an RxIFR state is reflected in the DLCBSVR. No indication of the EOD reception in made, since the RxIFR state will indicate that the main portion of the message has ended and the IFR portion has begun.

  The RxIFR interrupt is cleared when the received IFR byte is read from the DLCBDR. Once this is done, no further CPU intervention is necessary until the next IFR byte is received, and this step is repeated. As with a message reception, all bytes of the IFR, including the CRC byte, will be placed into the DLCBDR as they are received for the CPU to retrieve.

- When an EOF is Received, the IFR (and Message) is Complete

  Once all IFR bytes (including the possible CRC byte) have been received from the bus, the bus will again be idle for a time period equal to an EOD symbol. Following this, the BDLC module will determine whether or not the last byte of the IFR is a CRC byte, and if so verify that the CRC byte is correct. If the CRC byte is not correct, this will be reflected in the DLCBSVR.

  After an additional period of time the EOD symbol will transition into an EOF symbol. When the EOF is received it will be reflected in the DLCBSVR, indicating to the user that the IFR, and the message, is complete.

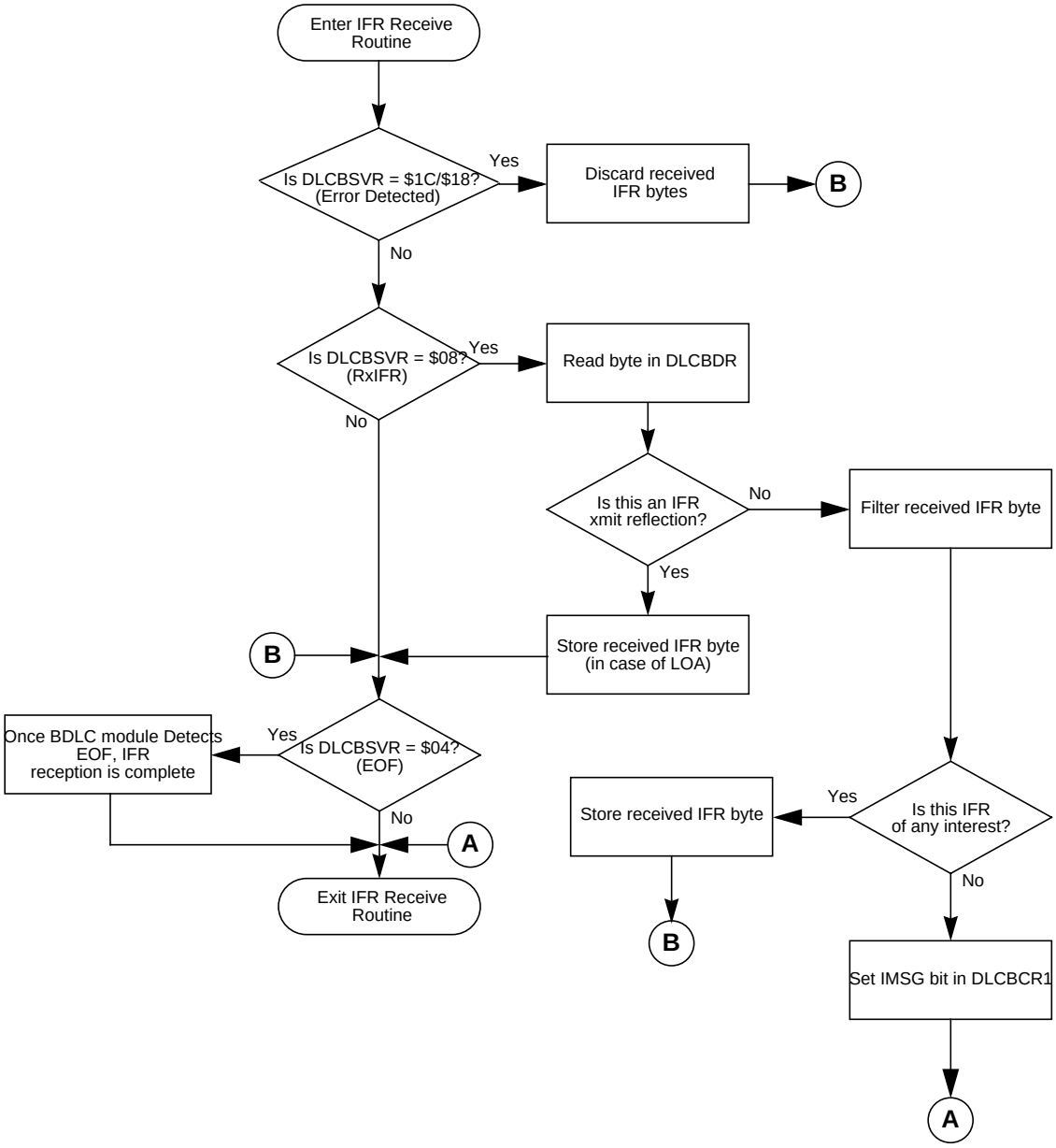


Figure 4-15 Receiving An IFR With the BDLC module

4.7.2 Receiving IFR Exceptions

This basic IFR receiving flow can be interrupted for the same reasons as a normal message reception. The IFR receiving process can be adversely affected due to a CRC error, an Invalid or Out of Range Symbol or due to a receiver overrun caused by the CPU failing to service an RxIFR interrupt in a timely fashion.

For a description of how these exceptions can affect the IFR receiving process, refer to Section 4.5.4 Receiving Exceptions on page 64.

4.8 Special BDLC Module Operations

There are a few special operations which the BDLC module can perform. What follows is a brief description of each of these functions and when they might be used.

4.8.1 Transmitting Or Receiving A Block Mode Message

The BDLC module, because it handles each message on a byte-by-byte basis, has the inherent capability of handling messages any number of bytes in length. While during normal operation this requires the user to carefully monitor message lengths to ensure compliance with SAE J1850 message limits, often in a production or diagnostic environment messages which exceed the SAE J1850 limits can be beneficial. This is especially true when large amounts of configuration data need to be downloaded over the SAE J1850 network.

Because of the BDLC module's architecture, it can both transmit and receive messages of unlimited length. The CRC calculations, both for transmitting and receiving, are not limited to eight bytes, but will instead be calculated and verified using all bytes in the message, regardless of the number. All control bits, including TEOD and IMSG, also work in an identical manner, regardless of the length of the message.

To transmit or receive these "Block Mode" messages, no extra BDLC module control functions must be performed. The user simply transmits or receives as many bytes as desired in one message frame, and the BDLC module will operate just as if a message of normal length was being used.

4.8.2 Receiving A Message In 4X Mode

In a diagnostic or production environment large amounts of data may need to be downloaded across the network to a component or module. This data is often sent in a large "Block Mode" message (see above) which violates the SAE J1850 limit for message length. In order to speed up the downloading of these large blocks of data, they are sometimes transmitted at four times (4X) the normal bit rate for the Variable Pulse Width modulation version of SAE J1850. This higher speed transmission, nominally 41.6kbps, allows these large blocks to be transmitted much more quickly.

The BDLC module is designed to receive (but not transmit) messages transmitted at this higher speed. By setting the RX4XE bit in DLCBCR2, the user can command the BDLC module to receive any message sent over the network at a 4X rate.

If the BDLC module is placed in this 4X mode, messages transmitted at the normal bit rate will not be received correctly. Likewise, 4X messages transmitted on the SAE J1850 bus when the BDLC module is in normal mode will be interpreted as noise on the network by the BDLC module. The RX4XE bit is not affected by entry or exit from BDLC module stop or wait modes. For more information on the RX4XE bit, refer to Section • 4X Mode on page 57.

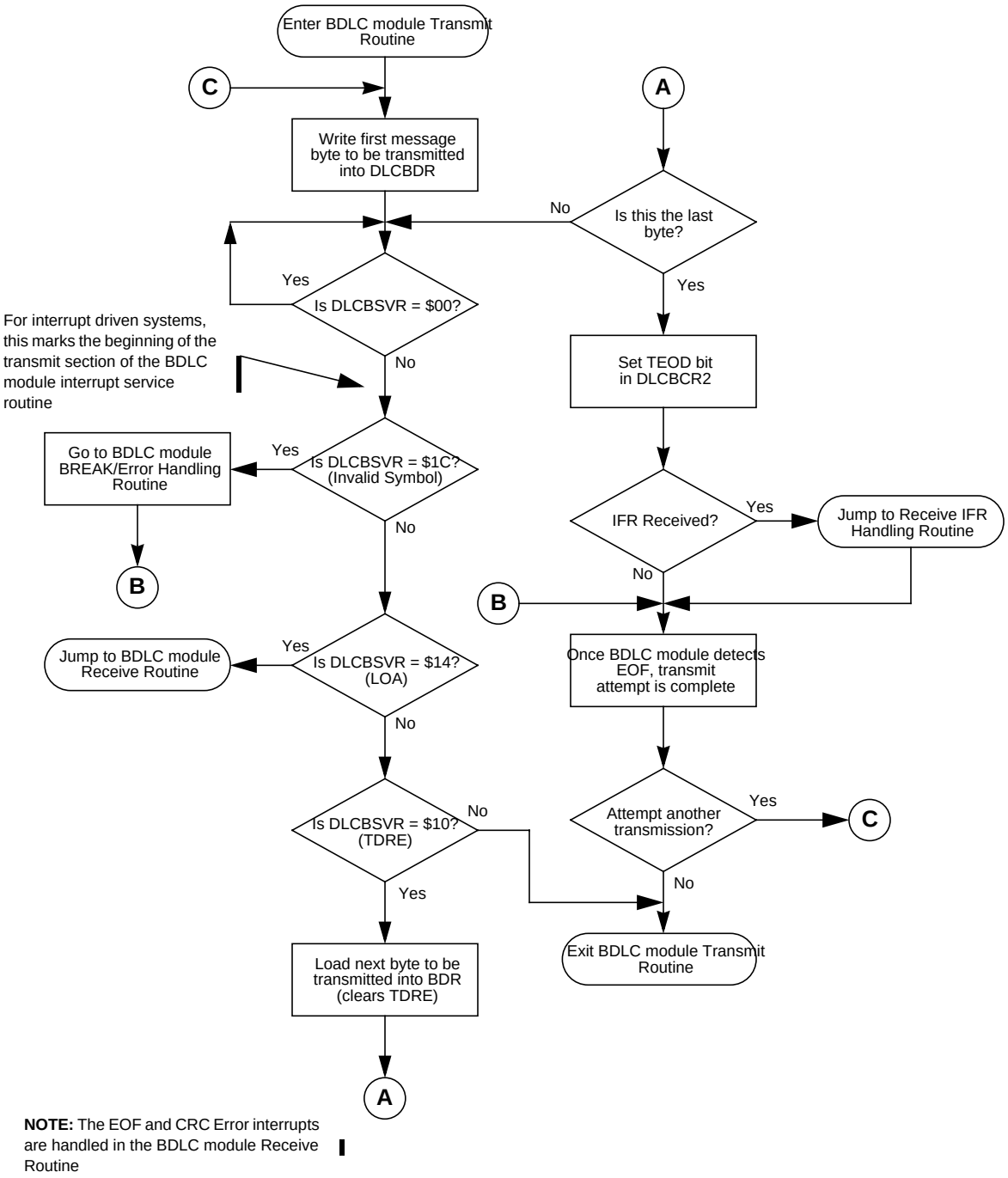


Figure 4-16 Basic BDLC Module Transmit Flowchart

4.9 BDLC Module Initialization

This section includes sample flows for initializing the BDLC module and using it to transmit and receive messages.

4.9.1 Initialization Sequence

To initialize the BDLC module, the user should first write the desired data to the configuration bits. The BDLC module should then be taken out of digital and analog loopback mode and enabled. Exiting from loopback mode will entail change of state indications in the DLCBSVR which must be dealt with. Once this is complete, CPU interrupts can be enabled (if desired), and then the BDLC module is capable of SAE J1850 serial network communication. For an illustration of the sequence necessary for initializing the BDLC module, refer to Figure 4-17 Basic BDLC Module Initialization Flowchart on page 85.

4.9.2 Initializing the Configuration Bits

The first step necessary for initializing the BDLC module following an MCU reset is to write the desired values to each of the BDLC module control registers. This is best done by storing predetermined initialization values directly into these registers. The following description outlines a basic flow for initializing the BDLC module. This basic flow does not detail more elaborate initialization routines, such as performing digital and analog loopback tests before enabling the BDLC module for SAE J1850 communication. However, from the following descriptions and the BDLC module specification, the user should be able to develop routines for performing various diagnostic procedures such as loopback tests.

- Step 1 - Initialize DLCBARD

  Begin initialization of the configuration bits by writing the desired analog transceiver configuration data into the DLCBARD register. Following this write to DLCBARD, all of these bits will become read only.

- Step 2- Initialize DLCBRSR

  The next step in BDLC module initialization is to write the desired bus clock divisor minus one into the DLCBRSR register. The divisor should be chosen to generate a 1 MHz or 1.048576 MHz mux interface clock (f_{bdlc}). Following this write to DLCBRSR, all of these bits will become read only.

- Step 3- Initialize DLCBCR2

  The next step in BDLC module initialization should be writing the configuration bits into the DLCBCR2 register. This initialization description assumes that the BDLC module will be put into normal mode (not 4X mode), and that the BDLC module should not yet exit either digital or analog loopback mode. Therefore, this step should write SMRST and DLOOP as logic ones, RX4XE as a logic zero, write NBFS to the desired level, and write TEOD, TSIFR, TMIFR1 and TMIFR0 as logic zeros. These last four bits MUST be written as logic zeros in order to prevent undesired operation of the BDLC module.

- Step 4- Initialize DLCBCR1

  The next step in BDLC module initialization is to write the configuration bits in DLCBCR1. The CLKS bit should be written to its desired values at this time, following which it will become read-only. The IE bit should be written as a logic zero at this time so BDLC module interrupts of the CPU will remain masked for the time being. The IMSG bit should be written as a logic one to prevent any receive events from setting the DLCBSVR until a valid SOF (or BREAK) symbol has been received by the BDLC module.

4.9.3 Exiting Loopback Mode and Enabling the BDLC module

Once the configuration bits have been written to the desired values, the BDLC module should be taken out of loopback and connected to the SAE J1850 bus. This is done by clearing the DLOOP bit and then setting the BDLCE bit in the DLCSCR.

- Step 5- Perform Loopback Tests (optional)

  Once the BDLC module is configured for desired operation, the user may wish to perform digital and/or analog loopback tests to determine the integrity of the link to the SAE J1850 network. This would involve leaving the DLOOP bit (DLCBCR2) set, setting the BDLCE bit, preforming the desired loopback tests and finally exiting digital loopback mode by clearing DLOOP in the DLCBCR2.

- Step 6- Exit Loopback Mode and enable the BDLC module

  If loopback mode tests are not to be preformed the BDLC module can be removed from digital loopback mode by clearing the DLOOP bit. The BDLC module can then be enabled by setting the BDLCE bit in the DLCSCR.

  Once DLOOP is cleared and BDLCE is set, the BDLC module is ready for SAE J1850 communication. However, to ensure that the BDLC module does not attempt to receive a message already in progress or to transmit a message while another device is transmitting, the BDLC module must first observe an EOF symbol on the bus before the receiver will be activated. To activate the transmitter, the BDLC module will need to observe an Inter-Frame Separator symbol.

4.9.4 Enabling BDLC Interrupts

The final step in readying the BDLC module for proper communication is to clear any pending interrupt sources and then, if desired, enable BDLC module interrupts of the CPU.

- Step 7- Clear Pending BDLC Interrupts

  In order to ensure that the BDLC module does not immediately generate a CPU interrupt when interrupts are enabled, the user should read the DLCBSVR to determine if any BDLC module interrupt sources are pending before setting the IE bit in the BCR1. If the BSVR reads as a %00000000, no interrupts are pending and the user is free to enable BDLC interrupts, if desired.

  If the DLCBSVR indicates that an interrupt is pending, the user should perform whatever actions are necessary to clear the interrupt source before enabling the interrupts. Whether any interrupts are pending will depend primarily upon how much time passes between the exit from loopback modes and enabling the BDLC module and the enabling of interrupts. It is a good practice to always clear any source of interrupts before enabling interrupts on any MCU subsystem.

  If any interrupts are pending (DLCBSVR not %00000000), then each interrupt source should be dealt with accordingly. Once all of the interrupt sources have been dealt with, the DLCBSVR should read %00000000, and the user is then free to enable BDLC interrupts.

- Step 8- Enable BDLC Interrupts

The last step in initializing the BDLC module is to enable interrupts to the CPU, if so desired. This is done by simply setting the IE bit in the DLCBCR1. Following this, the BDLC module is ready for operating in interrupt mode. If the user chooses not to enable interrupts, the DLCBSVR must be polled periodically to ensure that state changes in the BDLC module are detected and dealt with appropriately.

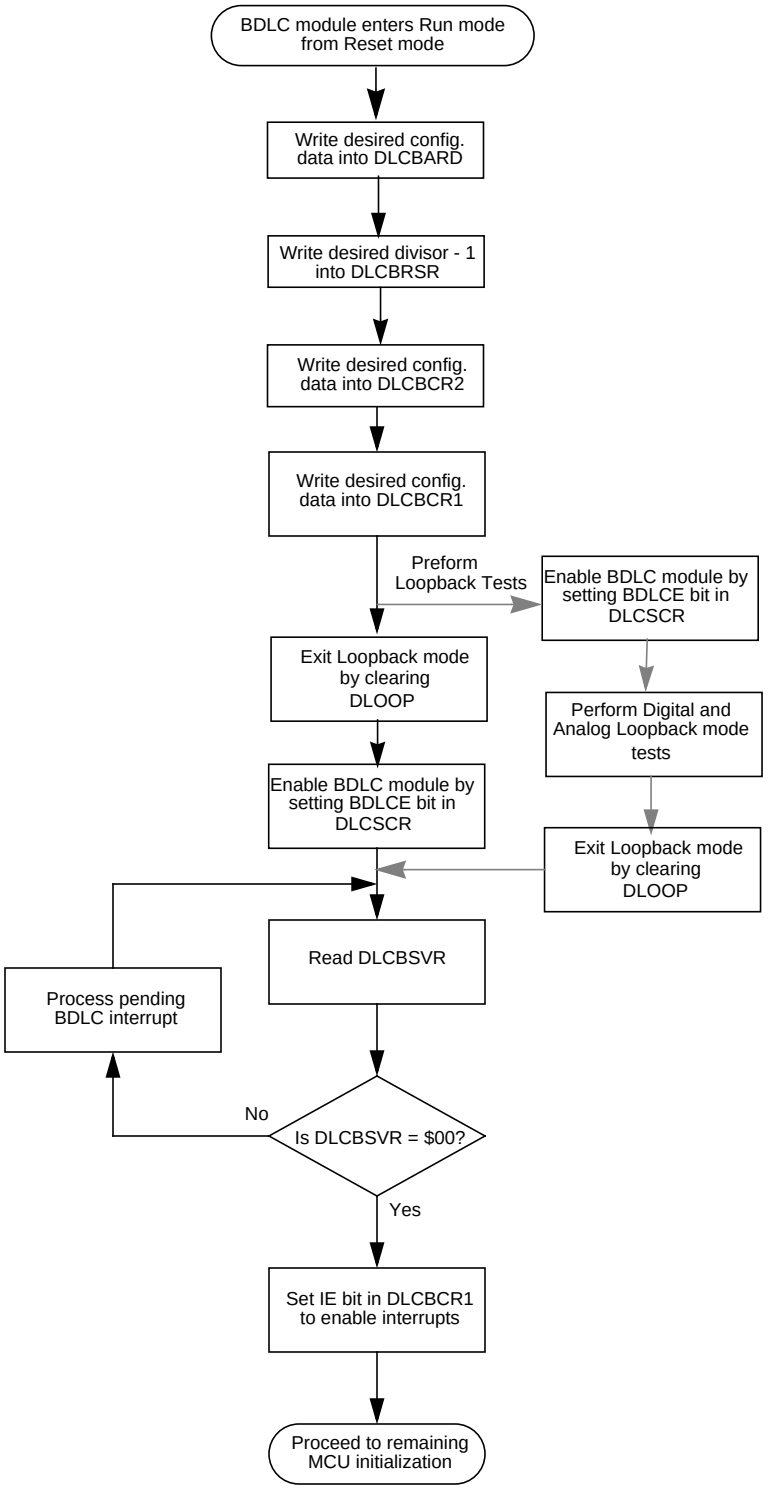


Figure 4-17 Basic BDLC Module Initialization Flowchart

Section 5 Resets

5.1 General

The reset state of each individual bit is listed within **Section 3 Memory Map and Registers** which details the registers and their bit-fields.

Section 6 Interrupts

6.1 General

Each change in status of the BDLC is encoded into the BDLC state vector register, (BSVR). Each state reflected in the BSVR can generate a CPU interrupt through the *ipi_bdlc_int* output, if the BDLC interrupts are enabled (IE = 1 in BCR1 Control register)

Table 6-1 shows this interrupt information for *ipi_bdlc_int*.

Table 6-1 Interrupt Summary

Interrupt	Interrupt Source	Priority
ipi_bdlc_int	Refer to table below	determined at chip-level

Refer to **3.3.2** for a listing of the interrupt sources.

Appendix A Electrical Specifications

N/A

User Guide End Sheet

FINAL PAGE OF
94
PAGES

MOTOROLA
intelligence everywhere™

digital dna™

Background Debug Module (BDM) V4

HCS12 Microcontrollers

S12BDMV4/D
Rev. 4.05
10/2004

MOTOROLA.COM/SEMICONDUCTORS

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
4.05	10/04/2004	10/04/2004		Added information for reserved register, which was added to solve MUCts entry "Possible manipulation of return address when exiting BDM active mode.".
+.1	11/15/2002	11/15/2002		A soft reset of the BDM and disable of the ACK function has been added during low power modes.
	10/31/2002	10/31/2002		Creation of block user guide from core user guide version 1.5 (Feb. 28, 2002). Changes include: updating format and adding a clock source table. The feature bullets were updated to more clearly show the differences from BDM2 - the predecessor of BDM3. Spell check now passes if conditional tags are turned off.
4.02	2/4/2003	2/4/2003		Original release

Table of Contents

List of Figures

List of Tables

Section 1 Introduction to Background Debug Module V4 (BDMV4)

Section 2 External Signal Description

Section 3 Memory Map/Register Definition

Section 4 Functional Description

Section 1 Introduction to Background Debug Module V4 (BDMV4)

This section describes the functionality of the Background Debug Module (BDM) sub-block of the HCS12 Core Platform.

A block diagram of the BDM is shown in **Figure 1-1**.

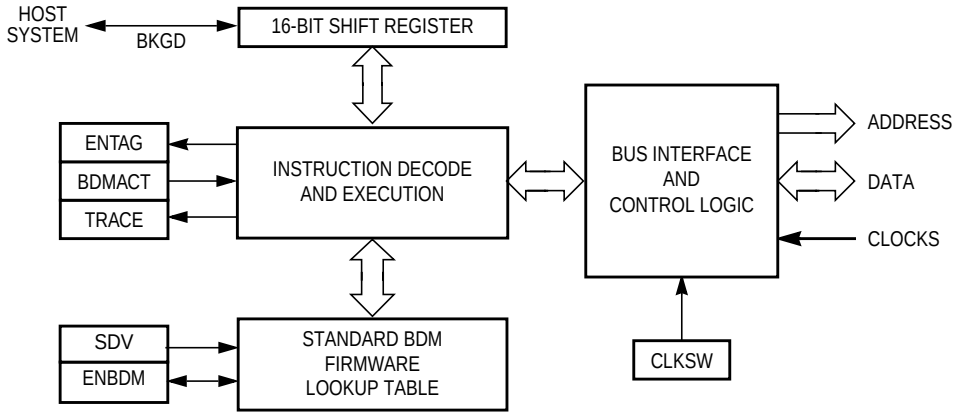


Figure 1-1 BDM Block Diagram

1.1 Overview

The Background Debug Module (BDM) sub-block is a single-wire, background debug system implemented in on-chip hardware for minimal CPU intervention. All interfacing with the BDM is done via the BKGD pin.

BDMV4 has enhanced capability for maintaining synchronization between the target and host while allowing more flexibility in clock rates. This includes a sync signal to show the clock rate and a handshake signal to indicate when an operation is complete. The system is backwards compatible with older external interfaces.

1.2 Features

- Single-wire communication with host development system
- BDMV4 (and BDM2): Enhanced capability for allowing more flexibility in clock rates
- BDMV4: SYNC command to determine communication rate
- BDMV4: GO_UNTIL command
- BDMV4: Hardware handshake protocol to increase the performance of the serial communication
- Active out of reset in special single-chip mode

- Nine hardware commands using free cycles, if available, for minimal CPU intervention
- Hardware commands not requiring active BDM
- 15 firmware commands execute from the standard BDM firmware lookup table
- Instruction tagging capability
- Software control of BDM operation during wait mode
- Software selectable clocks
- When secured, hardware commands are allowed to access the register space in Special Single-Chip mode, if the FLASH and EEPROM erase tests fail.

1.3 Modes of Operation

BDM is available in all operating modes but must be enabled before firmware commands are executed. Some system peripherals may have a control bit which allows suspending the peripheral function during background debug mode.

1.3.1 Regular Run Modes

All of these operations refer to the part in run mode. The BDM does not provide controls to conserve power during run mode.

- Normal Operation

  General operation of the BDM is available and operates the same in all normal modes.

- Special single-chip mode

  In special single-chip mode, background operation is enabled and active out of reset. This allows programming a system with blank memory.

- Special peripheral mode

  BDM is enabled and active immediately out of reset. BDM can be disabled by clearing the BDMACT bit in the BDM status (BDMSTS) register. The BDM serial system should not be used in special peripheral mode.

- Emulation Modes

  General operation of the BDM is available and operates the same as in normal modes.

1.3.2 Secure Mode Operation

If the part is in secure mode, the operation of the BDM is reduced to a small subset of it’s regular run mode operation. Secure operation prevents access to FLASH or EEPROM other than allowing erasure.

1.3.3 Low-Power Modes

- Wait Mode

  The BDM cannot be used in wait mode if the system disables the clocks to the BDM.

  There is a clearing mechanism associated with the WAIT instruction when the clocks to the BDM (CPU core platform) are disabled. As the clocks restart from wait mode, the BDM receives a soft reset (clearing any command in progress) and the ACK function will be disabled. This is a change from previous BDM modules.

- Stop Mode

  The BDM is completely shutdown in stop mode.

  There is a clearing mechanism associated with the STOP instruction. STOP must be enabled and the part must go into stop mode for this to occur. As the clocks restart from stop mode, the BDM receives a soft reset (clearing any command in progress) and the ACK function will be disabled. This is a change from previous BDM modules.

Section 2 External Signal Description

2.1 Overview

A single-wire interface pin is used to communicate with the BDM system. Two additional pins are used for instruction tagging. These pins are part of the Multiplexed External Bus Interface (MEBI) sub-block and all interfacing between the MEBI and BDM is done within the Core interface boundary. Functional descriptions of the pins are provided below for completeness.

- BKGD — Background interface pin
- $\overline{\text{TAGHI}}$ — High byte instruction tagging pin
- $\overline{\text{TAGLO}}$ — Low byte instruction tagging pin
- BKGD and $\overline{\text{TAGHI}}$ share the same pin.
- $\overline{\text{TAGLO}}$ and $\overline{\text{LSTRB}}$ share the same pin.

NOTE: *Generally these pins are shared as described, but it is best to check the chip description to make certain. All chips at the time of this writing have followed this pin sharing scheme.*

2.2 Detailed Signal Descriptions

2.2.1 Background Interface Pin (BKGD)

Debugging control logic communicates with external devices serially via the single-wire background interface pin (BKGD). During reset, this pin is a mode select input which selects between normal and special modes of operation. After reset, this pin becomes the dedicated serial interface pin for the background debug mode.

2.2.2 High Byte Instruction Tagging Pin ($\overline{\text{TAGHI}}$)

This pin is used to tag the high byte of an instruction. When instruction tagging is on, a logic 0 at the falling edge of the external clock (ECLK) tags the high half of the instruction word being read into the instruction queue.

2.2.3 Low Byte Instruction Tagging Pin ($\overline{\text{TAGLO}}$)

This pin is used to tag the low byte of an instruction. When instruction tagging is on and low strobe is enabled, a logic 0 at the falling edge of the external clock (ECLK) tags the low half of the instruction word being read into the instruction queue.

Section 3 Memory Map/Register Definition

A summary of the registers associated with the BDM is shown in **Table 3-1**. Registers are accessed by host-driven communications to the BDM hardware using READ_BD and WRITE_BD commands. Detailed descriptions of the registers and associated bits are given in the subsections that follow..

Table 3-1 BDM Register Map Summary

Address	Register Name		Bit 7	6	5	4	3	2	1	Bit 0
$FF00	Reserved	Read:	X	X	X	X	X	X	0	0
		Write:								
$FF01	BDMSTS	Read:	ENBDM	BDMACT	ENTAG	SDV	TRACE	CLKSW	UNSEC	0
		Write:								
$FF02	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								
$FF03	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								
$FF04	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								
$FF05	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								
$FF06	BDMCCR	Read:	CCR7	CCR6	CCR5	CCR4	CCR3	CCR2	CCR1	CCR0
		Write:								
$FF07	BDMINR	Read:	0	REG14	REG13	REG12	REG11	0	0	0
		Write:								
$FF08	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								
$FF09	Reserved	Read:	0	0	0	0	0	0	0	0
		Write:								

	= Unimplemented, Reserved		= Implemented (do not alter)
X	= Indeterminate	0	= Always read zero

Table 3-1 BDM Register Map Summary

Address	Register Name		Bit 7	6	5	4	3	2	1	Bit 0
$FF0A	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								
$FF0B	Reserved	Read:	X	X	X	X	X	X	X	X
		Write:								

= Unimplemented, Reserved
= Implemented (do not alter)
X = Indeterminate
0 = Always read zero

3.1 BDM Status Register

Register address $FF01

	7	6	5	4	3	2	1	0
R	ENBDM	BDMACT	ENTAG	SDV	TRACE	CLKSW	UNSEC	0
W								
Reset:								
Special single-chip mode:	1	1	0	0	0	0	0	0
Special peripheral mode:	0	1	0	0	0	0	0	0
All other modes:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved
= Implemented (do not alter)

Figure 3-1 BDM Status Register (BDMSTS)

Read: All modes through BDM operation

Write: All modes but subject to the following:

- BDMACT can only be set by BDM hardware upon entry into BDM. It can only be cleared by the standard BDM firmware lookup table upon exit from BDM active mode.
- CLKSW can only be written via BDM hardware or standard BDM firmware write commands.
- All other bits, while writable via BDM hardware or standard BDM firmware write commands, should only be altered by the BDM hardware or standard firmware lookup table as part of BDM command execution.
- ENBDM should only be set via a BDM hardware command if the BDM firmware commands are needed. (This does not apply in Special Single-Chip Mode).

ENBDM — Enable BDM

This bit controls whether the BDM is enabled or disabled. When enabled, BDM can be made active to allow firmware commands to be executed. When disabled, BDM cannot be made active but BDM hardware commands are still allowed.

1 = BDM enabled
0 = BDM disabled

NOTE: *ENBDM is set by the firmware immediately out of reset in special single-chip mode. In secure mode, this bit will not be set by the firmware until after the EEPROM and FLASH erase verify tests are complete.*

BDMACT — BDM active status

This bit becomes set upon entering BDM. The standard BDM firmware lookup table is then enabled and put into the memory map. BDMACT is cleared by a carefully timed store instruction in the standard BDM firmware as part of the exit sequence to return to user code and remove the BDM memory from the map.

1 = BDM active
0 = BDM not active

ENTAG — Tagging enable

This bit indicates whether instruction tagging in enabled or disabled. It is set when the TAGGO command is executed and cleared when BDM is entered. The serial system is disabled and the tag function enabled 16 cycles after this bit is written. BDM cannot process serial commands while tagging is active.

1 = Tagging enabled
0 = Tagging not enabled or BDM active

SDV — Shift data valid

This bit is set and cleared by the BDM hardware. It is set after data has been transmitted as part of a firmware read command or after data has been received as part of a firmware write command. It is cleared when the next BDM command has been received or BDM is exited. SDV is used by the standard BDM firmware to control program flow execution.

1 = Data phase of command is complete
0 = Data phase of command not complete

TRACE — TRACE1 BDM firmware command is being executed

This bit gets set when a BDM TRACE1 firmware command is first recognized. It will stay set as long as continuous back-to-back TRACE1 commands are executed. This bit will get cleared when the next command that is not a TRACE1 command is recognized.

1 = TRACE1 command is being executed
0 = TRACE1 command is not being executed

CLKSW — Clock switch

The CLKSW bit controls which clock the BDM operates with. It is only writable from a hardware BDM command. A 150 cycle delay at the clock speed that is active during the data portion of the command will occur before the new clock source is guaranteed to be active. The start of the next BDM command uses the new clock for timing subsequent BDM communications.

Table 3-2 shows the resulting BDM clock source based on the CLKSW and the PLLSEL (Pll select from the clock and reset generator) bits.

Table 3-2 BDM Clock Sources

PLLSEL	CLKSW	BDMCLK
0	0	Bus clock
0	1	Bus clock
1	0	Alternate clock (refer to the device specification to determine the alternate clock source)
1	1	Bus clock dependent on the PLL

NOTE: *The BDM alternate clock source can only be selected when CLKSW = 0 and PLLSEL = 1. The BDM serial interface is now fully synchronized to the alternate clock source, when enabled. This eliminates frequency restriction on the alternate clock which was required on previous versions. Refer to the device specification to determine which clock connects to the alternate clock source input.*

NOTE: *If the acknowledge function is turned on, changing the CLKSW bit will cause the ACK to be at the new rate for the write command which changes it.*

UNSEC — Unsecure

This bit is only writable in special single-chip mode from the BDM secure firmware and always gets reset to zero. It is in a zero state as secure mode is entered so that the secure BDM firmware lookup table is enabled and put into the memory map along with the standard BDM firmware lookup table.

The secure BDM firmware lookup table verifies that the on-chip EEPROM and FLASH EEPROM are erased. This being the case, the UNSEC bit is set and the BDM program jumps to the start of the standard BDM firmware lookup table and the secure BDM firmware lookup table is turned off. If the erase test fails, the UNSEC bit will not be asserted.

1 = System is in a unsecured mode
0 = System is in a secured mode

NOTE: *When UNSEC is set, security is off and the user can change the state of the secure bits in the on-chip FLASH EEPROM. Note that if the user does not change the state of the bits to “unsecured” mode, the system will be secured again when it is next taken out of reset.*

3.2 BDM CCR Holding Register

Register address $FF06

	7	6	5	4	3	2	1	0
R W	CCR7	CCR6	CCR5	CCR4	CCR3	CCR2	CCR1	CCR0
Reset:	0	0	0	0	0	0	0	0

Figure 3-2 BDM CCR Holding Register (BDMCCR)

Read: All modes

Write: All modes

NOTE: *When BDM is made active, the CPU stores the value of the CCR register in the BDMCCR register. However, out of special single-chip reset, the BDMCCR is set to $D8 and not $D0 which is the reset value of the CCR register.*

When entering background debug mode, the BDM CCR holding register is used to save the contents of the condition code register of the user's program. It is also used for temporary storage in the standard BDM firmware mode. The BDM CCR holding register can be written to modify the CCR value.

3.3 BDM Internal Register Position Register

Register address $FF07

	7	6	5	4	3	2	1	0
R	0	REG14	REG13	REG12	REG11	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 BDM Internal Register Position (BDMINR)

Read: All modes

Write: Never

REG14–REG11 — Internal register map position

These four bits show the state of the upper five bits of the base address for the system's relocatable register block. BDMINR is a shadow of the INITRG register which maps the register block to any 2K byte space within the first 32K bytes of the 64K byte address space.

Section 4 Functional Description

The BDM receives and executes commands from a host via a single wire serial interface. There are two types of BDM commands, namely, hardware commands and firmware commands.

Hardware commands are used to read and write target system memory locations and to enter active background debug mode, see **4.3 BDM Hardware Commands**. Target system memory includes all memory that is accessible by the CPU.

Firmware commands are used to read and write CPU resources and to exit from active background debug mode, see **4.4 Standard BDM Firmware Commands**. The CPU resources referred to are the accumulator (D), X index register (X), Y index register (Y), stack pointer (SP), and program counter (PC).

Hardware commands can be executed at any time and in any mode excluding a few exceptions as highlighted, see **4.3 BDM Hardware Commands**. Firmware commands can only be executed when the system is in active background debug mode (BDM).

4.1 Security

If the user resets into special single-chip mode with the system secured, a secured mode BDM firmware lookup table is brought into the map overlapping a portion of the standard BDM firmware lookup table. The secure BDM firmware verifies that the on-chip EEPROM and FLASH EEPROM are erased. This being the case, the UNSEC bit will get set. The BDM program jumps to the start of the standard BDM firmware and the secured mode BDM firmware is turned off and all BDM commands are allowed. If the EEPROM or FLASH do not verify as erased, the BDM firmware sets the ENBDM bit, without asserting UNSEC, and the firmware enters a loop. This causes the BDM hardware commands to become enabled, but does not enable the firmware commands. This allows the BDM hardware to be used to erase the EEPROM and FLASH. After execution of the secure firmware, regardless of the results of the erase tests, the CPU registers, INITEE and PPAGE, will no longer be in their reset state.

4.2 Enabling and Activating BDM

The system must be in active BDM to execute standard BDM firmware commands. BDM can be activated only after being enabled. BDM is enabled by setting the ENBDM bit in the BDM status (BDMSTS) register. The ENBDM bit is set by writing to the BDM status (BDMSTS) register, via the single-wire interface, using a hardware command such as WRITE_BD_BYTE.

After being enabled, BDM is activated by one of the following[1]:

- Hardware BACKGROUND command
- BDM external instruction tagging mechanism
- CPU BGND instruction
- Breakpoint sub-block's force or tag mechanism[2]

When BDM is activated, the CPU finishes executing the current instruction and then begins executing the firmware in the standard BDM firmware lookup table. When BDM is activated by the breakpoint sub-block, the type of breakpoint used determines if BDM becomes active before or after execution of the next instruction.

NOTE: *If an attempt is made to activate BDM before being enabled, the CPU resumes normal instruction execution after a brief delay. If BDM is not enabled, any hardware BACKGROUND commands issued are ignored by the BDM and the CPU is not delayed.*

In active BDM, the BDM registers and standard BDM firmware lookup table are mapped to addresses $FF00 to $FFFF. BDM registers are mapped to addresses $FF00 to $FF07. The BDM uses these registers which are readable anytime by the BDM. However, these registers are not readable by user programs.

4.3 BDM Hardware Commands

Hardware commands are used to read and write target system memory locations and to enter active background debug mode. Target system memory includes all memory that is accessible by the CPU such as on-chip RAM, EEPROM, FLASH EEPROM, I/O and control registers, and all external memory.

Hardware commands are executed with minimal or no CPU intervention and do not require the system to be in active BDM for execution, although, they can still be executed in this mode. When executing a hardware command, the BDM sub-block waits for a free CPU bus cycle so that the background access does not disturb the running application program. If a free cycle is not found within 128 clock cycles, the CPU is momentarily frozen so that the BDM can steal a cycle. When the BDM finds a free cycle, the operation does not intrude on normal CPU operation provided that it can be completed in a single cycle. However, if an operation requires multiple cycles the CPU is frozen until the operation is complete, even though the BDM found a free cycle.

The BDM hardware commands are listed in **Table 4-1**.

NOTES:
1. BDM is enabled and active immediately out of special single-chip reset.
2. This method is only available on systems that have a a Breakpoint or a Debug sub-block.

Table 4-1 Hardware Commands

Command	Opcode (hex)	Data	Description
BACKGROUND	90	None	Enter background mode if firmware is enabled. If enabled, an ACK will be issued when the part enters active background mode.
ACK_ENABLE	D5	None	Enable Handshake. Issues an ACK pulse after the command is executed.
ACK_DISABLE	D6	None	Disable Handshake. This command does not issue an ACK pulse.
READ_BD_BYTE	E4	16-bit address 16-bit data out	Read from memory with standard BDM firmware lookup table in map. Odd address data on low byte; even address data on high byte.
READ_BD_WORD	EC	16-bit address 16-bit data out	Read from memory with standard BDM firmware lookup table in map. Must be aligned access.
READ_BYTE	E0	16-bit address 16-bit data out	Read from memory with standard BDM firmware lookup table out of map. Odd address data on low byte; even address data on high byte.
READ_WORD	E8	16-bit address 16-bit data out	Read from memory with standard BDM firmware lookup table out of map. Must be aligned access.
WRITE_BD_BYTE	C4	16-bit address 16-bit data in	Write to memory with standard BDM firmware lookup table in map. Odd address data on low byte; even address data on high byte.
WRITE_BD_WORD	CC	16-bit address 16-bit data in	Write to memory with standard BDM firmware lookup table in map. Must be aligned access.
WRITE_BYTE	C0	16-bit address 16-bit data in	Write to memory with standard BDM firmware lookup table out of map. Odd address data on low byte; even address data on high byte.
WRITE_WORD	C8	16-bit address 16-bit data in	Write to memory with standard BDM firmware lookup table out of map. Must be aligned access.

NOTE:
If enabled, ACK will occur when data is ready for transmission for all BDM READ commands and will occur after the write is complete for all BDM WRITE commands.

The READ_BD and WRITE_BD commands allow access to the BDM register locations. These locations are not normally in the system memory map but share addresses with the application in memory. To distinguish between physical memory locations that share the same address, BDM memory resources are enabled just for the READ_BD and WRITE_BD access cycle. This allows the BDM to access BDM locations unobtrusively, even if the addresses conflict with the application memory map.

4.4 Standard BDM Firmware Commands

Firmware commands are used to access and manipulate CPU resources. The system must be in active BDM to execute standard BDM firmware commands, see **4.2 Enabling and Activating BDM**. Normal instruction execution is suspended while the CPU executes the firmware located in the standard BDM firmware lookup table. The hardware command BACKGROUND is the usual way to activate BDM.

As the system enters active BDM, the standard BDM firmware lookup table and BDM registers become visible in the on-chip memory map at $FF00–$FFFF, and the CPU begins executing the standard BDM firmware. The standard BDM firmware watches for serial commands and executes them as they are received.

The firmware commands are shown in **Table 4-2**.

Table 4-2 Firmware Commands

Command[1]	Opcode (hex)	Data	Description
READ_NEXT	62	16-bit data out	Increment X by 2 (X = X + 2), then read word X points to.
READ_PC	63	16-bit data out	Read program counter.
READ_D	64	16-bit data out	Read D accumulator.
READ_X	65	16-bit data out	Read X index register.
READ_Y	66	16-bit data out	Read Y index register.
READ_SP	67	16-bit data out	Read stack pointer.
WRITE_NEXT	42	16-bit data in	Increment X by 2 (X = X + 2), then write word to location pointed to by X.
WRITE_PC	43	16-bit data in	Write program counter.
WRITE_D	44	16-bit data in	Write D accumulator.
WRITE_X	45	16-bit data in	Write X index register.
WRITE_Y	46	16-bit data in	Write Y index register.
WRITE_SP	47	16-bit data in	Write stack pointer.
GO	08	none	Go to user program. If enabled, ACK will occur when leaving active background mode.
GO_UNTIL[2]	0C	none	Go to user program. If enabled, ACK will occur upon returning to active background mode.
TRACE1	10	none	Execute one user instruction then return to active BDM. If enabled, ACK will occur upon returning to active background mode.
TAGGO	18	none	Enable tagging and go to user program. There is no ACK pulse related to this command.

NOTES:
1. If enabled, ACK will occur when data is ready for transmission for all BDM READ commands and will occur after the write is complete for all BDM WRITE commands.
2. Both WAIT (with clocks to the S12 CPU core disabled) and STOP disable the ACK function. The GO_UNTIL command will not get an Acknowledge if one of these two CPU instructions occurs before the "UNTIL" instruction. This can be a problem for any instruction that uses ACK, but GO_UNTIL is a lot more difficult for the development tool to time-out.

4.5 BDM Command Structure

Hardware and firmware BDM commands start with an 8-bit opcode followed by a 16-bit address and/or a 16-bit data word depending on the command. All the read commands return 16 bits of data despite the byte or word implication in the command name.

NOTE: *8-bit reads return 16-bits of data, of which, only one byte will contain valid data. If reading an even address, the valid data will appear in the MSB. If reading an odd address, the valid data will appear in the LSB.*

NOTE: *16-bit misaligned reads and writes are not allowed. If attempted, the BDM will ignore the least significant bit of the address and will assume an even address from the remaining bits.*

For hardware data read commands, the external host must wait 150 bus clock cycles after sending the address before attempting to obtain the read data. This is to be certain that valid data is available in the BDM shift register, ready to be shifted out. For hardware write commands, the external host must wait 150 bus clock cycles after sending the data to be written before attempting to send a new command. This is to avoid disturbing the BDM shift register before the write has been completed. The 150 bus clock cycle delay in both cases includes the maximum 128 cycle delay that can be incurred as the BDM waits for a free cycle before stealing a cycle.

For firmware read commands, the external host should wait 44 bus clock cycles after sending the command opcode and before attempting to obtain the read data. This includes the potential of an extra 7 cycles when the access is external with a narrow bus access (+1 cycle) and / or a stretch (+1, 2, or 3 cycles), (7 cycles could be needed if both occur). The 44 cycle wait allows enough time for the requested data to be made available in the BDM shift register, ready to be shifted out.

NOTE: *This timing has increased from previous BDM modules due to the new capability in which the BDM serial interface can potentially run faster than the bus. On previous BDM modules this extra time could be hidden within the serial time.*

For firmware write commands, the external host must wait 32 bus clock cycles after sending the data to be written before attempting to send a new command. This is to avoid disturbing the BDM shift register before the write has been completed.

The external host should wait 64 bus clock cycles after a TRACE1 or GO command before starting any new serial command. This is to allow the CPU to exit gracefully from the standard BDM firmware lookup table and resume execution of the user code. Disturbing the BDM shift register prematurely may adversely affect the exit from the standard BDM firmware lookup table.

NOTE: *If the bus rate of the target processor is unknown or could be changing, it is recommended that the ACK (acknowledge function) be used to indicate when an operation is complete. When using ACK, the delay times are automated.*

Figure 4-1 represents the BDM command structure. The command blocks illustrate a series of eight bit times starting with a falling edge. The bar across the top of the blocks indicates that the BKGD line idles in the high state. The time for an 8-bit command is 8 × 16 target clock cycles.(1)

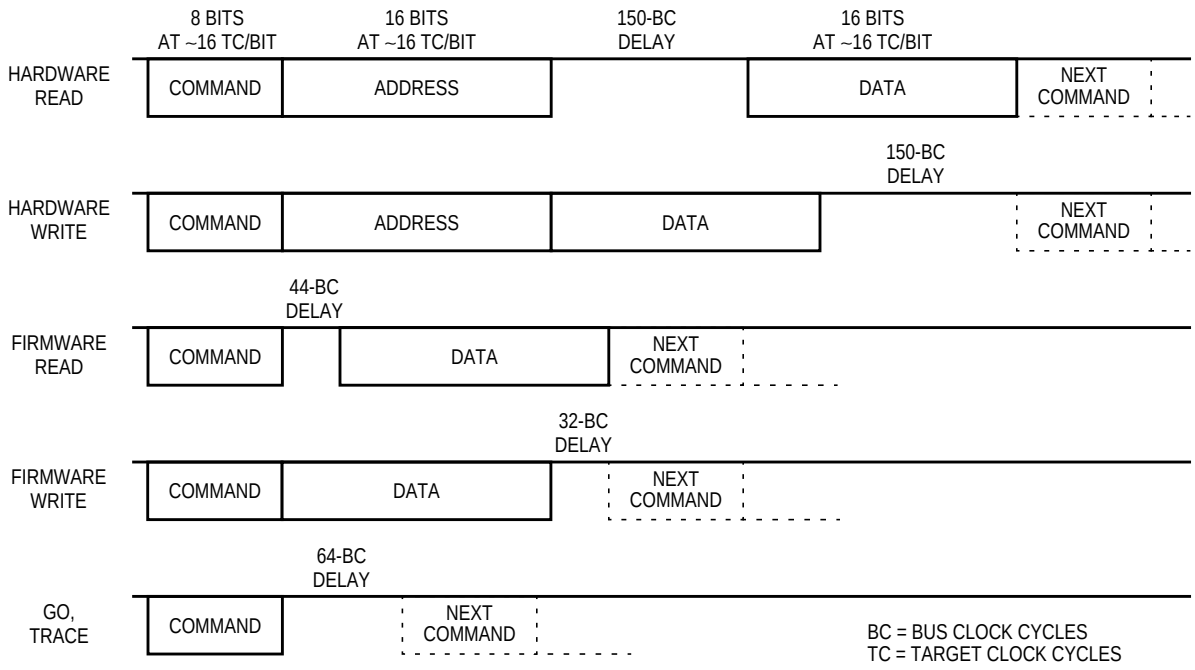


Figure 4-1 BDM Command Structure

4.6 BDM Serial Interface

The BDM communicates with external devices serially via the BKGD pin. During reset, this pin is a mode select input which selects between normal and special modes of operation. After reset, this pin becomes the dedicated serial interface pin for the BDM.

The BDM serial interface is timed using the clock selected by the CLKSW bit in the status register see **3.1 BDM Status Register**. This clock will be referred to as the target clock in the following explanation.

The BDM serial interface uses a clocking scheme in which the external host generates a falling edge on the BKGD pin to indicate the start of each bit time. This falling edge is sent for every bit whether data is transmitted or received. Data is transferred most significant bit (MSB) first at 16 target clock cycles per bit. The interface times out if 512 clock cycles occur between falling edges from the host.

The BKGD pin is a pseudo open-drain pin and has an weak on-chip active pull-up that is enabled at all times. It is assumed that there is an external pull-up and that drivers connected to BKGD do not typically drive the high level. Since R-C rise time could be unacceptably long, the target system and host provide

NOTES:
1. Target clock cycles are cycles measured using the target MCU's serial clock rate. See **4.6 BDM Serial Interface** and **3.1 BDM Status Register** for information on how serial clock rate is selected.

brief driven-high (speedup) pulses to drive BKGD to a logic 1. The source of this speedup pulse is the host for transmit cases and the target for receive cases.

The timing for host-to-target is shown in **Figure 4-2** and that of target-to-host in **Figure 4-3** and **Figure 4-4**. All four cases begin when the host drives the BKGD pin low to generate a falling edge. Since the host and target are operating from separate clocks, it can take the target system up to one full clock cycle to recognize this edge. The target measures delays from this perceived start of the bit time while the host measures delays from the point it actually drove BKGD low to start the bit up to one target clock cycle earlier. Synchronization between the host and target is established in this manner at the start of every bit time.

Figure 4-2 shows an external host transmitting a logic 1 and transmitting a logic 0 to the BKGD pin of a target system. The host is asynchronous to the target, so there is up to a one clock-cycle delay from the host-generated falling edge to where the target recognizes this edge as the beginning of the bit time. Ten target clock cycles later, the target senses the bit level on the BKGD pin. Internal glitch detect logic requires the pin be driven high no later that eight target clock cycles after the falling edge for a logic 1 transmission.

Since the host drives the high speedup pulses in these two cases, the rising edges look like digitally driven signals.

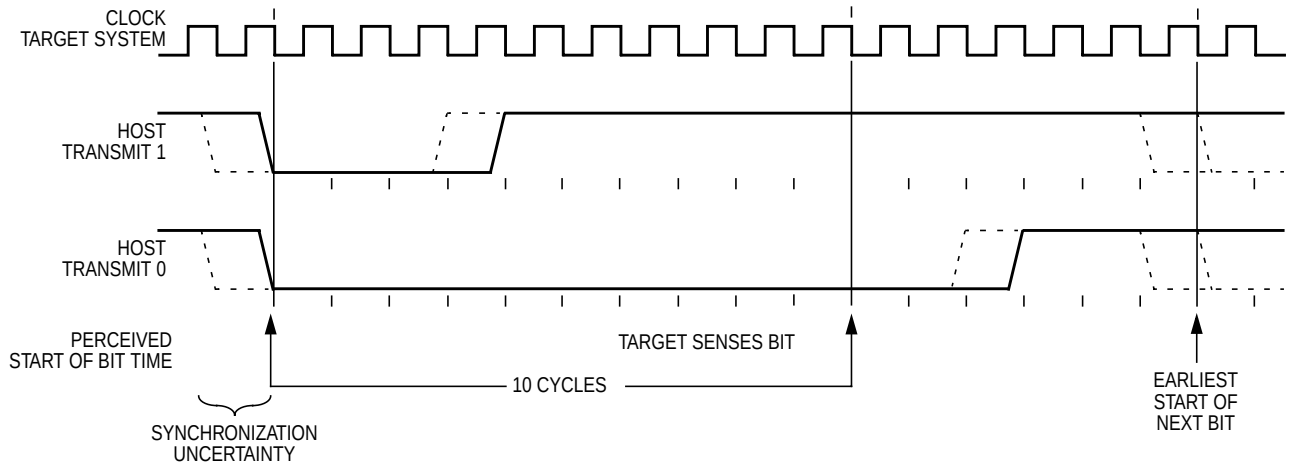


Figure 4-2 BDM Host-to-Target Serial Bit Timing

The receive cases are more complicated. **Figure 4-3** shows the host receiving a logic 1 from the target system. Since the host is asynchronous to the target, there is up to one clock-cycle delay from the host-generated falling edge on BKGD to the perceived start of the bit time in the target. The host holds the BKGD pin low long enough for the target to recognize it (at least two target clock cycles). The host must release the low drive before the target drives a brief high speedup pulse seven target clock cycles after the perceived start of the bit time. The host should sample the bit level about 10 target clock cycles after it started the bit time.

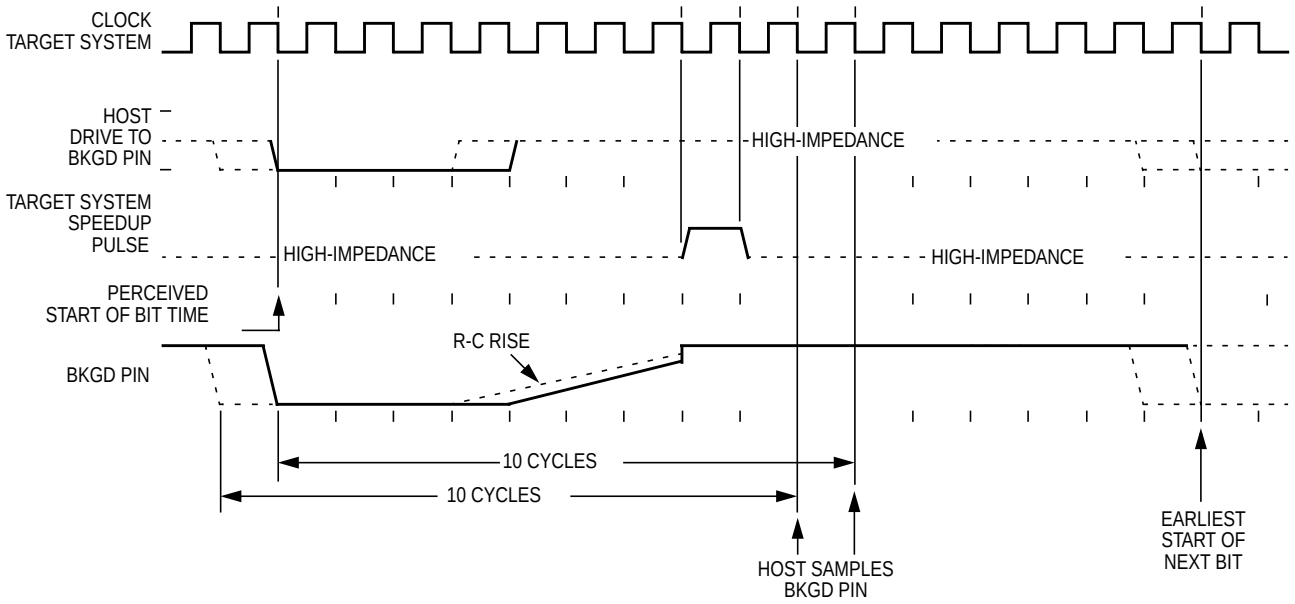


Figure 4-3 BDM Target-to-Host Serial Bit Timing (Logic 1)

Figure 4-4 shows the host receiving a logic 0 from the target. Since the host is asynchronous to the target, there is up to a one clock-cycle delay from the host-generated falling edge on BKGD to the start of the bit time as perceived by the target. The host initiates the bit time but the target finishes it. Since the target wants the host to receive a logic 0, it drives the BKGD pin low for 13 target clock cycles then briefly drives it high to speed up the rising edge. The host samples the bit level about 10 target clock cycles after starting the bit time.

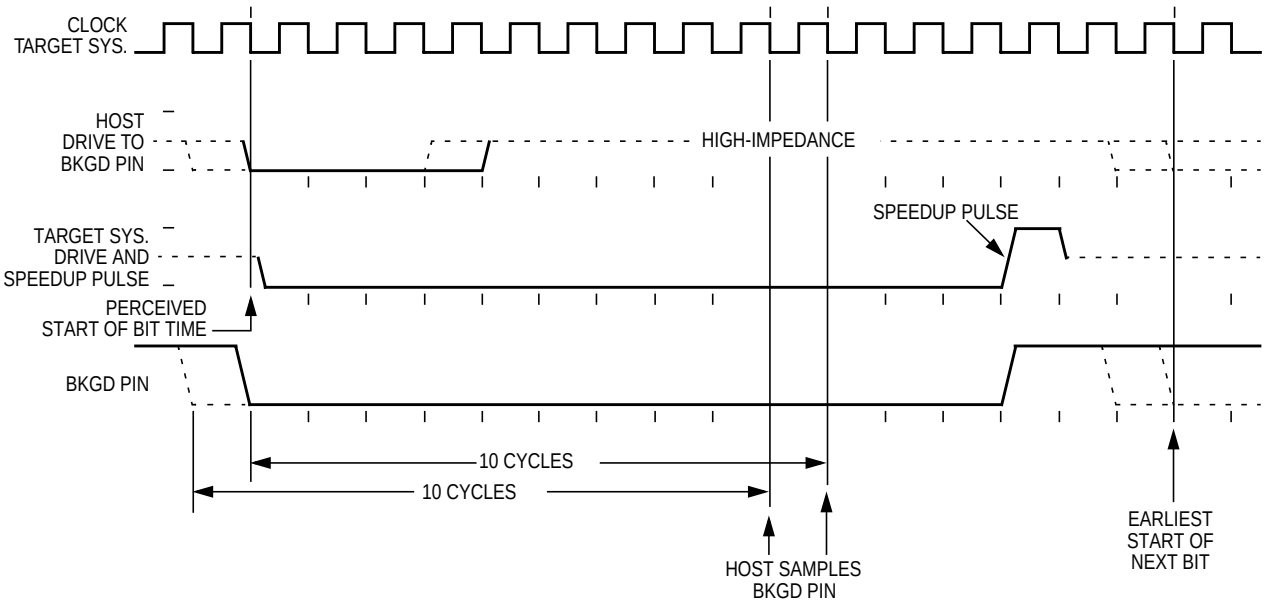


Figure 4-4 BDM Target-to-Host Serial Bit Timing (Logic 0)

4.7 Serial Interface Hardware Handshake Protocol

BDM commands that require CPU execution are ultimately treated at the MCU bus rate. Since the BDM clock source can be asynchronously related to the bus frequency, when CLKSW = 0, it is very helpful to provide a handshake protocol in which the host could determine when an issued command is executed by the CPU. The alternative is to always wait the amount of time equal to the appropriate number of cycles at the slowest possible rate the clock could be running. This sub-section will describe the hardware handshake protocol.

The hardware handshake protocol signals to the host controller when an issued command was successfully executed by the target. This protocol is implemented by a 16 serial clock cycle low pulse followed by a brief speedup pulse in the BKGD pin. This pulse is generated by the target MCU when a command, issued by the host, has been successfully executed (see **Figure 4-5**). This pulse is referred to as the ACK pulse. After the ACK pulse has finished: the host can start the bit retrieval if the last issued command was a read command, or start a new command if the last command was a write command or a control command (BACKGROUND, GO, GO_UNTIL or TRACE1). The ACK pulse is not issued earlier than 32 serial clock cycles after the BDM command was issued. The end of the BDM command is assumed to be the 16th tick of the last bit. This minimum delay assures enough time for the host to perceive the ACK pulse. Note also that, there is no upper limit for the delay between the command and the related ACK pulse, since the command execution depends upon the CPU bus frequency, which in some cases could be very slow

compared to the serial communication rate. This protocol allows a great flexibility for the POD designers, since it does not rely on any accurate time measurement or short response time to any event in the serial communication.

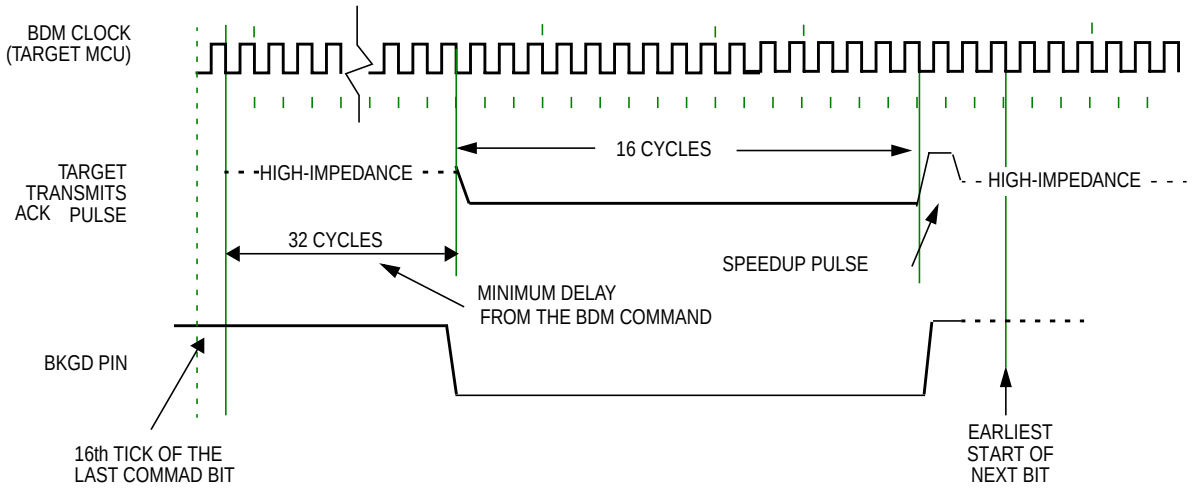


Figure 4-5 Target Acknowledge Pulse (ACK)

NOTE: *If the ACK pulse was issued by the target, the host assumes the previous command was executed. If the CPU enters WAIT or STOP prior to executing a hardware command, the ACK pulse will not be issued meaning that the BDM command was not executed. After entering wait or stop mode, the BDM command is no longer pending.*

Figure 4-6 shows the ACK handshake protocol in a command level timing diagram. The READ_BYTE instruction is used as an example. First, the 8-bit instruction opcode is sent by the host, followed by the address of the memory location to be read. The target BDM decodes the instruction. A bus cycle is grabbed (free or stolen) by the BDM and it executes the READ_BYTE operation. Having retrieved the data, the BDM issues an ACK pulse to the host controller, indicating that the addressed byte is ready to be retrieved. After detecting the ACK pulse, the host initiates the byte retrieval process. Note that data is sent in the form of a word and the host needs to determine which is the appropriate byte based on whether the address was odd or even.

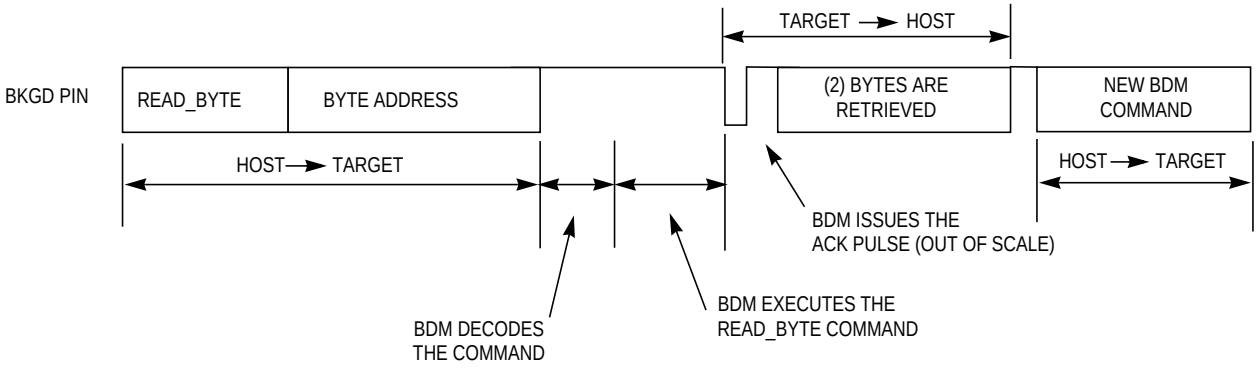


Figure 4-6 Handshake Protocol at Command Level

Differently from the normal bit transfer (where the host initiates the transmission), the serial interface ACK handshake pulse is initiated by the target MCU by issuing a negedge in the BKGD pin. The hardware handshake protocol in **Figure 4-5** specifies the timing when the BKGD pin is being driven, so the host should follow this timing constraint in order to avoid the risk of an electrical conflict in the BKGD pin.

NOTE: *The only place the BKGD pin can have an electrical conflict is when one side is driving low and the other side is issuing a speedup pulse (high). Other "highs" are pulled rather than driven. However, at low rates the time of the speedup pulse can become lengthy and so the potential conflict time becomes longer as well.*

The ACK handshake protocol does not support nested ACK pulses. If a BDM command is not acknowledge by an ACK pulse, the host needs to abort the pending command first in order to be able to issue a new BDM command. When the CPU enters WAIT or STOP while the host issues a command that requires CPU execution (e.g., WRITE_BYTE), the target discards the incoming command due to the WAIT or STOP being detected. Therefore, the command is not acknowledged by the target, which means that the ACK pulse will not be issued in this case. After a certain time the host should decide to abort the ACK sequence in order to be free to issue a new command. Therefore, the protocol should provide a mechanism in which a command, and therefore a pending ACK, could be aborted.

NOTE: *Differently from a regular BDM command, the ACK pulse does not provide a time out. This means that in the case of a WAIT or STOP instruction being executed, the ACK would be prevented from being issued. If not aborted, the ACK would remain pending indefinitely. See the handshake abort procedure described in* ***4.8 Hardware Handshake Abort Procedure****.*

4.8 Hardware Handshake Abort Procedure

The abort procedure is based on the SYNC command. In order to abort a command, which had not issued the corresponding ACK pulse, the host controller should generate a low pulse in the BKGD pin by driving it low for at least 128 serial clock cycles and then driving it high for one serial clock cycle, providing a speedup pulse. By detecting this long low pulse in the BKGD pin, the target executes the SYNC protocol, see **4.9 SYNC — Request Timed Reference Pulse**, and assumes that the pending command and therefore the related ACK pulse, are being aborted. Therefore, after the SYNC protocol has been completed the host is free to issue new BDM commands.

Although it is not recommended, the host could abort a pending BDM command by issuing a low pulse in the BKGD pin shorter than 128 serial clock cycles, which will not be interpreted as the SYNC command. The ACK is actually aborted when a negedge is perceived by the target in the BKGD pin. The short abort pulse should have at least 4 clock cycles keeping the BKGD pin low, in order to allow the negedge to be detected by the target. In this case, the target will not execute the SYNC protocol but the pending command will be aborted along with the ACK pulse. The potential problem with this abort procedure is when there is a conflict between the ACK pulse and the short abort pulse. In this case, the target may not perceive the abort pulse. The worst case is when the pending command is a read command (i.e., READ_BYTE). If the abort pulse is not perceived by the target the host will attempt to send a new command after the abort pulse was issued, while the target expects the host to retrieve the accessed memory byte. In this case, host and target will run out of synchronism. However, if the command to be aborted is not a read command the short

abort pulse could be used. After a command is aborted the target assumes the next negedge, after the abort pulse, is the first bit of a new BDM command.

NOTE: *The details about the short abort pulse are being provided only as a reference for the reader to better understand the BDM internal behavior. It is not recommended that this procedure be used in a real application.*

Since the host knows the target serial clock frequency, the SYNC command (used to abort a command) does not need to consider the lower possible target frequency. In this case, the host could issue a SYNC very close to the 128 serial clock cycles length. Providing a small overhead on the pulse length in order to assure the SYNC pulse will not be misinterpreted by the target. See **4.9 SYNC — Request Timed Reference Pulse**.

Figure 4-7 shows a SYNC command being issued after a READ_BYTE, which aborts the READ_BYTE command. Note that, after the command is aborted a new command could be issued by the host computer.

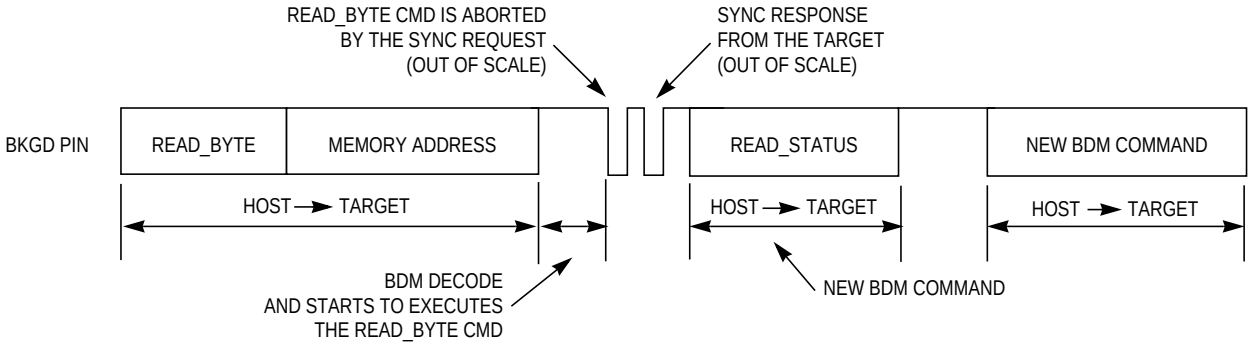


Figure 4-7 ACK Abort Procedure at the Command Level

NOTE: ***Figure 4-7*** *does not represent the signals in a true timing scale*

Figure 4-8 shows a conflict between the ACK pulse and the SYNC request pulse. This conflict could occur if a POD device is connected to the target BKGD pin and the target is already in debug active mode. Consider that the target CPU is executing a pending BDM command at the exact moment the POD is being connected to the BKGD pin. In this case, an ACK pulse is issued along with the SYNC command. In this case, there is an electrical conflict between the ACK speedup pulse and the SYNC pulse. Since this is not a probable situation, the protocol does not prevent this conflict from happening.

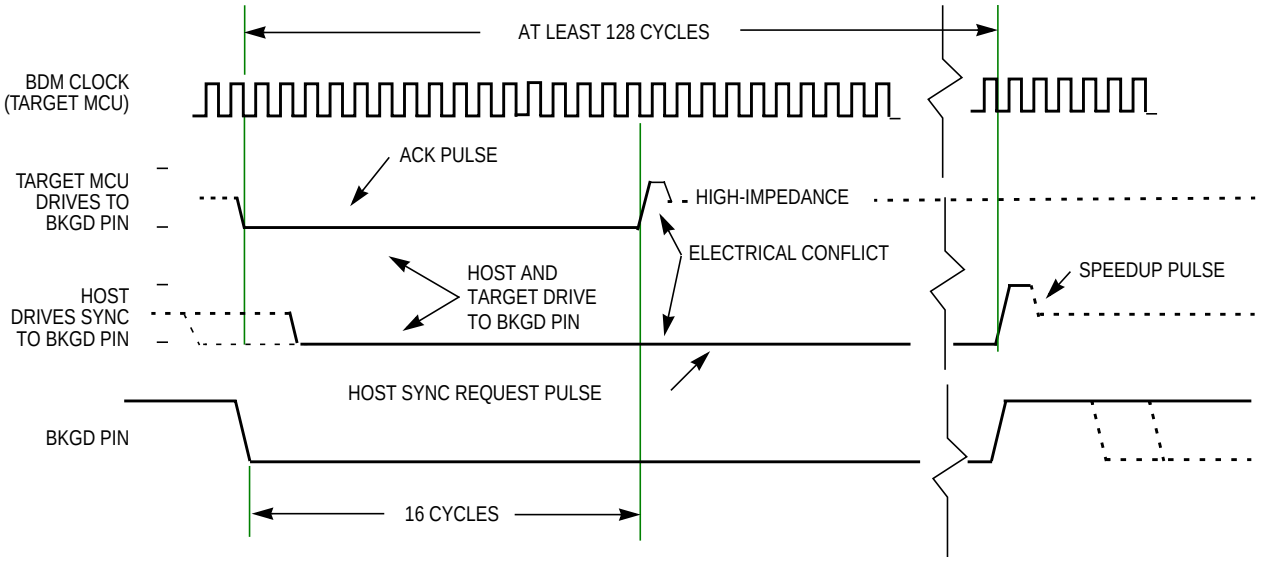


Figure 4-8 ACK Pulse and SYNC Request Conflict

NOTE: *This information is being provided so that the MCU integrator will be aware that such a conflict could eventually occur.*

The hardware handshake protocol is enabled by the ACK_ENABLE and disabled by the ACK_DISABLE BDM commands. This provides backwards compatibility with the existing POD devices which are not able to execute the hardware handshake protocol. It also allows for new POD devices, that support the hardware handshake protocol, to freely communicate with the target device. If desired, without the need for waiting for the ACK pulse.

The commands are described as follows:

- ACK_ENABLE — enables the hardware handshake protocol. The target will issue the ACK pulse when a CPU command is executed by the CPU. The ACK_ENABLE command itself also has the ACK pulse as a response.
- ACK_DISABLE — disables the ACK pulse protocol. In this case, the host needs to use the worst case delay time at the appropriate places in the protocol.

The default state of the BDM after reset is hardware handshake protocol disabled.

All the read commands will ACK (if enabled) when the data bus cycle has completed and the data is then ready for reading out by the BKGD serial pin. All the write commands will ACK (if enabled) after the data has been received by the BDM through the BKGD serial pin and when the data bus cycle is complete. See **4.3 BDM Hardware Commands** and **4.4 Standard BDM Firmware Commands** for more information on the BDM commands.

The ACK_ENABLE sends an ACK pulse when the command has been completed. This feature could be used by the host to evaluate if the target supports the hardware handshake protocol. If an ACK pulse is issued in response to this command, the host knows that the target supports the hardware handshake protocol. If the target does not support the hardware handshake protocol the ACK pulse is not issued. In this case, the ACK_ENABLE command is ignored by the target since it is not recognized as a valid command.

The BACKGROUND command will issue an ACK pulse when the CPU changes from normal to background mode. The ACK pulse related to this command could be aborted using the SYNC command.

The GO command will issue an ACK pulse when the CPU exits from background mode. The ACK pulse related to this command could be aborted using the SYNC command.

The GO_UNTIL command is equivalent to a GO command with exception that the ACK pulse, in this case, is issued when the CPU enters into background mode. This command is an alternative to the GO command and should be used when the host wants to trace if a breakpoint match occurs and causes the CPU to enter active background mode. Note that the ACK is issued whenever the CPU enters BDM, which could be caused by a Breakpoint match or by a BGND instruction being executed. The ACK pulse related to this command could be aborted using the SYNC command.

The TRACE1 command has the related ACK pulse issued when the CPU enters background active mode after one instruction of the application program is executed. The ACK pulse related to this command could be aborted using the SYNC command.

The TAGGO command will not issue an ACK pulse since this would interfere with the tagging function shared on the same pin.

4.9 SYNC — Request Timed Reference Pulse

The SYNC command is unlike other BDM commands because the host does not necessarily know the correct communication speed to use for BDM communications until after it has analyzed the response to the SYNC command. To issue a SYNC command, the host should perform the following steps:

1. Drive the BKGD pin low for at least 128 cycles at the lowest possible BDM serial communication frequency (the lowest serial communication frequency is determined by the crystal oscillator or the clock chosen by CLKSW.)
2. Drive BKGD high for a brief speedup pulse to get a fast rise time (this speedup pulse is typically one cycle of the host clock.)
3. Remove all drive to the BKGD pin so it reverts to high impedance.
4. Listen to the BKGD pin for the sync response pulse.

Upon detecting the SYNC request from the host, the target performs the following steps:

1. Discards any incomplete command received or bit retrieved.
2. Waits for BKGD to return to a logic one.
3. Delays 16 cycles to allow the host to stop driving the high speedup pulse.
4. Drives BKGD low for 128 cycles at the current BDM serial communication frequency.
5. Drives a one-cycle high speedup pulse to force a fast rise time on BKGD.
6. Removes all drive to the BKGD pin so it reverts to high impedance.

The host measures the low time of this 128 cycle SYNC response pulse and determines the correct speed for subsequent BDM communications. Typically, the host can determine the correct communication speed within a few percent of the actual target speed and the communication protocol can easily tolerate speed errors of several percent.

As soon as the SYNC request is detected by the target, any partially received command or bit retrieved is discarded. This is referred to as a soft-reset, equivalent to a time-out in the serial communication. After the SYNC response, the target will consider the next negedge (issued by the host) as the start of a new BDM command or the start of new SYNC request.

Another use of the SYNC command pulse is to abort a pending ACK pulse. The behavior is exactly the same as in a regular SYNC command. Note that one of the possible causes for a command to not be acknowledged by the target is a host-target synchronization problem. In this case, the command may not have been understood by the target and so an ACK response pulse will not be issued.

4.10 Instruction Tracing

When a TRACE1 command is issued to the BDM in active BDM, the CPU exits the standard BDM firmware and executes a single instruction in the user code. Once this has occurred, the CPU is forced to return to the standard BDM firmware and the BDM is active and ready to receive a new command. If the TRACE1 command is issued again, the next user instruction will be executed. This facilitates stepping or tracing through the user code one instruction at a time.

If an interrupt is pending when a TRACE1 command is issued, the interrupt stacking operation occurs but no user instruction is executed. Once back in standard BDM firmware execution, the program counter points to the first instruction in the interrupt service routine.

4.11 Instruction Tagging

The instruction queue and cycle-by-cycle CPU activity are reconstructible in real time or from trace history that is captured by a logic analyzer. However, the reconstructed queue cannot be used to stop the CPU at a specific instruction. This is because execution already has begun by the time an operation is visible outside the system. A separate instruction tagging mechanism is provided for this purpose.

The tag follows program information as it advances through the instruction queue. When a tagged instruction reaches the head of the queue, the CPU enters active BDM rather than executing the instruction.

NOTE: *Tagging is disabled when BDM becomes active and BDM serial commands are not processed while tagging is active.*

Executing the BDM TAGGO command configures two system pins for tagging. The $\overline{\text{TAGLO}}$ signal shares a pin with the $\overline{\text{LSTRB}}$ signal, and the $\overline{\text{TAGHI}}$ signal shares a pin with the BKGD signal.

Table 4-3 shows the functions of the two tagging pins. The pins operate independently, that is the state of one pin does not affect the function of the other. The presence of logic level 0 on either pin at the fall of the external clock (ECLK) performs the indicated function. High tagging is allowed in all modes. Low tagging is allowed only when low strobe is enabled (LSTRB is allowed only in wide expanded modes and emulation expanded narrow mode).

Table 4-3 Tag Pin Function

$\overline{\text{TAGHI}}$	$\overline{\text{TAGLO}}$	Tag
1	1	No tag
1	0	Low byte
0	1	High byte
0	0	Both bytes

4.12 Serial Communication Time-out

The host initiates a host-to-target serial transmission by generating a falling edge on the BKGD pin. If BKGD is kept low for more than 128 target clock cycles, the target understands that a SYNC command was issued. In this case, the target will keep waiting for a rising edge on BKGD in order to answer the SYNC request pulse. If the rising edge is not detected, the target will keep waiting forever without any time-out limit.

Consider now the case where the host returns BKGD to logic one before 128 cycles. This is interpreted as a valid bit transmission, and not as a SYNC request. The target will keep waiting for another falling edge marking the start of a new bit. If, however, a new falling edge is not detected by the target within 512 clock cycles since the last falling edge, a time-out occurs and the current command is discarded without affecting memory or the operating mode of the MCU. This is referred to as a soft-reset.

If a read command is issued but the data is not retrieved within 512 serial clock cycles, a soft-reset will occur causing the command to be disregarded. The data is not available for retrieval after the time-out has occurred. This is the expected behavior if the handshake protocol is not enabled. However, consider the behavior where the BDC is running in a frequency much greater than the CPU frequency. In this case, the command could time out before the data is ready to be retrieved. In order to allow the data to be retrieved even with a large clock frequency mismatch (between BDC and CPU) when the hardware handshake protocol is enabled, the time out between a read command and the data retrieval is disabled. Therefore, the host could wait for more then 512 serial clock cycles and still be able to retrieve the data from an issued read command. However, once the handshake pulse (ACK pulse) is issued, the time-out feature is re-activated, meaning that the target will time out after 512 clock cycles. Therefore, the host needs to retrieve the data within a 512 serial clock cycles time frame after the ACK pulse had been issued. After that period, the read command is discarded and the data is no longer available for retrieval. Any negedge in the BKGD pin after the time-out period is considered to be a new command or a SYNC request.

Note that whenever a partially issued command, or partially retrieved data, has occurred the time out in the serial communication is active. This means that if a time frame higher than 512 serial clock cycles is observed between two consecutive negative edges and the command being issued or data being retrieved is not complete, a soft-reset will occur causing the partially received command or data retrieved to be disregarded. The next negedge in the BKGD pin, after a soft-reset has occurred, is considered by the target as the start of a new BDC command, or the start of a SYNC request pulse.

2/2003

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

MOTOROLA
intelligence everywhere™

digital dna™

Breakpoint (BKP) Module V1

Block User Guide

HCS12 Microcontrollers

S12BKPV1/D
Rev. 1.02
5/2003

MOTOROLA.COM/SEMICONDUCTORS

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
1.02	5/1/2003	5/1/2003	John Langan	Original release

Table of Contents

List of Figures

List of Tables

Section 1 Introduction: Breakpoint (BKP) Module

Section 2 External Signal Description

Section 3 Memory Map/Register Definition

Section 4 Functional Description

Section 1 Introduction: Breakpoint (BKP) Module

This section describes the functionality of the Breakpoint (BKP) sub-block of the HCS12 Core Platform.

A block diagram of the Breakpoint sub-block is shown in **Figure 1-1**. The Breakpoint contains three main sub-blocks: the Register Block, the Compare Block, and the Control Block. The Register Block consists of the eight registers that make up the Breakpoint register space. The Compare Block performs all required address and data signal comparisons. The Control Block generates the signals for the CPU for the tag high, tag low, force SWI, and force BDM functions. In addition, it generates the register read and write signals and the comparator block enable signals.

NOTE: *There is a two-cycle latency for address compares and for forces, a two-cycle latency for write data compares, and a three-cycle latency for read data compares.*

1.1 Overview

The Breakpoint sub-block of the Core Platform provides for hardware breakpoints that are used to debug software on the CPU by comparing actual address and data values to predetermine data in setup registers. A successful comparison will place the CPU in Background Debug Mode or initiate a software interrupt (SWI). The choice between Background Debug Mode and SWI is software selectable.

There are two types of breakpoints, forced and tagged. Forced breakpoints occur at the next instruction boundary if a match occurs and tagged breakpoints allow for breaking just before a specific instruction executes. Tagged breakpoints will only occur on addresses of program fetches. Tagging on data is not allowed; however, if this occurs nothing will happen within the BKP.

The range function of the BKP allows breaking within a 256-byte address range. The page function allows breaking within expanded memory. In data matching operations, 8-bit or 16-bit data can be matched. Forced breakpoints are mainly used on a read or a write cycle, but can be used on any bus cycle.

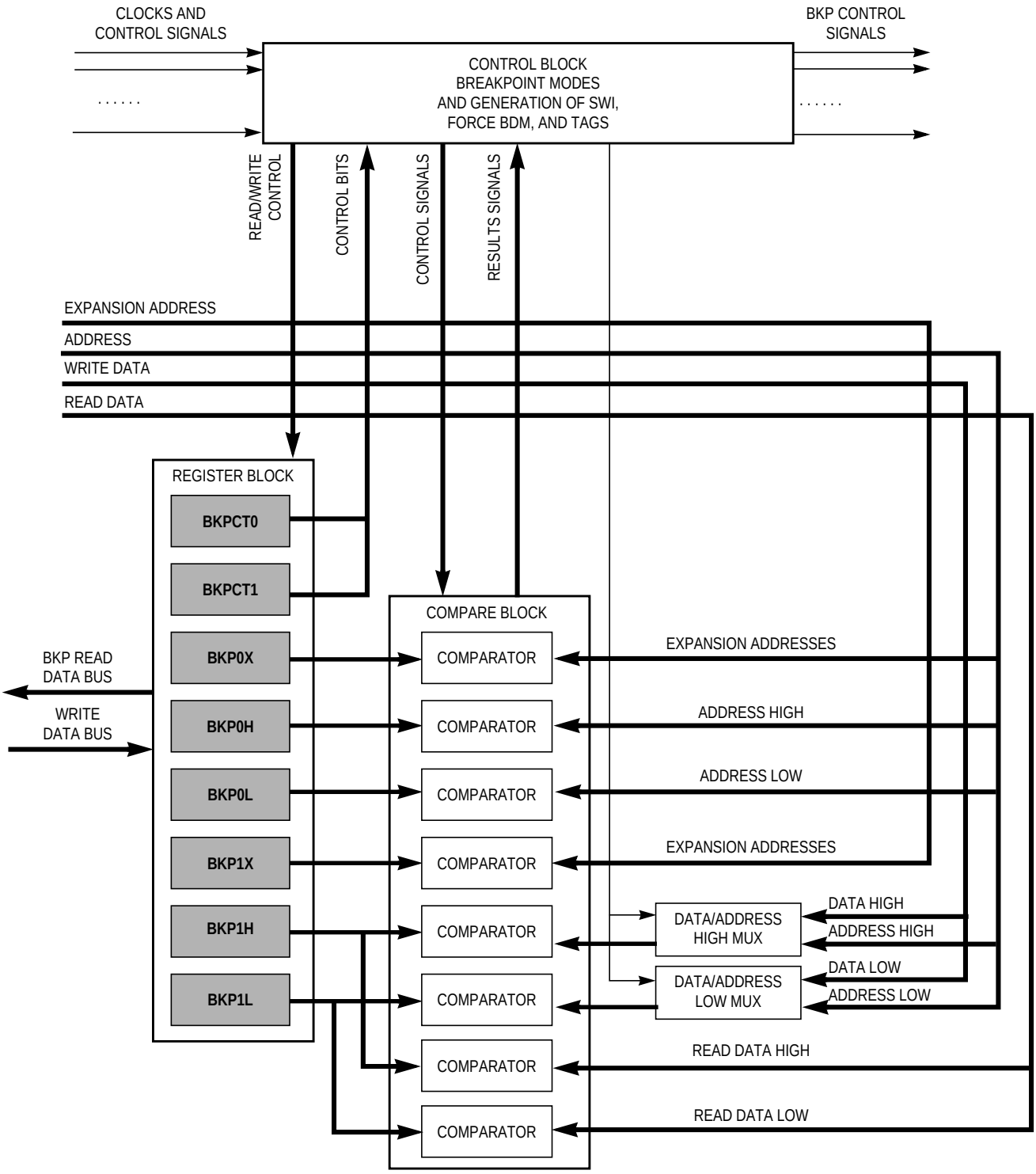


Figure 1-1 Breakpoint Block Diagram

1.2 Features

- Full or Dual Breakpoint Mode
  - Compare on address and data (Full)
  - Compare on either of two addresses (Dual)
- BDM or SWI Breakpoint
  - Enter BDM on breakpoint (BDM)
  - Execute SWI on breakpoint (SWI)
- Tagged or Forced Breakpoint
  - Break just before a specific instruction will begin execution (TAG)
  - Break on the first instruction boundary after a match occurs (Force)
- Single, Range, or Page address compares
  - Compare on address (Single)
  - Compare on address 256 byte (Range)
  - Compare on any 16K Page (Page)
- Compare address on read or write on forced breakpoints
- High and/or low byte data compares

1.3 Modes of Operation

The Breakpoint sub-block contains two modes of operation:

1. Dual Address Mode, where a match on either of two addresses will cause the system to enter Background Debug Mode or initiate a Software Interrupt (SWI).
2. Full Breakpoint Mode, where a match on address and data will cause the system to enter Background Debug Mode or initiate a Software Interrupt (SWI).

Section 2 External Signal Description

The breakpoint sub-module relies on the external bus interface (generally the MEBI) when the breakpoint is matching on the external bus.

The tag pins in **Table 2-1** (part of the MEBI) may also be a part of the breakpoint operation.

Table 2-1 External System Pins Associated With Breakpoint and MEBI

Pin Name	Pin Functions	Description
BKGD/MODC/ $\overline{\text{TAGHI}}$	$\overline{\text{TAGHI}}$	When instruction tagging is on, a 0 at the falling edge of PE4/ECLK tags the high half of the instruction word being read into the instruction queue.
PE3/$\overline{\text{LSTRB}}$/ $\overline{\text{TAGLO}}$	$\overline{\text{TAGLO}}$	In expanded wide mode or emulation narrow modes, when instruction tagging is on and low strobe is enabled, a 0 at the falling edge of PE4/ECLK tags the low half of the instruction word being read into the instruction queue.

Section 3 Memory Map/Register Definition

A summary of the registers associated with the Breakpoint sub-block is shown in **Figure 3-1**. Detailed descriptions of the registers and bits are given in the subsections that follow.

Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
$0028	BKPCT0	Read	BKEN	BKFULL	BKBDM	BKTAG	0	0	0	0
		Write								
$0029	BKPCT1	Read	BK0MBH	BK0MBL	BK1MBH	BK1MBL	BK0RWE	BK0RW	BK1RWE	BK1RW
		Write								
$002A	BKP0X	Read	0	0	BK0V5	BK0V4	BK0V3	BK0V2	BK0V1	BK0V0
		Write								
$002B	BKP0H	Read	Bit 15	14	13	12	11	10	9	Bit 8
		Write								
$002C	BKP0L	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$002D	BKP1X	Read	0	0	BK1V5	BK1V4	BK1V3	BK1V2	BK1V1	BK1V0
		Write								
$002E	BKP1H	Read	Bit 15	14	13	12	11	10	9	Bit 8
		Write								
$002F	BKP1L	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								

= Unimplemented X = Indeterminate

Figure 3-1 Breakpoint Register Summary

3.1 Breakpoint Control Register 0 (BKPCT0)

Read: anytime

Write: anytime

Register address $0028

	7	6	5	4	3	2	1	0
R	BKEN	BKFULL	BKBDM	BKTAG	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 Breakpoint Control Register 0 (BKPCT0)

This register is used to set the breakpoint modes.

BKEN — Breakpoint Enable

This bit enables the module
0 = Breakpoints disabled
1 = Breakpoints enabled, breakpoint mode is determined by bits BKFULL, BKBDM, and BKTAG

BKFULL — Full Breakpoint Mode Enable

This bit controls whether the breakpoint module is in Dual Mode or Full Mode
0 = Dual Address Mode enabled
1 = Full Breakpoint Mode enabled

BKBDM — Breakpoint Background Debug Mode Enable

This bit determines if the breakpoint causes the system to enter Background Debug Mode (BDM) or initiate a Software Interrupt (SWI)
0 = Go to Software Interrupt on a compare
1 = Go to BDM on a compare

BKTAG — Breakpoint on Tag

This bit controls whether the breakpoint will cause a break on the next instruction boundary (force) or on a match that will be an executable opcode (tagged). Non-executed opcodes cannot cause a tagged breakpoint
0 = On match, break at the next instruction boundary (force)
1 = On match, break if the match is an instruction that will be executed (tagged)

3.2 Breakpoint Control Register 1 (BKPCT1)

Read: anytime

Write: anytime

Register address $0029

	7	6	5	4	3	2	1	0
R W	BK0MBH	BK0MBL	BK1MBH	BK1MBL	BK0RWE	BK0RW	BK1RWE	BK1RW
Reset:	0	0	0	0	0	0	0	0

Figure 3-3 Breakpoint Control Register 1 (BKPCT1)

This register is used to configure the functionality of the Breakpoint sub-block within the Core.

BK0MBH:BK0MBL — Breakpoint Mask High Byte and Low Byte for First Address

In Dual or Full Mode, these bits may be used to mask (disable) the comparison of the high and low bytes of the first address breakpoint. The functionality is as given in **Table 3-1** below

Table 3-1 Breakpoint Mask Bits for First Address

BK0MBH:BK0MBL	Address Compare	BKP0X	BKP0H	BKP0L
x:0	Full address compare	Yes(1)	Yes	Yes
0:1	256 byte address range	Yes(1)	Yes	No
1:1	16K byte address range	Yes(1)	No	No

NOTES:
1. If page is selected.

The x:0 case is for a Full Address Compare. When a program page is selected, the full address compare will be based on bits for a 20-bit compare. The registers used for the compare are {BKP0X[5:0], BKP0H[5:0], BKP0L[7:0]}. When a program page is not selected, the full address compare will be based on bits for a 16-bit compare. The registers used for the compare are {BKP0H[7:0], BKP0L[7:0]}.

The 1:0 case is not sensible because it would ignore the high order address and compare the low order and expansion addresses. Logic forces this case to compare all address lines (effectively ignoring the BK0MBH control bit).

The 1:1 case is useful for triggering a breakpoint on any access to a particular expansion page. This only makes sense if a program page is being accessed so that the breakpoint trigger will occur only if BKP0X compares.

BK1MBH:BK1MBL — Breakpoint Mask High Byte and Low Byte of Data (Second Address)

In Dual Mode, these bits may be used to mask (disable) the comparison of the high and/or low bytes of the second address breakpoint. The functionality is as given in **Table 3-2**.

Table 3-2 Breakpoint Mask Bits for Second Address (Dual Mode)

BK1MBH:BK1MBL	Address Compare	BKP1X	BKP1H	BKP1L
x:0	Full address compare	Yes(1)	Yes	Yes
0:1	256 byte address range	Yes(1)	Yes	No
1:1	16K byte address range	Yes(1)	No	No

NOTES:
1. If page is selected.

The x:0 case is for a Full Address Compare. When a program page is selected, the full address compare will be based on bits for a 20-bit compare. The registers used for the compare are {BKP1X[5:0], BKP1H[5:0], BKP1L[7:0]}. When a program page is not selected, the full address compare will be based on bits for a 16-bit compare. The registers used for the compare are {BKP1H[7:0], BKP1L[7:0]}.

The 1:0 case is not sensible because it would ignore the high order address and compare the low order and expansion addresses. Logic forces this case to compare all address lines (effectively ignoring the BK1MBH control bit).

The 1:1 case is useful for triggering a breakpoint on any access to a particular expansion page. This only makes sense if a program page is being accessed so that the breakpoint trigger will occur only if BKP1X compares.

In Full Mode, these bits may be used to mask (disable) the comparison of the high and/or low bytes of the data breakpoint. The functionality is as given in **Table 3-3**.

Table 3-3 Breakpoint Mask Bits for Data Breakpoints (Full Mode)

BK1MBH:BK1MBL	Data Compare	BKP1X	BKP1H	BKP1L
0:0	High and low byte compare	No(1)	Yes	Yes
0:1	High byte	No(1)	Yes	No
1:0	Low byte	No(1)	No	Yes
1:1	No compare	No(1)	No	No

NOTES:
1. Expansion addresses for breakpoint 1 are not available in this mode.

BK0RWE — R/$\overline{W}$ Compare Enable

Enables the comparison of the R/$\overline{W}$ signal for first address breakpoint. This bit is not useful in tagged breakpoints.

0 = R/$\overline{W}$ is not used in the comparisons
1 = R/$\overline{W}$ is used in comparisons

BK0RW — R/$\overline{W}$ Compare Value

When BK0RWE = 1, this bit determines the type of bus cycle to match on first address breakpoint. When BK0RWE = 0, this bit has no effect.

0 = Write cycle will be matched
1 = Read cycle will be matched

BK1RWE — R/$\overline{W}$ Compare Enable

In Dual Mode, this bit enables the comparison of the R/$\overline{W}$ signal to further specify what causes a match for the second address breakpoint. This bit is not useful on tagged breakpoints or in Full Mode and is therefore a don't care.

0 = R/$\overline{W}$ is not used in comparisons
1 = R/$\overline{W}$ is used in comparisons

BK1RW — R/$\overline{W}$ Compare Value

When BK1RWE = 1, this bit determines the type of bus cycle to match on the second address breakpoint.When BK1RWE = 0, this bit has no effect.

0 = Write cycle will be matched
1 = Read cycle will be matched

3.3 Breakpoint First Address Expansion Register (BKP0X)

Read: anytime

Write: anytime

Register address $002A

	7	6	5	4	3	2	1	0
R	0	0	BK0V5	BK0V4	BK0V3	BK0V2	BK0V1	BK0V0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-4 Breakpoint First Address Expansion Register (BKP0X)

This register contains the data to be matched against expansion address lines for the first address breakpoint when a page is selected.

BK0V[5:0] — Value of first breakpoint address to be matched in memory expansion space.

3.4 Breakpoint First Address High Byte Register (BKP0H)

Read: anytime

Write: anytime

Register address $002B

	7	6	5	4	3	2	1	0
R W	Bit 15	14	13	12	11	10	9	Bit 8
Reset:	0	0	0	0	0	0	0	0

Figure 3-5 Breakpoint First Address High Byte Register (BKP0H)

This register is used to set the breakpoint when compared against the high byte of the address.

3.5 Breakpoint First Address Low Byte Register (BKP0L)

Read: anytime

Write: anytime

Register address $002C

	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-6 Breakpoint First Address Low Byte Register (BKP0L)

This register is used to set the breakpoint when compared against the low byte of the address.

3.6 Breakpoint Second Address Expansion Register (BKP1X)

Read: anytime

Write: anytime

Register address $002D

	7	6	5	4	3	2	1	0
R	0	0	BK1V5	BK1V4	BK1V3	BK1V2	BK1V1	BK1V0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-7 Breakpoint Second Address Expansion Register (BKP1X)

In Dual Mode, this register contains the data to be matched against expansion address lines for the second address breakpoint when a page is selected. In Full Mode, this register is not used.

BK1V[5:0] — Value of first breakpoint address to be matched in memory expansion space.

3.7 Breakpoint Data (Second Address) High Byte Register (BKP1H)

Read: anytime

Write: anytime

Register address $002E

	7	6	5	4	3	2	1	0
R	Bit 15	14	13	12	11	10	9	Bit 8
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-8 Breakpoint Data High Byte Register (BKP1H)

In Dual Mode, this register is used to compare against the high order address lines. In Full Mode, this register is used to compare against the high order data lines.

3.8 Breakpoint Data (Second Address) Low Byte Register (BKP1L)

Read: anytime

Write: anytime

Register address $002F

	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-9 Breakpoint Data Low Byte Register (BKP1L)

In Dual Mode, this register is used to compare against the low order address lines. In Full Mode, this register is used to compare against the low order data lines.

Section 4 Functional Description

The Breakpoint sub-block supports two modes of operation: Dual Address Mode and Full Breakpoint Mode. Within each of these modes, forced or tagged breakpoint types can be used. Forced breakpoints occur at the next instruction boundary if a match occurs and tagged breakpoints allow for breaking just before a specific instruction executes. The action taken upon a successful match can be to either place the CPU in Background Debug Mode or to initiate a software interrupt.

4.1 Modes of Operation

The Breakpoint can operate in Dual Address Mode or Full Breakpoint Mode. Each of these modes is discussed in the subsections below.

4.1.1 Dual Address Mode

When Dual Address Mode is enabled, two address breakpoints can be set. Each breakpoint can cause the system to enter Background Debug Mode or to initiate a software interrupt based upon the state of the BKBDM bit in the BKPCT0 Register being logic one or logic zero, respectively. BDM requests have a higher priority than SWI requests. No data breakpoints are allowed in this mode.

The BKTAG bit in the BKPCT0 register selects whether the breakpoint mode is forced or tagged. The BKxMBH:L bits in the BKPCT1 register select whether or not the breakpoint is matched exactly or is a range breakpoint. They also select whether the address is matched on the high byte, low byte, both bytes, and/or memory expansion. The BKxRW and BKxRWE bits in the BKPCT1 register select whether the type of bus cycle to match is a read, write, or both when performing forced breakpoints.

4.1.2 Full Breakpoint Mode

Full Breakpoint Mode requires a match on address and data for a breakpoint to occur. Upon a successful match, the system will enter Background Debug Mode or initiate a software interrupt based upon the state of the BKBDM bit in the BKPCT0 Register being logic one or logic zero, respectively. BDM requests have a higher priority than SWI requests. R/$\overline{W}$ matches are also allowed in this mode.

The BKTAG bit in the BKPCT0 register selects whether the breakpoint mode is forced or tagged. If the BKTAG bit is set in BKPCT0, then only address is matched, and data is ignored. The BK0MBH:L bits in the BKPCT1 register select whether or not the breakpoint is matched exactly, is a range breakpoint, or is in page space. The BK1MBH:L bits in the BKPCT1 register select whether the data is matched on the high byte, low byte, or both bytes. The BK0RW and BK0RWE bits in the BKPCT1 register select whether the type of bus cycle to match is a read or a write when performing forced breakpoints. BK1RW and BK1RWE bits in the BKPCT1 register are not used in Full Breakpoint Mode.

4.2 Breakpoint Priority

Breakpoint operation is first determined by the state of BDM. If BDM is already active, meaning the CPU is executing out of BDM firmware, Breakpoints are not allowed. In addition, while in BDM trace mode, tagging into BDM is not allowed. If BDM is not active, the Breakpoint will give priority to BDM requests over SWI requests. This condition applies to both forced and tagged breakpoints.

In all cases, BDM related breakpoints will have priority over those generated by the Breakpoint sub-block. This priority includes breakpoints enabled by the $\overline{\text{TAGLO}}$ and $\overline{\text{TAGHI}}$ external pins of the system that interface with the BDM directly and whose signal information passes through and is used by the Breakpoint sub-block.

NOTE: *BDM should not be entered from a breakpoint unless the BKEN bit is set in the BDM. Even if the ENABLE bit in the BDM is negated, the CPU actually executes the BDM firmware code. It checks the ENABLE and returns if enable is not set. If the BDM is not serviced by the monitor then the breakpoint would be re-asserted when the BDM returns to normal CPU flow.*

There is no hardware to enforce restriction of breakpoint operation if the BDM is not enabled.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

JAPAN:
Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

MOTOROLA

S12BKPV1/D
Rev. 1.02
5/2003

MOTOROLA
intelligence everywhere™

digital dna™

S12CPUV2

Reference Manual

HCS12
Microcontrollers

S12CPUV2/D
Rev. 0
7/2003

MOTOROLA.COM/SEMICONDUCTORS

S12CPUV2

Reference Manual

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Revision History

Date	Revision Level	Description	Page Number(s)
July, 2003	0	Initial release	N/A

List of Sections

List of Sections

Table of Contents

Section 3. Addressing Modes

Section 4. Instruction Queue

Section 5. Instruction Set Overview

Section 6. Instruction Glossary

Section 7. Exception Processing

Section 8. Instruction Queue

Section 9. Fuzzy Logic Support

Appendix A. Instruction Reference

Appendix B. M68HC11 to CPU12 Upgrade Path

Appendix C. High-Level Language Support

List of Figures

List of Figures

List of Tables

Table	Title	Page

Table	Title	Page

List of Tables

Section 1. Introduction

1.1 Introduction

This manual describes the features and operation of the core (central processing unit, or CPU, and development support functions) used in all HCS12 microcontrollers. For reference, information is provided for the M68HC12.

1.2 Features

The CPU12 is a high-speed, 16-bit processing unit that has a programming model identical to that of the industry standard M68HC11 central processor unit (CPU). The CPU12 instruction set is a proper superset of the M68HC11 instruction set, and M68HC11 source code is accepted by CPU12 assemblers with no changes.

- Full 16-bit data paths supports efficient arithmetic operation and high-speed math execution
- Supports instructions with odd byte counts, including many single-byte instructions. This allows much more efficient use of ROM space.
- An instruction queue buffers program information so the CPU has immediate access to at least three bytes of machine code at the start of every instruction.
- Extensive set of indexed addressing capabilities, including:
  - Using the stack pointer as an indexing register in all indexed operations
  - Using the program counter as an indexing register in all but auto increment/decrement mode
  - Accumulator offsets using A, B, or D accumulators
  - Automatic index predecrement, preincrement, postdecrement, and postincrement (by –8 to +8)

1.3 Symbols and Notation

The symbols and notation shown here are used throughout the manual. More specialized notation that applies only to the instruction glossary or instruction set summary are described at the beginning of those sections.

1.3.1 Abbreviations for System Resources

A — Accumulator A
B — Accumulator B
D — Double accumulator D (A : B)
X — Index register X
Y — Index register Y
SP — Stack pointer
PC — Program counter
CCR — Condition code register
- S — STOP instruction control bit
- X — Non-maskable interrupt control bit
- H — Half-carry status bit
- I — Maskable interrupt control bit
- N — Negative status bit
- Z — Zero status bit
- V — Two’s complement overflow status bit
- C — Carry/Borrow status bit

1.3.2 Memory and Addressing

M	—	8-bit memory location pointed to by the effective address of the instruction
M : M+1	—	16-bit memory location. Consists of the contents of the location pointed to by the effective address concatenated with the contents of the location at the next higher memory address. The most significant byte is at location M.
M~M+3 $M_{(Y)}$~$M_{(Y+3)}$	—	32-bit memory location. Consists of the contents of the effective address of the instruction concatenated with the contents of the next three higher memory locations. The most significant byte is at location M or $M_{(Y)}$.
$M_{(X)}$	—	Memory locations pointed to by index register X
$M_{(SP)}$	—	Memory locations pointed to by the stack pointer
$M_{(Y+3)}$	—	Memory locations pointed to by index register Y plus 3
PPAGE	—	Program overlay page (bank) number for extended memory (>64 Kbytes).
Page	—	Program overlay page
X_H	—	High-order byte
X_L	—	Low-order byte
( )	—	Content of register or memory location
$	—	Hexadecimal value
%	—	Binary value

1.3.3 Operators

+ — Addition

– — Subtraction

• — Logical AND

+ — Logical OR (inclusive)

⊕ — Logical exclusive OR

× — Multiplication

÷ — Division

$\overline{M}$ — Negation. One's complement (invert each bit of M)

: — Concatenate

Example: A : B means the 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B. A is in the high-order position.

⇒ — Transfer

Example: (A) ⇒ M means the content of accumulator A is transferred to memory location M.

⇔ — Exchange

Example: D ⇔ X means exchange the contents of D with those of X.

1.3.4 Definitions

Logic level 1 is the voltage that corresponds to the true (1) state.

Logic level 0 is the voltage that corresponds to the false (0) state.

Set refers specifically to establishing logic level 1 on a bit or bits.

Cleared refers specifically to establishing logic level 0 on a bit or bits.

Asserted means that a signal is in active logic state. An active low signal changes from logic level 1 to logic level 0 when asserted, and an active high signal changes from logic level 0 to logic level 1.

Negated means that an asserted signal changes logic state. An active low signal changes from logic level 0 to logic level 1 when negated, and an active high signal changes from logic level 1 to logic level 0.

ADDR is the mnemonic for address bus.

DATA is the mnemonic for data bus.

LSB means least significant bit or bits.

MSB means most significant bit or bits.

LSW means least significant word or words.

MSW means most significant word or words.

A specific bit location within a range is referred to by mnemonic and number. For example, A7 is bit 7 of accumulator A.

A range of bit locations is referred to by mnemonic and the numbers that define the range. For example, DATA[15:8] form the high byte of the data bus.

Introduction

Section 2. Overview

2.1 Introduction

This section describes the CPU12 programming model, register set, the data types used, and basic memory organization.

2.2 Programming Model

The CPU12 programming model, shown in **Figure 2-1**, is the same as that of the M68HC11 CPU. The CPU has two 8-bit general-purpose accumulators (A and B) that can be concatenated into a single 16-bit accumulator (D) for certain instructions. It also has:

- Two index registers (X and Y)
- 16-bit stack pointer (SP)
- 16-bit program counter (PC)
- 8-bit condition code register (CCR)

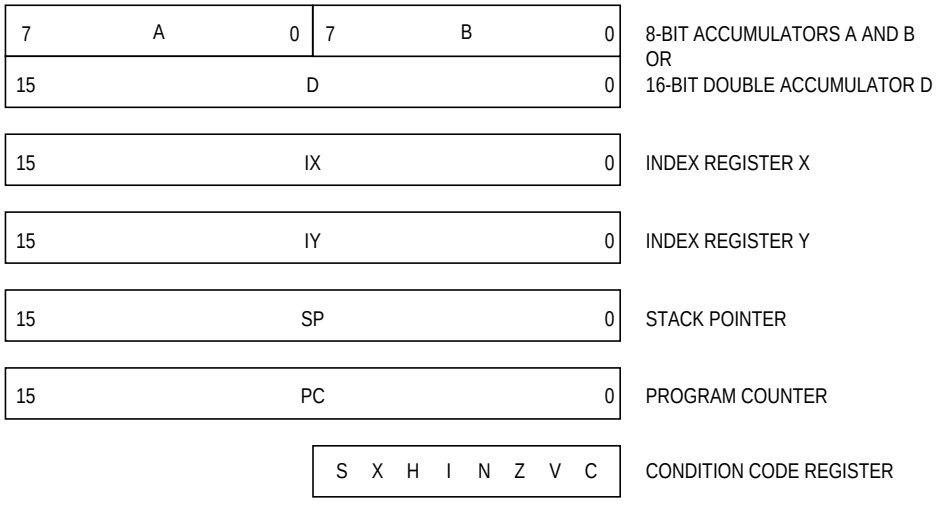


Figure 2-1. Programming Model

2.2.1 Accumulators

General-purpose 8-bit accumulators A and B are used to hold operands and results of operations. Some instructions treat the combination of these two 8-bit accumulators (A : B) as a 16-bit double accumulator (D).

Most operations can use accumulator A or B interchangeably. However, there are a few exceptions. Add, subtract, and compare instructions involving both A and B (ABA, SBA, and CBA) only operate in one direction, so it is important to make certain the correct operand is in the correct accumulator. The decimal adjust accumulator A (DAA) instruction is used after binary-coded decimal (BCD) arithmetic operations. There is no equivalent instruction to adjust accumulator B.

2.2.2 Index Registers

16-bit index registers X and Y are used for indexed addressing. In the indexed addressing modes, the contents of an index register are added to 5-bit, 9-bit, or 16-bit constants or to the content of an accumulator to form the effective address of the instruction operand. The second index register is especially useful for moves and in cases where operands from two separate tables are used in a calculation.

2.2.3 Stack Pointer

The CPU12 supports an automatic program stack. The stack is used to save system context during subroutine calls and interrupts and can also be used for temporary data storage. The stack can be located anywhere in the standard 64-Kbyte address space and can grow to any size up to the total amount of memory available in the system.

The stack pointer (SP) holds the 16-bit address of the last stack location used. Normally, the SP is initialized by one of the first instructions in an application program. The stack grows downward from the address pointed to by the SP. Each time a byte is pushed onto the stack, the stack pointer is automatically decremented, and each time a byte is pulled from the stack, the stack pointer is automatically incremented.

When a subroutine is called, the address of the instruction following the calling instruction is automatically calculated and pushed onto the stack. Normally, a return-from-subroutine (RTS) or a return-from-call (RTC)

instruction is executed at the end of a subroutine. The return instruction loads the program counter with the previously stacked return address and execution continues at that address.

When an interrupt occurs, the current instruction finishes execution. The address of the next instruction is calculated and pushed onto the stack, all the CPU registers are pushed onto the stack, the program counter is loaded with the address pointed to by the interrupt vector, and execution continues at that address. The stacked registers are referred to as an interrupt stack frame. The CPU12 stack frame is the same as that of the M68HC11.

NOTE: *These instructions can be interrupted, and they resume execution once the interrupt has been serviced:*

- *REV (fuzzy logic rule evaluation)*
- *REVW (fuzzy logic rule evaluation (weighted))*
- *WAV (weighted average)*

2.2.4 Program Counter

The program counter (PC) is a 16-bit register that holds the address of the next instruction to be executed. It is automatically incremented each time an instruction is fetched.

2.2.5 Condition Code Register

The condition code register (CCR), named for its five status indicators, contains:

- Five status indicators
- Two interrupt masking bits
- STOP instruction control bit

The status bits reflect the results of CPU operation as it executes instructions. The five flags are:

- Half carry (H)
- Negative (N)
- Zero (Z)
- Overflow (V)
- Carry/borrow (C)

The half-carry flag is used only for BCD arithmetic operations. The N, Z, V, and C status bits allow for branching based on the results of a previous operation.

In some architectures, only a few instructions affect condition codes, so that multiple instructions must be executed in order to load and test a variable. Since most CPU12 instructions automatically update condition codes, it is rarely necessary to execute an extra instruction for this purpose. The challenge in using the CPU12 lies in finding instructions that do not alter the condition codes. The most important of these instructions are pushes, pulls, transfers, and exchanges.

It is always a good idea to refer to an instruction set summary (see **Appendix A. Instruction Reference**) to check which condition codes are affected by a particular instruction.

The following paragraphs describe normal uses of the condition codes. There are other, more specialized uses. For instance, the C status bit is used to enable weighted fuzzy logic rule evaluation. Specialized usages are described in the relevant portions of this manual and in **Section 6. Instruction Glossary**.

2.2.5.1 S Control Bit

Clearing the S bit enables the STOP instruction. Execution of a STOP instruction normally causes the on-chip oscillator to stop. This may be undesirable in some applications. If the CPU encounters a STOP instruction while the S bit is set, it is treated like a no-operation (NOP) instruction and continues to the next instruction. Reset sets the S bit.

2.2.5.2 X Mask Bit

The $\overline{\text{XIRQ}}$ input is an updated version of the $\overline{\text{NMI}}$ input found on earlier generations of MCUs. Non-maskable interrupts are typically used to deal with major system failures, such as loss of power. However, enabling non-maskable interrupts before a system is fully powered and initialized can lead to spurious interrupts. The X bit provides a mechanism for enabling non-maskable interrupts after a system is stable.

By default, the X bit is set to 1 during reset. As long as the X bit remains set, interrupt service requests made via the $\overline{\text{XIRQ}}$ pin are not recognized. An instruction must clear the X bit to enable non-maskable interrupt service requests made via the $\overline{\text{XIRQ}}$ pin. Once the X bit has been cleared to 0, software cannot reset it to 1 by writing to the CCR. The X bit is not affected by maskable interrupts.

When an $\overline{\text{XIRQ}}$ interrupt occurs after non-maskable interrupts are enabled, both the X bit and the I bit are set automatically to prevent other interrupts from being recognized during the interrupt service routine. The mask bits are set after the registers are stacked, but before the interrupt vector is fetched.

Normally, a return-from-interrupt (RTI) instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the X bit is set, the RTI normally clears the X bit, and thus re-enables non-maskable interrupts. While it is possible to manipulate the stacked value of X so that X is set after an RTI, there is no software method to reset X (and disable $\overline{\text{XIRQ}}$) once X has been cleared.

2.2.5.3 H Status Bit

The H bit indicates a carry from accumulator A bit 3 during an addition operation. The DAA instruction uses the value of the H bit to adjust a result in accumulator A to correct BCD format. H is updated only by the add accumulator A to accumulator B (ABA), add without carry (ADD), and add with carry (ADC) instructions.

2.2.5.4 I Mask Bit

The I bit enables and disables maskable interrupt sources. By default, the I bit is set to 1 during reset. An instruction must clear the I bit to enable maskable interrupts. While the I bit is set, maskable interrupts can become pending and are remembered, but operation continues uninterrupted until the I bit is cleared.

When an interrupt occurs after interrupts are enabled, the I bit is automatically set to prevent other maskable interrupts during the interrupt service routine. The I bit is set after the registers are stacked, but before the first instruction in the interrupt service routine is executed.

Normally, an RTI instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the I bit is set, the RTI normally clears the I bit, and thus re-enables interrupts. Interrupts can be re-enabled by clearing the I bit within the service routine, but implementing a nested interrupt management scheme requires great care and seldom improves system performance.

2.2.5.5 N Status Bit

The N bit shows the state of the MSB of the result. N is most commonly used in two’s complement arithmetic, where the MSB of a negative number is 1 and the MSB of a positive number is 0, but it has other uses. For instance, if the MSB of a register or memory location is used as a status flag, the user can test status by loading an accumulator.

2.2.5.6 Z Status Bit

The Z bit is set when all the bits of the result are 0s. Compare instructions perform an internal implied subtraction, and the condition codes, including Z, reflect the results of that subtraction. The increment index register X (INX), decrement index register X (DEX), increment index register Y (INY), and decrement index register Y (DEY) instructions affect the Z bit and no other condition flags. These operations can only determine = (equal) and ≠ (not equal).

2.2.5.7 V Status Bit

The V bit is set when two’s complement overflow occurs as a result of an operation.

2.2.5.8 C Status Bit

The C bit is set when a carry occurs during addition or a borrow occurs during subtraction. The C bit also acts as an error flag for multiply and divide operations. Shift and rotate instructions operate through the C bit to facilitate multiple-word shifts.

2.3 Data Types

The CPU12 uses these types of data:

- Bits
- 5-bit signed integers
- 8-bit signed and unsigned integers
- 8-bit, 2-digit binary-coded decimal numbers
- 9-bit signed integers
- 16-bit signed and unsigned integers
- 16-bit effective addresses
- 32-bit signed and unsigned integers

Negative integers are represented in two’s complement form.

Five-bit and 9-bit signed integers are used only as offsets for indexed addressing modes.

Sixteen-bit effective addresses are formed during addressing mode computations.

Thirty-two-bit integer dividends are used by extended division instructions. Extended multiply and extended multiply-and-accumulate instructions produce 32-bit products.

2.4 Memory Organization

The standard CPU12 address space is 64 Kbytes. Some M68HC12 devices support a paged memory expansion scheme that increases the standard space by means of predefined windows in address space. The CPU12 has special instructions that support use of expanded memory.

Eight-bit values can be stored at any odd or even byte address in available memory.

Sixteen-bit values are stored in memory as two consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

Thirty-two-bit values are stored in memory as four consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

All input/output (I/O) and all on-chip peripherals are memory-mapped. No special instruction syntax is required to access these addresses. On-chip registers and memory typically are grouped in blocks which can be relocated within the standard 64-Kbyte address space. Refer to device documentation for specific information.

2.5 Instruction Queue

The CPU12 uses an instruction queue to buffer program information. The mechanism is called a queue rather than a pipeline because a typical pipelined CPU executes more than one instruction at the same time, while the CPU12 always finishes executing an instruction before beginning to execute another. Refer to **Section 4. Instruction Queue** for more information.

Section 3. Addressing Modes

3.1 Introduction

Addressing modes determine how the central processor unit (CPU) accesses memory locations to be operated upon. This section discusses the various modes and how they are used.

3.2 Mode Summary

Addressing modes are an implicit part of CPU12 instructions. Refer to **Appendix A. Instruction Reference** for the modes used by each instruction. All CPU12 addressing modes are shown in **Table 3-1**.

The CPU12 uses all M68HC11 modes as well as new forms of indexed addressing. Differences between M68HC11 and M68HC12 indexed modes are described in **3.9 Indexed Addressing Modes**. Instructions that use more than one mode are discussed in **3.10 Instructions Using Multiple Modes**.

3.3 Effective Address

Each addressing mode except inherent mode generates a 16-bit effective address which is used during the memory reference portion of the instruction. Effective address computations do not require extra execution cycles.

Table 3-1. M68HC12 Addressing Mode Summary

Addressing Mode	Source Format	Abbreviation	Description
Inherent	**INST** (no externally supplied operands)	INH	Operands (if any) are in CPU registers
Immediate	**INST** #*opr8i* or **INST** #*opr16i*	IMM	Operand is included in instruction stream 8- or 16-bit size implied by context
Direct	**INST** *opr8a*	DIR	Operand is the lower 8 bits of an address in the range $0000–$00FF
Extended	**INST** *opr16a*	EXT	Operand is a 16-bit address
Relative	**INST** *rel8* or **INST** *rel16*	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction
Indexed (5-bit offset)	**INST** *oprx5*,*xysp*	IDX	5-bit signed constant offset from X, Y, SP, or PC
Indexed (pre-decrement)	**INST** *oprx3*,–*xys*	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	**INST** *oprx3*,+*xys*	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	**INST** *oprx3*,*xys*–	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	**INST** *oprx3*,*xys*+	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	**INST** *abd*,*xysp*	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from X, Y, SP, or PC
Indexed (9-bit offset)	**INST** *oprx9*,*xysp*	IDX1	9-bit signed constant offset from X, Y, SP, or PC (lower 8 bits of offset in one extension byte)
Indexed (16-bit offset)	**INST** *oprx16*,*xysp*	IDX2	16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	**INST [***oprx16*,*xysp***]**	[IDX2]	Pointer to operand is found at... 16-bit constant offset from X, Y, SP, or PC (16-bit offset in two extension bytes)
Indexed-Indirect (D accumulator offset)	**INST [D,***xysp***]**	[D,IDX]	Pointer to operand is found at... X, Y, SP, or PC plus the value in D

3.4 Inherent Addressing Mode

Instructions that use this addressing mode either have no operands or all operands are in internal CPU registers. In either case, the CPU does not need to access any memory locations to complete the instruction.

Examples:

```
NOP   ;this instruction has no operands
INX   ;operand is a CPU register
```

3.5 Immediate Addressing Mode

Operands for immediate mode instructions are included in the instruction stream and are fetched into the instruction queue one 16-bit word at a time during normal program fetch cycles. Since program data is read into the instruction queue several cycles before it is needed, when an immediate addressing mode operand is called for by an instruction, it is already present in the instruction queue.

The pound symbol (#) is used to indicate an immediate addressing mode operand. One common programming error is to accidentally omit the # symbol. This causes the assembler to misinterpret the expression that follows it as an address rather than explicitly provided data. For example, LDAA #$55 means to load the immediate value $55 into the A accumulator, while LDAA $55 means to load the value from address $0055 into the A accumulator. Without the # symbol, the instruction is erroneously interpreted as a direct addressing mode instruction.

Examples:

```
LDAA        #$55
LDX         #$1234
LDY         #$67
```

These are common examples of 8-bit and 16-bit immediate addressing modes. The size of the immediate operand is implied by the instruction context. In the third example, the instruction implies a 16-bit immediate value but only an 8-bit value is supplied. In this case the assembler will generate the 16-bit value $0067 because the CPU expects a 16-bit value in the instruction stream.

Example:

```
BRSET       FOO,#$03,THERE
```

In this example, extended addressing mode is used to access the operand FOO, immediate addressing mode is used to access the mask value $03, and relative addressing mode is used to identify the destination address of a branch in case the branch-taken conditions are met. BRSET is listed as an extended mode instruction even though immediate and relative modes are also used.

3.6 Direct Addressing Mode

This addressing mode is sometimes called zero-page addressing because it is used to access operands in the address range $0000 through $00FF. Since these addresses always begin with $00, only the eight low-order bits of the address need to be included in the instruction, which saves program space and execution time. A system can be optimized by placing the most commonly accessed data in this area of memory. The eight low-order bits of the operand address are supplied with the instruction, and the eight high-order bits of the address are assumed to be 0.

Example:

```
LDAA        $55
```

This is a basic example of direct addressing. The value $55 is taken to be the low-order half of an address in the range $0000 through $00FF. The high order half of the address is assumed to be 0. During execution of this instruction, the CPU combines the value $55 from the instruction with the assumed value of $00 to form the address $0055, which is then used to access the data to be loaded into accumulator A.

Example:

```
LDX         $20
```

In this example, the value $20 is combined with the assumed value of $00 to form the address $0020. Since the LDX instruction requires a 16-bit value, a 16-bit word of data is read from addresses $0020 and $0021. After execution of this instruction, the X index register will have the value from address $0020 in its high-order half and the value from address $0021 in its low-order half.

3.7 Extended Addressing Mode

In this addressing mode, the full 16-bit address of the memory location to be operated on is provided in the instruction. This addressing mode can be used to access any location in the 64-Kbyte memory map.

Example:

```
LDAA        $F03B
```

This is a basic example of extended addressing. The value from address $F03B is loaded into the A accumulator.

3.8 Relative Addressing Mode

The relative addressing mode is used only by branch instructions. Short and long conditional branch instructions use relative addressing mode exclusively, but branching versions of bit manipulation instructions (branch if bits set (BRSET) and branch if bits cleared (BRCLR)) use multiple addressing modes, including relative mode. Refer to **3.10 Instructions Using Multiple Modes** for more information.

Short branch instructions consist of an 8-bit opcode and a signed 8-bit offset contained in the byte that follows the opcode. Long branch instructions consist of an 8-bit prebyte, an 8-bit opcode, and a signed 16-bit offset contained in the two bytes that follow the opcode.

Each conditional branch instruction tests certain status bits in the condition code register. If the bits are in a specified state, the offset is added to the address of the next memory location after the offset to form an effective address, and execution continues at that address. If the bits are not in the specified state, execution continues with the instruction immediately following the branch instruction.

Bit-condition branches test whether bits in a memory byte are in a specific state. Various addressing modes can be used to access the memory location. An 8-bit mask operand is used to test the bits. If each bit in memory that corresponds to a 1 in the mask is either set (BRSET) or clear (BRCLR), an 8-bit offset is added to the address of the next memory location after the offset to form an effective address, and execution continues at that address. If all the bits in memory that correspond to a 1 in the mask are not in the specified state, execution

continues with the instruction immediately following the branch instruction.

8-bit, 9-bit, and 16-bit offsets are signed two’s complement numbers to support branching upward and downward in memory. The numeric range of short branch offset values is $80 (–128) to $7F (127). Loop primitive instructions support a 9-bit offset which allows a range of $100 (–256) to $0FF (255). The numeric range of long branch offset values is $8000 (–32,768) to $7FFF (32,767). If the offset is 0, the CPU executes the instruction immediately following the branch instruction, regardless of the test involved.

Since the offset is at the end of a branch instruction, using a negative offset value can cause the program counter (PC) to point to the opcode and initiate a loop. For instance, a branch always (BRA) instruction consists of two bytes, so using an offset of $FE sets up an infinite loop; the same is true of a long branch always (LBRA) instruction with an offset of $FFFC.

An offset that points to the opcode can cause a bit-condition branch to repeat execution until the specified bit condition is satisfied. Since bit-condition branches can consist of four, five, or six bytes depending on the addressing mode used to access the byte in memory, the offset value that sets up a loop can vary. For instance, using an offset of $FC with a BRCLR that accesses memory using an 8-bit indexed postbyte sets up a loop that executes until all the bits in the specified memory byte that correspond to 1s in the mask byte are cleared.

3.9 Indexed Addressing Modes

The CPU12 uses redefined versions of M68HC11 indexed modes that reduce execution time and eliminate code size penalties for using the Y index register. In most cases, CPU12 code size for indexed operations is the same or is smaller than that for the M68HC11. Execution time is shorter in all cases. Execution time improvements are due to both a reduced number of cycles for all indexed instructions and to faster system clock speed.

The indexed addressing scheme uses a postbyte plus zero, one, or two extension bytes after the instruction opcode. The postbyte and extensions do the following tasks:

1. Specify which index register is used
2. Determine whether a value in an accumulator is used as an offset
3. Enable automatic pre- or post-increment or pre- or post-decrement
4. Specify size of increment or decrement
5. Specify use of 5-, 9-, or 16-bit signed offsets

This approach eliminates the differences between X and Y register use while dramatically enhancing the indexed addressing capabilities.

Major advantages of the CPU12 indexed addressing scheme are:

- The stack pointer can be used as an index register in all indexed operations.
- The program counter can be used as an index register in all but autoincrement and autodecrement modes.
- A, B, or D accumulators can be used for accumulator offsets.
- Automatic pre- or post-increment or pre- or post-decrement by –8 to +8
- A choice of 5-, 9-, or 16-bit signed constant offsets
- Use of two new indexed-indirect modes:
  - Indexed-indirect mode with 16-bit offset
  - Indexed-indirect mode with accumulator D offset

Table 3-2 is a summary of indexed addressing mode capabilities and a description of postbyte encoding. The postbyte is noted as xb in instruction descriptions. Detailed descriptions of the indexed addressing mode variations follow the table.

All indexed addressing modes use a 16-bit CPU register and additional information to create an effective address. In most cases the effective address specifies the memory location affected by the operation. In some variations of indexed addressing, the effective address specifies the location of a value that points to the memory location affected by the operation.

Table 3-2. Summary of Indexed Operations

Postbyte Code (xb)	Source Code Syntax	Comments rr; 00 = X, 01 = Y, 10 = SP, 11 = PC
rr0nnnnn	,r n,r –n,r	**5-bit constant offset** n = –16 to +15 r can specify X, Y, SP, or PC
111rr0zs	n,r –n,r	**Constant offset** (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte(s) –256 ≤ n ≤ 255 1 = 16-bit –32,768 ≤ n ≤ 65,535 if z = s = 1, 16-bit offset indexed-indirect (see below) r can specify X, Y, SP, or PC
111rr011	[n,r]	**16-bit offset indexed-indirect** rr can specify X, Y, SP, or PC –32,768 ≤ n ≤ 65,535
rr1pnnnn	n,–r n,+r n,r– n,r+	**Auto predecrement**, **preincrement**, **postdecrement**, or **postincrement**; p = pre-(0) or post-(1), n = –8 to –1, +1 to +8 r can specify X, Y, or SP (PC not a valid choice) +8 = 0111 … +1 = 0000 –1 = 1111 … –8 = 1000
111rr1aa	A,r B,r D,r	**Accumulator offset** (unsigned 8-bit or 16-bit) aa-00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect r can specify X, Y, SP, or PC
111rr111	[D,r]	**Accumulator D offset indexed-indirect** r can specify X, Y, SP, or PC

Indexed addressing mode instructions use a postbyte to specify index registers (X and Y), stack pointer (SP), or program counter (PC) as the base index register and to further classify the way the effective address is formed. A special group of instructions cause this calculated effective address to be loaded into an index register for further calculations:

- Load stack pointer with effective address (LEAS)
- Load X with effective address (LEAX)
- Load Y with effective address (LEAY)

3.9.1 5-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 5-bit signed offset which is included in the instruction postbyte. This short offset is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location that will be affected by the instruction. This gives a range of –16 through +15 from the value in the base index register. Although other indexed addressing modes allow 9- or 16-bit offsets, those modes also require additional extension bytes in the instruction for this extra information. The majority of indexed instructions in real programs use offsets that fit in the shortest 5-bit form of indexed addressing.

Examples:

```
LDAA    0,X
STAB    -8,Y
```

For these examples, assume X has a value of $1000 and Y has a value of $2000 before execution. The 5-bit constant offset mode does not change the value in the index register, so X will still be $1000 and Y will still be $2000 after execution of these instructions. In the first example, A will be loaded with the value from address $1000. In the second example, the value from the B accumulator will be stored at address $1FF8 ($2000 –$8).

3.9.2 9-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 9-bit signed offset which is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location affected by the instruction. This gives a range of –256 through +255 from the value in the base index register. The most significant bit (sign bit) of the offset is included in the instruction postbyte and the remaining eight bits are provided as an extension byte after the instruction postbyte in the instruction flow.

Examples:

```
LDAA    $FF,X
LDAB    -20,Y
```

For these examples, assume X is $1000 and Y is $2000 before execution of these instructions.

NOTE: *These instructions do not alter the index registers so they will still be $1000 and $2000, respectively, after the instructions.*

The first instruction will load A with the value from address $10FF and the second instruction will load B with the value from address $1FEC.

This variation of the indexed addressing mode in the CPU12 is similar to the M68HC11 indexed addressing mode, but is functionally enhanced. The M68HC11 CPU provides for unsigned 8-bit constant offset indexing from X or Y, and use of Y requires an extra instruction byte and thus, an extra execution cycle. The 9-bit signed offset used in the CPU12 covers the same range of positive offsets as the M68HC11, and adds negative offset capability. The CPU12 can use X, Y, SP, or PC as the base index register.

3.9.3 16-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 16-bit offset which is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location affected by the instruction. This allows access to any address in the 64-Kbyte address space. Since the address bus and the offset are both 16 bits, it does not matter whether the offset value is considered to be a signed or an unsigned value ($FFFF may be thought of as +65,535 or as –1). The 16-bit offset is provided as two extension bytes after the instruction postbyte in the instruction flow.

3.9.4 16-Bit Constant Indirect Indexed Addressing

This indexed addressing mode adds a 16-bit instruction-supplied offset to the base index register to form the address of a memory location that contains a pointer to the memory location affected by the instruction. The instruction itself does not point to the address of the memory location to be acted upon, but rather to the location of a pointer to the address to be acted on. The square brackets distinguish this addressing mode from 16-bit constant offset indexing.

Example:

```
LDAA        [10,X]
```

In this example, X holds the base address of a table of pointers. Assume that X has an initial value of $1000, and that the value $2000 is stored at addresses $100A and $100B. The instruction first adds the value 10 to the value in X to form the address $100A. Next, an address pointer ($2000) is fetched from memory at $100A. Then, the value stored in location $2000 is read and loaded into the A accumulator.

3.9.5 Auto Pre/Post Decrement/Increment Indexed Addressing

This indexed addressing mode provides four ways to automatically change the value in a base index register as a part of instruction execution. The index register can be incremented or decremented by an integer value either before or after indexing takes place. The base index register may be X, Y, or SP. (Auto-modify modes would not make sense on PC.)

Pre-decrement and pre-increment versions of the addressing mode adjust the value of the index register before accessing the memory location affected by the instruction — the index register retains the changed value after the instruction executes. Post-decrement and post-increment versions of the addressing mode use the initial value in the index register to access the memory location affected by the instruction, then change the value of the index register.

The CPU12 allows the index register to be incremented or decremented by any integer value in the ranges –8 through –1 or 1 through 8. The value need not be related to the size of the operand for the current instruction. These instructions can be used to incorporate an index adjustment into an existing instruction rather than using an additional instruction and increasing execution time. This addressing mode is also used to perform operations on a series of data structures in memory.

When an LEAS, LEAX, or LEAY instruction is executed using this addressing mode, and the operation modifies the index register that is being loaded, the final value in the register is the value that would have been used to access a memory operand. (Premodification is seen in the result but postmodification is not.)

Examples:

```
STAA    1,-SP     ;equivalent to PSHA
STX     2,-SP     ;equivalent to PSHX
LDX     2,SP+     ;equivalent to PULX
LDAA    1,SP+     ;equivalent to PULA
```

For a “last-used” type of stack like the CPU12 stack, these four examples are equivalent to common push and pull instructions.

For a “next-available” stack like the M68HC11 stack, push A onto stack (PSHA) is equivalent to store accumulator A (STAA) 1,SP– and pull A from stack (PULA) is equivalent to load accumulator A (LDAA) 1,+SP. However, in the M68HC11, 16-bit operations like push register X onto

stack (PSHX) and pull register X from stack (PULX) require multiple instructions to decrement the SP by one, then store X, then decrement SP by one again.

In the STAA 1,–SP example, the stack pointer is pre-decremented by one and then A is stored to the address contained in the stack pointer. Similarly the LDX 2,SP+ first loads X from the address in the stack pointer, then post-increments SP by two.

Example:

```
MOVW        2,X+,4,+Y
```

This example demonstrates how to work with data structures larger than bytes and words. With this instruction in a program loop, it is possible to move words of data from a list having one word per entry into a second table that has four bytes per table element. In this example the source pointer is updated after the data is read from memory (post-increment) while the destination pointer is updated before it is used to access memory (pre-increment).

3.9.6 Accumulator Offset Indexed Addressing

In this indexed addressing mode, the effective address is the sum of the values in the base index register and an unsigned offset in one of the accumulators. The value in the index register itself is not changed. The index register can be X, Y, SP, or PC and the accumulator can be either of the 8-bit accumulators (A or B) or the 16-bit D accumulator.

Example:

```
LDAA        B,X
```

This instruction internally adds B to X to form the address from which A will be loaded. B and X are not changed by this instruction. This example is similar to the following 2-instruction combination in an M68HC11.

Examples:

```
ABX
LDAA        0,X
```

However, this 2-instruction sequence alters the index register. If this sequence was part of a loop where B changed on each pass, the index register would have to be reloaded with the reference value on each loop pass. The use of LDAA B,X is more efficient in the CPU12.

3.9.7 Accumulator D Indirect Indexed Addressing

This indexed addressing mode adds the value in the D accumulator to the value in the base index register to form the address of a memory location that contains a pointer to the memory location affected by the instruction. The instruction operand does not point to the address of the memory location to be acted upon, but rather to the location of a pointer to the address to be acted upon. The square brackets distinguish this addressing mode from D accumulator offset indexing.

Examples:

```
JMP         [D,PC]
GO1         DC.W            PLACE1
GO2         DC.W            PLACE2
GO3         DC.W            PLACE3
```

This example is a computed GOTO. The values beginning at GO1 are addresses of potential destinations of the jump (JMP) instruction. At the time the JMP [D,PC] instruction is executed, PC points to the address GO1, and D holds one of the values $0000, $0002, or $0004 (determined by the program some time before the JMP).

Assume that the value in D is $0002. The JMP instruction adds the values in D and PC to form the address of GO2. Next the CPU reads the address PLACE2 from memory at GO2 and jumps to PLACE2. The locations of PLACE1 through PLACE3 were known at the time of program assembly but the destination of the JMP depends upon the value in D computed during program execution.

3.10 Instructions Using Multiple Modes

Several CPU12 instructions use more than one addressing mode in the course of execution.

3.10.1 Move Instructions

Move instructions use separate addressing modes to access the source and destination of a move. There are move variations for all practical combinations of immediate, extended, and indexed addressing modes.

The only combinations of addressing modes that are not allowed are those with an immediate mode destination (the operand of an immediate mode instruction is data, not an address). For indexed moves, the reference index register may be X, Y, SP, or PC.

Move instructions do not support indirect modes, 9-bit, or 16-bit offset modes requiring extra extension bytes. There are special considerations when using PC-relative addressing with move instructions. The original M68HC12 implemented the instruction queue slightly differently than the newer HCS12. In the older M68HC12 implementation, the CPU did not maintain a pointer to the start of the instruction after the current instruction (what the user thinks of as the PC value during execution). This caused an offset for PC-relative move instructions.

PC-relative addressing uses the address of the location immediately following the last byte of object code for the current instruction as a reference point. The CPU12 normally corrects for queue offset and for instruction alignment so that queue operation is transparent to the user. However, in the original M68HC12, move instructions pose three special problems:

- Some moves use an indexed source and an indexed destination.
- Some moves have object code that is too long to fit in the queue all at one time, so the PC value changes during execution.
- All moves do not have the indexed postbyte as the last byte of object code.

These cases are not handled by automatic queue pointer maintenance, but it is still possible to use PC-relative indexing with move instructions by providing for PC offsets in source code.

Table 3-3 shows PC offsets from the location immediately following the current instruction by addressing mode.

Table 3-3. PC Offsets for MOVE Instructions (M68HC12 Only)

MOVE Instruction	Addressing Modes	Offset Value
MOVB	IMM ⇒ IDX	+1
	EXT ⇒ IDX	+2
	IDX ⇒ EXT	–2
	IDX ⇒ IDX	–1 for first operand +1 for second operand
MOVW	IMM ⇒ IDX	+2
	EXT ⇒ IDX	+2
	IDX ⇒ EXT	–2
	IDX ⇒ IDX	–1 for first operand +1 for second operand

Example:

```
1000   18 09 C2 20 00    MOVB  $2000 2,PC
```

Moves a byte of data from $2000 to $1009

The expected location of the PC = $1005. The offset = +2.
[1005 + 2 (for 2,PC) + 2 (for correction) = 1009]

$18 is the page pre-byte, 09 is the MOVB opcode for ext-idx, C2 is the indexed postbyte for 2,PC (without correction).

The Motorola MCUasm assembler produces corrected object code for PC-relative moves (18 09 C0 20 00 for the example shown).

NOTE: *Instead of assembling the 2,PC as C2, the correction has been applied to make it C0. Check whether an assembler makes the correction before using PC-relative moves.*

On the newer HCS12, the instruction queue was implemented such that an internal pointer, to the start of the next instruction, is always available. On the HCS12, PC-relative move instructions work as expected without any offset adjustment. Although this is different from the original M68HC12, it is unlikely to be a problem because PC-relative indexing is rarely, if ever, used with move instructions.

3.10.2 Bit Manipulation Instructions

Bit manipulation instructions use either a combination of two or a combination of three addressing modes.

The clear bits in memory (BCLR) and set bits in memory (BSET) instructions use an 8-bit mask to determine which bits in a memory byte are to be changed. The mask must be supplied with the instruction as an immediate mode value. The memory location to be modified can be specified by means of direct, extended, or indexed addressing modes.

The branch if bits cleared (BRCLR) and branch if bits set (BRSET) instructions use an 8-bit mask to test the states of bits in a memory byte. The mask is supplied with the instruction as an immediate mode value. The memory location to be tested is specified by means of direct, extended, or indexed addressing modes. Relative addressing mode is used to determine the branch address. A signed 8-bit offset must be supplied with the instruction.

3.11 Addressing More than 64 Kbytes

Some M68HC12 devices incorporate hardware that supports addressing a larger memory space than the standard 64 Kbytes. The expanded memory system uses fast on-chip logic to implement a transparent bank-switching scheme.

Increased code efficiency is the greatest advantage of using a switching scheme instead of a large linear address space. In systems with large linear address spaces, instructions require more bits of information to address a memory location, and CPU overhead is greater. Other advantages include the ability to change the size of system memory and the ability to use various types of external memory.

However, the add-on bank switching schemes used in other microcontrollers have known weaknesses. These include the cost of external glue logic, increased programming overhead to change banks, and the need to disable interrupts while banks are switched.

The M68HC12 system requires no external glue logic. Bank switching overhead is reduced by implementing control logic in the MCU. Interrupts do not need to be disabled during switching because switching tasks are incorporated in special instructions that greatly simplify program access to extended memory.

MCUs with expanded memory treat the 16 Kbytes of memory space from $8000 to $BFFF as a program memory window. Expanded-memory architecture includes an 8-bit program page register (PPAGE), which allows up to 256 16-Kbyte program memory pages to be switched into and out of the program memory window. This provides for up to 4 Megabytes of paged program memory.

The CPU12 instruction set includes call subroutine in expanded memory (CALL) and return from call (RTC) instructions, which greatly simplify the use of expanded memory space. These instructions also execute correctly on devices that do not have expanded-memory addressing capability, thus providing for portable code.

The CALL instruction is similar to the jump-to-subroutine (JSR) instruction. When CALL is executed, the current value in PPAGE is pushed onto the stack with a return address, and a new instruction-supplied value is written to PPAGE. This value selects the page the called subroutine resides upon and can be considered part of

the effective address. For all addressing mode variations except indexed indirect modes, the new page value is provided by an immediate operand in the instruction. For indexed indirect variations of CALL, a pointer specifies memory locations where the new page value and the address of the called subroutine are stored. Use of indirect addressing for both the page value and the address within the page frees the program from keeping track of explicit values for either address.

The RTC instruction restores the saved program page value and the return address from the stack. This causes execution to resume at the next instruction after the original CALL instruction.

Addressing Modes

Section 4. Instruction Queue

4.1 Introduction

The CPU12 uses an instruction queue to increase execution speed. This section describes queue operation during normal program execution and changes in execution flow. These concepts augment the descriptions of instructions and cycle-by-cycle instruction execution in subsequent sections, but it is important to note that queue operation is automatic, and generally transparent to the user.

The material in this section is general. **Section 6. Instruction Glossary** contains detailed information concerning cycle-by-cycle execution of each instruction. **Section 8. Instruction Queue** contains detailed information about tracking queue operation and instruction execution.

4.2 Queue Description

The fetching mechanism in the CPU12 is best described as a queue rather than as a pipeline. Queue logic fetches program information and positions it for execution, but instructions are executed sequentially. A typical pipelined central processor unit (CPU) can execute more than one instruction at the same time, but interactions between the prefetch and execution mechanisms can make tracking and debugging difficult. The CPU12 thus gains the advantages of independent fetches, yet maintains a straightforward relationship between bus and execution cycles.

Each instruction refills the queue by fetching the same number of bytes that the instruction uses. Program information is fetched in aligned 16-bit words. Each program fetch (P) indicates that two bytes need to be replaced in the instruction queue. Each optional fetch (0) indicates that only one byte needs to be replaced. For example, an instruction composed of five bytes does two program fetches and one optional fetch. If the first byte of the five-byte instruction was even-aligned, the

optional fetch is converted into a free cycle. If the first byte was odd-aligned, the optional fetch is executed as a program fetch.

Two external pins, IPIPE[1:0], provide time-multiplexed information about data movement in the queue and instruction execution. Decoding and use of these signals is discussed in **Section 8. Instruction Queue**.

4.2.1 Original M68HC12 Queue Implementation

There are two 16-bit queue stages and one 16-bit buffer. Program information is fetched in aligned 16-bit words. Unless buffering is required, program information is first queued into stage 1, then advanced to stage 2 for execution.

At least two words of program information are available to the CPU when execution begins. The first byte of object code is in either the even or odd half of the word in stage 2, and at least two more bytes of object code are in the queue.

The buffer is used when a program word arrives before the queue can advance. This occurs during execution of single-byte and odd-aligned instructions. For instance, the queue cannot advance after an aligned, single-byte instruction is executed, because the first byte of the next instruction is also in stage 2. In these cases, information is latched into the buffer until the queue can advance.

4.2.2 HCS12 Queue Implementation

There are three 16-bit stages in the instruction queue. Instructions enter the queue at stage 1 and shift out of stage 3 as the CPU executes instructions and fetches new ones into stage 1. Each byte in the queue is selectable. An opcode prediction algorithm determines the location of the next opcode in the instruction queue.

4.3 Data Movement in the Queue

All queue operations are combinations of four basic queue movement cycles. Descriptions of each of these cycles follows. Queue movement cycles are only one factor in instruction execution time and should not be confused with bus cycles.

4.3.1 No Movement

There is no data movement in the instruction queue during the cycle. This occurs during execution of instructions that must perform a number of internal operations, such as division instructions.

4.3.2 Latch Data from Bus (Applies Only to the M68HC12 Queue Implementation)

All instructions initiate fetches to refill the queue as execution proceeds. However, a number of conditions, including instruction alignment and the length of previous instructions, affect when the queue advances. If the queue is not ready to advance when fetched information arrives, the information is latched into the buffer. Later, when the queue does advance, stage 1 is refilled from the buffer. If more than one latch cycle occurs before the queue advances, the buffer is filled on the first latch event and subsequent latch events are ignored until the queue advances.

4.3.3 Advance and Load from Data Bus

The content of queue is advanced by one stage, and stage 1 is loaded with a word of program information from the data bus. The information was requested two bus cycles earlier but has only become available this cycle, due to access delay.

4.3.4 Advance and Load from Buffer (Applies Only to M68HC12 Queue Implementation)

The content of queue stage 1 advances to stage 2, and stage 1 is loaded with a word of program information from the buffer. The information in the buffer was latched from the data bus during a previous cycle because the queue was not ready to advance when it arrived.

4.4 Changes in Execution Flow

During normal instruction execution, queue operations proceed as a continuous sequence of queue movement cycles. However, situations arise which call for changes in flow. These changes are categorized as resets, interrupts, subroutine calls, conditional branches, and jumps. Generally speaking, resets and interrupts are considered to be related to events outside the current program context that require special

processing, while subroutine calls, branches, and jumps are considered to be elements of program structure.

During design, great care is taken to assure that the mechanism that increases instruction throughput during normal program execution does not cause bottlenecks during changes of program flow, but internal queue operation is largely transparent to the user. The following information is provided to enhance subsequent descriptions of instruction execution.

4.4.1 Exceptions

Exceptions are events that require processing outside the normal flow of instruction execution. CPU12 exceptions include five types of exceptions:

- Reset (including COP, clock monitor, and pin)
- Unimplemented opcode trap
- Software interrupt instruction
- X-bit interrupts
- I-bit interrupts

All exceptions use the same microcode, but the CPU follows different execution paths for each type of exception.

CPU12 exception handling is designed to minimize the effect of queue operation on context switching. Thus, an exception vector fetch is the first part of exception processing, and fetches to refill the queue from the address pointed to by the vector are interleaved with the stacking operations that preserve context, so that program access time does not delay the switch. Refer to **Section 7. Exception Processing** for detailed information.

4.4.2 Subroutines

The CPU12 can branch to (BSR), jump to (JSR), or call (CALL) subroutines. BSR and JSR are used to access subroutines in the normal 64-Kbyte address space. The CALL instruction is intended for use in MCUs with expanded memory capability.

BSR uses relative addressing mode to generate the effective address of the subroutine, while JSR can use various other addressing modes. Both instructions calculate a return address, stack the address, then perform three program word fetches to refill the queue.

Subroutines in the normal 64-Kbyte address space are terminated with a return-from-subroutine (RTS) instruction. RTS unstacks the return address, then performs three program word fetches from that address to refill the queue.

CALL is similar to JSR. MCUs with expanded memory treat 16 Kbytes of addresses from $8000 to $BFFF as a memory window. An 8-bit PPAGE register switches memory pages into and out of the window. When CALL is executed, a return address is calculated, then it and the current PPAGE value are stacked, and a new instruction-supplied value is written to PPAGE. The subroutine address is calculated, then three program word fetches are made from that address to refill the instruction queue.

The return-from-call (RTC) instruction is used to terminate subroutines in expanded memory. RTC unstacks the PPAGE value and the return address, then performs three program word fetches from that address to refill the queue.

CALL and RTC execute correctly in the normal 64-Kbyte address space, thus providing for portable code. However, since extra execution cycles are required, routinely substituting CALL/RTC for JSR/RTS is not recommended.

4.4.3 Branches

Branch instructions cause execution flow to change when specific pre-conditions exist. The CPU12 instruction set includes:

- Short conditional branches
- Long conditional branches
- Bit-condition branches

Types and conditions of branch instructions are described in **5.19 Branch Instructions**. All branch instructions affect the queue similarly, but there are differences in overall cycle counts between the

various types. Loop primitive instructions are a special type of branch instruction used to implement counter-based loops.

Branch instructions have two execution cases:

- The branch condition is satisfied, and a change of flow takes place.
- The branch condition is not satisfied, and no change of flow occurs.

4.4.3.1 Short Branches

The “not-taken” case for short branches is simple. Since the instruction consists of a single word containing both an opcode and an 8-bit offset, the queue advances, another program word is fetched, and execution continues with the next instruction.

The “taken” case for short branches requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is calculated using the relative offset in the instruction. Then, the address is loaded into the program counter, and the CPU performs three program word fetches at the new address to refill the instruction queue.

4.4.3.2 Long Branches

The “not-taken” case for all long branches requires three cycles, while the “taken” case requires four cycles. This is due to differences in the amount of program information needed to fill the queue.

Long branch instructions begin with a $18 prebyte which indicates that the opcode is on page 2 of the opcode map. The CPU12 treats the prebyte as a special one-byte instruction. If the prebyte is not aligned, the first cycle is used to perform a program word access; if the prebyte is aligned, the first cycle is used to perform a free cycle. The first cycle for the prebyte is executed whether or not the branch is taken.

The first cycle of the branch instruction is an optional cycle. Optional cycles make the effects of byte-sized and misaligned instructions consistent with those of aligned word-length instructions. Program information is always fetched as aligned 16-bit words. When an instruction has an odd number of bytes, and the first byte is not aligned with an even byte boundary, the optional cycle makes an additional

program word access that maintains queue order. In all other cases, the optional cycle is a free cycle.

In the “not-taken” case, the queue must advance so that execution can continue with the next instruction. Two cycles are used to refill the queue. Alignment determines how the second of these cycles is used.

In the “taken” case, the effective address of the branch is calculated using the 16-bit relative offset contained in the second word of the instruction. This address is loaded into the program counter, then the CPU performs three program word fetches at the new address.

4.4.3.3 Bit Condition Branches

Bit condition branch instructions read a location in memory, and branch if the bits in that location are in a certain state. These instructions can use direct, extended, or indexed addressing modes. Indexed operations require varying amounts of information to determine the effective address, so instruction length varies according to the mode used, which in turn affects the amount of program information fetched. To shorten execution time, these branches perform one program word fetch in anticipation of the “taken” case. The data from this fetch is ignored in the “not-taken” case. If the branch is taken, the CPU fetches three program word fetches at the new address to fill the instruction queue.

4.4.3.4 Loop Primitives

The loop primitive instructions test a counter value in a register or accumulator and branch to an address specified by a 9-bit relative offset contained in the instruction if a specified condition is met. There are auto-increment and auto-decrement versions of these instructions. The test and increment/decrement operations are performed on internal CPU registers, and require no additional program information. To shorten execution time, these branches perform one program word fetch in anticipation of the “taken” case. The data from this fetch is ignored if the branch is not taken, and the CPU does one program fetch and one optional fetch to refill the queue[1]. If the branch is taken, the CPU finishes refilling the queue with two additional program word fetches at the new address.

1. In the original M68HC12, the implementation of these two cycles are both program word fetches.

4.4.4 Jumps

Jump (JMP) is the simplest change of flow instruction. JMP can use extended or indexed addressing. Indexed operations require varying amounts of information to determine the effective address, so instruction length varies according to the mode used, which in turn affects the amount of program information fetched. All forms of JMP perform three program word fetches at the new address to refill the instruction queue.

Section 5. Instruction Set Overview

5.1 Introduction

This section contains general information about the central processor unit (CPU12) instruction set. It is organized into instruction categories grouped by function.

5.2 Instruction Set Description

CPU12 instructions are a superset of the M68HC11 instruction set. Code written for an M68HC11 can be reassembled and run on a CPU12 with no changes. The CPU12 provides expanded functionality and increased code efficiency. There are two implementations of the CPU12, the original M68HC12 and the newer HCS12. Both implementations have the same instruction set, although there are small differences in cycle-by-cycle access details (the order of some bus cycles changed to accommodate differences in the way the instruction queue was implemented). These minor differences are transparent for most users.

In the M68HC12 and HCS12 architecture, all memory and input/output (I/O) are mapped in a common 64-Kbyte address space (memory-mapped I/O). This allows the same set of instructions to be used to access memory, I/O, and control registers. General-purpose load, store, transfer, exchange, and move instructions facilitate movement of data to and from memory and peripherals.

The CPU12 has a full set of 8-bit and 16-bit mathematical instructions. There are instructions for signed and unsigned arithmetic, division, and multiplication with 8-bit, 16-bit, and some larger operands.

Special arithmetic and logic instructions aid stacking operations, indexing, binary-coded decimal (BCD) calculation, and condition code register manipulation. There are also dedicated instructions for multiply and accumulate operations, table interpolation, and specialized fuzzy logic operations that involve mathematical calculations.

Refer to **Section 6. Instruction Glossary** for detailed information about individual instructions. **Appendix A. Instruction Reference** contains quick-reference material, including an opcode map and postbyte encoding for indexed addressing, transfer/exchange instructions, and loop primitive instructions.

5.3 Load and Store Instructions

Load instructions copy memory content into an accumulator or register. Memory content is not changed by the operation. Load instructions (but not LEA_ instructions) affect condition code bits so no separate test instructions are needed to check the loaded values for negative or 0 conditions.

Store instructions copy the content of a CPU register to memory. Register/accumulator content is not changed by the operation. Store instructions automatically update the N and Z condition code bits, which can eliminate the need for a separate test instruction in some programs.

Table 5-1 is a summary of load and store instructions.

Table 5-1. Load and Store Instructions

Mnemonic	Function	Operation
Load Instructions		
LDAA	Load A	(M) ⇒ A
LDAB	Load B	(M) ⇒ B
LDD	Load D	(M : M + 1) ⇒ (A:B)
LDS	Load SP	(M : M + 1) ⇒ SP_H:SP_L
LDX	Load index register X	(M : M + 1) ⇒ X_H:X_L
LDY	Load index register Y	(M : M + 1) ⇒ Y_H:Y_L
LEAS	Load effective address into SP	Effective address ⇒ SP
LEAX	Load effective address into X	Effective address ⇒ X
LEAY	Load effective address into Y	Effective address ⇒ Y

Continued on next page

Table 5-1. Load and Store Instructions (Continued)

Store Instructions		
STAA	Store A	(A) ⇒ M
STAB	Store B	(B) ⇒ M
STD	Store D	(A) ⇒ M, (B) ⇒ M + 1
STS	Store SP	(SP_H:SP_L) ⇒ M : M + 1
STX	Store X	(X_H:X_L) ⇒ M : M + 1
STY	Store Y	(Y_H:Y_L) ⇒ M : M + 1

5.4 Transfer and Exchange Instructions

Transfer instructions copy the content of a register or accumulator into another register or accumulator. Source content is not changed by the operation. Transfer register to register (TFR) is a universal transfer instruction, but other mnemonics are accepted for compatibility with the M68HC11. The transfer A to B (TAB) and transfer B to A (TBA) instructions affect the N, Z, and V condition code bits in the same way as M68HC11 instructions. The TFR instruction does not affect the condition code bits.

The sign extend 8-bit operand (SEX) instruction is a special case of the universal transfer instruction that is used to sign extend 8-bit two's complement numbers so that they can be used in 16-bit operations. The 8-bit number is copied from accumulator A, accumulator B, or the condition code register to accumulator D, the X index register, the Y index register, or the stack pointer. All the bits in the upper byte of the 16-bit result are given the value of the most-significant bit (MSB) of the 8-bit number.

Exchange instructions exchange the contents of pairs of registers or accumulators. When the first operand in an EXG instruction is 8-bits and the second operand is 16 bits, a zero-extend operation is performed on the 8-bit register as it is copied into the 16-bit register.

Section 6. Instruction Glossary contains information concerning other transfers and exchanges between 8- and 16-bit registers.

Table 5-2 is a summary of transfer and exchange instructions.

Table 5-2. Transfer and Exchange Instructions

Mnemonic	Function	Operation
Transfer Instructions		
TAB	Transfer A to B	(A) ⇒ B
TAP	Transfer A to CCR	(A) ⇒ CCR
TBA	Transfer B to A	(B) ⇒ A
TFR	Transfer register to register	(A, B, CCR, D, X, Y, or SP) ⇒ A, B, CCR, D, X, Y, or SP
TPA	Transfer CCR to A	(CCR) ⇒ A
TSX	Transfer SP to X	(SP) ⇒ X
TSY	Transfer SP to Y	(SP) ⇒ Y
TXS	Transfer X to SP	(X) ⇒ SP
TYS	Transfer Y to SP	(Y) ⇒ SP
Exchange Instructions		
EXG	Exchange register to register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	Exchange D with X	(D) ⇔ (X)
XGDY	Exchange D with Y	(D) ⇔ (Y)
Sign Extension Instruction		
SEX	Sign extend 8-Bit operand	Sign-extended (A, B, or CCR) ⇒ D, X, Y, or SP

5.5 Move Instructions

Move instructions move (copy) data bytes or words from a source (M_1 or M : M +1_1) to a destination (M_2 or M : M +1_2) in memory. Six combinations of immediate, extended, and indexed addressing are allowed to specify source and destination addresses (IMM ⇒ EXT, IMM ⇒ IDX, EXT ⇒ EXT, EXT ⇒ IDX, IDX ⇒ EXT, IDX ⇒ IDX). Addressing mode combinations with immediate for the destination would not be useful.

Table 5-3 shows byte and word move instructions.

Table 5-3. Move Instructions

Mnemonic	Function	Operation
MOVB	Move byte (8-bit)	(M_1) ⇒ M_2
MOVW	Move word (16-bit)	(M : M + 1_1) ⇒ M : M + 1_2

5.6 Addition and Subtraction Instructions

Signed and unsigned 8- and 16-bit addition can be performed between registers or between registers and memory. Special instructions support index calculation. Instructions that add the carry bit in the condition code register (CCR) facilitate multiple precision computation.

Signed and unsigned 8- and 16-bit subtraction can be performed between registers or between registers and memory. Special instructions support index calculation. Instructions that subtract the carry bit in the CCR facilitate multiple precision computation. Refer to **Table 5-4** for addition and subtraction instructions.

Load effective address (LEAS, LEAX, and LEAY) instructions could also be considered as specialized addition and subtraction instructions. See **5.25 Pointer and Index Calculation Instructions** for more information.

Table 5-4. Addition and Subtraction Instructions

Mnemonic	Function	Operation
Addition Instructions		
ABA	Add B to A	(A) + (B) ⇒ A
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB	Add with carry to B	(B) + (M) + C ⇒ B
ADDA	Add without carry to A	(A) + (M) ⇒ A
ADDB	Add without carry to B	(B) + (M) ⇒ B
ADDD	Add to D	(A:B) + (M : M + 1) ⇒ A : B
Subtraction Instructions		
SBA	Subtract B from A	(A) – (B) ⇒ A
SBCA	Subtract with borrow from A	(A) – (M) – C ⇒ A
SBCB	Subtract with borrow from B	(B) – (M) – C ⇒ B
SUBA	Subtract memory from A	(A) – (M) ⇒ A
SUBB	Subtract memory from B	(B) – (M) ⇒ B
SUBD	Subtract memory from D (A:B)	(D) – (M : M + 1) ⇒ D

5.7 Binary-Coded Decimal Instructions

To add binary-coded decimal (BCD) operands, use addition instructions that set the half-carry bit in the CCR, then adjust the result with the decimal adjust A (DAA) instruction. **Table 5-5** is a summary of instructions that can be used to perform BCD operations.

Table 5-5. BCD Instructions

Mnemonic	Function	Operation
ABA	Add B to A	(A) + (B) ⇒ A
ADCA	Add with carry to A	(A) + (M) + C ⇒ A
ADCB(1)	Add with carry to B	(B) + (M) + C ⇒ B
ADDA(1)	Add memory to A	(A) + (M) ⇒ A
ADDB	Add memory to B	(B) + (M) ⇒ B
DAA	Decimal adjust A	$(A)_{10}$

1. These instructions are not normally used for BCD operations because, although they affect H correctly, they do not leave the result in the correct accumulator (A) to be used with the DAA instruction. Thus additional steps would be needed to adjust the result to correct BCD form.

5.8 Decrement and Increment Instructions

The decrement and increment instructions are optimized 8- and 16-bit addition and subtraction operations. They are generally used to implement counters. Because they do not affect the carry bit in the CCR, they are particularly well suited for loop counters in multiple-precision computation routines. Refer to **5.20 Loop Primitive Instructions** for information concerning automatic counter branches. **Table 5-6** is a summary of decrement and increment instructions.

Table 5-6. Decrement and Increment Instructions

Mnemonic	Function	Operation
Decrement Instructions		
DEC	Decrement memory	(M) – $01 ⇒ M
DECA	Decrement A	(A) – $01 ⇒ A
DECB	Decrement B	(B) – $01 ⇒ B
DES	Decrement SP	(SP) – $0001 ⇒ SP
DEX	Decrement X	(X) – $0001 ⇒ X
DEY	Decrement Y	(Y) – $0001 ⇒ Y
Increment Instructions		
INC	Increment memory	(M) + $01 ⇒ M
INCA	Increment A	(A) + $01 ⇒ A
INCB	Increment B	(B) + $01 ⇒ B
INS	Increment SP	(SP) + $0001 ⇒ SP
INX	Increment X	(X) + $0001 ⇒ X
INY	Increment Y	(Y) + $0001 ⇒ Y

5.9 Compare and Test Instructions

Compare and test instructions perform subtraction between a pair of registers or between a register and memory. The result is not stored, but condition codes are set by the operation. These instructions are generally used to establish conditions for branch instructions. In this architecture, most instructions update condition code bits automatically, so it is often unnecessary to include separate test or compare instructions. **Table 5-7** is a summary of compare and test instructions.

Table 5-7. Compare and Test Instructions

Mnemonic	Function	Operation
Compare Instructions		
CBA	Compare A to B	(A) – (B)
CMPA	Compare A to memory	(A) – (M)
CMPB	Compare B to memory	(B) – (M)
CPD	Compare D to memory (16-bit)	(A : B) – (M : M + 1)
CPS	Compare SP to memory (16-bit)	(SP) – (M : M + 1)
CPX	Compare X to memory (16-bit)	(X) – (M : M + 1)
CPY	Compare Y to memory (16-bit)	(Y) – (M : M + 1)
Test Instructions		
TST	Test memory for zero or minus	(M) – $00
TSTA	Test A for zero or minus	(A) – $00
TSTB	Test B for zero or minus	(B) – $00

5.10 Boolean Logic Instructions

The Boolean logic instructions perform a logic operation between an 8-bit accumulator or the CCR and a memory value. AND, OR, and exclusive OR functions are supported. **Table 5-8** summarizes logic instructions.

Table 5-8. Boolean Logic Instructions

Mnemonic	Function	Operation
ANDA	AND A with memory	(A) • (M) ⇒ A
ANDB	AND B with memory	(B) • (M) ⇒ B
ANDCC	AND CCR with memory (clear CCR bits)	(CCR) • (M) ⇒ CCR
EORA	Exclusive OR A with memory	(A) ⊕ (M) ⇒ A
EORB	Exclusive OR B with memory	(B) ⊕ (M) ⇒ B
ORAA	OR A with memory	(A) + (M) ⇒ A
ORAB	OR B with memory	(B) + (M) ⇒ B
ORCC	OR CCR with memory (set CCR bits)	(CCR) + (M) ⇒ CCR

5.11 Clear, Complement, and Negate Instructions

Each of the clear, complement, and negate instructions performs a specific binary operation on a value in an accumulator or in memory. Clear operations clear the value to 0, complement operations replace the value with its one's complement, and negate operations replace the value with its two's complement. **Table 5-9** is a summary of clear, complement, and negate instructions.

Table 5-9. Clear, Complement, and Negate Instructions

Mnemonic	Function	Operation
CLC	Clear C bit in CCR	$0 \Rightarrow C$
CLI	Clear I bit in CCR	$0 \Rightarrow I$
CLR	Clear memory	$\$00 \Rightarrow M$
CLRA	Clear A	$\$00 \Rightarrow A$
CLRB	Clear B	$\$00 \Rightarrow B$
CLV	Clear V bit in CCR	$0 \Rightarrow V$
COM	One's complement memory	$\$FF - (M) \Rightarrow M$ or $(\overline{M}) \Rightarrow M$
COMA	One's complement A	$\$FF - (A) \Rightarrow A$ or $(\overline{A}) \Rightarrow A$
COMB	One's complement B	$\$FF - (B) \Rightarrow B$ or $(\overline{B}) \Rightarrow B$
NEG	Two's complement memory	$\$00 - (M) \Rightarrow M$ or $(\overline{M}) + 1 \Rightarrow M$
NEGA	Two's complement A	$\$00 - (A) \Rightarrow A$ or $(\overline{A}) + 1 \Rightarrow A$
NEGB	Two's complement B	$\$00 - (B) \Rightarrow B$ or $(\overline{B}) + 1 \Rightarrow B$

5.12 Multiplication and Division Instructions

There are instructions for signed and unsigned 8- and 16-bit multiplication. Eight-bit multiplication operations have a 16-bit product. Sixteen-bit multiplication operations have 32-bit products.

Integer and fractional division instructions have 16-bit dividend, divisor, quotient, and remainder. Extended division instructions use a 32-bit dividend and a 16-bit divisor to produce a 16-bit quotient and a 16-bit remainder.

Table 5-10 is a summary of multiplication and division instructions.

Table 5-10. Multiplication and Division Instructions

Mnemonic	Function	Operation
Multiplication Instructions		
EMUL	16 by 16 multiply (unsigned)	(D) × (Y) ⇒ Y : D
EMULS	16 by 16 multiply (signed)	(D) × (Y) ⇒ Y : D
MUL	8 by 8 multiply (unsigned)	(A) × (B) ⇒ A : B
Division Instructions		
EDIV	32 by 16 divide (unsigned)	(Y : D) ÷ (X) ⇒ Y Remainder ⇒ D
EDIVS	32 by 16 divide (signed)	(Y : D) ÷ (X) ⇒ Y Remainder ⇒ D
FDIV	16 by 16 fractional divide	(D) ÷ (X) ⇒ X Remainder ⇒ D
IDIV	16 by 16 integer divide (unsigned)	(D) ÷ (X) ⇒ X Remainder ⇒ D
IDIVS	16 by 16 integer divide (signed)	(D) ÷ (X) ⇒ X Remainder ⇒ D

5.13 Bit Test and Manipulation Instructions

The bit test and manipulation operations use a mask value to test or change the value of individual bits in an accumulator or in memory. Bit test A (BITA) and bit test B (BITB) provide a convenient means of testing bits without altering the value of either operand. **Table 5-11** is a summary of bit test and manipulation instructions.

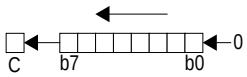
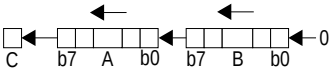
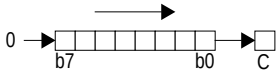
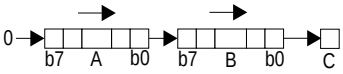
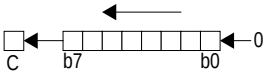
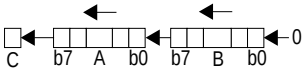
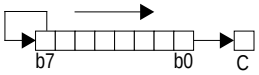
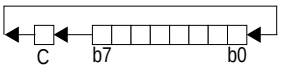
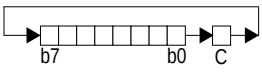
Table 5-11. Bit Test and Manipulation Instructions

Mnemonic	Function	Operation
BCLR	Clear bits in memory	(M) • ($\overline{mm}$) ⇒ M
BITA	Bit test A	(A) • (M)
BITB	Bit test B	(B) • (M)
BSET	Set bits in memory	(M) + (mm) ⇒ M

5.14 Shift and Rotate Instructions

There are shifts and rotates for all accumulators and for memory bytes. All pass the shifted-out bit through the C status bit to facilitate multiple-byte operations. Because logical and arithmetic left shifts are identical, there are no separate logical left shift operations. Logic shift left (LSL) mnemonics are assembled as arithmetic shift left memory (ASL) operations. **Table 5-12** shows shift and rotate instructions.

Table 5-12. Shift and Rotate Instructions

Mnemonic	Function	Operation
Logical Shifts		
LSL LSLA LSLB	Logic shift left memory Logic shift left A Logic shift left B	
LSLD	Logic shift left D	
LSR LSRA LSRB	Logic shift right memory Logic shift right A Logic shift right B	
LSRD	Logic shift right D	
Arithmetic Shifts		
ASL ASLA ASLB	Arithmetic shift left memory Arithmetic shift left A Arithmetic shift left B	
ASLD	Arithmetic shift left D	
ASR ASRA ASRB	Arithmetic shift right memory Arithmetic shift right A Arithmetic shift right B	
Rotates		
ROL ROLA ROLB	Rotate left memory through carry Rotate left A through carry Rotate left B through carry	
ROR RORA RORB	Rotate right memory through carry Rotate right A through carry Rotate right B through carry	

5.15 Fuzzy Logic Instructions

The CPU12 instruction set includes instructions that support efficient processing of fuzzy logic operations. The descriptions of fuzzy logic instructions given here are functional overviews. **Table 5-13** summarizes the fuzzy logic instructions. Refer to **Section 9. Fuzzy Logic Support** for detailed discussion.

5.15.1 Fuzzy Logic Membership Instruction

The membership function (MEM) instruction is used during the fuzzification process. During fuzzification, current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y value for the current input on a trapezoidal membership function for each label of each system input. The MEM instruction performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure.

5.15.2 Fuzzy Logic Rule Evaluation Instructions

The MIN-MAX rule evaluation (REV and REVW) instructions perform MIN-MAX rule evaluations that are central elements of a fuzzy logic inference program. Fuzzy input values are processed using a list of rules from the knowledge base to produce a list of fuzzy outputs. The REV instruction treats all rules as equally important. The REVW instruction allows each rule to have a separate weighting factor. The two rule evaluation instructions also differ in the way rules are encoded into the knowledge base. Because they require a number of cycles to execute, rule evaluation instructions can be interrupted. Once the interrupt has been serviced, instruction execution resumes at the point the interrupt occurred.

5.15.3 Fuzzy Logic Weighted Average Instruction

The weighted average (WAV) instruction computes a sum-of-products and a sum-of-weights used for defuzzification. To be usable, the fuzzy outputs produced by rule evaluation must be defuzzified to produce a single output value which represents the combined effect of all of the fuzzy outputs. Fuzzy outputs correspond to the labels of a system output and each is defined by a membership function in the knowledge base. The CPU12 typically uses singletons for output membership functions rather than the trapezoidal shapes used for inputs. As with inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function. The WAV instruction calculates the numerator and denominator sums for a weighted average of the fuzzy outputs. Because WAV requires a number of cycles to execute, it can be interrupted. The WAVR pseudo-instruction causes execution to resume at the point where it was interrupted.

Table 5-13. Fuzzy Logic Instructions

Mnemonic	Function	Operation
MEM	Membership function	μ (grade) $\Rightarrow M_{(Y)}$ (X) + 4 $\Rightarrow$ X; (Y) + 1 $\Rightarrow$ Y; A unchanged if (A) < P1 or (A) > P2, then μ = 0, else μ = MIN [((A) – P1) $\times$ S1, (P2 – (A)) $\times$ S2, $FF] where: A = current crisp input value X points to a 4-byte data structure that describes a trapezoidal membership function as base intercept points and slopes (P1, P2, S1, S2) Y points at fuzzy input (RAM location)

Continued on next page

Table 5-13. Fuzzy Logic Instructions (Continued)

Mnemonic	Function	Operation
REV	MIN-MAX rule evaluation	Find smallest rule input (MIN) Store to rule outputs unless fuzzy output is larger (MAX) Rules are unweighted Each rule input is an 8-bit offset from a base address in Y Each rule output is an 8-bit offset from a base address in Y $FE separates rule inputs from rule outputs $FF terminates the rule list REV can be interrupted
REVW	MIN-MAX rule evaluation	Find smallest rule input (MIN) Multiply by a rule weighting factor (optional) Store to rule outputs unless fuzzy output is larger (MAX) Each rule input is the 16-bit address of a fuzzy input Each rule output is the 16-bit address of a fuzzy output Address $FFFE separates rule inputs from rule outputs $FFFF terminates the rule list Weights are 8-bit values in a separate table REVW can be interrupted
WAV	Calculates numerator (sum of products) and denominator (sum of weights) for weighted average calculation Results are placed in correct registers for EDIV immediately after WAV	$\sum_{i=1}^{B} S_i F_i \Rightarrow Y{:}D$ $\sum_{i=1}^{B} F_i \Rightarrow X$
WAVR	Resumes execution of interrupted WAV instruction	Recover immediate results from stack rather than initializing them to 0.

5.16 Maximum and Minimum Instructions

The maximum (MAX) and minimum (MIN) instructions are used to make comparisons between an accumulator and a memory location. These instructions can be used for linear programming operations, such as simplex-method optimization, or for fuzzification.

MAX and MIN instructions use accumulator A to perform 8-bit comparisons, while EMAX and EMIN instructions use accumulator D to perform 16-bit comparisons. The result (maximum or minimum value) can be stored in the accumulator (EMAXD, EMIND, MAXA, MINA) or the memory address (EMAXM, EMINM, MAXM, MINM).

Table 5-14 is a summary of minimum and maximum instructions.

Table 5-14. Minimum and Maximum Instructions

Mnemonic	Function	Operation
Minimum Instructions		
EMIND	MIN of two unsigned 16-bit values result to accumulator	MIN ((D), (M : M + 1)) ⇒ D
EMINM	MIN of two unsigned 16-bit values result to memory	MIN ((D), (M : M + 1)) ⇒ M : M+1
MINA	MIN of two unsigned 8-bit values result to accumulator	MIN ((A), (M)) ⇒ A
MINM	MIN of two unsigned 8-bit values result to memory	MIN ((A), (M)) ⇒ M
Maximum Instructions		
EMAXD	MAX of two unsigned 16-bit values result to accumulator	MAX ((D), (M : M + 1)) ⇒ D
EMAXM	MAX of two unsigned 16-bit values result to memory	MAX ((D), (M : M + 1)) ⇒ M : M + 1
MAXA	MAX of two unsigned 8-bit values result to accumulator	MAX ((A), (M)) ⇒ A
MAXM	MAX of two unsigned 8-bit values result to memory	MAX ((A), (M)) ⇒ M

5.17 Multiply and Accumulate Instruction

The multiply and accumulate (EMACS) instruction multiplies two 16-bit operands stored in memory and accumulates the 32-bit result in a third memory location. EMACS can be used to implement simple digital filters and defuzzification routines that use 16-bit operands. The WAV instruction incorporates an 8- to 16-bit multiply and accumulate operation that obtains a numerator for the weighted average calculation. The EMACS instruction can automate this portion of the averaging operation when 16-bit operands are used. **Table 5-15** shows the EMACS instruction.

Table 5-15. Multiply and Accumulate Instructions

Mnemonic	Function	Operation
EMACS	Multiply and accumulate (signed) 16 bit by 16 bit ⇒ 32 bit	$((M_{(X)}:M_{(X+1)}) \times (M_{(Y)}:M_{(Y+1)}))$ + (M ~ M + 3) ⇒ M ~ M + 3

5.18 Table Interpolation Instructions

The table interpolation instructions (TBL and ETBL) interpolate values from tables stored in memory. Any function that can be represented as a series of linear equations can be represented by a table of appropriate size. Interpolation can be used for many purposes, including tabular fuzzy logic membership functions. TBL uses 8-bit table entries and returns an 8-bit result; ETBL uses 16-bit table entries and returns a 16-bit result. Use of indexed addressing mode provides great flexibility in structuring tables.

Consider each of the successive values stored in a table to be y-values for the endpoint of a line segment. The value in the B accumulator before instruction execution begins represents the change in x from the beginning of the line segment to the lookup point divided by total change in x from the beginning to the end of the line segment. B is treated as an 8-bit binary fraction with radix point left of the MSB, so each line segment is effectively divided into 256 smaller segments. During instruction execution, the change in y between the beginning and end of the segment (a signed byte for TBL or a signed word for ETBL) is multiplied by the content of the B accumulator to obtain an intermediate delta-y term. The result (stored in the A accumulator by TBL, and in the D

accumulator by ETBL) is the y-value of the beginning point plus the signed intermediate delta-y value. **Table 5-16** shows the table interpolation instructions.

Table 5-16. Table Interpolation Instructions

Mnemonic	Function	Operation
ETBL	16-bit table lookup and interpolate (no indirect addressing modes allowed)	(M : M + 1) + [(B) × ((M + 2 : M + 3) – (M : M + 1))] ⇒ D Initialize B, and index before ETBL. <ea> points to the first table entry (M : M + 1) B is fractional part of lookup value
TBL	8-bit table lookup and interpolate (no indirect addressing modes allowed)	(M) + [(B) × ((M + 1) – (M))] ⇒ A Initialize B, and index before TBL. <ea> points to the first 8-bit table entry (M) B is fractional part of lookup value.

5.19 Branch Instructions

Branch instructions cause a sequence to change when specific conditions exist. The CPU12 uses three kinds of branch instructions. These are short branches, long branches, and bit condition branches.

Branch instructions can also be classified by the type of condition that must be satisfied in order for a branch to be taken. Some instructions belong to more than one classification. For example:

- Unary branch instructions always execute.
- Simple branches are taken when a specific bit in the condition code register is in a specific state as a result of a previous operation.
- Unsigned branches are taken when comparison or test of unsigned quantities results in a specific combination of condition code register bits.
- Signed branches are taken when comparison or test of signed quantities results in a specific combination of condition code register bits.

5.19.1 Short Branch Instructions

Short branch instructions operate this way: When a specified condition is met, a signed 8-bit offset is added to the value in the program counter. Program execution continues at the new address.

The numeric range of short branch offset values is $80 (–128) to $7F (127) from the address of the next memory location after the offset value.

Table 5-17 is a summary of the short branch instructions.

Table 5-17. Short Branch Instructions

Mnemonic	Function		Equation or Operation
Unary Branches			
BRA	Branch always		1 = 1
BRN	Branch never		1 = 0
Simple Branches			
BCC	Branch if carry clear		C = 0
BCS	Branch if carry set		C = 1
BEQ	Branch if equal		Z = 1
BMI	Branch if minus		N = 1
BNE	Branch if not equal		Z = 0
BPL	Branch if plus		N = 0
BVC	Branch if overflow clear		V = 0
BVS	Branch if overflow set		V = 1
Unsigned Branches			
		Relation	
BHI	Branch if higher	R > M	C + Z = 0
BHS	Branch if higher or same	R ≥ M	C = 0
BLO	Branch if lower	R < M	C = 1
BLS	Branch if lower or same	R ≤ M	C + Z = 1
Signed Branches			
BGE	Branch if greater than or equal	R ≥ M	N ⊕ V = 0
BGT	Branch if greater than	R > M	Z + (N ⊕ V) = 0
BLE	Branch if less than or equal	R ≤ M	Z + (N ⊕ V) = 1
BLT	Branch if less than	R < M	N ⊕ V = 1

5.19.2 Long Branch Instructions

Long branch instructions operate this way: When a specified condition is met, a signed 16-bit offset is added to the value in the program counter. Program execution continues at the new address. Long branches are used when large displacements between decision-making steps are necessary.

The numeric range of long branch offset values is $8000 (–32,768) to $7FFF (32,767) from the address of the next memory location after the offset value. This permits branching from any location in the standard 64-Kbyte address map to any other location in the 64-Kbyte map.

Table 5-18 is a summary of the long branch instructions.

Table 5-18. Long Branch Instructions

Mnemonic	Function	Equation or Operation
Unary Branches		
LBRA	Long branch always	1 = 1
LBRN	Long branch never	1 = 0
Simple Branches		
LBCC	Long branch if carry clear	C = 0
LBCS	Long branch if carry set	C = 1
LBEQ	Long branch if equal	Z = 1
LBMI	Long branch if minus	N = 1
LBNE	Long branch if not equal	Z = 0
LBPL	Long branch if plus	N = 0
LBVC	Long branch if overflow clear	V = 0
LBVS	Long branch if overflow set	V = 1
Unsigned Branches		
LBHI	Long branch if higher	C + Z = 0
LBHS	Long branch if higher or same	C = 0
LBLO	Long branch if lower	Z = 1
LBLS	Long branch if lower or same	C + Z = 1
Signed Branches		
LBGE	Long branch if greater than or equal	N ⊕ V = 0
LBGT	Long branch if greater than	Z + (N ⊕ V) = 0
LBLE	Long branch if less than or equal	Z + (N ⊕ V) = 1
LBLT	Long branch if less than	N ⊕ V = 1

5.19.3 Bit Condition Branch Instructions

The bit condition branches are taken when bits in a memory byte are in a specific state. A mask operand is used to test the location. If all bits in that location that correspond to ones in the mask are set (BRSET) or cleared (BRCLR), the branch is taken.

The numeric range of 8-bit offset values is $80 (–128) to $7F (127) from the address of the next memory location after the offset value.

Table 5-19 is a summary of bit condition branches.

Table 5-19. Bit Condition Branch Instructions

Mnemonic	Function	Equation or Operation
BRCLR	Branch if selected bits clear	(M) • (mm) = 0
BRSET	Branch if selected bits set	($\overline{M}$) • (mm) = 0

5.20 Loop Primitive Instructions

The loop primitives can also be thought of as counter branches. The instructions test a counter value in a register or accumulator (A, B, D, X, Y, or SP) for zero or non-zero value as a branch condition. There are predecrement, preincrement, and test-only versions of these instructions.

The numeric range of 9-bit offset values is $100 (–256) to $0FF (255) from the address of the next memory location after the offset value.

Table 5-20 is a summary of loop primitive branches.

Table 5-20. Loop Primitive Instructions

Mnemonic	Function	Equation or Operation
DBEQ	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1⇒ counter If (counter) = 0, then branch; else continue to next instruction
DBNE	Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1⇒ counter If (counter) not = 0, then branch; else continue to next instruction
IBEQ	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1⇒ counter If (counter) = 0, then branch; else continue to next instruction
IBNE	Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1⇒ counter If (counter) not = 0, then branch; else continue to next instruction
TBEQ	Test counter and branch if = 0 (counter = A, B, D, X,Y, or SP)	If (counter) = 0, then branch; else continue to next instruction
TBNE	Test counter and branch if ≠ 0 (counter = A, B, D, X,Y, or SP)	If (counter) not = 0, then branch; else continue to next instruction

5.21 Jump and Subroutine Instructions

Jump (JMP) instructions cause immediate changes in sequence. The JMP instruction loads the PC with an address in the 64-Kbyte memory map, and program execution continues at that address. The address can be provided as an absolute 16-bit address or determined by various forms of indexed addressing.

Subroutine instructions optimize the process of transferring control to a code segment that performs a particular task. A short branch (BSR), a jump to subroutine (JSR), or an expanded-memory call (CALL) can be used to initiate subroutines. There is no LBSR instruction, but a PC-relative JSR performs the same function. A return address is stacked, then execution begins at the subroutine address. Subroutines in the normal 64-Kbyte address space are terminated with a return-from-subroutine (RTS) instruction. RTS unstacks the return address so that execution resumes with the instruction after BSR or JSR.

The call subroutine in expanded memory (CALL) instruction is intended for use with expanded memory. CALL stacks the value in the PPAGE register and the return address, then writes a new value to PPAGE to select the memory page where the subroutine resides. The page value is an immediate operand in all addressing modes except indexed indirect modes; in these modes, an operand points to locations in memory where the new page value and subroutine address are stored. The return from call (RTC) instruction is used to terminate subroutines in expanded memory. RTC unstacks the PPAGE value and the return address so that execution resumes with the next instruction after CALL. For software compatibility, CALL and RTC execute correctly on devices that do not have expanded addressing capability. **Table 5-21** summarizes the jump and subroutine instructions.

Table 5-21. Jump and Subroutine Instructions

Mnemonic	Function	Operation
BSR	Branch to subroutine	SP – 2 ⇒ SP $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address ⇒ PC
CALL	Call subroutine in expanded memory	SP – 2 ⇒ SP $RTN_H:RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ SP – 1 ⇒ SP (PPAGE) ⇒ $M_{(SP)}$ Page ⇒ PPAGE Subroutine address ⇒ PC
JMP	Jump	Address ⇒ PC
JSR	Jump to subroutine	SP – 2 ⇒ SP $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address ⇒ PC
RTC	Return from call	$M_{(SP)}$ ⇒ PPAGE SP + 1 ⇒ SP $M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ SP + 2 ⇒ SP
RTS	Return from subroutine	$M_{(SP)} : M_{(SP+1)} \Rightarrow PC_H : PC_L$ SP + 2 ⇒ SP

5.22 Interrupt Instructions

Interrupt instructions handle transfer of control to a routine that performs a critical task. Software interrupts are a type of exception. **Section 7. Exception Processing** covers interrupt exception processing in detail.

The software interrupt (SWI) instruction initiates synchronous exception processing. First, the return PC value is stacked. After CPU context is stacked, execution continues at the address pointed to by the SWI vector.

Execution of the SWI instruction causes an interrupt without an interrupt service request. SWI is not inhibited by global mask bits I and X in the CCR, and execution of SWI sets the I mask bit. Once an SWI interrupt begins, maskable interrupts are inhibited until the I bit in the CCR is cleared. This typically occurs when a return from interrupt (RTI) instruction at the end of the SWI service routine restores context.

The CPU12 uses a variation of the software interrupt for unimplemented opcode trapping. There are opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the unimplemented opcodes on page 2, an opcode trap interrupt occurs. Traps are essentially interrupts that share the $FFF8:$FFF9 interrupt vector.

The RTI instruction is used to terminate all exception handlers, including interrupt service routines. RTI first restores the CCR, B:A, X, Y, and the return address from the stack. If no other interrupt is pending, normal execution resumes with the instruction following the last instruction that executed prior to interrupt.

Table 5-22 is a summary of interrupt instructions.

Table 5-22. Interrupt Instructions

Mnemonic	Function	Operation
RTI	Return from interrupt	$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$
SWI	Software interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$
TRAP	Unimplemented opcode interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$

5.23 Index Manipulation Instructions

The index manipulation instructions perform 8- and 16-bit operations on the three index registers and accumulators, other registers, or memory, as shown in **Table 5-23**.

Table 5-23. Index Manipulation Instructions

Mnemonic	Function	Operation
Addition Instructions		
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
Compare Instructions		
CPS	Compare SP to memory	(SP) – (M : M + 1)
CPX	Compare X to memory	(X) – (M : M + 1)
CPY	Compare Y to memory	(Y) – (M : M + 1)
Load Instructions		
LDS	Load SP from memory	M : M+1 ⇒ SP
LDX	Load X from memory	(M : M + 1) ⇒ X
LDY	Load Y from memory	(M : M + 1) ⇒ Y
LEAS	Load effective address into SP	Effective address ⇒ SP
LEAX	Load effective address into X	Effective address ⇒ X
LEAY	Load effective address into Y	Effective address ⇒ Y
Store Instructions		
STS	Store SP in memory	(SP) ⇒ M:M+1
STX	Store X in memory	(X) ⇒ M : M + 1
STY	Store Y in memory	(Y) ⇒ M : M + 1
Transfer Instructions		
TFR	Transfer register to register	(A, B, CCR, D, X, Y, or SP) ⇒ A, B, CCR, D, X, Y, or SP
TSX	Transfer SP to X	(SP) ⇒ X
TSY	Transfer SP to Y	(SP) ⇒ Y
TXS	transfer X to SP	(X) ⇒ SP
TYS	transfer Y to SP	(Y) ⇒ SP
Exchange Instructions		
EXG	Exchange register to register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	EXchange D with X	(D) ⇔ (X)
XGDY	EXchange D with Y	(D) ⇔ (Y)

5.24 Stacking Instructions

The two types of stacking instructions, are shown in **Table 5-24**. Stack pointer instructions use specialized forms of mathematical and data transfer instructions to perform stack pointer manipulation. Stack operation instructions save information on and retrieve information from the system stack.

Table 5-24. Stacking Instructions

Mnemonic	Function	Operation
Stack Pointer Instructions		
CPS	Compare SP to memory	(SP) – (M : M + 1)
DES	Decrement SP	(SP) – 1 ⇒ SP
INS	Increment SP	(SP) + 1 ⇒ SP
LDS	Load SP	(M : M + 1) ⇒ SP
LEAS	Load effective address into SP	Effective address ⇒ SP
STS	Store SP	(SP) ⇒ M : M + 1
TSX	Transfer SP to X	(SP) ⇒ X
TSY	Transfer SP to Y	(SP) ⇒ Y
TXS	Transfer X to SP	(X) ⇒ SP
TYS	Transfer Y to SP	(Y) ⇒ SP
Stack Operation Instructions		
PSHA	Push A	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$
PSHB	Push B	(SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$
PSHC	Push CCR	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$
PSHD	Push D	(SP) – 2 ⇒ SP; (A : B) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PSHX	Push X	(SP) – 2 ⇒ SP; (X) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PSHY	Push Y	(SP) – 2 ⇒ SP; (Y) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PULA	Pull A	($M_{(SP)}$) ⇒ A; (SP) + 1 ⇒ SP
PULB	Pull B	($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP
PULC	Pull CCR	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP
PULD	Pull D	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ A : B; (SP) + 2 ⇒ SP
PULX	Pull X	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ X; (SP) + 2 ⇒ SP
PULY	Pull Y	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ Y; (SP) + 2 ⇒ SP

5.25 Pointer and Index Calculation Instructions

The load effective address instructions allow 5-, 8-, or 16-bit constants or the contents of 8-bit accumulators A and B or 16-bit accumulator D to be added to the contents of the X and Y index registers, or to the SP.

Table 5-25 is a summary of pointer and index instructions.

Table 5-25. Pointer and Index Calculation Instructions

Mnemonic	Function	Operation
LEAS	Load result of indexed addressing mode effective address calculation into stack pointer	r ± constant ⇒ SP or (r) + (accumulator) ⇒ SP r = X, Y, SP, or PC
LEAX	Load result of indexed addressing mode effective address calculation into x index register	r ± constant ⇒X or (r) + (accumulator) ⇒X r = X, Y, SP, or PC
LEAY	Load result of indexed addressing mode effective address calculation into y index register	r ± constant ⇒Y or (r) + (accumulator) ⇒ Y r = X, Y, SP, or PC

5.26 Condition Code Instructions

Condition code instructions are special forms of mathematical and data transfer instructions that can be used to change the condition code register. **Table 5-26** shows instructions that can be used to manipulate the CCR.

Table 5-26. Condition Code Instructions

Mnemonic	Function	Operation
ANDCC	Logical AND CCR with memory	(CCR) • (M) ⇒ CCR
CLC	Clear C bit	0 ⇒ C
CLI	Clear I bit	0 ⇒ I
CLV	Clear V bit	0 ⇒ V
ORCC	Logical OR CCR with memory	(CCR) + (M) ⇒ CCR
PSHC	Push CCR onto stack	(SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$
PULC	Pull CCR from stack	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP
SEC	Set C bit	1 ⇒ C
SEI	Set I bit	1 ⇒ I
SEV	Set V bit	1 ⇒ V
TAP	Transfer A to CCR	(A) ⇒ CCR
TPA	Transfer CCR to A	(CCR) ⇒ A

5.27 Stop and Wait Instructions

As shown in **Table 5-27**, two instructions put the CPU12 in an inactive state that reduces power consumption.

The stop instruction (STOP) stacks a return address and the contents of CPU registers and accumulators, then halts all system clocks.

The wait instruction (WAI) stacks a return address and the contents of CPU registers and accumulators, then waits for an interrupt service request; however, system clock signals continue to run.

Both STOP and WAI require that either an interrupt or a reset exception occur before normal execution of instructions resumes. Although both instructions require the same number of clock cycles to resume normal program execution after an interrupt service request is made, restarting after a STOP requires extra time for the oscillator to reach operating speed.

Table 5-27. Stop and Wait Instructions

Mnemonic	Function	Operation
STOP	Stop	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$ Stop CPU clocks
WAI	Wait for interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$

5.28 Background Mode and Null Operations

Background debug mode (BDM) is a special CPU12 operating mode that is used for system development and debugging. Executing enter background debug mode (BGND) when BDM is enabled puts the CPU12 in this mode. For complete information, refer to **Section 8. Instruction Queue**.

Null operations are often used to replace other instructions during software debugging. Replacing conditional branch instructions with branch never (BRN), for instance, permits testing a decision-making routine by disabling the conditional branch without disturbing the offset value.

Null operations can also be used in software delay programs to consume execution time without disturbing the contents of other CPU registers or memory.

Table 5-28 shows the BGND and null operation (NOP) instructions.

Table 5-28. Background Mode and Null Operation Instructions

Mnemonic	Function	Operation
BGND	Enter background debug mode	If BDM enabled, enter BDM; else resume normal processing
BRN	Branch never	Does not branch
LBRN	Long branch never	Does not branch
NOP	Null operation	—

Section 6. Instruction Glossary

6.1 Introduction

This section is a comprehensive reference to the CPU12 instruction set.

6.2 Glossary Information

The glossary contains an entry for each assembler mnemonic, in alphabetic order. **Figure 6-1** is a representation of a glossary page.

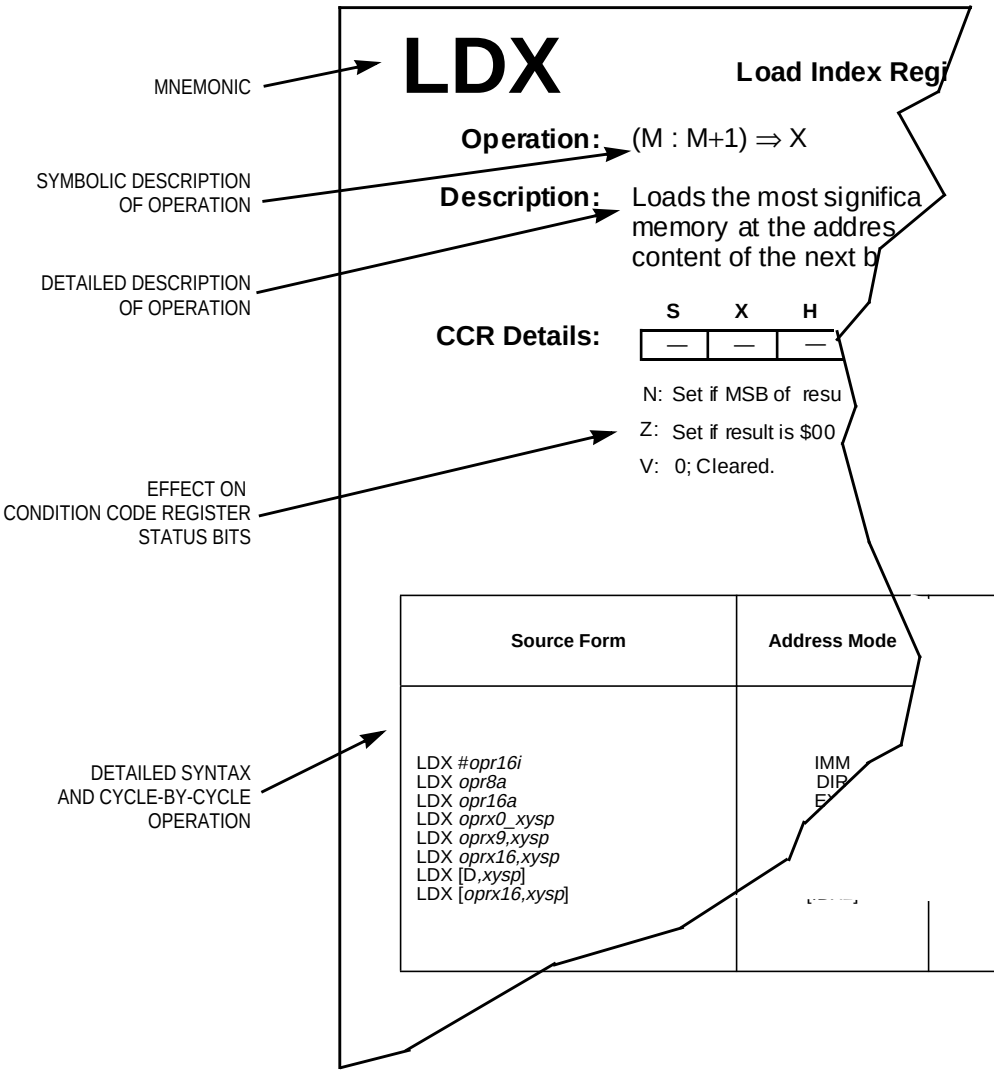


Figure 6-1. Example Glossary Page

Each entry contains symbolic and textual descriptions of operation, information concerning the effect of operation on status bits in the condition code register, and a table that describes assembler syntax, address mode variations, and cycle-by-cycle execution of the instruction.

6.3 Condition Code Changes

The following special characters are used to describe the effects of instruction execution on the status bits in the condition code register.

– — Status bit not affected by operation

0 — Status bit cleared by operation

1 — Status bit set by operation

Δ — Status bit affected by operation

⇓ — Status bit may be cleared or remain set, but is not set by operation.

⇑ — Status bit may be set or remain cleared, but is not cleared by operation.

? — Status bit may be changed by operation, but the final state is not defined.

! — Status bit used for a special purpose

6.4 Object Code Notation

The digits 0 to 9 and the uppercase letters A to F are used to express hexadecimal values. Pairs of lowercase letters represent the 8-bit values as described here.

- `dd` — 8-bit direct address $0000 to $00FF; high byte assumed to be $00
- `ee` — High-order byte of a 16-bit constant offset for indexed addressing
- `eb` — Exchange/transfer post-byte
- `ff` — Low-order eight bits of a 9-bit signed constant offset for indexed addressing, or low-order byte of a 16-bit constant offset for indexed addressing
- `hh` — High-order byte of a 16-bit extended address
- `ii` — 8-bit immediate data value
- `jj` — High-order byte of a 16-bit immediate data value
- `kk` — Low-order byte of a 16-bit immediate data value
- `lb` — Loop primitive (DBNE) post-byte
- `ll` — Low-order byte of a 16-bit extended address
- `mm` — 8-bit immediate mask value for bit manipulation instructions; set bits indicate bits to be affected
- `pg` — Program overlay page (bank) number used in CALL instruction
- `qq` — High-order byte of a 16-bit relative offset for long branches
- `tn` — Trap number $30–$39 or $40–$FF
- `rr` — Signed relative offset $80 (–128) to $7F (+127) offset relative to the byte following the relative offset byte, or low-order byte of a 16-bit relative offset for long branches
- `xb` — Indexed addressing post-byte

6.5 Source Forms

The glossary pages provide only essential information about assembler source forms. Assemblers generally support a number of assembler directives, allow definition of program labels, and have special conventions for comments. For complete information about writing source files for a particular assembler, refer to the documentation provided by the assembler vendor.

Assemblers are typically flexible about the use of spaces and tabs. Often, any number of spaces or tabs can be used where a single space is shown on the glossary pages. Spaces and tabs are also normally allowed before and after commas. When program labels are used, there must also be at least one tab or space before all instruction mnemonics. This required space is not apparent in the source forms.

Everything in the source forms columns, *except expressions in italic characters*, is literal information which must appear in the assembly source file exactly as shown. The initial 3- to 5-letter mnemonic is always a literal expression. All commas, pound signs (#), parentheses, square brackets ([or]), plus signs (+), minus signs (–), and the register designation D (as in [D,...), are literal characters.

Groups of italic characters in the columns represent variable information to be supplied by the programmer. These groups can include any alphanumeric character or the underscore character, but cannot include a space or comma. For example, the groups *xysp* and *oprx0_xysp* are both valid, but the two groups *oprx0 xysp* are not valid because there is a space between them. Permitted syntax is described here.

The definition of a legal label or expression varies from assembler to assembler. Assemblers also vary in the way CPU registers are specified. Refer to assembler documentation for detailed information. Recommended register designators are a, A, b, B, ccr, CCR, d, D, x, X, y, Y, sp, SP, pc, and PC.

abc — Any one legal register designator for accumulators A or B or the CCR

abcdxys — Any one legal register designator for accumulators A or B, the CCR, the double accumulator D, index registers X or Y, or the SP. Some assemblers may accept t2, T2, t3, or T3 codes in certain cases of transfer and exchange

instructions, but these forms are intended for Motorola use only.

abd — Any one legal register designator for accumulators A or B or the double accumulator D

abdxys — Any one legal register designator for accumulators A or B, the double accumulator D, index register X or Y, or the SP

dxys — Any one legal register designation for the double accumulator D, index registers X or Y, or the SP

msk8 — Any label or expression that evaluates to an 8-bit value. Some assemblers require a # symbol before this value.

opr8i — Any label or expression that evaluates to an 8-bit immediate value

opr16i — Any label or expression that evaluates to a 16-bit immediate value

opr8a — Any label or expression that evaluates to an 8-bit value. The instruction treats this 8-bit value as the low-order 8 bits of an address in the direct page of the 64-Kbyte address space ($00xx).

opr16a — Any label or expression that evaluates to a 16-bit value. The instruction treats this value as an address in the 64-Kbyte address space.

oprx0_xysp — This word breaks down into one of the following alternative forms that assemble to an 8-bit indexed addressing postbyte code. These forms generate the same object code except for the value of the postbyte code, which is designated as xb in the object code columns of the glossary pages. As with the source forms, treat all commas, plus signs, and minus signs as literal syntax elements. The italicized words used in these forms are included in this key.

oprx5,*xysp*
oprx3,–*xys*
oprx3,+*xys*
oprx3,*xys*–
oprx3,*xys*+
abd,*xysp*

oprx3 — Any label or expression that evaluates to a value in the range +1 to +8

oprx5 — Any label or expression that evaluates to a 5-bit value in the range –16 to +15

oprx9 — Any label or expression that evaluates to a 9-bit value in the range –256 to +255

oprx16 — Any label or expression that evaluates to a 16-bit value. Since the CPU12 has a 16-bit address bus, this can be either a signed or an unsigned value.

page — Any label or expression that evaluates to an 8-bit value. The CPU12 recognizes up to an 8-bit page value for memory expansion but not all MCUs that include the CPU12 implement all of these bits. It is the programmer's responsibility to limit the page value to legal values for the intended MCU system. Some assemblers require a # symbol before this value.

rel8 — Any label or expression that refers to an address that is within –128 to +127 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 8-bit signed offset and include it in the object code for this instruction.

rel9 — Any label or expression that refers to an address that is within –256 to +255 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 9-bit signed offset and include it in the object code for this instruction. The sign bit for this 9-bit value is encoded by the assembler as a bit in the looping postbyte (lb) of one of the loop control instructions DBEQ, DBNE, IBEQ, IBNE, TBEQ, or TBNE. The remaining eight bits of the offset are included as an extra byte of object code.

rel16 — Any label or expression that refers to an address anywhere in the 64-Kbyte address space. The assembler will calculate the 16-bit signed offset between this address and the next address after the last byte of object code for this instruction and include it in the object code for this instruction.

trapnum — Any label or expression that evaluates to an 8-bit number in the range $30–$39 or $40–$FF. Used for TRAP instruction.

xys — Any one legal register designation for index registers X or Y or the SP

xysp — Any one legal register designation for index registers X or Y, the SP, or the PC. The reference point for PC-relative instructions is the next address after the last byte of object code for the current instruction.

6.6 Cycle-by-Cycle Execution

This information is found in the tables at the bottom of each instruction glossary page. Entries show how many bytes of information are accessed from different areas of memory during the course of instruction execution. With this information and knowledge of the type and speed of memory in the system, a user can determine the execution time for any instruction in any system.

A single letter code in the column represents a single CPU cycle. Uppercase letters indicate 16-bit access cycles. There are cycle codes for each addressing mode variation of each instruction. Simply count code letters to determine the execution time of an instruction in a best-case system. An example of a best-case system is a single-chip 16-bit system with no 16-bit off-boundary data accesses to any locations other than on-chip RAM.

Many conditions can cause one or more instruction cycles to be stretched, but the CPU is not aware of the stretch delays because the clock to the CPU is temporarily stopped during these delays.

The following paragraphs explain the cycle code letters used and note conditions that can cause each type of cycle to be stretched.

f — Free cycle. This indicates a cycle where the CPU does not require use of the system buses. An f cycle is always one cycle of the system bus clock. These cycles can be used by a queue controller or the background debug system to perform single cycle accesses without disturbing the CPU.

g — Read 8-bit PPAGE register. These cycles are used only with the CALL instruction to read the current value of the PPAGE register and are not visible on the external bus. Since the PPAGE register is an internal 8-bit register, these cycles are never stretched.

l — Read indirect pointer. Indexed indirect instructions use this 16-bit pointer from memory to address the operand for the instruction. These are always 16-bit reads but they can be either aligned or misaligned. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned access to a memory that is not designed for single-cycle misaligned access.

i — Read indirect PPAGE value. These cycles are only used with indexed indirect versions of the CALL instruction, where the 8-bit value for the memory expansion page register of the CALL destination is fetched from an indirect memory location. These cycles are stretched only when controlled by a chip-select circuit that is programmed for slow memory.

n — Write 8-bit PPAGE register. These cycles are used only with the CALL and RTC instructions to write the destination value of the PPAGE register and are not visible on the external bus. Since the PPAGE register is an internal 8-bit register, these cycles are never stretched.

O — Optional cycle. Program information is always fetched as aligned 16-bit words. When an instruction consists of an odd number of bytes, and the first byte is misaligned, an O cycle is used to make an additional program word access (P) cycle that maintains queue order. In all other cases, the O cycle appears as a free (f) cycle. The $18 prebyte for page two opcodes is treated as a special 1-byte instruction. If the prebyte is misaligned, the O cycle is used as a program word access for the prebyte; if the prebyte is aligned, the O cycle appears as a free cycle. If the remainder of the instruction consists of an odd number of bytes, another O cycle is required some time before the instruction is completed. If the O cycle for the prebyte is treated as a P cycle, any subsequent O cycle in the same instruction is treated as an f cycle; if the O cycle for the prebyte is treated as an f cycle, any subsequent O cycle in the same instruction is treated as a P cycle. Optional cycles used for program word accesses can be extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. Optional cycles used as free cycles are never stretched.

P — Program word access. Program information is fetched as aligned 16-bit words. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored externally. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.

r — 8-bit data read. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

R — 16-bit data read. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to memory that is not designed for single-cycle misaligned access.

s — Stack 8-bit data. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

S — Stack 16-bit data. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching if the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single cycle misaligned access. The internal RAM is designed to allow single cycle misaligned word access.

w — 8-bit data write. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

W — 16-bit data write. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned access to a memory that is not designed for single-cycle misaligned access.

u — Unstack 8-bit data. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

U — Unstack 16-bit data. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single-cycle misaligned access. The internal RAM is designed to allow single-cycle misaligned word access.

V — Vector fetch. Vectors are always aligned 16-bit words. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.

t — 8-bit conditional read. These cycles are either data read cycles or unused cycles, depending on the data and flow of the REVW instruction. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

T — 16-bit conditional read. These cycles are either data read cycles or free cycles, depending on the data and flow of the REV or REVW instruction. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single-cycle misaligned access.

x — 8-bit conditional write. These cycles are either data write cycles or free cycles, depending on the data and flow of the REV or REVW instruction. These cycles are only stretched when controlled by a chip-select circuit programmed for slow memory.

Special Notation for Branch Taken/Not Taken Cases

PPP/P — Short branches require three cycles if taken, one cycle if not taken. Since the instruction consists of a single word containing both an opcode and an 8-bit offset, the not-taken case is simple — the queue advances, another program word fetch is made, and execution continues with the next instruction. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.

OPPP/OPO — Long branches require four cycles if taken, three cycles if not taken. Optional cycles are required because all long branches are page two opcodes, and thus include the $18 prebyte. The CPU12 treats the prebyte as a special 1-byte instruction. If the prebyte is misaligned, the optional cycle is used to perform a program word access; if the prebyte is aligned, the optional cycle is used to perform a free cycle. As a result, both the taken and not-taken cases use one optional cycle for the prebyte. In the not-taken case, the queue must advance so that execution can continue with the next instruction, and another optional cycle is required to maintain the queue. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.

6.7 Glossary

This subsection contains an entry for each assembler mnemonic, in alphabetic order.

ABA — Add Accumulator B to Accumulator A — ABA

Operation: (A) + (B) ⇒ A

Description: Adds the content of accumulator B to the content of accumulator A and places the result in A. The content of B is not changed. This instruction affects the H status bit so it is suitable for use in BCD arithmetic operations. See **DAA** instruction for additional information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	Δ	–	Δ	Δ	Δ	Δ

H: $A3 \bullet B3 + B3 \bullet \overline{R3} + \overline{R3} \bullet A3$
Set if there was a carry from bit 3; cleared otherwise

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet B7 \bullet \overline{R7} + \overline{A7} \bullet \overline{B7} \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $A7 \bullet B7 + B7 \bullet \overline{R7} + \overline{R7} \bullet A7$
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ABA	INH	18 06	00	00

ABX — Add Accumulator B to Index Register X — ABX

Operation: (B) + (X) ⇒ X

Description: Adds the 8-bit unsigned content of accumulator B to the content of index register X considering the possible carry out of the low-order byte of X; places the result in X. The content of B is not changed.

This mnemonic is implemented by the LEAX B,X instruction. The LEAX instruction allows A, B, D, or a constant to be added to X. For compatibility with the M68HC11, the mnemonic ABX is translated into the LEAX B,X instruction by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ABX *translates to...* LEAX B,X	IDX	1A E5	Pf	PP(1)

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

ABY — Add Accumulator B to Index Register Y — ABY

Operation: (B) + (Y) ⇒ Y

Description: Adds the 8-bit unsigned content of accumulator B to the content of index register Y considering the possible carry out of the low-order byte of Y; places the result in Y. The content of B is not changed.

This mnemonic is implemented by the LEAY B,Y instruction. The LEAY instruction allows A, B, D, or a constant to be added to Y. For compatibility with the M68HC11, the mnemonic ABY is translated into the LEAY B,Y instruction by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ABY *translates to...* LEAY B,Y	IDX	19 ED	Pf	PP(1)

1. Due to internal M68HC12CPU requirements, the program word fetch is performed twice to the same address during this instruction.

ADCA Add with Carry to A ADCA

Operation: (A) + (M) + C ⇒ A

Description: Adds the content of accumulator A to the content of memory location M, then adds the value of the C bit and places the result in A. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations. See **DAA** instruction for additional information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	Δ	–	Δ	Δ	Δ	Δ

H: A3 • M3 + M3 • $\overline{R3}$ + $\overline{R3}$ • A3
Set if there was a carry from bit 3; cleared otherwise

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: A7 • M7 • $\overline{R7}$ + $\overline{A7}$ • $\overline{M7}$ • R7
Set if two's complement overflow resulted from the operation; cleared otherwise

C: A7 • M7 + M7 • $\overline{R7}$ + $\overline{R7}$ • A7
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ADCA #*opr8i*	IMM	89 ii	P	P
ADCA *opr8a*	DIR	99 dd	rPf	rfP
ADCA *opr16a*	EXT	B9 hh ll	rPO	rOP
ADCA *oprx0_xysp*	IDX	A9 xb	rPf	rfP
ADCA *oprx9,xysp*	IDX1	A9 xb ff	rPO	rPO
ADCA *oprx16,xysp*	IDX2	A9 xb ee ff	frPP	frPP
ADCA [D,*xysp*]	[D,IDX]	A9 xb	fIfrPf	fIfrfP
ADCA [*oprx16,xysp*]	[IDX2]	A9 xb ee ff	fIPrPf	fIPrfP

ADCB Add with Carry to B ADCB

Operation: (B) + (M) + C ⇒ B

Description: Adds the content of accumulator B to the content of memory location M, then adds the value of the C bit and places the result in B. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations. See **DAA** instruction for additional information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	Δ	–	Δ	Δ	Δ	Δ

H: $X3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet X3$
Set if there was a carry from bit 3; cleared otherwise

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $X7 \bullet M7 \bullet \overline{R7} + \overline{X7} \bullet \overline{M7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $X7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet X7$
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
ADCB #*opr8i*	IMM	C9 ii	P	P
ADCB *opr8a*	DIR	D9 dd	rPf	rfP
ADCB *opr16a*	EXT	F9 hh ll	rPO	rOP
ADCB *oprx0_xysp*	IDX	E9 xb	rPf	rfP
ADCB *oprx9,xysp*	IDX1	E9 xb ff	rPO	rPO
ADCB *oprx16,xysp*	IDX2	E9 xb ee ff	frPP	frPP
ADCB [D,*xysp*]	[D,IDX]	E9 xb	fIfrPf	fIfrfP
ADCB [*oprx16,xysp*]	[IDX2]	E9 xb ee ff	fIPrPf	fIPrfP

ADDA — Add without Carry to A — ADDA

Operation: (A) + (M) ⇒ A

Description: Adds the content of memory location M to accumulator A and places the result in A. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations. See **DAA** instruction for additional information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	Δ	–	Δ	Δ	Δ	Δ

H: $A3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet A3$
Set if there was a carry from bit 3; cleared otherwise

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet M7 \bullet \overline{R7} + \overline{A7} \bullet \overline{M7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $A7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet A7$
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ADDA #*opr8i*	IMM	8B ii	P	P
ADDA *opr8a*	DIR	9B dd	rPf	rfP
ADDA *opr16a*	EXT	BB hh ll	rPO	rOP
ADDA *oprx0_xysp*	IDX	AB xb	rPf	rfP
ADDA *oprx9,xysp*	IDX1	AB xb ff	rPO	rPO
ADDA *oprx16,xysp*	IDX2	AB xb ee ff	frPP	frPP
ADDA [D,*xysp*]	[D,IDX]	AB xb	fIfrPf	fIfrfP
ADDA [*oprx16,xysp*]	[IDX2]	AB xb ee ff	fIPrPf	fIPrfP

ADDB — Add without Carry to B — ADDB

Operation: (B) + (M) ⇒ B

Description: Adds the content of memory location M to accumulator B and places the result in B. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations. See **DAA** instruction for additional information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	Δ	–	Δ	Δ	Δ	Δ

H: $B3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet B3$
Set if there was a carry from bit 3; cleared otherwise

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $B7 \bullet M7 \bullet \overline{R7} + \overline{B7} \bullet \overline{M7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $B7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet B7$
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
ADDB #*opr8i*	IMM	CB ii	P	P
ADDB *opr8a*	DIR	DB dd	rPf	rfP
ADDB *opr16a*	EXT	FB hh ll	rPO	rOP
ADDB *oprx0_xysp*	IDX	EB xb	rPf	rfP
ADDB *oprx9,xysp*	IDX1	EB xb ff	rPO	rPO
ADDB *oprx16,xysp*	IDX2	EB xb ee ff	frPP	frPP
ADDB [D,*xysp*]	[D,IDX]	EB xb	fIfrPf	fIfrfP
ADDB [*oprx16,xysp*]	[IDX2]	EB xb ee ff	fIPrPf	fIPrfP

ADDD — Add Double Accumulator — ADDD

Operation: (A : B) + (M : M+1) ⇒ A : B

Description: Adds the content of memory location M concatenated with the content of memory location M +1 to the content of double accumulator D and places the result in D. Accumulator A forms the high-order half of 16-bit double accumulator D; accumulator B forms the low-order half.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet M15 \bullet \overline{R15} + \overline{D15} \bullet \overline{M15} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $D15 \bullet M15 + M15 \bullet \overline{R15} + \overline{R15} \bullet D15$
Set if there was a carry from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ADDD #*opr16i*	IMM	C3 jj kk	PO	OP
ADDD *opr8a*	DIR	D3 dd	RPf	RfP
ADDD *opr16a*	EXT	F3 hh ll	RPO	ROP
ADDD *oprx0_xysp*	IDX	E3 xb	RPf	RfP
ADDD *oprx9,xysp*	IDX1	E3 xb ff	RPO	RPO
ADDD *oprx16,xysp*	IDX2	E3 xb ee ff	fRPP	fRPP
ADDD [D,*xysp*]	[D,IDX]	E3 xb	fIfRPF	fIfRfP
ADDD [*oprx16,xysp*]	[IDX2]	E3 xb ee ff	fIPRPf	fIPRfP

ANDA — Logical AND A — ANDA

Operation: (A) • (M) ⇒ A

Description: Performs logical AND between the content of memory location M and the content of accumulator A. The result is placed in A. After the operation is performed, each bit of A is the logical AND of the corresponding bits of M and of A before the operation began.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared.

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
ANDA #*opr8i*	IMM	84 ii	P	P
ANDA *opr8a*	DIR	94 dd	rPf	rfP
ANDA *opr16a*	EXT	B4 hh ll	rPO	rOP
ANDA *oprx0_xysp*	IDX	A4 xb	rPf	rfP
ANDA *oprx9,xysp*	IDX1	A4 xb ff	rPO	rPO
ANDA *oprx16,xysp*	IDX2	A4 xb ee ff	frPP	frPP
ANDA [D,*xysp*]	[D,IDX]	A4 xb	fIfrPf	fIfrfP
ANDA [*oprx16,xysp*]	[IDX2]	A4 xb ee ff	fIPrPf	fIPrfP

ANDB Logical AND B ANDB

Operation: (B) • (M) ⇒ B

Description: Performs logical AND between the content of memory location M and the content of accumulator B. The result is placed in B. After the operation is performed, each bit of B is the logical AND of the corresponding bits of M and of B before the operation began.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ANDB #*opr8i*	IMM	C4 ii	P	P
ANDB *opr8a*	DIR	D4 dd	rPf	rfP
ANDB *opr16a*	EXT	F4 hh ll	rPO	rOP
ANDB *oprx0_xysp*	IDX	E4 xb	rPf	rfP
ANDB *oprx9,xysp*	IDX1	E4 xb ff	rPO	rPO
ANDB *oprx16,xysp*	IDX2	E4 xb ee ff	frPP	frPP
ANDB [D,*xysp*]	[D,IDX]	E4 xb	fIfrPf	fIfrfP
ANDB [*oprx16,xysp*]	[IDX2]	E4 xb ee ff	fIPrPf	fIPrfP

ANDCC Logical AND CCR with Mask ANDCC

Operation: (CCR) • (Mask) ⇒ CCR

Description: Performs bitwise logical AND between the content of a mask operand and the content of the CCR. The result is placed in the CCR. After the operation is performed, each bit of the CCR is the result of a logical AND with the corresponding bits of the mask. To clear CCR bits, clear the corresponding mask bits. CCR bits that correspond to ones in the mask are not changed by the ANDCC operation.

If the I mask bit is cleared, there is a 1-cycle delay before the system allows interrupt requests. This prevents interrupts from occurring between instructions in the sequences CLI, WAI and CLI, SEI (CLI is equivalent to ANDCC #$EF).

CCR Details:

S	X	H	I	N	Z	V	C
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓

Condition code bits are cleared if the corresponding bit was 0 before the operation or if the corresponding bit in the mask is 0.

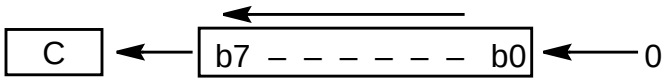
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ANDCC #*opr8i*	IMM	`10 ii`	P	P

ASL

Arithmetic Shift Left Memory (same as LSL)

ASL

Operation:



Description: Shifts all bits of memory location M one bit position to the left. Bit 0 is loaded with a 0. The C status bit is loaded from the most significant bit of M.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M7
Set if the MSB of M was set before the shift; cleared otherwise

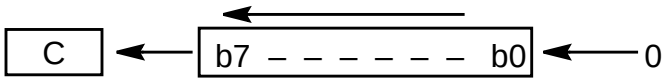
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASL *opr16a*	EXT	78 hh ll	rPw0	r0Pw
ASL *oprx0_xysp*	IDX	68 xb	rPw	rPw
ASL *oprx9,xysp*	IDX1	68 xb ff	rPw0	rP0w
ASL *oprx16,xysp*	IDX2	68 xb ee ff	frPwP	frPPw
ASL [D,*xysp*]	[D,IDX]	68 xb	fIfrPw	fIfrPw
ASL [*oprx16,xysp*]	[IDX2]	68 xb ee ff	fIPrPw	fIPrPw

ASLA

Arithmetic Shift Left A
(same as LSLA)

ASLA

Operation:



Description: Shifts all bits of accumulator A one bit position to the left. Bit 0 is loaded with a 0. TheC status bit is loaded from the most significant bit of A.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A7
Set if the MSB of A was set before the shift; cleared otherwise

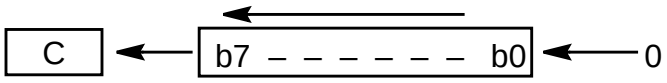
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASLA	INH	48	0	0

ASLB

Arithmetic Shift Left B
(same as LSLB)

ASLB

Operation:



Description: Shifts all bits of accumulator B one bit position to the left. Bit 0 is loaded with a 0. The C status bit is loaded from the most significant bit of B.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B7
Set if the MSB of B was set before the shift; cleared otherwise

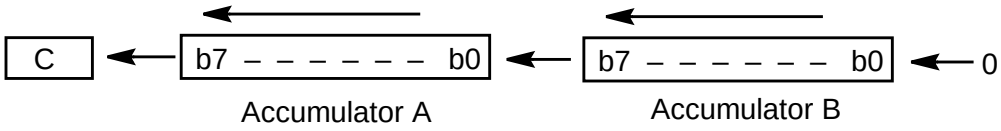
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASLB	INH	58	1	0

ASLD

Arithmetic Shift Left Double Accumulator (same as LSLD)

ASLD

Operation:



Description: Shifts all bits of double accumulator D one bit position to the left. Bit 0 is loaded with a 0. The C status bit is loaded from the most significant bit of D.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

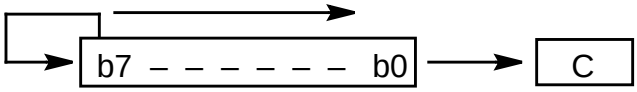
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: D15
Set if the MSB of D was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASLD	INH	59	0	0

ASR — Arithmetic Shift Right Memory — ASR

Operation:



Description: Shifts all bits of memory location M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

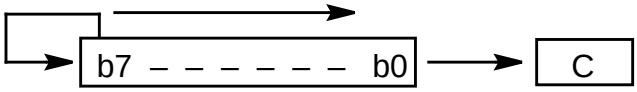
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M0
Set if the LSB of M was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASR *opr16a*	EXT	`77 hh ll`	`rPw0`	`r0Pw`
ASR *oprx0_xysp*	IDX	`67 xb`	`rPw`	`rPw`
ASR *oprx9,xysp*	IDX1	`67 xb ff`	`rPw0`	`rP0w`
ASR *oprx16,xysp*	IDX2	`67 xb ee ff`	`frPwP`	`frPPw`
ASR [D,*xysp*]	[D,IDX]	`67 xb`	`fIfrPw`	`fIfrPw`
ASR [*oprx16,xysp*]	[IDX2]	`67 xb ee ff`	`fIPrPw`	`fIPrPw`

ASRA — Arithmetic Shift Right A — ASRA

Operation:



Description: Shifts all bits of accumulator A one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

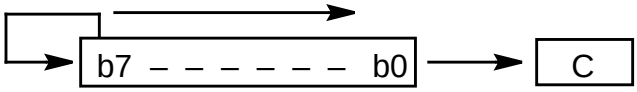
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A0
Set if the LSB of A was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
ASRA	INH	47	0	0

ASRB Arithmetic Shift Right B ASRB

Operation:



Description: Shifts all bits of accumulator B one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B0
Set if the LSB of B was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ASRB	INH	57	0	0

BCC

Branch if Carry Cleared
(Same as BHS)

BCC

Operation: If C = 0, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the C status bit and branches if C = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BCC *rel8*	REL	24 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BCLR Clear Bits in Memory BCLR

Operation: (M) • ($\overline{\text{Mask}}$) ⇒ M

Description: Clears bits in location M. To clear a bit, set the corresponding bit in the mask byte. Bits in M that correspond to 0s in the mask byte are not changed. Mask bytes can be located at PC + 2, PC + 3, or PC + 4, depending on addressing mode used.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode(1)	Object Code	Access Detail	
			HCS12	M68HC12
BCLR *opr8a, msk8*	DIR	4D dd mm	rPwO	rPOw
BCLR *opr16a, msk8*	EXT	1D hh ll mm	rPwP	rPPw
BCLR *oprx0_xysp, msk8*	IDX	0D xb mm	rPwO	rPOw
BCLR *oprx9,xysp, msk8*	IDX1	0D xb ff mm	rPwP	rPwP
BCLR *oprx16,xysp, msk8*	IDX2	0D xb ee ff mm	frPwPO	frPwOP

1. Indirect forms of indexed addressing cannot be used with this instruction.

BCS

Branch if Carry Set
(Same as BLO)

BCS

Operation: If C = 1, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the C status bit and branches if C = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BCS *rel8*	REL	25 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BEQ — Branch if Equal — BEQ

Operation: If Z = 1, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the Z status bit and branches if Z = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BEQ *rel8*	REL	27 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BGE Branch if Greater than or Equal to Zero BGE

Operation: If N ⊕ V = 0, then (PC) + $0002 + Rel ⇒ PC

For signed two's complement values
if (Accumulator) ≥ (Memory), then branch

Description: BGE can be used to branch after comparing or subtracting signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is greater than or equal to the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BGE *rel8*	REL	2C rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BGND Enter Background Debug Mode BGND

Description: BGND operates like a software interrupt, except that no registers are stacked. First, the current PC value is stored in internal CPU register TMP2. Next, the BDM ROM and background register block become active. The BDM ROM contains a substitute vector, mapped to the address of the software interrupt vector, which points to routines in the BDM ROM that control background operation. The substitute vector is fetched, and execution continues from the address that it points to. Finally, the CPU checks the location that TMP2 points to. If the value stored in that location is $00 (the BGND opcode), TMP2 is incremented, so that the instruction that follows the BGND instruction is the first instruction executed when normal program execution resumes.

For all other types of BDM entry, the CPU performs the same sequence of operations as for a BGND instruction, but the value stored in TMP2 already points to the instruction that would have executed next had BDM not become active. If active BDM is triggered just as a BGND instruction is about to execute, the BDM firmware does increment TMP2, but the change does not affect resumption of normal execution.

While BDM is active, the CPU executes debugging commands received via a special single-wire serial interface. BDM is terminated by the execution of specific debugging commands. Upon exit from BDM, the background/boot ROM and registers are disabled, the instruction queue is refilled starting with the return address pointed to by TMP2, and normal processing resumes.

BDM is normally disabled to avoid accidental entry. While BDM is disabled, BGND executes as described, but the firmware causes execution to return to the user program. Refer to **Section 8. Instruction Queue** for more information concerning BDM.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BGND	INH	00	VfPPP	VfPPP

BGT — Branch if Greater than Zero — BGT

Operation: If Z + (N ⊕ V) = 0, then (PC) + $0002 + Rel ⇒ PC

For signed two's complement values
if (Accumulator) > (Memory), then branch

Description: BGT can be used to branch after comparing or subtracting signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than the value in M. After CBA or SBA, the branch occurs if the value in B is greater than the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BGT *rel8*	REL	2E rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BHI — Branch if Higher — BHI

Operation: If C + Z = 0, then (PC) + $0002 + Rel ⇒ PC

For unsigned values, if (Accumulator) > (Memory), then branch

Description: BHI can be used to branch after comparing or subtracting unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than the value in M. After CBA or SBA, the branch occurs if the value in B is greater than the value in A. BHI should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BHI *rel8*	REL	22 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BHS

Branch if Higher or Same
(Same as BCC)

BHS

Operation: If C = 0, then (PC) + $0002 + Rel ⇒ PC

For unsigned values, if (Accumulator) ≥ (Memory), then branch

Description: BHS can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is greater than or equal to the value in A. BHS should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BHS *rel8*	REL	24 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BITA — Bit Test A — BITA

Operation: (A) • (M)

Description: Performs bitwise logical AND on the content of accumulator A and the content of memory location M and modifies the condition codes accordingly. Each bit of the result is the logical AND of the corresponding bits of the accumulator and the memory location. Neither the content of the accumulator nor the content of the memory location is affected.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BITA #*opr8i*	IMM	85 ii	P	P
BITA *opr8a*	DIR	95 dd	rPf	rfP
BITA *opr16a*	EXT	B5 hh ll	rPO	rOP
BITA *oprx0_xysp*	IDX	A5 xb	rPf	rfP
BITA *oprx9,xysp*	IDX1	A5 xb ff	rPO	rPO
BITA *oprx16,xysp*	IDX2	A5 xb ee ff	frPP	frPP
BITA [D,*xysp*]	[D,IDX]	A5 xb	fIfrPf	fIfrfP
BITA [*oprx16,xysp*]	[IDX2]	A5 xb ee ff	fIPrPf	fIPrfP

BITB — Bit Test B — BITB

Operation: (B) • (M)

Description: Performs bitwise logical AND on the content of accumulator B and the content of memory location M and modifies the condition codes accordingly. Each bit of the result is the logical AND of the corresponding bits of the accumulator and the memory location. Neither the content of the accumulator nor the content of the memory location is affected.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
BITB #*opr8i*	IMM	C5 ii	P	P
BITB *opr8a*	DIR	D5 dd	rPf	rfP
BITB *opr16a*	EXT	F5 hh ll	rPO	rOP
BITB *oprx0_xysp*	IDX	E5 xb	rPf	rfP
BITB *oprx9,xysp*	IDX1	E5 xb ff	rPO	rPO
BITB *oprx16,xysp*	IDX2	E5 xb ee ff	frPP	frPP
BITB [D,*xysp*]	[D,IDX]	E5 xb	fIfrPf	fIfrfP
BITB [*oprx16,xysp*]	[IDX2]	E5 xb ee ff	fIPrPf	fIPrfP

BLE Branch if Less Than or Equal to Zero BLE

Operation: If Z + (N ⊕ V) = 1, then (PC) + $0002 + Rel ⇒ PC

For signed two's complement numbers
if (Accumulator) ≤ (Memory), then branch

Description: BLE can be used to branch after subtracting or comparing signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is less than or equal to the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BLE *rel8*	REL	2F rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BLO

Branch if Lower
(Same as BCS)

BLO

Operation: If C = 1, then (PC) + $0002 + Rel ⇒ PC

For unsigned values, if (Accumulator) < (Memory), then branch

Description: BLO can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than the value in M. After CBA or SBA, the branch occurs if the value in B is less than the value in A. BLO should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BLO *rel8*	REL	25 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BLS Branch if Lower or Same BLS

Operation: If C + Z = 1, then (PC) + $0002 + Rel ⇒ PC

For unsigned values, if (Accumulator) ≤ (Memory), then branch

Description: If BLS is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator is less than or equal to the unsigned binary number in memory. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM because these instructions do not affect the C status bit.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BLS *rel8*	REL	23 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BLT — Branch if Less than Zero — BLT

Operation: If N ⊕ V = 1, then (PC) + $0002 + Rel ⇒ PC

For signed two's complement numbers
if (Accumulator) < (Memory), then branch

Description: BLT can be used to branch after subtracting or comparing signed two's complement values. After CMPA, CMPB, CMPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than the value in M. After CBA or SBA, the branch occurs if the value in B is less than the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BLT *rel8*	REL	2D rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BMI — Branch if Minus — BMI

Operation: If N = 1, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the N status bit and branches if N = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
BMI *rel8*	REL	2B rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BNE — Branch if Not Equal to Zero — BNE

Operation: If Z = 0, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the Z status bit and branches if Z = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BNE *rel8*	REL	26 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BPL — Branch if Plus — BPL

Operation: If N = 0, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the N status bit and branches if N = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
BPL *rel8*	REL	2A rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BRA Branch Always BRA

Operation: (PC) + $0002 + Rel ⇒ PC

Description: Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two's complement number in the second byte of the branch instruction.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the BRA branch condition is always satisfied, the branch is always taken, and the instruction queue must always be refilled.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BRA *rel8*	REL	`20 rr`	PPP	PPP

BRCLR Branch if Bits Cleared BRCLR

Operation: If (M) • (Mask) = 0, then branch

Description: Performs a bitwise logical AND of memory location M and the mask supplied with the instruction, then branches if and only if all bits with a value of 1 in the mask byte correspond to bits with a value of 0 in the tested byte. Mask operands can be located at PC + 1, PC + 2, or PC + 4, depending on addressing mode. The branch offset is referenced to the next address after the relative offset (rr) which is the last byte of the instruction object code.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode(1)	Object Code	Access Detail	
			HCS12	M68HC12
BRCLR *opr8a, msk8, rel8*	DIR	`4F dd mm rr`	`rPPP`	`rPPP`
BRCLR *opr16a, msk8, rel8*	EXT	`1F hh ll mm rr`	`rfPPP`	`rfPPP`
BRCLR *oprx0_xysp, msk8, rel8*	IDX	`0F xb mm rr`	`rPPP`	`rPPP`
BRCLR *oprx9,xysp, msk8, rel8*	IDX1	`0F xb ff mm rr`	`rfPPP`	`rffPPP`
BRCLR *oprx16,xysp, msk8, rel8*	IDX2	`0F xb ee ff mm rr`	`PrfPPP`	`frPffPPP`

1. Indirect forms of indexed addressing cannot be used with this instruction.

BRN Branch Never BRN

Operation: (PC) + $0002 ⇒ PC

Description: Never branches. BRN is effectively a 2-byte NOP that requires one cycle to execute. BRN is included in the instruction set to provide a complement to the BRA instruction. The instruction is useful during program debug, to negate the effect of another branch instruction without disturbing the offset byte. A complement for BRA is also useful in compiler implementations.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the BRN branch condition is never satisfied, the branch is never taken, and only a single program fetch is needed to update the instruction queue.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BRN *rel8*	REL	21 rr	P	P

BRSET Branch if Bits Set BRSET

Operation: If ($\overline{M}$) • (Mask) = 0, then branch

Description: Performs a bitwise logical AND of the inverse of memory location M and the mask supplied with the instruction, then branches if and only if all bits with a value of 1 in the mask byte correspond to bits with a value of one in the tested byte. Mask operands can be located at PC + 1, PC + 2, or PC + 4, depending on addressing mode. The branch offset is referenced to the next address after the relative offset (rr) which is the last byte of the instruction object code.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode(1)	Object Code	Access Detail	
			HCS12	M68HC12
BRSET *opr8a, msk8, rel8*	DIR	`4E dd mm rr`	`rPPP`	`rPPP`
BRSET *opr16a, msk8, rel8*	EXT	`1E hh ll mm rr`	`rfPPP`	`rfPPP`
BRSET *oprx0_xysp, msk8, rel8*	IDX	`0E xb mm rr`	`rPPP`	`rPPP`
BRSET *oprx9,xysp, msk8, rel8*	IDX1	`0E xb ff mm rr`	`rfPPP`	`rffPPP`
BRSET *oprx16,xysp, msk8, rel8*	IDX2	`0E xb ee ff mm rr`	`PrfPPP`	`frPffPPP`

1. Indirect forms of indexed addressing cannot be used with this instruction.

BSET — Set Bit(s) in Memory — BSET

Operation: (M) + (Mask) ⇒ M

Description: Sets bits in memory location M. To set a bit, set the corresponding bit in the mask byte. All other bits in M are unchanged. The mask byte can be located at PC + 2, PC + 3, or PC + 4, depending upon addressing mode.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode(1)	Object Code	Access Detail HCS12	Access Detail M68HC12
BSET *opr8a, msk8*	DIR	4C dd mm	rPw0	rP0w
BSET *opr16a, msk8*	EXT	1C hh ll mm	rPwP	rPPw
BSET *oprx0_xysp, msk8*	IDX	0C xb mm	rPw0	rP0w
BSET *oprx9,xysp, msk8*	IDX1	0C xb ff mm	rPwP	rPwP
BSET *oprx16,xysp, msk8*	IDX2	0C xb ee ff mm	frPwP0	frPw0P

1. Indirect forms of indexed addressing cannot be used with this instruction.

BSR — Branch to Subroutine — BSR

Operation:

(SP) – $0002 ⇒ SP
$RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$
(PC) + Rel ⇒ PC

Description:

Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address.

Decrements the SP by two, to allow the two bytes of the return address to be stacked.

Stacks the return address (the SP points to the high-order byte of the return address).

Branches to a location determined by the branch offset.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BSR *rel8*	REL	07 rr	SPPP	PPPS

BVC Branch if Overflow Cleared BVC

Operation: If V = 0, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the V status bit and branches if V = 0.

BVC causes a branch when a previous operation on two's complement binary values does not cause an overflow. That is, when BVC follows a two's complement operation, a branch occurs when the result of the operation is valid.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BVC *rel8*	REL	28 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

BVS — Branch if Overflow Set — BVS

Operation: If V = 1, then (PC) + $0002 + Rel ⇒ PC

Simple branch

Description: Tests the V status bit and branches if V = 1.

BVS causes a branch when a previous operation on two's complement binary values causes an overflow. That is, when BVS follows a two's complement operation, a branch occurs when the result of the operation is invalid.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
BVS *rel8*	REL	29 rr	PPP/P(1)	PPP/P(1)

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

CALL — Call Subroutine in Expanded Memory — CALL

Operation:

(SP) – $0002 ⇒ SP; RTN_H : RTN_L ⇒ $M_{(SP)}$: $M_{(SP+1)}$
(SP) – $0001 ⇒ SP; (PPAGE) ⇒ $M_{(SP)}$
page ⇒ PPAGE; Subroutine Address ⇒ PC

Description:

Sets up conditions to return to normal program flow, then transfers control to a subroutine in expanded memory. Uses the address of the instruction following the CALL as a return address. For code compatibility, CALL also executes correctly in devices that do not have expanded memory capability.

Decrements the SP by two, then stores the return address on the stack. The SP points to the high-order byte of the return address.

Decrements the SP by one, then stacks the current memory page value from the PPAGE register on the stack.

Writes a new page value supplied by the instruction to PPAGE and transfers control to the subroutine.

In indexed-indirect modes, the subroutine address and the PPAGE value are fetched from memory in the order M high byte, M low byte, and new PPAGE value.

Expanded-memory subroutines must be terminated by an RTC instruction, which restores the return address and PPAGE value from the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CALL *opr16a, page*	EXT	`4A hh ll pg`	`gnSsPPP`	`gnfSsPPP`
CALL *oprx0_xysp, page*	IDX	`4B xb pg`	`gnSsPPP`	`gnfSsPPP`
CALL *oprx9,xysp, page*	IDX1	`4B xb ff pg`	`gnSsPPP`	`gnfSsPPP`
CALL *oprx16,xysp, page*	IDX2	`4B xb ee ff pg`	`fgnSsPPP`	`fgnfSsPPP`
CALL [D,*xysp*]	[D,IDX]	`4B xb`	`fIignSsPPP`	`fIignSsPPP`
CALL [*oprx16,xysp*]	[IDX2]	`4B xb ee ff`	`fIignSsPPP`	`fIignSsPPP`

CBA — Compare Accumulators — CBA

Operation: (A) – (B)

Description: Compares the content of accumulator A to the content of accumulator B and sets the condition codes, which may then be used for arithmetic and logical conditional branches. The contents of the accumulators are not changed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{B7} \bullet \overline{R7} + \overline{A7} \bullet B7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet B7 + B7 \bullet R7 + R7 \bullet \overline{A7}$
Set if there was a borrow from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CBA	INH	18 17	00	00

CLC — Clear Carry — CLC

Operation: $0 \Rightarrow$ C bit

Description: Clears the C status bit. This instruction is assembled as ANDCC #$FE. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

CLC can be used to set up the C bit prior to a shift or rotate instruction involving the C bit.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	0

C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CLC *translates to...* ANDCC #$FE	IMM	10 FE	P	P

CLI — Clear Interrupt Mask — CLI

Operation: $0 \Rightarrow$ I bit

Description: Clears the I mask bit. This instruction is assembled as ANDCC #$EF. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

When the I bit is cleared, interrupts are enabled. There is a 1-cycle (bus clock) delay in the clearing mechanism for the I bit so that, if interrupts were previously disabled, the next instruction after a CLI will always be executed, even if there was an interrupt pending prior to execution of the CLI instruction.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	0	–	–	–	–

I: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CLI *translates to...* ANDCC #$EF	IMM	10 EF	P	P

CLR — Clear Memory — CLR

Operation: 0 ⇒ M

Description: All bits in memory location M are cleared to 0.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	1	0	0

N: 0; cleared
Z: 1; set
V: 0; cleared
C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
CLR *opr16a*	EXT	79 hh ll	PwO	wOP
CLR *oprx0_xysp*	IDX	69 xb	Pw	Pw
CLR *oprx9,xysp*	IDX1	69 xb ff	PwO	PwO
CLR *oprx16,xysp*	IDX2	69 xb ee ff	PwP	PwP
CLR [D,*xysp*]	[D,IDX]	69 xb	PIfw	PIfPw
CLR [*oprx16,xysp*]	[IDX2]	69 xb ee ff	PIPw	PIPPw

CLRA Clear A CLRA

Operation: $0 \Rightarrow A$

Description: All bits in accumulator A are cleared to 0.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	1	0	0

N: 0; cleared
Z: 1; set
V: 0; cleared
C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CLRA	INH	87	0	0

CLRB Clear B CLRB

Operation: $0 \Rightarrow B$

Description: All bits in accumulator B are cleared to 0.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	1	0	0

N: 0; cleared
Z: 1; set
V: 0; cleared
C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CLRB	INH	C7	0	0

CLV — Clear Two's Complement Overflow Bit — CLV

Operation: $0 \Rightarrow$ V bit

Description: Clears the V status bit. This instruction is assembled as ANDCC #$FD. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	0	–

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CLV *translates to...* ANDCC #$FD	IMM	10 FD	P	P

CMPA — Compare A — CMPA

Operation: (A) – (M)

Description: Compares the content of accumulator A to the content of memory location M and sets the condition codes, which may then be used for arithmetic and logical conditional branching. The contents of A and location M are not changed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if there was a borrow from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
CMPA #*opr8i*	IMM	81 ii	P	P
CMPA *opr8a*	DIR	91 dd	rPf	rfP
CMPA *opr16a*	EXT	B1 hh ll	rPO	rOP
CMPA *oprx0_xysp*	IDX	A1 xb	rPf	rfP
CMPA *oprx9,xysp*	IDX1	A1 xb ff	rPO	rPO
CMPA *oprx16,xysp*	IDX2	A1 xb ee ff	frPP	frPP
CMPA [D,*xysp*]	[D,IDX]	A1 xb	fIfrPf	fIfrfP
CMPA [*oprx16,xysp*]	[IDX2]	A1 xb ee ff	fIPrPf	fIPrfP

CMPB Compare B CMPB

Operation: (B) – (M)

Description: Compares the content of accumulator B to the content of memory location M and sets the condition codes, which may then be used for arithmetic and logical conditional branching. The contents of B and location M are not changed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $B7 \bullet \overline{M7} \bullet \overline{R7} + \overline{B7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{B7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{B7}$
Set if there was a borrow from the MSB of the result; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CMPB #*opr8i*	IMM	C1 ii	P	P
CMPB *opr8a*	DIR	D1 dd	rPf	rfP
CMPB *opr16a*	EXT	F1 hh ll	rPO	rOP
CMPB *oprx0_xysp*	IDX	E1 xb	rPf	rfP
CMPB *oprx9*,*xysp*	IDX1	E1 xb ff	rPO	rPO
CMPB *oprx16*,*xysp*	IDX2	E1 xb ee ff	frPP	frPP
CMPB [D,*xysp*]	[D,IDX]	E1 xb	fIfrPf	fIfrfP
CMPB [*oprx16*,*xysp*]	[IDX2]	E1 xb ee ff	fIPrPf	fIPrfP

COM — Complement Memory — COM

Operation: $(\overline{M}) = \$FF - (M) \Rightarrow M$

Description: Replaces the content of memory location M with its one's complement. Each bit of M is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	1

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 1; set (for M6800 compatibility)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
COM *opr16a*	EXT	71 hh ll	rPw0	r0Pw
COM *oprx0_xysp*	IDX	61 xb	rPw	rPw
COM *oprx9,xysp*	IDX1	61 xb ff	rPw0	rP0w
COM *oprx16,xysp*	IDX2	61 xb ee ff	frPwP	frPPw
COM [D,*xysp*]	[D,IDX]	61 xb	fIfrPw	fIfrPw
COM [*oprx16,xysp*]	[IDX2]	61 xb ee ff	fIPrPw	fIPrPw

COMA — Complement A — COMA

Operation: $(\overline{A}) = \$FF - (A) \Rightarrow A$

Description: Replaces the content of accumulator A with its one's complement. Each bit of A is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	1

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 1; set (for M6800 compatibility)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
COMA	INH	41	0	0

COMB Complement B COMB

Operation: $(\overline{B}) = \$FF - (B) \Rightarrow B$

Description: Replaces the content of accumulator B with its one's complement. Each bit of B is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	1

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 1; set (for M6800 compatibility)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
COMB	INH	51	0	0

CPD — Compare Double Accumulator — CPD

Operation: (A : B) – (M : M + 1)

Description: Compares the content of double accumulator D with a 16-bit value at the address specified and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of (M : M + 1) from D without modifying either D or (M : M + 1).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the absolute value of the content of memory is larger than the absolute value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CPD #*opr16i*	IMM	8C jj kk	PO	OP
CPD *opr8a*	DIR	9C dd	RPf	RfP
CPD *opr16a*	EXT	BC hh ll	RPO	ROP
CPD *oprx0_xysp*	IDX	AC xb	RPf	RfP
CPD *oprx9,xysp*	IDX1	AC xb ff	RPO	RPO
CPD *oprx16,xysp*	IDX2	AC xb ee ff	fRPP	fRPP
CPD [D,*xysp*]	[D,IDX]	AC xb	fIfRPf	fIfRfP
CPD [*oprx16,xysp*]	[IDX2]	AC xb ee ff	fIPRPf	fIPRfP

CPS — Compare Stack Pointer — CPS

Operation: (SP) – (M : M + 1)

Description: Compares the content of the SP with a 16-bit value at the address specified, and sets the condition codes accordingly. The compare is accomplished internally by doing a 16-bit subtract of (M : M + 1) from the SP without modifying either the SP or (M : M + 1).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $S15 \bullet \overline{M15} \bullet \overline{R15} + \overline{S15} \bullet M15 \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{S15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{S15}$
Set if the absolute value of the content of memory is larger than the absolute value of the SP; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
CPS #*opr16i*	IMM	8F jj kk	PO	OP
CPS *opr8a*	DIR	9F dd	RPf	RfP
CPS *opr16a*	EXT	BF hh ll	RPO	ROP
CPS *oprx0_xysp*	IDX	AF xb	RPf	RfP
CPS *oprx9,xysp*	IDX1	AF xb ff	RPO	RPO
CPS *oprx16,xysp*	IDX2	AF xb ee ff	fRPP	fRPP
CPS [D,*xysp*]	[D,IDX]	AF xb	fIfRPf	fIfRfP
CPS [*oprx16,xysp*]	[IDX2]	AF xb ee ff	fIPRPf	fIPRfP

CPX — Compare Index Register X — CPX

Operation: (X) – (M : M + 1)

Description: Compares the content of index register X with a 16-bit value at the address specified and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of (M : M + 1) from index register X without modifying either index register X or (M : M + 1).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $X15 \bullet \overline{M15} \bullet \overline{R15} + \overline{X15} \bullet M15 \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{X15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{X15}$
Set if the absolute value of the content of memory is larger than the absolute value of the index register; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CPX #*opr16i*	IMM	8E jj kk	PO	OP
CPX *opr8a*	DIR	9E dd	RPf	RfP
CPX *opr16a*	EXT	BE hh ll	RPO	ROP
CPX *oprx0_xysp*	IDX	AE xb	RPf	RfP
CPX *oprx9,xysp*	IDX1	AE xb ff	RPO	RPO
CPX *oprx16,xysp*	IDX2	AE xb ee ff	fRPP	fRPP
CPX [D,*xysp*]	[D,IDX]	AE xb	fIfRPf	fIfRfP
CPX [*oprx16,xysp*]	[IDX2]	AE xb ee ff	fIPRPf	fIPRfP

CPY — Compare Index Register Y — CPY

Operation: (Y) – (M : M + 1)

Description: Compares the content of index register Y to a 16-bit value at the address specified and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of (M : M + 1) from Y without modifying either Y or (M : M + 1).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $Y15 \bullet \overline{M15} \bullet \overline{R15} + \overline{Y15} \bullet M15 \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{Y15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{Y15}$
Set if the absolute value of the content of memory is larger than the absolute value of the index register; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
CPY #*opr16i*	IMM	8D jj kk	PO	OP
CPY *opr8a*	DIR	9D dd	RPf	RfP
CPY *opr16a*	EXT	BD hh ll	RPO	ROP
CPY *oprx0_xysp*	IDX	AD xb	RPf	RfP
CPY *oprx9,xysp*	IDX1	AD xb ff	RPO	RPO
CPY *oprx16,xysp*	IDX2	AD xb ee ff	fRPP	fRPP
CPY [D,*xysp*]	[D,IDX]	AD xb	fIfRPf	fIfRfP
CPY [*oprx16,xysp*]	[IDX2]	AD xb ee ff	fIPRPf	fIPRfP

DAA Decimal Adjust A DAA

Description: DAA adjusts the content of accumulator A and the state of the C status bit to represent the correct binary-coded-decimal sum and the associated carry when a BCD calculation has been performed. To execute DAA, the content of accumulator A, the state of the C status bit, and the state of the H status bit must all be the result of performing an ABA, ADD, or ADC on BCD operands, with or without an initial carry.

The table shows DAA operation for all legal combinations of input operands. Columns 1 through 4 represent the results of ABA, ADC, or ADD operations on BCD operands. The correction factor in column 5 is added to the accumulator to restore the result of an operation on two BCD operands to a valid BCD value and to set or clear the C bit. All values are in hexadecimal.

1	2	3	4	5	6
Initial C Bit Value	**Value of A[7:4]**	**Initial H Bit Value**	**Value of A[3:0]**	**Correction Factor**	**Corrected C Bit Value**
0	0–9	0	0–9	00	0
0	0–8	0	A–F	06	0
0	0–9	1	0–3	06	0
0	A–F	0	0–9	60	1
0	9–F	0	A–F	66	1
0	A–F	1	0–3	66	1
1	0–2	0	0–9	60	1
1	0–2	0	A–F	66	1
1	0–3	1	0–3	66	1

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	?	Δ

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is $00; cleared otherwise
V: Undefined
C: Represents BCD carry. See bit table

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
DAA	INH	18 07	0f0	0f0

DBEQ Decrement and Branch if Equal to Zero DBEQ

Operation: (Counter) – 1 ⇒ Counter
If (Counter) = 0, then (PC) + $0003 + Rel ⇒ PC

Description: Subtract one from the specified counter register A, B, D, X, Y, or SP. If the counter register has reached zero, execute a branch to the specified relative destination. The DBEQ instruction is encoded into three bytes of machine code including the 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

IBEQ and TBEQ instructions are similar to DBEQ except that the counter is incremented or tested rather than being decremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code(1)	Access Detail	
			HCS12	M68HC12
DBEQ *abdxys, rel9*	REL	`04 lb rr`	`PPP/PPO`	`PPP`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (DBEQ – 0) or not zero (DBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 would be 0:0 for DBEQ.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	DBEQ A, *rel9*	`04 00 rr`	`04 10 rr`
B	001	DBEQ B, *rel9*	`04 01 rr`	`04 11 rr`
D	100	DBEQ D, *rel9*	`04 04 rr`	`04 14 rr`
X	101	DBEQ X, *rel9*	`04 05 rr`	`04 15 rr`
Y	110	DBEQ Y, *rel9*	`04 06 rr`	`04 16 rr`
SP	111	DBEQ SP, *rel9*	`04 07 rr`	`04 17 rr`

DBNE Decrement and Branch if Not Equal to Zero DBNE

Operation: (Counter) – 1 ⇒ Counter
If (Counter) not = 0, then (PC) + $0003 + Rel ⇒ PC

Description: Subtract one from the specified counter register A, B, D, X, Y, or SP. If the counter register has not been decremented to zero, execute a branch to the specified relative destination. The DBNE instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

IBNE and TBNE instructions are similar to DBNE except that the counter is incremented or tested rather than being decremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code(1)	Access Detail	
			HCS12	M68HC12
DBNE *abdxys, rel9*	REL	`04 lb rr`	`PPP/PPO`	`PPP`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (DBEQ – 0) or not zero (DBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 would be 0:0 for DBNE.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	DBNE A, *rel9*	`04 20 rr`	`04 30 rr`
B	001	DBNE B, *rel9*	`04 21 rr`	`04 31 rr`
D	100	DBNE D, *rel9*	`04 24 rr`	`04 34 rr`
X	101	DBNE X, *rel9*	`04 25 rr`	`04 35 rr`
Y	110	DBNE Y, *rel9*	`04 26 rr`	`04 36 rr`
SP	111	DBNE SP, *rel9*	`04 27 rr`	`04 37 rr`

DEC — Decrement Memory — DEC

Operation: (M) – $01 ⇒ M

Description: Subtract one from the content of memory location M.

The N, Z, and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there was a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (M) was $80 before the operation.

Source Form	Address Mode	Object Code[1]	Access Detail HCS12	Access Detail M68HC12
DEC *opr16a*	EXT	`73 hh ll`	`rPw0`	`r0Pw`
DEC *oprx0_xysp*	IDX	`63 xb`	`rPw`	`rPw`
DEC *oprx9,xysp*	IDX1	`63 xb ff`	`rPw0`	`rP0w`
DEC *oprx16,xysp*	IDX2	`63 xb ee ff`	`frPwP`	`frPPw`
DEC [D,*xysp*]	[D,IDX]	`63 xb`	`fIfrPw`	`fIfrPw`
DEC [*oprx16,xysp*]	[IDX2]	`63 xb ee ff`	`fIPrPw`	`fIPrPw`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (DBEQ – 0) or not zero (DBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 would be 0:0 for DBNE.

DECA Decrement A DECA

Operation: (A) – $01 ⇒ A

Description: Subtract one from the content of accumulator A.

The N, Z, and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there was a two’s complement overflow as a result of the operation; cleared otherwise. Two’s complement overflow occurs if and only if (A) was $80 before the operation.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
DECA	INH	43	0	0

DECB — Decrement B — DECB

Operation: (B) – $01 ⇒ B

Description: Subtract one from the content of accumulator B.

The N, Z, and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there was a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (B) was $80 before the operation.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
DECB	INH	53	0	0

DES Decrement Stack Pointer DES

Operation: (SP) – $0001 ⇒ SP

Description: Subtract one from the SP. This instruction assembles to LEAS –1,SP. The LEAS instruction does not affect condition codes as DEX or DEY instructions do.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
DES *translates to...* LEAS –1,SP	IDX	1B 9F	Pf	PP(1)

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

DEX — Decrement Index Register X — DEX

Operation: (X) – $0001 ⇒ X

Description: Subtract one from index register X. LEAX –1,X can produce the same result, but LEAX does not affect the Z bit. Although the LEAX instruction is more flexible, DEX requires only one byte of object code.

Only the Z bit is set or cleared according to the result of this operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	–	–

Z: Set if result is $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
DEX	INH	09	0	0

DEY — Decrement Index Register Y — DEY

Operation: (Y) – $0001 ⇒ Y

Description: Subtract one from index register Y. LEAY –1,Y can produce the same result, but LEAY does not affect the Z bit. Although the LEAY instruction is more flexible, DEY requires only one byte of object code.

Only the Z bit is set or cleared according to the result of this operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	–	–

Z: Set if result is $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
DEY	INH	03	0	0

EDIV

Extended Divide 32-Bit by 16-Bit (Unsigned)

EDIV

Operation: (Y : D) ÷ (X) ⇒ Y; Remainder ⇒ D

Description: Divides a 32-bit unsigned dividend by a 16-bit divisor, producing a 16-bit unsigned quotient and an unsigned 16-bit remainder. All operands and results are located in CPU registers. If an attempt to divide by zero is made, C is set and the states of the N, Z, and V bits in the CCR are undefined. In case of an overflow or a divide by zero, the contents of the registers D and Y do not change.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise
Undefined after overflow or division by zero

Z: Set if result is $0000; cleared otherwise
Undefined after overflow or division by zero

V: Set if the result was > $FFFF; cleared otherwise Undefined after division by zero

C: Set if divisor was $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EDIV	INH	11	fffffffff0	fffffffff0

EDIVS — Extended Divide 32-Bit by 16-Bit (Signed) — EDIVS

Operation: (Y : D) ÷ (X) ⇒ Y; Remainder ⇒ D

Description: Divides a signed 32-bit dividend by a 16-bit signed divisor, producing a signed 16-bit quotient and a signed 16-bit remainder. All operands and results are located in CPU registers. If an attempt to divide by zero is made, C is set and the states of the N, Z, and V bits in the CCR are undefined. In case of an overflow or a divide by zero, the contents of the registers D and Y do not change.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise
Undefined after overflow or division by zero

Z: Set if result is $0000; cleared otherwise
Undefined after overflow or division by zero

V: Set if the result was > $7FFF or < $8000; cleared otherwise
Undefined after division by zero

C: Set if divisor was $0000; cleared otherwise
Indicates division by zero

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EDIVS	INH	18 14	Offfffffff0	Offfffffff0

EMACS

Extended Multiply and Accumulate (Signed) 16-Bit by 16-Bit to 32-Bit

EMACS

Operation: $(M_{(X)} : M_{(X+1)}) \times (M_{(Y)} : M_{(Y+1)}) + (M \sim M+3) \Rightarrow M \sim M+3$

Description: A 16-bit value is multiplied by a 16-bit value to produce a 32-bit intermediate result. This 32-bit intermediate result is then added to the content of a 32-bit accumulator in memory. EMACS is a signed integer operation. All operands and results are located in memory. When the EMACS instruction is executed, the first source operand is fetched from an address pointed to by X, and the second source operand is fetched from an address pointed to by index register Y. Before the instruction is executed, the X and Y index registers must contain values that point to the most significant bytes of the source operands. The most significant byte of the 32-bit result is specified by an extended address supplied with the instruction.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00000000; cleared otherwise

V: $M31 \bullet I31 \bullet \overline{R31} + \overline{M31} \bullet \overline{I31} \bullet R31$
Set if result > $7FFFFFFF (+ overflow) or
< $80000000 (– underflow)
Indicates two's complement overflow

C: $M15 \bullet I15 + I15 \bullet \overline{R15} + \overline{R15} \bullet M15$
Set if there was a carry from bit 15 of the result; cleared otherwise
Indicates a carry from low word to high word of the result occurred

Source Form[1]	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMACS *opr16a*	Special	18 12 hh ll	ORROfffRRfWWP	ORROfffRRfWWP

1. *opr16a* is an extended address specification. Both X and Y point to source operands.

EMAXD

Place Larger of Two Unsigned 16-Bit Values in Accumulator D

EMAXD

Operation: MAX ((D), (M : M + 1)) ⇒ D

Description: Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the larger of the two values in D. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in D has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = D – M : M + 1)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMAXD *oprx0_xysp*	IDX	18 1A xb	ORPf	ORfP
EMAXD *oprx9,xysp*	IDX1	18 1A xb ff	ORPO	ORPO
EMAXD *oprx16,xysp*	IDX2	18 1A xb ee ff	OfRPP	OfRPP
EMAXD [D,*xysp*]	[D,IDX]	18 1A xb	OfIfRPf	OfIfRfP
EMAXD [*oprx16,xysp*]	[IDX2]	18 1A xb ee ff	OfIPRPf	OfIPRfP

EMAXM

Place Larger of Two Unsigned 16-Bit Values in Memory

EMAXM

Operation: MAX ((D), (M : M + 1)) ⇒ M : M + 1

Description: Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the larger of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in D has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = D – M : M + 1)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMAXM *oprx0_xysp*	IDX	18 1E xb	ORPW	ORPW
EMAXM *oprx9,xysp*	IDX1	18 1E xb ff	ORPWO	ORPWO
EMAXM *oprx16,xysp*	IDX2	18 1E xb ee ff	OfRPWP	OfRPWP
EMAXM [D,*xysp*]	[D,IDX]	18 1E xb	OfIfRPW	OfIfRPW
EMAXM [*oprx16,xysp*]	[IDX2]	18 1E xb ee ff	OfIPRPW	OfIPRPW

EMIND

Place Smaller of Two Unsigned 16-Bit Values in Accumulator D

EMIND

Operation: MIN ((D), (M : M + 1)) ⇒ D

Description: Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the smaller of the two values in D. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in D has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the smallest value in a list of values.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = D – M : M + 1)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMIND *oprx0_xysp*	IDX	18 1B xb	ORPf	ORfP
EMIND *oprx9,xysp*	IDX1	18 1B xb ff	ORPO	ORPO
EMIND *oprx16,xysp*	IDX2	18 1B xb ee ff	OfRPP	OfRPP
EMIND [D,*xysp*]	[D,IDX]	18 1B xb	OfIfRPf	OfIfRfP
EMIND [*oprx16,xysp*]	[IDX2]	18 1B xb ee ff	OfIPRPf	OfIPRfP

EMINM

Place Smaller of Two Unsigned 16-Bit Values in Memory

EMINM

Operation: MIN ((D), (M : M + 1)) ⇒ M : M + 1

Description: Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger and leaves the smaller of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in D has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = D – M : M + 1)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMINM *oprx0_xysp*	IDX	`18 1F xb`	`ORPW`	`ORPW`
EMINM *oprx9,xysp*	IDX1	`18 1F xb ff`	`ORPWO`	`ORPWO`
EMINM *oprx16,xysp*	IDX2	`18 1F xb ee ff`	`OfRPWP`	`OfRPWP`
EMINM [D,*xysp*]	[D,IDX]	`18 1F xb`	`OfIfRPW`	`OfIfRPW`
EMINM [*oprx16,xysp*]	[IDX2]	`18 1F xb ee ff`	`OfIPRPW`	`OfIPRPW`

EMUL

Extended Multiply
16-Bit by 16-Bit (Unsigned)

EMUL

Operation: (D) × (Y) ⇒ Y : D

Description: An unsigned 16-bit value is multiplied by an unsigned 16-bit value to produce an unsigned 32-bit result. The first source operand must be loaded into 16-bit double accumulator D and the second source operand must be loaded into index register Y before executing the instruction. When the instruction is executed, the value in D is multiplied by the value in Y. The upper 16-bits of the 32-bit result are stored in Y and the low-order 16-bits of the result are stored in D.

The C status bit can be used to round the high-order 16 bits of the result.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	–	Δ

N: Set if the MSB of the result is set; cleared otherwise

Z: Set if result is $00000000; cleared otherwise

C: Set if bit 15 of the result is set; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMUL	INH	13	ff0	ff0

EMULS — Extended Multiply 16-Bit by 16-Bit (Signed) — EMULS

Operation: (D) × (Y) ⇒ Y : D

Description: A signed 16-bit value is multiplied by a signed 16-bit value to produce a signed 32-bit result. The first source operand must be loaded into 16-bit double accumulator D, and the second source operand must be loaded into index register Y before executing the instruction. When the instruction is executed, D is multiplied by the value Y. The 16 high-order bits of the 32-bit result are stored in Y and the 16 low-order bits of the result are stored in D.

The C status bit can be used to round the high-order 16 bits of the result.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	–	Δ

N: Set if the MSB of the result is set; cleared otherwise

Z: Set if result is $00000000; cleared otherwise

C: Set if bit 15 of the result is set; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EMULS	INH	18 13	0f0 0ff0(1)	0f0

1. EMULS has an extra free cycle if it is followed by another PAGE TWO instruction.

EORA — Exclusive OR A — EORA

Operation: (A) ⊕ (M) ⇒ A

Description: Performs the logical exclusive OR between the content of accumulator A and the content of memory location M. The result is placed in A. Each bit of A after the operation is the logical exclusive OR of the corresponding bits of M and A before the operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
EORA #*opr8i*	IMM	88 ii	P	P
EORA *opr8a*	DIR	98 dd	rPf	rfP
EORA *opr16a*	EXT	B8 hh ll	rPO	rOP
EORA *oprx0_xysp*	IDX	A8 xb	rPf	rfP
EORA *oprx9,xysp*	IDX1	A8 xb ff	rPO	rPO
EORA *oprx16,xysp*	IDX2	A8 xb ee ff	frPP	frPP
EORA [D,*xysp*]	[D,IDX]	A8 xb	fIfrPf	fIfrfP
EORA [*oprx16,xysp*]	[IDX2]	A8 xb ee ff	fIPrPf	fIPrfP

EORB Exclusive OR B EORB

Operation: (B) ⊕ (M) ⇒ B

Description: Performs the logical exclusive OR between the content of accumulator B and the content of memory location M. The result is placed in A. Each bit of A after the operation is the logical exclusive OR of the corresponding bits of M and B before the operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
EORB #*opr8i*	IMM	C8 ii	P	P
EORB *opr8a*	DIR	D8 dd	rPf	rfP
EORB *opr16a*	EXT	F8 hh ll	rPO	rOP
EORB *oprx0_xysp*	IDX	E8 xb	rPf	rfP
EORB *oprx9*,*xysp*	IDX1	E8 xb ff	rPO	rPO
EORB *oprx16*,*xysp*	IDX2	E8 xb ee ff	frPP	frPP
EORB [D,*xysp*]	[D,IDX]	E8 xb	fIfrPf	fIfrfP
EORB [*oprx16*,*xysp*]	[IDX2]	E8 xb ee ff	fIPrPf	fIPrfP

ETBL Extended Table Lookup and Interpolate ETBL

Operation: $(M : M + 1) + [(B) \times ((M + 2 : M + 3) – (M : M + 1))] \Rightarrow D$

Description: ETBL linearly interpolates one of 256 result values that fall between each pair of data entries in a lookup table stored in memory. Data entries in the table represent the y values of endpoints of equally-spaced line segments. Table entries and the interpolated result are 16-bit values. The result is stored in the D accumulator.

Before executing ETBL, an index register points to the table entry corresponding to the x value (X1 that is closest to, but less than or equal to, the desired lookup point (XL, YL). This defines the left end of a line segment and the right end is defined by the next data entry in the table. Prior to execution, accumulator B holds a binary fraction (radix left of MSB) representing the ratio of (XL–X1) ÷ (X2–X1).

The 16-bit unrounded result is calculated using the following expression:

$$D = Y1 + [(B) \times (Y2 – Y1)]$$

Where:

(B) = (XL – X1) ÷ (X2 – X1)

Y1 = 16-bit data entry pointed to by <effective address>

Y2 = 16-bit data entry pointed to by <effective address> + 2

The intermediate value $[(B) \times (Y2 – Y1)]$ produces a 24-bit result with the radix point between bits 7 and 8. Any indexed addressing mode, except indirect modes or 9-bit and 16-bit offset modes, can be used to identify the first data point (X1,Y1). The second data point is the next table entry.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	–	Δ(1)

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

C: Set if result can be rounded up; cleared otherwise

1. C-bit was undefined in original M68HC12

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ETBL *oprx0_xysp*	IDX	18 3F xb	ORRfffffP	ORRfffffP

EXG — Exchange Register Contents — EXG

Operation: See table

Description: Exchanges the contents of registers specified in the instruction as shown below. Note that the order in which exchanges between 8-bit and 16-bit registers are specified affects the high byte of the 16-bit registers differently. Exchanges of D with A or B are ambiguous. Cases involving TMP2 and TMP3 are reserved for Motorola use, so some assemblers may not permit their use, but it is possible to generate these cases by using DC.B or DC.W assembler directives.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Or:

S	X	H	I	N	Z	V	C
Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ

None affected, unless the CCR is the destination register. Condition codes take on the value of the corresponding source bits, except that the X mask bit cannot change from 0 to 1. Software can leave the X bit set, leave it cleared, or change it from 1 to 0, but it can be set only in response to any reset or by recognition of an $\overline{\text{XIRQ}}$ interrupt.

Source Form	Address Mode	Object Code[1]	Access Detail HCS12	Access Detail M68HC12
EXG *abcdxys,abcdxys*	INH	B7 eb	P	P

1. Legal coding for eb is summarized in the following table. Columns represent the high-order source digit. Rows represent the low-order destination digit (bit 3 is a don't care). Values are in hexadecimal.

	8	9	A	B	C	D	E	F
0	A ⇔ A	B ⇔ A	CCR ⇔ A	$TMP3_L$ ⇒ A $00:A ⇒ TMP3	B ⇒ A A ⇒ B	X_L ⇒ A $00:A ⇒ X	Y_L ⇒ A $00:A ⇒ Y	SP_L ⇒ A $00:A ⇒ SP
1	A ⇔ B	B ⇔ B	CCR ⇔ B	$TMP3_L$ ⇒ B $FF:B ⇒ TMP3	B ⇒ B $FF ⇒ A	X_L ⇒ B $FF:B ⇒ X	Y_L ⇒ B $FF:B ⇒ Y	SP_L ⇒ B $FF:B ⇒ SP
2	A ⇔ CCR	B ⇔ CCR	CCR ⇔ CCR	$TMP3_L$ ⇒ CCR $FF:CCR ⇒ TMP3	B ⇒ CCR $FF:CCR ⇒ D	X_L ⇒ CCR $FF:CCR ⇒ X	Y_L ⇒ CCR $FF:CCR ⇒ Y	SP_L ⇒ CCR $FF:CCR ⇒ SP
3	$00:A ⇒ TMP2 $TMP2_L$ ⇒ A	$00:B ⇒ TMP2 $TMP2_L$ ⇒ B	$00:CCR ⇒ TMP2 $TMP2_L$ ⇒ CCR	TMP3 ⇔ TMP2	D ⇔ TMP2	X ⇔ TMP2	Y ⇔ TMP2	SP ⇔ TMP2
4	$00:A ⇒ D	$00:B ⇒ D	$00:CCR ⇒ D B ⇒ CCR	TMP3 ⇔ D	D ⇔ D	X ⇔ D	Y ⇔ D	SP ⇔ D
5	$00:A ⇒ X X_L ⇒ A	$00:B ⇒ X X_L ⇒ B	$00:CCR ⇒ X X_L ⇒ CCR	TMP3 ⇔ X	D ⇔ X	X ⇔ X	Y ⇔ X	SP ⇔ X
6	$00:A ⇒ Y Y_L ⇒ A	$00:B ⇒ Y Y_L ⇒ B	$00:CCR ⇒ Y Y_L ⇒ CCR	TMP3 ⇔ Y	D ⇔ Y	X ⇔ Y	Y ⇔ Y	SP ⇔ Y
7	$00:A ⇒ SP SP_L ⇒ A	$00:B ⇒ SP SP_L ⇒ B	$00:CCR ⇒ SP SP_L ⇒ CCR	TMP3 ⇔ SP	D ⇔ SP	X ⇔ SP	Y ⇔ SP	SP ⇔ SP

FDIV Fractional Divide FDIV

Operation: (D) ÷ (X) ⇒ X; Remainder ⇒ D

Description: Divides an unsigned 16-bit numerator in double accumulator D by an unsigned 16-bit denominator in index register X, producing an unsigned 16-bit quotient in X and an unsigned 16-bit remainder in D. If both the numerator and the denominator are assumed to have radix points in the same positions, the radix point of the quotient is to the left of bit 15. The numerator must be less than the denominator. In the case of overflow (denominator is less than or equal to the numerator) or division by zero, the quotient is set to $FFFF, and the remainder is indeterminate.

FDIV is equivalent to multiplying the numerator by 2^{16} and then performing 32 by 16-bit integer division. The result is interpreted as a binary-weighted fraction, which resulted from the division of a 16-bit integer by a larger 16-bit integer. A result of $0001 corresponds to 0.000015, and $FFFF corresponds to 0.9998. The remainder of an IDIV instruction can be resolved into a binary-weighted fraction by an FDIV instruction. The remainder of an FDIV instruction can be resolved into the next 16 bits of binary-weighted fraction by another FDIV instruction.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	Δ	Δ

Z: Set if quotient is $0000; cleared otherwise

V: 1 if X ≤ D
Set if the denominator was less than or equal to the numerator; cleared otherwise

C: $\overline{X15} \bullet \overline{X14} \bullet \overline{X13} \bullet \overline{X12} \bullet ... \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$
Set if denominator was $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
FDIV	INH	18 11	0fffffffff0	0fffffffff0

IBEQ — Increment and Branch if Equal to Zero — IBEQ

Operation: (Counter) + 1 ⇒ Counter
If (Counter) = 0, then (PC) + $0003 + Rel ⇒ PC

Description: Add one to the specified counter register A, B, D, X, Y, or SP. If the counter register has reached zero, branch to the specified relative destination. The IBEQ instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBEQ and TBEQ instructions are similar to IBEQ except that the counter is decremented or tested rather than being incremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code[1]	Access Detail HCS12	Access Detail M68HC12
IBEQ *abdxys, rel9*	REL	`04 lb rr`	`PPP/PPO`	`PPP`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (IBEQ – 0) or not zero (IBNE – 1) versions, and bit 0 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 1:0 for IBEQ.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	IBEQ A, *rel9*	`04 80 rr`	`04 90 rr`
B	001	IBEQ B, *rel9*	`04 81 rr`	`04 91 rr`
D	100	IBEQ D, *rel9*	`04 84 rr`	`04 94 rr`
X	101	IBEQ X, *rel9*	`04 85 rr`	`04 95 rr`
Y	110	IBEQ Y, *rel9*	`04 86 rr`	`04 96 rr`
SP	111	IBEQ SP, *rel9*	`04 87 rr`	`04 97 rr`

IBNE Increment and Branch if Not Equal to Zero IBNE

Operation: (Counter) + 1 ⇒ Counter
If (Counter) not = 0, then (PC) + $0003 + Rel ⇒ PC

Description: Add one to the specified counter register A, B, D, X, Y, or SP. If the counter register has not been incremented to zero, branch to the specified relative destination. The IBNE instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBNE and TBNE instructions are similar to IBNE except that the counter is decremented or tested rather than being incremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code(1)	Access Detail	
			HCS12	M68HC12
IBNE *abdxys, rel9*	REL	`04 lb rr`	PPP/PPO	PPP

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (IBEQ – 0) or not zero (IBNE – 1) versions, and bit 0 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 1:0 for IBNE.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	IBNE A, *rel9*	`04 A0 rr`	`04 B0 rr`
B	001	IBNE B, *rel9*	`04 A1 rr`	`04 B1 rr`
D	100	IBNE D, *rel9*	`04 A4 rr`	`04 B4 rr`
X	101	IBNE X, *rel9*	`04 A5 rr`	`04 B5 rr`
Y	110	IBNE Y, *rel9*	`04 A6 rr`	`04 B6 rr`
SP	111	IBNE SP, *rel9*	`04 A7 rr`	`04 B7 rr`

IDIV — Integer Divide — IDIV

Operation: (D) ÷ (X) ⇒ X; Remainder ⇒ D

Description: Divides an unsigned 16-bit dividend in double accumulator D by an unsigned 16-bit divisor in index register X, producing an unsigned 16-bit quotient in X, and an unsigned 16-bit remainder in D. If both the divisor and the dividend are assumed to have radix points in the same positions, the radix point of the quotient is to the right of bit 0. In the case of division by zero, C is set, the quotient is set to $FFFF, and the remainder is indeterminate.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	0	Δ

Z: Set if quotient is $0000; cleared otherwise

V: 0; cleared

C: $\overline{X15} \bullet \overline{X14} \bullet \overline{X13} \bullet \overline{X12} \bullet ... \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$
Set if denominator was $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
IDIV	INH	18 10	Offfffffff0	Offfffffff0

IDIVS Integer Divide (Signed) IDIVS

Operation: (D) ÷ (X) ⇒ X; Remainder ⇒ D

Description: Performs signed integer division of a signed 16-bit numerator in double accumulator D by a signed 16-bit denominator in index register X, producing a signed 16-bit quotient in X, and a signed 16-bit remainder in D. If division by zero is attempted, the values in D and X are not changed, C is set, and the values of the N, Z, and V status bits are undefined.

Other than division by zero, which is not legal and causes the C status bit to be set, the only overflow case is:

$$\frac{\$8000}{\$FFFF} = \frac{-32{,}768}{-1} = +32{,}768$$

But the highest positive value that can be represented in a 16-bit two's complement number is 32,767 ($7FFFF).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise
Undefined after overflow or division by zero

Z: Set if quotient is $0000; cleared otherwise
Undefined after overflow or division by zero

V: Set if the result was > $7FFF or < $8000; cleared otherwise
Undefined after division by zero

C: $\overline{X15} \bullet \overline{X14} \bullet \overline{X13} \bullet \overline{X12} \bullet ... \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$
Set if denominator was $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
IDIVS	INH	18 15	0fffffffff0	0fffffffff0

INC — Increment Memory — INC

Operation: (M) + $01 ⇒ M

Description: Add one to the content of memory location M.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (M) was $7F before the operation.

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
INC *opr16a*	EXT	72 hh ll	rPw0	r0Pw
INC *oprx0_xysp*	IDX	62 xb	rPw	rPw
INC *oprx9,xysp*	IDX1	62 xb ff	rPw0	rP0w
INC *oprx16,xysp*	IDX2	62 xb ee ff	frPwP	frPPw
INC [D,*xysp*]	[D,IDX]	62 xb	fIfrPw	fIfrPw
INC [*oprx16,xysp*]	[IDX2]	62 xb ee ff	fIPrPw	fIPrPw

INCA — Increment A — INCA

Operation: (A) + $01 ⇒ A

Description: Add one to the content of accumulator A.

The N, Z, and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (A) was $7F before the operation.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INCA	INH	42	0	0

INCB — Increment B — INCB

Operation: (B) + $01 ⇒ B

Description: Add one to the content of accumulator B.

The N, Z, and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (B) was $7F before the operation.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INCB	INH	52	0	0

INS — Increment Stack Pointer — INS

Operation: (SP) + $0001 ⇒ SP

Description: Add one to the SP. This instruction is assembled to LEAS 1,SP. The LEAS instruction does not affect condition codes as an INX or INY instruction would.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INS *translates to...* LEAS 1,SP	IDX	1B 81	Pf	PP(1)

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

INX — Increment Index Register X — INX

Operation: (X) + $0001 ⇒ X

Description: Add one to index register X. LEAX 1,X can produce the same result but LEAX does not affect the Z status bit. Although the LEAX instruction is more flexible, INX requires only one byte of object code.

INX operation affects only the Z status bit.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	–	–

Z: Set if result is $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INX	INH	08	0	0

INY — Increment Index Register Y — INY

Operation: (Y) + $0001 ⇒ Y

Description: Add one to index register Y. LEAY 1,Y can produce the same result but LEAY does not affect the Z status bit. Although the LEAY instruction is more flexible, INY requires only one byte of object code.

INY operation affects only the Z status bit.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	Δ	–	–

Z: Set if result is $0000; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
INY	INH	02	0	0

JMP — Jump — JMP

Operation: Effective Address ⇒ PC

Description: Jumps to the instruction stored at the effective address. The effective address is obtained according to the rules for extended or indexed addressing.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
JMP *opr16a*	EXT	06 hh ll	PPP	PPP
JMP *oprx0_xysp*	IDX	05 xb	PPP	PPP
JMP *oprx9,xysp*	IDX1	05 xb ff	PPP	PPP
JMP *oprx16,xysp*	IDX2	05 xb ee ff	fPPP	fPPP
JMP [D,*xysp*]	[D,IDX]	05 xb	fIfPPP	fIfPPP
JMP [*oprx16,xysp*]	[IDX2]	05 xb ee ff	fIfPPP	fIfPPP

JSR — Jump to Subroutine — JSR

Operation:

(SP) – $0002 ⇒ SP
$RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP + 1)}$
Subroutine Address ⇒ PC

Description:

Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction following the JSR as a return address.

Decrements the SP by two to allow the two bytes of the return address to be stacked.

Stacks the return address. The SP points to the high order byte of the return address.

Calculates an effective address according to the rules for extended, direct, or indexed addressing.

Jumps to the location determined by the effective address.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
JSR *opr8a*	DIR	17 dd	SPPP	PPPS
JSR *opr16a*	EXT	16 hh ll	SPPP	PPPS
JSR *oprx0_xysp*	IDX	15 xb	PPPS	PPPS
JSR *oprx9,xysp*	IDX1	15 xb ff	PPPS	PPPS
JSR *oprx16,xysp*	IDX2	15 xb ee ff	fPPPS	fPPPS
JSR [D,*xysp*]	[D,IDX]	15 xb	fIfPPPS	fIfPPPS
JSR [*oprx16,xysp*]	[IDX2]	15 xb ee ff	fIfPPPS	fIfPPPS

LBCC

Long Branch if Carry Cleared (Same as LBHS)

LBCC

Operation: If C = 0, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the C status bit and branches if C = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBCC *rel16*	REL	18 24 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBCS

Long Branch if Carry Set (Same as LBLO)

LBCS

Operation: If C = 1, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the C status bit and branches if C = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBCS *rel16*	REL	18 25 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBEQ Long Branch if Equal LBEQ

Operation: If Z = 1, (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the Z status bit and branches if Z = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBEQ *rel16*	REL	18 27 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBGE Long Branch if Greater Than or Equal to Zero LBGE

Operation: If N ⊕ V = 0, (PC) + $0004 + Rel ⇒ PC

For signed two's complement numbers, if (Accumulator) ≥ Memory), then branch

Description: LBGE can be used to branch after subtracting or comparing signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is greater than or equal to the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBGE *rel16*	REL	18 2C qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBGT Long Branch if Greater Than Zero LBGT

Operation: If Z + (N ⊕ V) = 0, then (PC) + $0004 + Rel ⇒ PC

For signed two's complement numbers, If (Accumulator) > (Memory), then branch

Description: LBGT can be used to branch after subtracting or comparing signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is greater than or equal to the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBGT *rel16*	REL	18 2E qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBHI — Long Branch if Higher — LBHI

Operation: If C + Z = 0, then (PC) + $0004 + Rel ⇒ PC

For unsigned binary numbers, if (Accumulator) > (Memory), then branch

Description: LBHI can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than the value in M. After CBA or SBA, the branch occurs if the value in B is greater than the value in A. LBHI should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBHI *rel16*	REL	18 22 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBHS

Long Branch if Higher or Same (Same as LBCC)

LBHS

Operation: If C = 0, then (PC) + $0004 + Rel ⇒ PC

For unsigned binary numbers, if (Accumulator) ≥ (Memory), then branch

Description: LBHS can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is greater than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is greater than or equal to the value in A. LBHS should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBHS *rel16*	REL	18 24 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBLE Long Branch if Less Than or Equal to Zero LBLE

Operation: If Z + (N ⊕ V) = 1, then (PC) + $0004 + Rel ⇒ PC

For signed two's complement numbers, if (Accumulator) ≤ (Memory), then branch.

Description: LBLE can be used to branch after subtracting or comparing signed two's complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is less than or equal to the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBLE *rel16*	REL	18 2F qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBLO

Long Branch if Lower
(Same as LBCS)

LBLO

Operation: If C = 1, then (PC) + $0004 + Rel ⇒ PC

For unsigned binary numbers, if (Accumulator) < (Memory), then branch

Description: LBLO can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than the value in M. After CBA or SBA, the branch occurs if the value in B is less than the value in A. LBLO should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBLO *rel16*	REL	18 25 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBLS Long Branch if Lower or Same LBLS

Operation: If C + Z = 1, then (PC) + $0004 + Rel ⇒ PC

For unsigned binary numbers, if (Accumulator) ≤ (Memory), then branch

Description: LBLS can be used to branch after subtracting or comparing unsigned values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than or equal to the value in M. After CBA or SBA, the branch occurs if the value in B is less than or equal to the value in A. LBLS should not be used for branching after instructions that do not affect the C bit, such as increment, decrement, load, store, test, clear, or complement.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBLS *rel16*	REL	18 23 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBLT Long Branch if Less Than Zero LBLT

Operation: If N ⊕ V = 1, (PC) + $0004 + Rel ⇒ PC

For signed two's complement numbers, if (Accumulator) < (Memory), then branch

Description: LBLT can be used to branch after subtracting or comparing signed two-s complement values. After CMPA, CMPB, CPD, CPS, CPX, CPY, SBCA, SBCB, SUBA, SUBB, or SUBD, the branch occurs if the CPU register value is less than the value in M. After CBA or SBA, the branch occurs if the value in B is less than the value in A.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBLT *rel16*	REL	18 2D qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBMI — Long Branch if Minus — LBMI

Operation: If N = 1, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the N status bit and branches if N = 1.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBMI *rel16*	REL	18 2B qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBNE Long Branch if Not Equal to Zero LBNE

Operation: If Z = 0, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the Z status bit and branches if Z = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBNE *rel16*	REL	18 26 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBPL — Long Branch if Plus — LBPL

Operation: If N = 0, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the N status bit and branches if N = 0.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBPL *rel16*	REL	18 2A qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBRA Long Branch Always LBRA

Operation: (PC) + $0004 + Rel ⇒ PC

Description: Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two's complement number in the second and third bytes of machine code corresponding to the long branch instruction.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the LBRA branch condition is always satisfied, the branch is always taken, and the instruction queue must always be refilled, so execution time is always the larger value.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
LBRA *rel16*	REL	18 20 qq rr	OPPP	OPPP

LBRN Long Branch Never LBRN

Operation: (PC) + $0004 ⇒ PC

Description: Never branches. LBRN is effectively a 4-byte NOP that requires three cycles to execute. LBRN is included in the instruction set to provide a complement to the LBRA instruction. The instruction is useful during program debug, to negate the effect of another branch instruction without disturbing the offset byte. A complement for LBRA is also useful in compiler implementations.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the LBRN branch condition is never satisfied, the branch is never taken, and the queue does not need to be refilled, so execution time is always the smaller value.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBRN *rel16*	REL	18 21 qq rr	OPO	OPO

LBVC Long Branch if Overflow Cleared LBVC

Operation: If V = 0, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the V status bit and branches if V = 0.

LBVC causes a branch when a previous operation on two's complement binary values does not cause an overflow. That is, when LBVC follows a two's complement operation, a branch occurs when the result of the operation is valid.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBVC *rel16*	REL	18 28 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBVS Long Branch if Overflow Set LBVS

Operation: If V = 1, then (PC) + $0004 + Rel ⇒ PC

Simple branch

Description: Tests the V status bit and branches if V = 1.

LBVS causes a branch when a previous operation on two's complement binary values causes an overflow. That is, when LBVS follows a two's complement operation, a branch occurs when the result of the operation is invalid.

See **3.8 Relative Addressing Mode** for details of branch execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LBVS *rel16*	REL	18 29 qq rr	OPPP/OPO(1)	OPPP/OPO(1)

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N ⊕ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N ⊕ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N ⊕ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N ⊕ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LDAA — Load Accumulator A — LDAA

Operation: (M) ⇒ A

Description: Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDAA #*opr8i*	IMM	86 ii	P	P
LDAA *opr8a*	DIR	96 dd	rPf	rfP
LDAA *opr16a*	EXT	B6 hh ll	rPO	rOP
LDAA *oprx0_xysp*	IDX	A6 xb	rPf	rfP
LDAA *oprx9,xysp*	IDX1	A6 xb ff	rPO	rPO
LDAA *oprx16,xysp*	IDX2	A6 xb ee ff	frPP	frPP
LDAA [D,*xysp*]	[D,IDX]	A6 xb	fIfrPf	fIfrfP
LDAA [*oprx16,xysp*]	[IDX2]	A6 xb ee ff	fIPrPf	fIPrfP

LDAB Load Accumulator B LDAB

Operation: (M) ⇒ B

Description: Loads the content of memory location M into accumulator B. The condition codes are set according to the data.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDAB #*opr8i*	IMM	C6 ii	P	P
LDAB *opr8a*	DIR	D6 dd	rPf	rfP
LDAB *opr16a*	EXT	F6 hh ll	rPO	rOP
LDAB *oprx0_xysp*	IDX	E6 xb	rPf	rfP
LDAB *oprx9*,*xysp*	IDX1	E6 xb ff	rPO	rPO
LDAB *oprx16*,*xysp*	IDX2	E6 xb ee ff	frPP	frPP
LDAB [D,*xysp*]	[D,IDX]	E6 xb	fIfrPf	fIfrfP
LDAB [*oprx16,xysp*]	[IDX2]	E6 xb ee ff	fIPrPf	fIPrfP

LDD — Load Double Accumulator — LDD

Operation: (M : M+1) ⇒ A : B

Description: Loads the contents of memory locations M and M+1 into double accumulator D. The condition codes are set according to the data. The information from M is loaded into accumulator A, and the information from M+1 is loaded into accumulator B.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDD #*opr16i*	IMM	CC jj kk	PO	OP
LDD *opr8a*	DIR	DC dd	RPf	RfP
LDD *opr16a*	EXT	FC hh ll	RPO	ROP
LDD *oprx0_xysp*	IDX	EC xb	RPf	RfP
LDD *oprx9,xysp*	IDX1	EC xb ff	RPO	RPO
LDD *oprx16,xysp*	IDX2	EC xb ee ff	fRPP	fRPP
LDD [D,*xysp*]	[D,IDX]	EC xb	fIfRPf	fIfRfP
LDD [*oprx16,xysp*]	[IDX2]	EC xb ee ff	fIPRPf	fIPRfP

LDS — Load Stack Pointer — LDS

Operation: (M : M+1) ⇒ SP

Description: Loads the most significant byte of the SP with the content of memory location M, and loads the least significant byte of the SP with the content of the next byte of memory at M+1.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDS #*opr16i*	IMM	CF jj kk	PO	OP
LDS *opr8a*	DIR	DF dd	RPf	RfP
LDS *opr16a*	EXT	FF hh ll	RPO	ROP
LDS *oprx0_xysp*	IDX	EF xb	RPf	RfP
LDS *oprx9,xysp*	IDX1	EF xb ff	RPO	RPO
LDS *oprx16,xysp*	IDX2	EF xb ee ff	fRPP	fRPP
LDS [D,*xysp*]	[D,IDX]	EF xb	fIfRPf	fIfRfP
LDS [*oprx16,xysp*]	[IDX2]	EF xb ee ff	fIPRPf	fIPRfP

LDX — Load Index Register X — LDX

Operation: (M : M+1) ⇒ X

Description: Loads the most significant byte of index register X with the content of memory location M, and loads the least significant byte of X with the content of the next byte of memory at M+1.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDX #*opr16i*	IMM	CE jj kk	PO	OP
LDX *opr8a*	DIR	DE dd	RPf	RfP
LDX *opr16a*	EXT	FE hh ll	RPO	ROP
LDX *oprx0_xysp*	IDX	EE xb	RPf	RfP
LDX *oprx9,xysp*	IDX1	EE xb ff	RPO	RPO
LDX *oprx16,xysp*	IDX2	EE xb ee ff	fRPP	fRPP
LDX [D,*xysp*]	[D,IDX]	EE xb	fIfRPf	fIfRfP
LDX [*oprx16,xysp*]	[IDX2]	EE xb ee ff	fIPRPf	fIPRfP

LDY — Load Index Register Y — LDY

Operation: (M : M+1) ⇒ Y

Description: Loads the most significant byte of index register Y with the content of memory location M, and loads the least significant byte of Y with the content of the next memory location at M+1.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LDY #*opr16i*	IMM	CD jj kk	PO	OP
LDY *opr8a*	DIR	DD dd	RPf	RfP
LDY *opr16a*	EXT	FD hh ll	RPO	ROP
LDY *oprx0_xysp*	IDX	ED xb	RPf	RfP
LDY *oprx9,xysp*	IDX1	ED xb ff	RPO	RPO
LDY *oprx16,xysp*	IDX2	ED xb ee ff	fRPP	fRPP
LDY [D,*xysp*]	[D,IDX]	ED xb	fIfRPf	fIfRfP
LDY [*oprx16,xysp*]	[IDX2]	ED xb ee ff	fIPRPf	fIPRfP

LEAS Load Stack Pointer with Effective Address LEAS

Operation: Effective Address ⇒ SP

Description: Loads the stack pointer with an effective address specified by the program. The effective address can be any indexed addressing mode operand address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.9 Indexed Addressing Modes** for more details.

LEAS does not alter condition code bits. This allows stack modification without disturbing CCR bits changed by recent arithmetic operations.

Operation is a bit more complex when LEAS is used with auto-increment or auto-decrement operand specifications and the SP is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load index instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

In the unusual case where LEAS involves two different index registers and post-increment or post-decrement, both index registers are modified as demonstrated by the following example. Consider the instruction LEAS 4,Y+. First S is loaded with the value of Y, then Y is incremented by 4.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LEAS *oprx0_xysp*	IDX	1B xb	Pf	PP(1)
LEAS *oprx9,xysp*	IDX1	1B xb ff	P0	P0
LEAS *oprx16,xysp*	IDX2	1B xb ee ff	PP	PP

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

LEAX — Load X with Effective Address — LEAX

Operation: Effective Address ⇒ X

Description: Loads index register X with an effective address specified by the program. The effective address can be any indexed addressing mode operand address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.9 Indexed Addressing Modes** for more details.

Operation is a bit more complex when LEAX is used with auto-increment or auto-decrement operand specifications and index register X is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load indexed instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

In the unusual case where LEAX involves two different index registers and post-increment and post-decrement, both index registers are modified as demonstrated by the following example. Consider the instruction LEAX 4,Y+. First X is loaded with the value of Y, then Y is incremented by 4.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LEAX *oprx0_xysp*	IDX	1A xb	Pf	PP(1)
LEAX *oprx9,xysp*	IDX1	1A xb ff	P0	P0
LEAX *oprx16,xysp*	IDX2	1A xb ee ff	PP	PP

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

LEAY — Load Y with Effective Address — LEAY

Operation: Effective Address ⇒ Y

Description: Loads index register Y with an effective address specified by the program. The effective address can be any indexed addressing mode operand address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.9 Indexed Addressing Modes** for more details.

Operation is a bit more complex when LEAY is used with auto-increment or auto-decrement operand specifications and index register Y is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load indexed instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

In the unusual case where LEAY involves two different index registers and post-increment and post-decrement, both index registers are modified as demonstrated by the following example. Consider the instruction LEAY 4,X+. First Y is loaded with the value of X, then X is incremented by 4.

CCR Details:

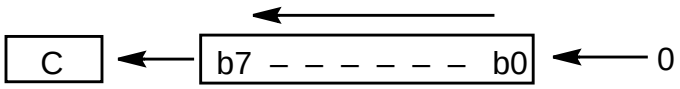
S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LEAY *oprx0_xysp*	IDX	19 xb	Pf	PP(1)
LEAY *oprx9,xysp*	IDX1	19 xb ff	P0	P0
LEAY *oprx16,xysp*	IDX2	19 xb ee ff	PP	PP

1. Due to internal M68HC12 CPU requirements, the program word fetch is performed twice to the same address during this instruction.

LSL — Logical Shift Left Memory (Same as ASL) — LSL

Operation:



Description: Shifts all bits of the memory location M one place to the left. Bit 0 is loaded with 0. The C status bit is loaded from the most significant bit of M.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M7
Set if the LSB of M was set before the shift; cleared otherwise

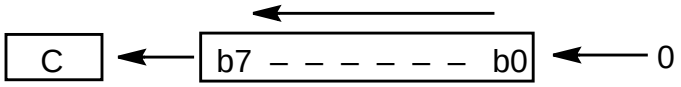
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSL *opr16a*	EXT	78 hh ll	rPw0	r0Pw
LSL *oprx0_xysp*	IDX	68 xb	rPw	rPw
LSL *oprx9,xysp*	IDX1	68 xb ff	rPw0	rP0w
LSL *oprx16,xysp*	IDX2	68 xb ee ff	frPwP	frPPw
LSL [D,*xysp*]	[D,IDX]	68 xb	fIfrPw	fIfrPw
LSL [*oprx16,xysp*]	[IDX2]	68 xb ee ff	fIPrPw	fIPrPw

LSLA

Logical Shift Left A
(Same as ASLA)

LSLA

Operation:



Description: Shifts all bits of accumulator A one place to the left. Bit 0 is loaded with 0. The C status bit is loaded from the most significant bit of A.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A7
Set if the LSB of A was set before the shift; cleared otherwise

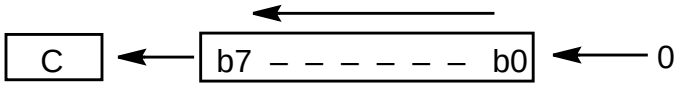
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSLA	INH	48	0	0

LSLB

Logical Shift Left B
(Same as ASLB)

LSLB

Operation:



Description: Shifts all bits of accumulator B one place to the left. Bit 0 is loaded with 0. The C status bit is loaded from the most significant bit of B.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B7
Set if the LSB of B was set before the shift; cleared otherwise

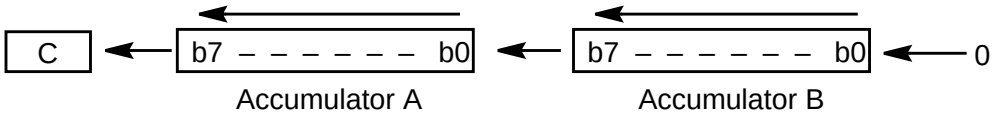
Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSLB	INH	58	0	0

LSLD

Logical Shift Left Double
(Same as ASLD)

LSLD

Operation:



Description: Shifts all bits of double accumulator D one place to the left. Bit 0 is loaded with 0. The C status bit is loaded from the most significant bit of accumulator A.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

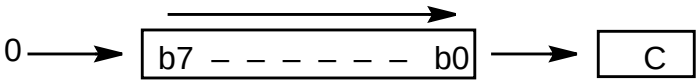
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: D15
Set if the MSB of D was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSLD	INH	59	0	0

LSR — Logical Shift Right Memory — LSR

Operation:



Description: Shifts all bits of memory location M one place to the right. Bit 7 is loaded with 0. The C status bit is loaded from the least significant bit of M.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	Δ	Δ	Δ

N: 0; cleared

Z: Set if result is $00; cleared otherwise

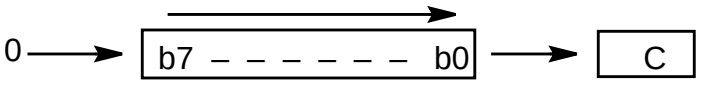
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M0
Set if the LSB of M was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSR *opr16a*	EXT	74 hh ll	rPw0	r0Pw
LSR *oprx0_xysp*	IDX	64 xb	rPw	rPw
LSR *oprx9,xysp*	IDX1	64 xb ff	rPw0	rP0w
LSR *oprx16,xysp*	IDX2	64 xb ee ff	frPwP	frPPw
LSR [D,*xysp*]	[D,IDX]	64 xb	fIfrPw	fIfrPw
LSR [*oprx16,xysp*]	[IDX2]	64 xb ee ff	fIPrPw	fIPrPw

LSRA — Logical Shift Right A — LSRA

Operation:



Description: Shifts all bits of accumulator A one place to the right. Bit 7 is loaded with 0. The C status bit is loaded from the least significant bit of A.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	Δ	Δ	Δ

N: 0; cleared

Z: Set if result is $00; cleared otherwise

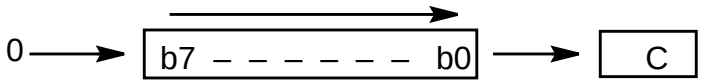
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A0
Set if the LSB of A was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSRA	INH	44	0	0

LSRB — Logical Shift Right B — LSRB

Operation:



Description: Shifts all bits of accumulator B one place to the right. Bit 7 is loaded with 0. The C status bit is loaded from the least significant bit of B.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	Δ	Δ	Δ

N: 0; cleared

Z: Set if result is $00; cleared otherwise

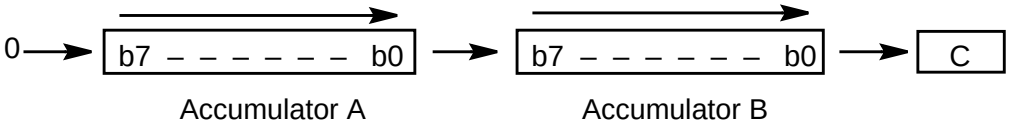
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B0
Set if the LSB of B was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSRB	INH	54	0	0

LSRD Logical Shift Right Double LSRD

Operation:



Description: Shifts all bits of double accumulator D one place to the right. D15 (MSB of A) is loaded with 0. The C status bit is loaded from D0 (LSB of B).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	0	Δ	Δ	Δ

N: 0; cleared

Z: Set if result is $0000; cleared otherwise

V: D0
Set if, after the shift operation, C is set; cleared otherwise

C: D0
Set if the LSB of D was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
LSRD	INH	49	0	0

MAXA — Place Larger of Two Unsigned 8-Bit Values in Accumulator A — MAXA

Operation: MAX ((A), (M)) ⇒ A

Description: Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger and leaves the larger of the two values in A. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in A has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = A – M)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MAXA *oprx0_xysp*	IDX	18 18 xb	OrPf	OrfP
MAXA *oprx9,xysp*	IDX1	18 18 xb ff	OrPO	OrPO
MAXA *oprx16,xysp*	IDX2	18 18 xb ee ff	OfrPP	OfrPP
MAXA [D,*xysp*]	[D,IDX]	18 18 xb	OfIfrPf	OfIfrfP
MAXA [*oprx16,xysp*]	[IDX2]	18 18 xb ee ff	OfIPrPf	OfIPrfP

MAXM — Place Larger of Two Unsigned 8-Bit Values in Memory — MAXM

Operation: MAX ((A), (M)) ⇒ M

Description: Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger and leaves the larger of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in accumulator A has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = A – M)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MAXM *oprx0_xysp*	IDX	18 1C xb	OrPw	OrPw
MAXM *oprx9,xysp*	IDX1	18 1C xb ff	OrPwO	OrPwO
MAXM *oprx16,xysp*	IDX2	18 1C xb ee ff	OfrPwP	OfrPwP
MAXM [D,*xysp*]	[D,IDX]	18 1C xb	OfIfrPw	OfIfrPw
MAXM [*oprx16,xysp*]	[IDX2]	18 1C xb ee ff	OfIPrPw	OfIPrPw

MEM — Determine Grade of Membership (Fuzzy Logic) — MEM

Operation: Grade of Membership ⇒ $M_{(Y)}$
(Y) + $0001 ⇒ Y
(X) + $0004 ⇒ X

Description: Before executing MEM, initialize A, X, and Y. Load A with the current crisp value of a system input variable. Load Y with the fuzzy input RAM location where the grade of membership is to be stored. Load X with the first address of a 4-byte data structure that describes a trapezoidal membership function. The data structure consists of:

- Point_1 — The x-axis starting point for the leading side (at M_X)
- Slope_1 — The slope of the leading side (at M_{X+1})
- Point_2 — The x-axis position of the rightmost point (at M_{X+2})
- Slope_2 — The slope of the trailing side (at M_{X+3}); the right side slopes up and to the left from point_2

A slope_1 or slope_2 value of $00 is a special case in which the membership function either starts with a grade of $FF at input = point_1, or ends with a grade of $FF at input = point_2 (infinite slope).

During execution, the value of A remains unchanged. X is incremented by four and Y is incremented by one.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	?	–	?	?	?	?

H, N, Z, V, and C may be altered by this instruction.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MEM	Special	01	RRfOw	RRfOw

MINA

Place Smaller of Two Unsigned 8-Bit Values in Accumulator A

MINA

Operation: MIN ((A), (M)) ⇒ A

Description: Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger, and leaves the smaller of the two values in accumulator A. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in accumulator A has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the smallest value in a list of values.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = A – M)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MINA *oprx0_xysp*	IDX	18 19 xb	OrPf	OrfP
MINA *oprx9,xysp*	IDX1	18 19 xb ff	OrPO	OrPO
MINA *oprx16,xysp*	IDX2	18 19 xb ee ff	OfrPP	OfrPP
MINA [D,*xysp*]	[D,IDX]	18 19 xb	OfIfrPf	OfIfrfP
MINA [*oprx16,xysp*]	[IDX2]	18 19 xb ee ff	OfIPrPf	OfIPrfP

MINM

Place Smaller of Two Unsigned 8-Bit Values in Memory

MINM

Operation: MIN ((A), (M)) ⇒ M

Description: Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger and leaves the smaller of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in accumulator A has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Condition codes reflect internal subtraction (R = A – M)

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MINM *oprx0_xysp*	IDX	`18 1D xb`	`OrPw`	`OrPw`
MINM *oprx9,xysp*	IDX1	`18 1D xb ff`	`OrPwO`	`OrPwO`
MINM *oprx16,xysp*	IDX2	`18 1D xb ee ff`	`OfrPwP`	`OfrPwP`
MINM [D*,xysp*]	[D,IDX]	`18 1D xb`	`OfIfrPw`	`OfIfrPw`
MINM [*oprx16,xysp*]	[IDX2]	`18 1D xb ee ff`	`OfIPrPw`	`OfIPrPw`

MOVB — Move a Byte of Data from One Memory Location to Another — MOVB

Operation: $(M_1) \Rightarrow M_2$

Description: Moves the content of one memory location to another memory location. The content of the source memory location is not changed.

Move instructions use separate addressing modes to access the source and destination of a move. The following combinations of addressing modes are supported: IMM–EXT, IMM–IDX, EXT–EXT, EXT–IDX, IDX–EXT, and IDX–IDX. IDX operands allow indexed addressing mode specifications that fit in a single postbyte including 5-bit constant, accumulator offsets, and auto increment/decrement modes. Nine-bit and 16-bit constant offsets would require additional extension bytes and are not allowed. Indexed indirect modes (for example [D,r]) are also not allowed.

There are special considerations when using PC-relative addressing with move instructions. These are discussed in **3.10 Instructions Using Multiple Modes**.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form[1]	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MOVB #*opr8*, *opr16a*	IMM–EXT	`18 0B ii hh ll`	`OPwP`	`OPwP`
MOVB #*opr8i*, *oprx0_xysp*	IMM–IDX	`18 08 xb ii`	`OPwO`	`OPwO`
MOVB *opr16a*, *opr16a*	EXT–EXT	`18 0C hh ll hh ll`	`OrPwPO`	`OrPwPO`
MOVB *opr16a*, *oprx0_xysp*	EXT–IDX	`18 09 xb hh ll`	`OPrPw`	`OPrPw`
MOVB *oprx0_xysp*, *opr16a*	IDX–EXT	`18 0D xb hh ll`	`OrPwP`	`OrPwP`
MOVB *oprx0_xysp*, *oprx0_xysp*	IDX–IDX	`18 0A xb xb`	`OrPwO`	`OrPwO`

1. The first operand in the source code statement specifies the source for the move.

MOVW — Move a Word of Data from One Memory Location to Another — MOVW

Operation: $(M : M + 1_1) \Rightarrow M : M + 1_2$

Description: Moves the content of one 16-bit location in memory to another 16-bit location in memory. The content of the source memory location is not changed.

Move instructions use separate addressing modes to access the source and destination of a move. The following combinations of addressing modes are supported: IMM–EXT, IMM–IDX, EXT–EXT, EXT–IDX, IDX–EXT, and IDX–IDX. IDX operands allow indexed addressing mode specifications that fit in a single postbyte including 5-bit constant, accumulator offsets, and auto increment/decrement modes. Nine-bit and 16-bit constant offsets would require additional extension bytes and are not allowed. Indexed indirect modes (for example [D,r]) are also not allowed.

There are special considerations when using PC-relative addressing with move instructions. These are discussed in **3.10 Instructions Using Multiple Modes**.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form[1]	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MOVW *#opr16i, opr16a*	IMM–EXT	`18 03 jj kk hh ll`	`OPWPO`	`OPWPO`
MOVW *#opr16i, oprx0_xysp*	IMM–IDX	`18 00 xb jj kk`	`OPPW`	`OPPW`
MOVW *opr16a, opr16a*	EXT–EXT	`18 04 hh ll hh ll`	`ORPWPO`	`ORPWPO`
MOVW *opr16a, oprx0_xysp*	EXT–IDX	`18 01 xb hh ll`	`OPRPW`	`OPRPW`
MOVW *oprx0_xysp, opr16a*	IDX–EXT	`18 05 xb hh ll`	`ORPWP`	`ORPWP`
MOVW *oprx0_xysp, oprx0_xysp*	IDX–IDX	`18 02 xb xb`	`ORPWO`	`ORPWO`

1. The first operand in the source code statement specifies the source for the move.

MUL

Multiply
8-Bit by 8-Bit (Unsigned)

MUL

Operation: (A) × (B) ⇒ A : B

Description: Multiplies the 8-bit unsigned binary value in accumulator A by the 8-bit unsigned binary value in accumulator B and places the 16-bit unsigned result in double accumulator D. The carry flag allows rounding the most significant byte of the result through the sequence MUL, ADCA #0.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	Δ

C: R7
Set if bit 7 of the result (B bit 7) is set; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
MUL	INH	12	0	ff0

NEG — Negate Memory — NEG

Operation: $0 - (M) = (\overline{M}) + 1 \Rightarrow M$

Description: Replaces the content of memory location M with its two's complement (the value $80 is left unchanged).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is $00; cleared otherwise.

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. Two's complement overflow occurs if and only if (M) = $80

C: $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$
Set if there is a borrow in the implied subtraction from zero; cleared otherwise. Set in all cases except when (M) = $00.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
NEG *opr16a*	EXT	`70 hh ll`	`rPwO`	`rOPw`
NEG *oprx0_xysp*	IDX	`60 xb`	`rPw`	`rPw`
NEG *oprx9,xysp*	IDX1	`60 xb ff`	`rPwO`	`rPOw`
NEG *oprx16,xysp*	IDX2	`60 xb ee ff`	`frPwP`	`frPPw`
NEG [D,*xysp*]	[D,IDX]	`60 xb`	`fIfrPw`	`fIfrPw`
NEG [*oprx16,xysp*]	[IDX2]	`60 xb ee ff`	`fIPrPw`	`fIPrPw`

NEGA — Negate A — NEGA

Operation: $0 - (A) = (\overline{A}) + 1 \Rightarrow A$

Description: Replaces the content of accumulator A with its two's complement (the value $80 is left unchanged).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise
Two's complement overflow occurs if and only if (A) = $80

C: $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$
Set if there is a borrow in the implied subtraction from zero; cleared otherwise
Set in all cases except when (A) = $00

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
NEGA	INH	40	0	0

NEGB — Negate B — NEGB

Operation: $0 - (B) = (\overline{B}) + 1 \Rightarrow B$

Description: Replaces the content of accumulator B with its two's complement (the value $80 is left unchanged).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $R7 \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise
Two's complement overflow occurs if and only if (B) = $80

C: $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$
Set if there is a borrow in the implied subtraction from zero; cleared otherwise
Set in all cases except when (B) = $00

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
NEGB	INH	50	0	0

NOP — Null Operation — NOP

Operation: No operation

Description: This single-byte instruction increments the PC and does nothing else. No other CPU registers are affected. NOP is typically used to produce a time delay, although some software disciplines discourage CPU frequency-based time delays. During debug, NOP instructions are sometimes used to temporarily replace other machine code instructions, thus disabling the replaced instruction(s).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
NOP	INH	A7	0	0

ORAA — Inclusive OR A — ORAA

Operation: (A) + (M) ⇒ A

Description: Performs bitwise logical inclusive OR between the content of accumulator A and the content of memory location M and places the result in A. Each bit of A after the operation is the logical inclusive OR of the corresponding bits of M and of A before the operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ORAA #*opr8i*	IMM	8A ii	P	P
ORAA *opr8a*	DIR	9A dd	rPf	rfP
ORAA *opr16a*	EXT	BA hh ll	rPO	rOP
ORAA *oprx0_xysp*	IDX	AA xb	rPf	rfP
ORAA *oprx9,xysp*	IDX1	AA xb ff	rPO	rPO
ORAA *oprx16,xysp*	IDX2	AA xb ee ff	frPP	frPP
ORAA [D,*xysp*]	[D,IDX]	AA xb	fIfrPf	fIfrfP
ORAA [*oprx16,xysp*]	[IDX2]	AA xb ee ff	fIPrPf	fIPrfP

ORAB Inclusive OR B ORAB

Operation: (B) + (M) ⇒ B

Description: Performs bitwise logical inclusive OR between the content of accumulator B and the content of memory location M. The result is placed in B. Each bit of B after the operation is the logical inclusive OR of the corresponding bits of M and of B before the operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ORAB #*opr8i*	IMM	CA ii	P	P
ORAB *opr8a*	DIR	DA dd	rPf	rfP
ORAB *opr16a*	EXT	FA hh ll	rPO	rOP
ORAB *oprx0_xysp*	IDX	EA xb	rPf	rfP
ORAB *oprx9*,*xysp*	IDX1	EA xb ff	rPO	rPO
ORAB *oprx16*,*xysp*	IDX2	EA xb ee ff	frPP	frPP
ORAB [D,*xysp*]	[D,IDX]	EA xb	fIfrPf	fIfrfP
ORAB [*oprx16*,*xysp*]	[IDX2]	EA xb ee ff	fIPrPf	fIPrfP

ORCC Logical OR CCR with Mask ORCC

Operation: (CCR) + (M) ⇒ CCR

Description: Performs bitwise logical inclusive OR between the content of memory location M and the content of the CCR and places the result in the CCR. Each bit of the CCR after the operation is the logical OR of the corresponding bits of M and of CCR before the operation. To set one or more bits, set the corresponding bit of the mask equal to 1. Bits corresponding to 0s in the mask are not changed by the ORCC operation.

CCR Details:

S	X	H	I	N	Z	V	C
⇑	–	⇑	⇑	⇑	⇑	⇑	⇑

Condition code bits are set if the corresponding bit was 1 before the operation or if the corresponding bit in the instruction-provided mask is 1. The X interrupt mask cannot be set by any software instruction.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ORCC #*opr8i*	IMM	`14 ii`	P	P

PSHA Push A onto Stack PSHA

Operation: (SP) – $0001 ⇒ SP
(A) ⇒ $M_{(SP)}$

Description: Stacks the content of accumulator A. The stack pointer is decremented by one. The content of A is then stored at the address the SP points to.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHA	INH	36	0s	0s

PSHB Push B onto Stack PSHB

Operation: (SP) – $0001 ⇒ SP
(B) ⇒ $M_{(SP)}$

Description: Stacks the content of accumulator B. The stack pointer is decremented by one. The content of B is then stored at the address the SP points to.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHB	INH	37	0s	0s

PSHC — Push CCR onto Stack — PSHC

Operation: (SP) – $0001 ⇒ SP
(CCR) ⇒ $M_{(SP)}$

Description: Stacks the content of the condition codes register. The stack pointer is decremented by one. The content of the CCR is then stored at the address to which the SP points.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHC	INH	39	0s	0s

PSHD Push Double Accumulator onto Stack PSHD

Operation: (SP) – $0002 ⇒ SP
(A : B) ⇒ $M_{(SP)}$: $M_{(SP+1)}$

Description: Stacks the content of double accumulator D. The stack pointer is decremented by two, then the contents of accumulators A and B are stored at the location to which the SP points.

After PSHD executes, the SP points to the stacked value of accumulator A. This stacking order is the opposite of the order in which A and B are stacked when an interrupt is recognized. The interrupt stacking order is backward-compatible with the M6800, which had no 16-bit accumulator.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHD	INH	3B	0S	0S

PSHX Push Index Register X onto Stack PSHX

Operation: (SP) – $0002 ⇒ SP
$(X_H : X_L) ⇒ M_{(SP)} : M_{(SP+1)}$

Description: Stacks the content of index register X. The stack pointer is decremented by two. The content of X is then stored at the address to which the SP points. After PSHX executes, the SP points to the stacked value of the high-order half of X.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHX	INH	34	0S	0S

PSHY Push Index Register Y onto Stack PSHY

Operation: (SP) – $0002 ⇒ SP
$(Y_H : Y_L) ⇒ M_{(SP)} : M_{(SP+1)}$

Description: Stacks the content of index register Y. The stack pointer is decremented by two. The content of Y is then stored at the address to which the SP points. After PSHY executes, the SP points to the stacked value of the high-order half of Y.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PSHY	INH	35	0S	0S

PULA — Pull A from Stack — PULA

Operation: $(M_{(SP)}) \Rightarrow A$
$(SP) + \$0001 \Rightarrow SP$

Description: Accumulator A is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
PULA	INH	32	uf0	uf0

PULB Pull B from Stack PULB

Operation: $(M_{(SP)}) \Rightarrow B$
$(SP) + \$0001 \Rightarrow SP$

Description: Accumulator B is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
PULB	INH	33	uf0	uf0

PULC Pull Condition Code Register from Stack PULC

Operation: $(M_{(SP)}) \Rightarrow CCR$
$(SP) + \$0001 \Rightarrow SP$

Description: The condition code register is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ

Condition codes take on the value pulled from the stack, except that the X mask bit cannot change from 0 to 1. Software can leave the X bit set, leave it cleared, or change it from 1 to 0, but it can be set only by a reset or by recognition of an $\overline{\text{XIRQ}}$ interrupt.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PULC	INH	38	`uf0`	`uf0`

PULD Pull Double Accumulator from Stack PULD

Operation: $(M_{(SP)} : M_{(SP+1)}) \Rightarrow A : B$
$(SP) + \$0002 \Rightarrow SP$

Description: Double accumulator D is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

The order in which A and B are pulled from the stack is the opposite of the order in which A and B are pulled when an RTI instruction is executed. The interrupt stacking order for A and B is backward-compatible with the M6800, which had no 16-bit accumulator.

Pull instructions are commonly used at the end of a subroutine to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
PULD	INH	3A	Uf0	Uf0

PULX Pull Index Register X from Stack PULX

Operation: $(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L$
$(SP) + \$0002 \Rightarrow SP$

Description: Index register X is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

Pull instructions are commonly used at the end of a subroutine to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
PULX	INH	30	Uf0	Uf0

PULY — Pull Index Register Y from Stack — PULY

Operation: $(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L$
$(SP) + \$0002 \Rightarrow SP$

Description: Index register Y is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

Pull instructions are commonly used at the end of a subroutine to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
PULY	INH	31	Uf0	Uf0

REV — Fuzzy Logic Rule Evaluation — REV

Operation: MIN-MAX Rule Evaluation

Description: Performs an unweighted evaluation of a list of rules, using fuzzy input values to produce fuzzy outputs. REV can be interrupted, so it does not adversely affect interrupt latency.

The REV instruction uses an 8-bit offset from a base address stored in index register Y to determine the address of each fuzzy input and fuzzy output. For REV to execute correctly, each rule in the knowledge base must consist of a table of 8-bit antecedent offsets followed by a table of 8-bit consequent offsets. The value $FE marks boundaries between antecedents and consequents and between successive rules. The value $FF marks the end of the rule list. REV can evaluate any number of rules with any number of inputs and outputs.

Beginning with the address pointed to by the first rule antecedent, REV evaluates each successive fuzzy input value until it encounters an $FE separator. Operation is similar to that of a MINA instruction. The smallest input value is the truth value of the rule. Then, beginning with the address pointed to by the first rule consequent, the truth value is compared to each successive fuzzy output value until another $FE separator is encountered; if the truth value is greater than the current output value, it is written to the output. Operation is similar to that of a MAXM instruction. Rules are processed until an $FF terminator is encountered.

Before executing REV, perform these set up operations.

- X must point to the first 8-bit element in the rule list.
- Y must point to the base address for fuzzy inputs and fuzzy outputs.
- A must contain the value $FF, and the CCR V bit must = 0. (LDAA #$FF places the correct value in A and clears V.)
- Clear fuzzy outputs to 0s.

Index register X points to the element in the rule list that is being evaluated. X is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, X points to the next address after the $FF separator at the end of the rule list.

REV

Fuzzy Logic Rule Evaluation (Continued)

REV

Index register Y points to the base address for the fuzzy inputs and fuzzy outputs. The value in Y does not change during execution.

Accumulator A holds intermediate results. During antecedent processing, a MIN function compares each fuzzy input to the value stored in A, and writes the smaller of the two to A. When all antecedents have been evaluated, A contains the smallest input value. This is the truth value used during consequent processing. Accumulator A must be initialized to $FF for the MIN function to evaluate the inputs of the first rule correctly. For subsequent rules, the value $FF is written to A when an $FE marker is encountered. At the end of execution, accumulator A holds the truth value for the last rule.

The V status bit signals whether antecedents (0) or consequents (1) are being processed. V must be initialized to 0 for processing to begin with the antecedents of the first rule. Once execution begins, the value of V is automatically changed as $FE separators are encountered. At the end of execution, V should equal 1, because the last element before the $FF end marker should be a rule consequent. If V is equal to 0 at the end of execution, the rule list is incorrect.

Fuzzy outputs must be cleared to $00 before processing begins in order for the MAX algorithm used during consequent processing to work correctly. Residual output values would cause incorrect comparison.

Refer to **Section 9. Fuzzy Logic Support** for details.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	?	–	?	?	Δ	?

V: 1; Normally set, unless rule structure is erroneous

H, N, Z, and C may be altered by this instruction

Source Form	Address Mode	Object Code	Access Detail(1) HCS12	Access Detail(1) M68HC12
REV (replace comma if interrupted)	Special	18 3A	Orf(t,tx)O ff + Orf(t,	Orf(t,tx)O ff + Orf(t,

1. The 3-cycle loop in parentheses is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.

REVW Fuzzy Logic Rule Evaluation (Weighted) REVW

Operation: MIN-MAX Rule Evaluation with Optional Rule Weighting

Description: REVW performs either weighted or unweighted evaluation of a list of rules, using fuzzy inputs to produce fuzzy outputs. REVW can be interrupted, so it does not adversely affect interrupt latency.

For REVW to execute correctly, each rule in the knowledge base must consist of a table of 16-bit antecedent pointers followed by a table of 16-bit consequent pointers. The value $FFFE marks boundaries between antecedents and consequents, and between successive rules. The value $FFFF marks the end of the rule list. REVW can evaluate any number of rules with any number of inputs and outputs.

Setting the C status bit enables weighted evaluation. To use weighted evaluation, a table of 8-bit weighting factors, one per rule, must be stored in memory. Index register Y points to the weighting factors.

Beginning with the address pointed to by the first rule antecedent, REVW evaluates each successive fuzzy input value until it encounters an $FFFE separator. Operation is similar to that of a MINA instruction. The smallest input value is the truth value of the rule. Next, if weighted evaluation is enabled, a computation is performed, and the truth value is modified. Then, beginning with the address pointed to by the first rule consequent, the truth value is compared to each successive fuzzy output value until another $FFFE separator is encountered; if the truth value is greater than the current output value, it is written to the output. Operation is similar to that of a MAXM instruction. Rules are processed until an $FFFF terminator is encountered.

Perform these set up operations before execution:

- X must point to the first 16-bit element in the rule list.
- A must contain the value $FF, and the CCR V bit must = 0 (LDAA #$FF places the correct value in A and clears V).
- Clear fuzzy outputs to 0s.
- Set or clear the CCR C bit. When weighted evaluation is enabled, Y must point to the first item in a table of 8-bit weighting factors.

REVW Fuzzy Logic Rule Evaluation (Weighted) (Continued) REVW

Index register X points to the element in the rule list that is being evaluated. X is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, X points to the address after the $FFFF separator at the end of the rule list.

Index register Y points to the weighting factor being used. Y is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, Y points to the last weighting factor used. When weighting is not used (C = 0), Y is not changed.

Accumulator A holds intermediate results. During antecedent processing, a MIN function compares each fuzzy input to the value stored in A and writes the smaller of the two to A. When all antecedents have been evaluated, A contains the smallest input value. For unweighted evaluation, this is the truth value used during consequent processing. For weighted evaluation, the value in A is multiplied by the quantity (Rule Weight + 1) and the upper eight bits of the result replace the content of A. Accumulator A must be initialized to $FF for the MIN function to evaluate the inputs of the first rule correctly. For subsequent rules, the value $FF is automatically written to A when an $FFFE marker is encountered. At the end of execution, accumulator A holds the truth value for the last rule.

The V status bit signals whether antecedents (0) or consequents (1) are being processed. V must be initialized to 0 for processing to begin with the antecedents of the first rule. Once execution begins, the value of V is automatically changed as $FFFE separators are encountered. At the end of execution, V should equal 1, because the last element before the $FF end marker should be a rule consequent. If V is equal to 0 at the end of execution, the rule list is incorrect.

Fuzzy outputs must be cleared to $00 before processing begins in order for the MAX algorithm used during consequent processing to work correctly. Residual output values would cause incorrect comparison.

Refer to **Section 9. Fuzzy Logic Support** for details.

REVW

Fuzzy Logic Rule Evaluation (Weighted) (Concluded)

REVW

CCR Details:

S	X	H	I	N	Z	V	C
–	–	?	–	?	?	Δ	!

V: 1; Normally set, unless rule structure is erroneous

C: Selects weighted (1) or unweighted (0) rule evaluation

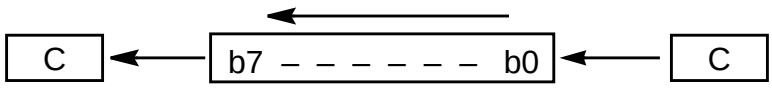
H, N, Z, and C may be altered by this instruction

Source Form	Address Mode	Object Code	Access Detail[1]	
			HCS12	M68HC12
REVW (add 2 at end of ins if wts) (replace comma if interrupted)	Special	18 3B	ORf(t,Tx)O (r,RfRf) ffff + ORf(t,	ORf(tTx)O (r,RfRf) ffff + ORf(t,

1. The 3-cycle loop in parentheses expands to five cycles for separators when weighting is enabled. The loop is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.

ROL — Rotate Left Memory — ROL

Operation:



Description: Shifts all bits of memory location M one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of M. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, ROL HIGH could be used where LOW, MID and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

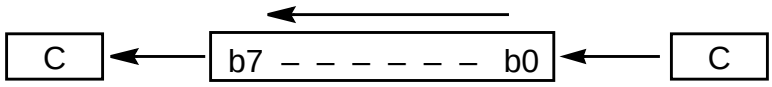
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M7
Set if the MSB of M was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ROL *opr16a*	EXT	`75 hh ll`	`rPw0`	`r0Pw`
ROL *oprx0_xysp*	IDX	`65 xb`	`rPw`	`rPw`
ROL *oprx9,xysp*	IDX1	`65 xb ff`	`rPw0`	`rP0w`
ROL *oprx16,xysp*	IDX2	`65 xb ee ff`	`frPwP`	`frPPw`
ROL [D,*xysp*]	[D,IDX]	`65 xb`	`fIfrPw`	`fIfrPw`
ROL [*oprx16,xysp*]	[IDX2]	`65 xb ee ff`	`fIPrPw`	`fIPrPw`

ROLA — Rotate Left A — ROLA

Operation:



Description: Shifts all bits of accumulator A one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of A. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, and ROL HIGH could be used where LOW, MID, and HIGH refer to the low-order, middle, and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

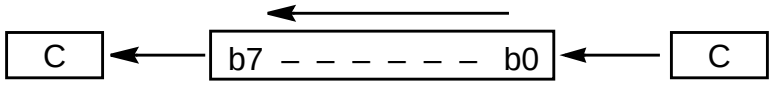
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A7
Set if the MSB of A was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ROLA	INH	45	0	0

ROLB Rotate Left B ROLB

Operation:



Description: Shifts all bits of accumulator B one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of B. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, and ROL HIGH could be used where LOW, MID, and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

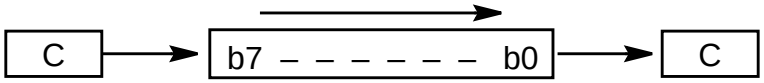
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B7
Set if the MSB of B was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ROLB	INH	55	0	0

ROR — Rotate Right Memory — ROR

Operation:



Description: Shifts all bits of memory location M one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of M. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, and ROR LOW could be used where LOW, MID, and HIGH refer to the low-order, middle, and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

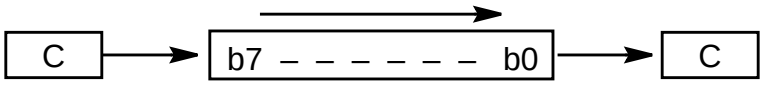
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: M0
Set if the LSB of M was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
ROR *opr16a*	EXT	76 hh ll	rPw0	r0Pw
ROR *oprx0_xysp*	IDX	66 xb	rPw	rPw
ROR *oprx9,xysp*	IDX1	66 xb ff	rPw0	rP0w
ROR *oprx16,xysp*	IDX2	66 xb ee ff	frPwP	frPPw
ROR [D,*xysp*]	[D,IDX]	66 xb	fIfrPw	fIfrPw
ROR [*oprx16,xysp*]	[IDX2]	66 xb ee ff	fIPrPw	fIPrPw

RORA — Rotate Right A — RORA

Operation:



Description: Shifts all bits of accumulator A one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of A. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, and ROR LOW could be used where LOW, MID, and HIGH refer to the low-order, middle, and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

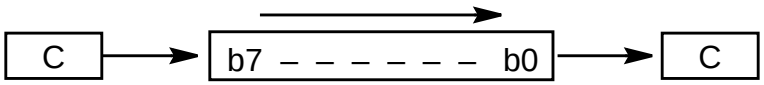
V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: A0
Set if the LSB of A was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
RORA	INH	46	0	0

RORB — Rotate Right B — RORB

Operation:



Description: Shifts all bits of accumulator B one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of B. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, and ROR LOW could be used where LOW, MID, and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$ (for N and C after the shift)
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift)

C: B0
Set if the LSB of B was set before the shift; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
RORB	INH	56	0	0

RTC — Return from Call — RTC

Operation:

$(M_{(SP)}) \Rightarrow PPAGE$
$(SP) + \$0001 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L$
$(SP) + \$0002 \Rightarrow SP$

Description:

Terminates subroutines in expanded memory invoked by the CALL instruction. Returns execution flow from the subroutine to the calling program. The program overlay page (PPAGE) register and the return address are restored from the stack; program execution continues at the restored address. For code compatibility purposes, CALL and RTC also execute correctly in devices that do not have expanded memory capability.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
RTC	INH	0A	uUnfPPP	uUnPPP

RTI — Return from Interrupt — RTI

Operation:

$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) - \$0002 \Rightarrow SP$
$(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$

Description:

Restores system context after interrupt service processing is completed. The condition codes, accumulators B and A, index register X, the PC, and index register Y are restored to a state pulled from the stack. The X mask bit may be cleared as a result of an RTI instruction, but cannot be set if it was cleared prior to execution of the RTI instruction.

If another interrupt is pending when RTI has finished restoring registers from the stack, the SP is adjusted to preserve stack content, and the new vector is fetched. This operation is functionally identical to the same operation in the M68HC11, where registers actually are re-stacked, but is faster.

CCR Details:

S	X	H	I	N	Z	V	C
Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ

Condition codes take on the value pulled from the stack, except that the X mask bit cannot change from 0 to 1. Software can leave the X bit set, leave it cleared, or change it from 1 to 0, but it can be set only by a reset or by recognition of an $\overline{\text{XIRQ}}$ interrupt.

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
RTI (with interrupt pending)	INH	0B	uUUUUPPP uUUUUfVfPPP	uUUUUPPP uUUUUVfPPP

RTS — Return from Subroutine — RTS

Operation: $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L$; (SP) + $0002 ⇒ SP

Description: Restores context at the end of a subroutine. Loads the program counter with a 16-bit value pulled from the stack and increments the stack pointer by two. Program execution continues at the address restored from the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
RTS	INH	3D	UfPPP	UfPPP

SBA Subtract Accumulators SBA

Operation: (A) – (B) ⇒ A

Description: Subtracts the content of accumulator B from the content of accumulator A and places the result in A. The content of B is not affected. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{B7} \bullet \overline{R7} + \overline{A7} \bullet B7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet B7 + B7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the absolute value of B is larger than the absolute value of A; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SBA	INH	18 16	00	00

SBCA Subtract with Carry from A SBCA

Operation: (A) – (M) – C ⇒ A

Description: Subtracts the content of memory location M and the value of the C status bit from the content of accumulator A. The result is placed in A. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the absolute value of the content of memory plus previous carry is larger than the absolute value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SBCA #*opr8i*	IMM	82 ii	P	P
SBCA *opr8a*	DIR	92 dd	rPf	rfP
SBCA *opr16a*	EXT	B2 hh ll	rPO	rOP
SBCA *oprx0_xysp*	IDX	A2 xb	rPf	rfP
SBCA *oprx9,xysp*	IDX1	A2 xb ff	rPO	rPO
SBCA *oprx16,xysp*	IDX2	A2 xb ee ff	frPP	frPP
SBCA [D,*xysp*]	[D,IDX]	A2 xb	fIfrPf	fIfrfP
SBCA [*oprx16,xysp*]	[IDX2]	A2 xb ee ff	fIPrPf	fIPrfP

SBCB Subtract with Carry from B SBCB

Operation: (B) – (M) – C ⇒ B

Description: Subtracts the content of memory location M and the value of the C status bit from the content of accumulator B. The result is placed in B. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $B7 \bullet \overline{M7} \bullet \overline{R7} + \overline{B7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{B7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{B7}$
Set if the absolute value of the content of memory plus previous carry is larger than the absolute value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SBCB #*opr8i*	IMM	C2 ii	P	P
SBCB *opr8a*	DIR	D2 dd	rPf	rfP
SBCB *opr16a*	EXT	F2 hh ll	rPO	rOP
SBCB *oprx0_xysp*	IDX	E2 xb	rPf	rfP
SBCB *oprx9,xysp*	IDX1	E2 xb ff	rPO	rPO
SBCB *oprx16,xysp*	IDX2	E2 xb ee ff	frPP	frPP
SBCB [D,*xysp*]	[D,IDX]	E2 xb	fIfrPf	fIfrfP
SBCB [*oprx16,xysp*]	[IDX2]	E2 xb ee ff	fIPrPf	fIPrfP

SEC — Set Carry — SEC

Operation: 1 ⇒ C bit

Description: Sets the C status bit. This instruction is assembled as ORCC #$01. The ORCC instruction can be used to set any combination of bits in the CCR in one operation.

SEC can be used to set up the C bit prior to a shift or rotate instruction involving the C bit.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	1

C: 1; set

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
SEC *translates to...* ORCC #$01	IMM	14 01	P	P

SEI — Set Interrupt Mask — SEI

Operation: 1 ⇒ I bit

Description: Sets the I mask bit. This instruction is assembled as ORCC #$10. The ORCC instruction can be used to set any combination of bits in the CCR in one operation. When the I bit is set, all maskable interrupts are inhibited, and the CPU will recognize only non-maskable interrupt sources or an SWI.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	1	–	–	–	–

I: 1; set

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SEI *translates to...* ORCC #$10	IMM	14 10	P	P

SEV — Set Two's Complement Overflow Bit — SEV

Operation: 1 ⇒ V bit

Description: Sets the V status bit. This instruction is assembled as ORCC #$02. The ORCC instruction can be used to set any combination of bits in the CCR in one operation.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	1	–

V: 1; set

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SEV *translates to...* ORCC #$02	IMM	14 02	P	P

SEX — Sign Extend into 16-Bit Register — SEX

Operation: If r1 bit 7 = 0, then $00 : (r1) ⇒ r2
If r1 bit 7 = 1, then $FF : (r1) ⇒ r2

Description: This instruction is an alternate mnemonic for the TFR r1,r2 instruction, where r1 is an 8-bit register and r2 is a 16-bit register. The result in r2 is the 16-bit sign extended representation of the original two's complement number in r1. The content of r1 is unchanged in all cases except that of SEX A,D (D is A : B).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code(1)	Access Detail HCS12	Access Detail M68HC12
SEX *abc,dxys*	INH	B7 eb	P	P

1. Legal coding for eb is summarized in the following table. Columns represent the high-order source digit. Rows represent the low-order destination digit (MSB is a don't care). Values are in hexadecimal.

	0	1	2
3	sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2
4	sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D
5	sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X
6	sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y
7	sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP

STAA Store Accumulator A STAA

Operation: (A) ⇒ M

Description: Stores the content of accumulator A in memory location M. The content of A is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STAA *opr8a*	DIR	5A dd	Pw	Pw
STAA *opr16a*	EXT	7A hh ll	PwO	wOP
STAA *oprx0_xysp*	IDX	6A xb	Pw	Pw
STAA *oprx9,xysp*	IDX1	6A xb ff	PwO	PwO
STAA *oprx16,xysp*	IDX2	6A xb ee ff	PwP	PwP
STAA [D,*xysp*]	[D,IDX]	6A xb	PIfw	PIfPw
STAA [*oprx16,xysp*]	[IDX2]	6A xb ee ff	PIPw	PIPPw

STAB Store Accumulator B STAB

Operation: (B) ⇒ M

Description: Stores the content of accumulator B in memory location M. The content of B is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STAB *opr8a*	DIR	5B dd	Pw	Pw
STAB *opr16a*	EXT	7B hh ll	PwO	wOP
STAB *oprx0_xysp*	IDX	6B xb	Pw	Pw
STAB *oprx9,xysp*	IDX1	6B xb ff	PwO	PwO
STAB *oprx16,xysp*	IDX2	6B xb ee ff	PwP	PwP
STAB [D*,xysp*]	[D,IDX]	6B xb	PIfw	PIfPw
STAB [*oprx16,xysp*]	[IDX2]	6B xb ee ff	PIPw	PIPPw

STD — Store Double Accumulator — STD

Operation: (A : B) ⇒ M : M + 1

Description: Stores the content of double accumulator D in memory location M : M + 1. The content of D is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STD *opr8a*	DIR	5C dd	PW	PW
STD *opr16a*	EXT	7C hh ll	PWO	WOP
STD *oprx0_xysp*	IDX	6C xb	PW	PW
STD *oprx9,xysp*	IDX1	6C xb ff	PWO	PWO
STD *oprx16,xysp*	IDX2	6C xb ee ff	PWP	PWP
STD [D,*xysp*]	[D,IDX]	6C xb	PIfW	PIfPW
STD [*oprx16,xysp*]	[IDX2]	6C xb ee ff	PIPW	PIPPW

STOP Stop Processing STOP

Operation:

(SP) – $0002 ⇒ SP; $RTN_H : RTN_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $Y_H : Y_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $X_H : X_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; B : A⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0001 ⇒ SP; CCR ⇒ $(M_{(SP)})$
Stop All Clocks

Description:

When the S control bit is set, STOP is disabled and operates like a 2-cycle NOP instruction. When the S bit is cleared, STOP stacks CPU context, stops all system clocks, and puts the device in standby mode.

Standby operation minimizes system power consumption. The contents of registers and the states of I/O pins remain unchanged.

Asserting the $\overline{\text{RESET}}$, $\overline{\text{XIRQ}}$, or $\overline{\text{IRQ}}$ signals ends standby mode. Stacking on entry to STOP allows the CPU to recover quickly when an interrupt is used, provided a stable clock is applied to the device. If the system uses a clock reference crystal that also stops during low-power mode, crystal startup delay lengthens recovery time.

If $\overline{\text{XIRQ}}$ is asserted while the X mask bit = 0 ($\overline{\text{XIRQ}}$ interrupts enabled), execution resumes with a vector fetch for the $\overline{\text{XIRQ}}$ interrupt. If the X mask bit = 1 ($\overline{\text{XIRQ}}$ interrupts disabled), a 2-cycle recovery sequence including an O cycle is used to adjust the instruction queue, and execution continues with the next instruction after STOP.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STOP (entering STOP)	INH	18 3E	00SSSSsf	00SSSfSs
(exiting STOP)			fVfPPP	fVfPPP
(continue)			ff	f0
(if STOP disabled)			00	00

STS — Store Stack Pointer — STS

Operation: $(SP_H : SP_L) \Rightarrow M : M + 1$

Description: Stores the content of the stack pointer in memory. The most significant byte of the SP is stored at the specified address, and the least significant byte of the SP is stored at the next higher byte address (the specified address plus one).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STS *opr8a*	DIR	5F dd	PW	PW
STS *opr16a*	EXT	7F hh ll	PWO	WOP
STS *oprx0_xysp*	IDX	6F xb	PW	PW
STS *oprx9,xysp*	IDX1	6F xb ff	PWO	PWO
STS *oprx16,xysp*	IDX2	6F xb ee ff	PWP	PWP
STS [D,*xysp*]	[D,IDX]	6F xb	PIfW	PIfPW
STS [*oprx16,xysp*]	[IDX2]	6F xb ee ff	PIPW	PIPPW

STX — Store Index Register X — STX

Operation: $(X_H : X_L) \Rightarrow M : M + 1$

Description: Stores the content of index register X in memory. The most significant byte of X is stored at the specified address, and the least significant byte of X is stored at the next higher byte address (the specified address plus one).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STX *opr8a*	DIR	5E dd	PW	PW
STX *opr16a*	EXT	7E hh ll	PWO	WOP
STX *oprx0_xysp*	IDX	6E xb	PW	PW
STX *oprx9,xysp*	IDX1	6E xb ff	PWO	PWO
STX *oprx16,xysp*	IDX2	6E xb ee ff	PWP	PWP
STX [D,*xysp*]	[D,IDX]	6E xb	PIfW	PIfPW
STX [*oprx16,xysp*]	[IDX2]	6E xb ee ff	PIPW	PIPPW

STY Store Index Register Y STY

Operation: $(Y_H : Y_L) \Rightarrow M : M + 1$

Description: Stores the content of index register Y in memory. The most significant byte of Y is stored at the specified address, and the least significant byte of Y is stored at the next higher byte address (the specified address plus one).

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
STY *opr8a*	DIR	5D dd	PW	PW
STY *opr16a*	EXT	7D hh ll	PWO	WOP
STY *oprx0_xysp*	IDX	6D xb	PW	PW
STY *oprx9,xysp*	IDX1	6D xb ff	PWO	PWO
STY *oprx16,xysp*	IDX2	6D xb ee ff	PWP	PWP
STY [D,*xysp*]	[D,IDX]	6D xb	PIfW	PIfPW
STY [*oprx16,xysp*]	[IDX2]	6D xb ee ff	PIPW	PIPPW

SUBA — Subtract A — SUBA

Operation: (A) – (M) ⇒ A

Description: Subtracts the content of memory location M from the content of accumulator A, and places the result in A. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $A7 \bullet \overline{M7} \bullet \overline{R7} + \overline{A7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{A7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{A7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SUBA #*opr8i*	IMM	80 ii	P	P
SUBA *opr8a*	DIR	90 dd	rPf	rfP
SUBA *opr16a*	EXT	B0 hh ll	rPO	rOP
SUBA *oprx0_xysp*	IDX	A0 xb	rPf	rfP
SUBA *oprx9,xysp*	IDX1	A0 xb ff	rPO	rPO
SUBA *oprx16,xysp*	IDX2	A0 xb ee ff	frPP	frPP
SUBA [D,*xysp*]	[D,IDX]	A0 xb	fIfrPf	fIfrfP
SUBA [*oprx16,xysp*]	[IDX2]	A0 xb ee ff	fIPrPf	fIPrfP

SUBB Subtract B SUBB

Operation: (B) – (M) ⇒ B

Description: Subtracts the content of memory location M from the content of accumulator B and places the result in B. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: $B7 \bullet \overline{M7} \bullet \overline{R7} + \overline{B7} \bullet M7 \bullet R7$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{B7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{B7}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SUBB #*opr8i*	IMM	C0 ii	P	P
SUBB *opr8a*	DIR	D0 dd	rPf	rfP
SUBB *opr16a*	EXT	F0 hh ll	rPO	rOP
SUBB *oprx0_xysp*	IDX	E0 xb	rPf	rfP
SUBB *oprx9,xysp*	IDX1	E0 xb ff	rPO	rPO
SUBB *oprx16,xysp*	IDX2	E0 xb ee ff	frPP	frPP
SUBB [D,*xysp*]	[D,IDX]	E0 xb	fIfrPf	fIfrfP
SUBB [*oprx16,xysp*]	[IDX2]	E0 xb ee ff	fIPrPf	fIPrfP

SUBD Subtract Double Accumulator SUBD

Operation: (A : B) – (M : M + 1) ⇒ A : B

Description: Subtracts the content of memory location M : M + 1 from the content of double accumulator D and places the result in D. For subtraction instructions, the C status bit represents a borrow.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $0000; cleared otherwise

V: $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise

C: $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SUBD #*opr16i*	IMM	83 jj kk	PO	OP
SUBD *opr8a*	DIR	93 dd	RPf	RfP
SUBD *opr16a*	EXT	B3 hh ll	RPO	ROP
SUBD *oprx0_xysp*	IDX	A3 xb	RPf	RfP
SUBD *oprx9,xyssp*	IDX1	A3 xb ff	RPO	RPO
SUBD *oprx16,xysp*	IDX2	A3 xb ee ff	fRPP	fRPP
SUBD [D,*xysp*]	[D,IDX]	A3 xb	fIfRPf	fIfRfP
SUBD [*oprx16,xysp*]	[IDX2]	A3 xb ee ff	fIPRPf	fIPRfP

SWI — Software Interrupt — SWI

Operation:

(SP) – $0002 ⇒ SP; $RTN_H : RTN_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $Y_H : Y_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $X_H : X_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; B : A⇒ $(M_{(SP)} : M_{(SP+1)})$
(SP) – $0001 ⇒ SP; CCR ⇒ $(M_{(SP)})$
1 ⇒ I
(SWI Vector) ⇒ PC

Description:

Causes an interrupt without an external interrupt service request. Uses the address of the next instruction after SWI as a return address. Stacks the return address, index registers Y and X, accumulators B and A, and the CCR, decrementing the SP before each item is stacked. The I mask bit is then set, the PC is loaded with the SWI vector, and instruction execution resumes at that location. SWI is not affected by the I mask bit. Refer to **Section 7. Exception Processing** for more information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	1	–	–	–	–

I: 1; set

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
SWI	INH	3F	VSPSSPSsP(1)	VSPSSPSsP(1)

1. The CPU also uses the SWI processing sequence for hardware interrupts and unimplemented opcode traps. A variation of the sequence (VfPPP) is used for resets.

TAB

Transfer from Accumulator A to Accumulator B

TAB

Operation: (A) ⇒ B

Description: Moves the content of accumulator A to accumulator B. The former content of B is lost; the content of A is not affected. Unlike the general transfer instruction TFR A,B which does not affect condition codes, the TAB instruction affects the N, Z, and V status bits for compatibility with M68HC11.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TAB	INH	18 0E	00	00

TAP — Transfer from Accumulator A to Condition Code Register — TAP

Operation: (A) ⇒ CCR

Description: Transfers the logic states of bits [7:0] of accumulator A to the corresponding bit positions of the CCR. The content of A remains unchanged. The X mask bit can be cleared as a result of a TAP, but cannot be set if it was cleared prior to execution of the TAP. If the I bit is cleared, there is a 1-cycle delay before the system allows interrupt requests. This prevents interrupts from occurring between instructions in the sequences CLI, WAI and CLI, SEI.

This instruction is accomplished with the TFR A,CCR instruction. For compatibility with the M68HC11, the mnemonic TAP is translated by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ

Condition codes take on the value of the corresponding bit of accumulator A, except that the X mask bit cannot change from 0 to 1. Software can leave the X bit set, leave it cleared, or change it from 1 to 0, but it can only be set by a reset or by recognition of an $\overline{\text{XIRQ}}$ interrupt.

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
TAP *translates to...* TFR A,CCR	INH	B7 02	P	P

TBA

Transfer from Accumulator B to Accumulator A

TBA

Operation: (B) ⇒ A

Description: Moves the content of accumulator B to accumulator A. The former content of A is lost; the content of B is not affected. Unlike the general transfer instruction TFR B,A, which does not affect condition codes, the TBA instruction affects N, Z, and V for compatibility with M68HC11.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TBA	INH	18 0F	00	00

TBEQ Test and Branch if Equal to Zero TBEQ

Operation: If (Counter) = 0, then (PC) + $0003 + Rel ⇒ PC

Description: Tests the specified counter register A, B, D, X, Y, or SP. If the counter register is zero, branches to the specified relative destination. TBEQ is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBEQ and IBEQ instructions are similar to TBEQ, except that the counter is decremented or incremented rather than simply being tested. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code(1)	Access Detail	
			HCS12	M68HC12
TBEQ *abdxys,rel9*	REL	`04 lb rr`	`PPP/PPO`	`PPP`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (TBEQ – 0) or not zero (TBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 0:1 for TBEQ.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	TBEQ A, *rel9*	`04 40 rr`	`04 50 rr`
B	001	TBEQ B, *rel9*	`04 41 rr`	`04 51 rr`
D	100	TBEQ D, *rel9*	`04 44 rr`	`04 54 rr`
X	101	TBEQ X, *rel9*	`04 45 rr`	`04 55 rr`
Y	110	TBEQ Y, *rel9*	`04 46 rr`	`04 56 rr`
SP	111	TBEQ SP, *rel9*	`04 47 rr`	`04 57 rr`

TBL — Table Lookup and Interpolate — TBL

Operation: (M) + [(B) × ((M+1) – (M))] ⇒ A

Description: Linearly interpolates one of 256 result values that fall between each pair of data entries in a lookup table stored in memory. Data entries in the table represent the Y values of endpoints of equally spaced line segments. Table entries and the interpolated result are 8-bit values. The result is stored in accumulator A.

Before executing TBL, an index register points to the table entry corresponding to the X value (X1) that is closest to, but less than or equal to, the desired lookup point (XL, YL). This defines the left end of a line segment and the right end is defined by the next data entry in the table. Prior to execution, accumulator B holds a binary fraction (radix point to left of MSB), representing the ratio (XL–X1) ÷ (X2–X1).

The 8-bit unrounded result is calculated using the following expression:

$$A = Y1 + [(B) \times (Y2 - Y1)]$$

Where

(B) = (XL – X1) ÷ (X2 – X1)
Y1 = 8-bit data entry pointed to by <effective address>
Y2 = 8-bit data entry pointed to by <effective address> + 1

The intermediate value [(B) × (Y2 – Y1)] produces a 16-bit result with the radix point between bits 7 and 8. Any indexed addressing mode referenced to X, Y, SP, or PC, except indirect modes or 9-bit and 16-bit offset modes, can be used to identify the first data point (X1,Y1). The second data point is the next table entry.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	–	Δ(1)

1. C-bit was undefined in original M68HC12.

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is $00; cleared otherwise
C: Set if result can be rounded up; cleared otherwise

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TBL *oprx0_xysp*	IDX	18 3D xb	ORfffP	OrrfffP

TBNE Test and Branch if Not Equal to Zero TBNE

Operation: If (Counter) ≠ 0, then (PC) + $0003 + Rel ⇒ PC

Description: Tests the specified counter register A, B, D, X, Y, or SP. If the counter register is not zero, branches to the specified relative destination. TBNE is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBNE and IBNE instructions are similar to TBNE, except that the counter is decremented or incremented rather than simply being tested. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code[1]	Access Detail HCS12	Access Detail M68HC12
TBNE *abdxys,rel9*	REL	`04 lb rr`	`PPP/PPO`	`PPP`

1. Encoding for `lb` is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (TBEQ – 0) or not zero (TBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 0:1 for TBNE.

Count Register	Bits 2:0	Source Form	Object Code (If Offset is Positive)	Object Code (If Offset is Negative)
A	000	TBNE A, *rel9*	`04 60 rr`	`04 70 rr`
B	001	TBNE B, *rel9*	`04 61 rr`	`04 71 rr`
D	100	TBNE D, *rel9*	`04 64 rr`	`04 74 rr`
X	101	TBNE X, *rel9*	`04 65 rr`	`04 75 rr`
Y	110	TBNE Y, *rel9*	`04 66 rr`	`04 76 rr`
SP	111	TBNE SP, *rel9*	`04 67 rr`	`04 77 rr`

TFR

Transfer Register Content to Another Register

TFR

Operation: See table.

Description: Transfers the content of a source register to a destination register specified in the instruction. The order in which transfers between 8-bit and 16-bit registers are specified affects the high byte of the 16-bit registers differently. Cases involving TMP2 and TMP3 are reserved for Motorola use, so some assemblers may not permit their use. It is possible to generate these cases by using DC.B or DC.W assembler directives.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Or:

S	X	H	I	N	Z	V	C
Δ	⇓	Δ	Δ	Δ	Δ	Δ	Δ

None affected, unless the CCR is the destination register. Condition codes take on the value of the corresponding source bits, except that the X mask bit cannot change from 0 to 1. Software can leave the X bit set, leave it cleared, or change it from 1 to 0, but it can be set only by a reset or by recognition of an $\overline{\text{XIRQ}}$ interrupt.

Source Form	Address Mode	Object Code(1)	Access Detail HCS12	Access Detail M68HC12
TFR *abcdxys,abcdxys*	INH	B7 eb	P	P

1. Legal coding for eb is summarized in the following table. Columns represent the high-order source digit. Rows represent the low-order destination digit (MSB is a don't-care). Values are in hexadecimal.

	0	1	2	3	4	5	6	7
0	A ⇒ A	B ⇒ A	CCR ⇒ A	$TMP3_L$ ⇒ A	B ⇒ A	X_L ⇒ A	Y_L ⇒ A	SP_L ⇒ A
1	A ⇒ B	B ⇒ B	CCR ⇒ B	$TMP3_L$ ⇒ B	B ⇒ B	X_L ⇒ B	Y_L ⇒ B	SP_L ⇒ B
2	A ⇒ CCR	B ⇒ CCR	CCR ⇒ CCR	$TMP3_L$ ⇒ CCR	B ⇒ CCR	X_L ⇒ CCR	Y_L ⇒ CCR	SP_L ⇒ CCR
3	sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2	TMP3 ⇒ TMP2	D ⇒ TMP2	X ⇒ TMP2	Y ⇒ TMP2	SP ⇒ TMP2
4	sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D	TMP3 ⇒ D	D ⇒ D	X ⇒ D	Y ⇒ D	SP ⇒ D
5	sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X	TMP3 ⇒ X	D ⇒ X	X ⇒ X	Y ⇒ X	SP ⇒ X
6	sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y	TMP3 ⇒ Y	D ⇒ Y	X ⇒ Y	Y ⇒ Y	SP ⇒ Y
7	sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP	TMP3 ⇒ SP	D ⇒ SP	X ⇒ SP	Y ⇒ SP	SP ⇒ SP

TPA

Transfer from Condition Code Register to Accumulator A

TPA

Operation: (CCR) ⇒ A

Description: Transfers the content of the condition code register to corresponding bit positions of accumulator A. The CCR remains unchanged.

This mnemonic is implemented by the TFR CCR,A instruction. For compatibility with the M68HC11, the mnemonic TPA is translated into the TFR CCR,A instruction by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TPA *translates to...* TFR CCR,A	INH	B7 20	P	P

TRAP — Unimplemented Opcode Trap — TRAP

Operation:

(SP) – $0002 ⇒ SP; $RTN_H : RTN_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $Y_H : Y_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; $X_H : X_L ⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0002 ⇒ SP; B : A⇒ (M_{(SP)} : M_{(SP+1)})$
(SP) – $0001 ⇒ SP; CCR ⇒ (M_{(SP)})$
1 ⇒ I
(Trap Vector) ⇒ PC

Description:

Traps unimplemented opcodes. There are opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the unimplemented opcodes on page 2, an opcode trap interrupt occurs. Unimplemented opcode traps are essentially interrupts that share the $FFF8:$FFF9 interrupt vector.

TRAP uses the next address after the unimplemented opcode as a return address. It stacks the return address, index registers Y and X, accumulators B and A, and the CCR, automatically decrementing the SP before each item is stacked. The I mask bit is then set, the PC is loaded with the trap vector, and instruction execution resumes at that location. This instruction is not maskable by the I bit. Refer to **Section 7. Exception Processing** for more information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	1	–	–	–	–

I: 1; set

Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
TRAP *trapnum*	INH	$18 tn(1)	OVSPSSPSsP	OfVSPSSPSsP

1. The value tn represents an unimplemented page 2 opcode in either of the two ranges $30 to $39 or $40 to $FF.

TST — Test Memory — TST

Operation: (M) – $00

Description: Subtracts $00 from the content of memory location M and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying M.

The TST instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TST. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	0

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TST *opr16a*	EXT	F7 hh ll	rPO	r0P
TST *oprx0_xysp*	IDX	E7 xb	rPf	rfP
TST *oprx9,xysp*	IDX1	E7 xb ff	rPO	rPO
TST *oprx16,xysp*	IDX2	E7 xb ee ff	frPP	frPP
TST [D,*xysp*]	[D,IDX]	E7 xb	fIfrPf	fIfrfP
TST [*oprx16,xysp*]	[IDX2]	E7 xb ee ff	fIPrPf	fIPrfP

TSTA Test A TSTA

Operation: (A) – $00

Description: Subtracts $00 from the content of accumulator A and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying A.

The TSTA instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TSTA. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	0

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TSTA	INH	97	0	0

TSTB Test B TSTB

Operation: (B) – $00

Description: Subtracts $00 from the content of accumulator B and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying B.

The TSTB instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TSTB. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	0

N: Set if MSB of result is set; cleared otherwise

Z: Set if result is $00; cleared otherwise

V: 0; cleared

C: 0; cleared

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TSTB	INH	D7	0	0

TSX

Transfer from Stack Pointer to Index Register X

TSX

Operation: (SP) ⇒ X

Description: This is an alternate mnemonic to transfer the stack pointer value to index register X. The content of the SP remains unchanged. After a TSX instruction, X points at the last value that was stored on the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TSX *translates to...* TFR SP,X	INH	B7 75	P	P

TSY

Transfer from Stack Pointer to Index Register Y

TSY

Operation: (SP) ⇒ Y

Description: This is an alternate mnemonic to transfer the stack pointer value to index register Y. The content of the SP remains unchanged. After a TSY instruction, Y points at the last value that was stored on the stack.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TSY *translates to...* TFR SP,Y	INH	B7 76	P	P

TXS

Transfer from Index Register X to Stack Pointer

TXS

Operation: (X) ⇒ SP

Description: This is an alternate mnemonic to transfer index register X value to the stack pointer. The content of X is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TXS *translates to...* TFR X,SP	INH	B7 57	P	P

TYS

Transfer from Index Register Y to Stack Pointer

TYS

Operation: (Y) ⇒ SP

Description: This is an alternate mnemonic to transfer index register Y value to the stack pointer. The content of Y is unchanged.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
TYS *translates to...* TFR Y,SP	INH	B7 67	P	P

WAI — Wait for Interrupt — WAI

Operation:

$(SP) - \$0002 \Rightarrow SP; RTN_H : RTN_L \Rightarrow (M_{(SP)} : M_{(SP+1)})$
$(SP) - \$0002 \Rightarrow SP; Y_H : Y_L \Rightarrow (M_{(SP)} : M_{(SP+1)})$
$(SP) - \$0002 \Rightarrow SP; X_H : X_L \Rightarrow (M_{(SP)} : M_{(SP+1)})$
$(SP) - \$0002 \Rightarrow SP; B : A \Rightarrow (M_{(SP)} : M_{(SP+1)})$
$(SP) - \$0001 \Rightarrow SP; CCR \Rightarrow (M_{(SP)})$
Stop CPU Clocks

Description:

Puts the CPU into a wait state. Uses the address of the instruction following WAI as a return address. Stacks the return address, index registers Y and X, accumulators B and A, and the CCR, decrementing the SP before each item is stacked.

The CPU then enters a wait state for an integer number of bus clock cycles. During the wait state, CPU clocks are stopped, but other MCU clocks can continue to run. The CPU leaves the wait state when it senses an interrupt that has not been masked.

Upon leaving the wait state, the CPU sets the appropriate interrupt mask bit(s), fetches the vector corresponding to the interrupt sensed, and instruction execution continues at the location the vector points to.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
WAI (before interrupt)	INH	3E	`OSSSSsf`	`OSSSfSsf`
WAI (when interrupt comes)			`fVfPPP`	`VfPPP`

Although the WAI instruction itself does not alter the condition codes, the interrupt that causes the CPU to resume processing also causes the I mask bit (and the X mask bit, if the interrupt was $\overline{XIRQ}$) to be set as the interrupt vector is fetched.

WAV — Weighted Average — WAV

Operation: Do until B = 0, leave SOP in Y : D, SOW in X

Partial Product = (M pointed to by X) × (M pointed to by Y)
Sum-of-Products (24-bit SOP) = Previous SOP + Partial Product
Sum-of-Weights (16-bit SOW) = Previous SOW + (M pointed to by Y)
(X) + $0001 ⇒ X; (Y) + $0001 ⇒ Y
(B) – $01 ⇒ B

Description: Performs weighted average calculations on values stored in memory. Uses indexed (X) addressing mode to reference one source operand list, and indexed (Y) addressing mode to reference a second source operand list. Accumulator B is used as a counter to control the number of elements to be included in the weighted average.

For each pair of data points, a 24-bit sum of products (SOP) and a 16-bit sum of weights (SOW) is accumulated in temporary registers. When B reaches zero (no more data pairs), the SOP is placed in Y : D. The SOW is placed in X. To arrive at the final weighted average, divide the content of Y : D by X by executing an EDIV after the WAV.

This instruction can be interrupted. If an interrupt occurs during WAV execution, the intermediate results (six bytes) are stacked in the order $SOW_{[15:0]}$, $SOP_{[15:0]}$, $00:$SOP_{[23:16]}$ before the interrupt is processed. The wavr pseudo-instruction is used to resume execution after an interrupt. The mechanism is re-entrant. New WAV instructions can be started and interrupted while a previous WAV instruction is interrupted.

This instruction is often used in fuzzy logic rule evaluation. Refer to **Section 9. Fuzzy Logic Support** for more information.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	?	–	?	1	?	?

Z: 1; set
H, N, V and C may be altered by this instruction

Source Form	Address Mode	Object Code	Access Detail(1) HCS12	Access Detail(1) M68HC12
WAV	Special	18 3C	`Of(frr,ffff)O` (replace comma if interrupted) `SSS + UUUrr`	`Off(frr,fffff)O` (replace comma if interrupted) `SSSf + UUUrr`

1. The replace comma sequence in parentheses represents the loop for one iteration of SOP and SOW accumulation.

XGDX

Exchange Double Accumulator and Index Register X

XGDX

Operation: (D) ⇔ (X)

Description: Exchanges the content of double accumulator D and the content of index register X. For compatibility with the M68HC11, the XGDX instruction is translated into an EXG D,X instruction by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
XGDX *translates to...* EXG D,X	INH	B7 C5	P	P

XGDY — Exchange Double Accumulator and Index Register Y — XGDY

Operation: (D) ⇔ (Y)

Description: Exchanges the content of double accumulator D and the content of index register Y. For compatibility with the M68HC11, the XGDY instruction is translated into an EXG D,Y instruction by the assembler.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

Source Form	Address Mode	Object Code	Access Detail	
			HCS12	M68HC12
XGDY *translates to...* EXG D,Y	INH	B7 C6	P	P

Instruction Glossary

Section 7. Exception Processing

7.1 Introduction

Exceptions are events that require processing outside the normal flow of instruction execution. This section describes exceptions and the way each is handled.

7.2 Types of Exceptions

Central processor unit (CPU12) exceptions include:

- Resets
  - Power-on reset and $\overline{\text{RESET}}$ pin
  - Clock monitor reset
  - COP watchdog reset
- An unimplemented opcode trap
- A software interrupt instruction (SWI)
- Non-maskable (X-bit) interrupts
- Non-maskable (I-bit) interrupts

Each exception has an associated 16-bit vector, which points to the memory location where the routine that handles the exception is located. As shown in **Table 7-1**, vectors are stored in the upper bytes of the standard 64-Kbyte address map.

The six highest vector addresses are used for resets and unmaskable interrupt sources. The remaining vectors are used for maskable interrupts. All vectors must be programmed to point to the address of the appropriate service routine.

Table 7-1. CPU12 Exception Vector Map(1)

Vector Address	Source
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Clock monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI)
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ signal
$FF00–$FFF1	Device-specific interrupt sources (HCS12)
$FFC0–$FFF1	Device-specific interrupt sources (M68HC12)

1. See Device User Guide and Interrupt Block Guide for further details

The CPU12 can handle up to 128 exception vectors, but the number actually used varies from device to device, and some vectors are reserved for Motorola use. Refer to Device User Guide for more information.

Exceptions can be classified by the effect of the X and I interrupt mask bits on recognition of a pending request.

- Resets, the unimplemented opcode trap, and the SWI instruction are not affected by the X and I mask bits.
- Interrupt service requests from the $\overline{\text{XIRQ}}$ pin are inhibited when X = 1, but are not affected by the I bit.
- All other interrupts are inhibited when I = 1.

7.3 Exception Priority

A hardware priority hierarchy determines which reset or interrupt is serviced first when simultaneous requests are made. Six sources are not maskable. The remaining sources are maskable, and the device integration module typically can change the relative priorities of maskable interrupts. Refer to **7.5 Interrupts** for more detail concerning interrupt priority and servicing.

The priorities of the unmaskable sources are:

1. $\overline{\text{RESET}}$ pin or power-on reset (POR)
2. Clock monitor reset
3. Computer operating properly (COP) watchdog reset
4. Non-maskable interrupt request ($\overline{\text{XIRQ}}$) signal
5. Unimplemented opcode trap
6. Software interrupt instruction (SWI)

External reset and POR share the highest exception-processing priority, followed by clock monitor reset, and then the on-chip watchdog reset.

The $\overline{\text{XIRQ}}$ interrupt is pseudo-non-maskable. After reset, the X bit in the CCR is set, which inhibits all interrupt service requests from the $\overline{\text{XIRQ}}$ pin until the X bit is cleared. The X bit can be cleared by a program instruction, but program instructions cannot change X from 0 to 1. Once the X bit is cleared, interrupt service requests made via the $\overline{\text{XIRQ}}$ pin become non-maskable.

The unimplemented page 2 opcode trap (TRAP) and the SWI are special cases. In one sense, these two exceptions have very low priority, because any enabled interrupt source that is pending prior to the time exception processing begins will take precedence. However, once the CPU begins processing a TRAP or SWI, neither can be interrupted. Also, since these are mutually exclusive instructions, they have no relative priority.

All remaining interrupts are subject to masking via the I bit in the CCR. Most HCS12 microcontroller units (MCU) have an external $\overline{\text{IRQ}}$ pin, which is assigned the highest I-bit interrupt priority and an internal periodic real-time interrupt generator, which has the next highest priority. The other maskable sources have default priorities that follow the address order of the interrupt vectors — the higher the address, the higher the priority of the interrupt. Other maskable interrupts are associated with on-chip peripherals such as timers or serial ports. Typically, logic in the device integration module can give one I-masked source priority over other I-masked sources. Refer to the documentation for the specific HCS12 derivative for more information.

7.4 Resets

M68HC12 devices perform resets with a combination of hardware and software. Integration module circuitry determines the type of reset that has occurred, performs basic system configuration, then passes control to the CPU12. The CPU fetches a vector determined by the type of reset that has occurred, jumps to the address pointed to by the vector, and begins to execute code at that address.

The are four possible sources of reset are:

- Power-on reset (POR)
- External reset ($\overline{\text{RESET}}$ pin)
- COP reset
- Clock monitor reset

Power-on reset (POR) and external reset share the same reset vector. The computer operating properly (COP) reset and the clock monitor reset each have a vector.

7.4.1 Power-On Reset

The HCS12 incorporate circuitry to detect a positive transition in the V_{DD} supply and initialize the device during cold starts, generally by asserting the reset signal internally. The signal is typically released after a delay that allows the device clock generator to stabilize.

7.4.2 External Reset

The MCU distinguishes between internal and external resets by sensing how quickly the signal on the $\overline{\text{RESET}}$ pin rises to logic level 1 after it has been asserted. When the MCU senses any of the four reset conditions, internal circuitry drives the $\overline{\text{RESET}}$ signal low for N clock cycles, then releases. M clock cycles later, the MCU samples the state of the signal applied to the $\overline{\text{RESET}}$ pin. If the signal is still low, an external reset has occurred. If the signal is high, reset is assumed to have been initiated internally by either the COP system or the clock monitor.

7.4.3 COP Reset

The MCU includes a computer operating properly (COP) system to help protect against software failures. When the COP is enabled, software must write a particular code sequence to a specific address to keep a watchdog timer from timing out. If software fails to execute the sequence properly, a reset occurs.

7.4.4 Clock Monitor Reset

The clock monitor circuit uses an internal RC circuit to determine whether clock frequency is above a predetermined limit. If clock frequency falls below the limit when the clock monitor is enabled, a reset occurs.

7.5 Interrupts

Each HCS12 device can recognize a number of interrupt sources. Each source has a vector in the vector table. The $\overline{\text{XIRQ}}$ signal, the unimplemented opcode trap, and the SWI instruction are non-maskable, and have a fixed priority. The remaining interrupt sources can be masked by the I bit. In most devices, the external interrupt request pin is assigned the highest maskable interrupt priority, and the internal periodic real-time interrupt generator has the next highest priority. Other maskable interrupts are associated with on-chip peripherals such as timers or serial ports. These maskable sources have default priorities that follow the address order of the interrupt vectors. The higher the vector address, the higher the priority of the interrupt. Typically, a device integration module incorporates logic that can give any one maskable source priority over other maskable sources.

7.5.1 Non-Maskable Interrupt Request ($\overline{\text{XIRQ}}$)

The $\overline{\text{XIRQ}}$ input is an updated version of the non-maskable interrupt ($\overline{\text{NMI}}$) input of earlier MCUs. The $\overline{\text{XIRQ}}$ function is disabled during system reset and upon entering the interrupt service routine for an $\overline{\text{XIRQ}}$ interrupt.

During reset, both the I bit and the X bit in the CCR are set. This disables maskable interrupts and interrupt service requests made by asserting the $\overline{\text{XIRQ}}$ signal. After minimum system initialization, software can clear the X bit using an instruction such as ANDCC #$BF. Software cannot set the X bit from 0 to 1 once it has been cleared, and interrupt requests made via the $\overline{\text{XIRQ}}$ pin become non-maskable. When a non-maskable interrupt is recognized, both the X and I bits are set after context is saved. The X bit is not affected by maskable interrupts. Execution of an return-from-interrupt (RTI) instruction at the end of the interrupt service routine normally restores the X and I bits to the pre-interrupt request state.

7.5.2 Maskable Interrupts

Maskable interrupt sources include on-chip peripheral systems and external interrupt service requests. Interrupts from these sources are recognized when the global interrupt mask bit (I) in the CCR is cleared. The default state of the I bit out of reset is 1, but it can be written at any time.

The interrupt module manages maskable interrupt priorities. Typically, an on-chip interrupt source is subject to masking by associated bits in control registers in addition to global masking by the I bit in the CCR. Sources generally must be enabled by writing one or more bits in associated control registers. There may be other interrupt-related control bits and flags, and there may be specific register read-write sequences associated with interrupt service. Refer to individual on-chip peripheral descriptions for details.

7.5.3 Interrupt Recognition

Once enabled, an interrupt request can be recognized at any time after the I mask bit is cleared. When an interrupt service request is recognized, the CPU responds at the completion of the instruction being executed. Interrupt latency varies according to the number of cycles required to complete the current instruction. Because the fuzzy logic rule evaluation (REV), fuzzy logic rule evaluation weighted (REVW), and weighted average (WAV) instructions can take many cycles to complete, they are designed so that they can be interrupted. Instruction execution resumes when interrupt execution is complete. When the CPU begins to

service an interrupt, the instruction queue is refilled, a return address is calculated, and then the return address and the contents of the CPU registers are stacked as shown in **Table 7-2**.

Table 7-2. Stacking Order on Entry to Interrupts

Memory Location	CPU Registers
SP + 7	RTN_H : RTN_L
SP + 5	Y_H : Y_L
SP + 3	X_H : X_L
SP + 1	B : A
SP	CCR

After the CCR is stacked, the I bit (and the X bit, if an $\overline{XIRQ}$ interrupt service request caused the interrupt) is set to prevent other interrupts from disrupting the interrupt service routine. Execution continues at the address pointed to by the vector for the highest-priority interrupt that was pending at the beginning of the interrupt sequence. At the end of the interrupt service routine, an RTI instruction restores context from the stacked registers, and normal program execution resumes.

7.5.4 External Interrupts

External interrupt service requests are made by asserting an active-low signal connected to the $\overline{IRQ}$ pin. Typically, control bits affect how the signal is detected and recognized.

The I bit serves as the $\overline{IRQ}$ interrupt enable flag. When an $\overline{IRQ}$ interrupt is recognized, the I bit is set to inhibit interrupts during the interrupt service routine. Before other maskable interrupt requests can be recognized, the I bit must be cleared. This is generally done by an RTI instruction at the end of the service routine.

7.5.5 Return-from-Interrupt Instruction (RTI)

RTI is used to terminate interrupt service routines. RTI is an 8-cycle instruction when no other interrupt is pending and 11 cycles (10 cycles in M68HC12) when another interrupt is pending. In either case, the first five cycles are used to restore (pull) the CCR, B:A, X, Y, and the return

address from the stack. If no other interrupt is pending at this point, three program words are fetched to refill the instruction queue from the area of the return address and processing proceeds from there.

If another interrupt is pending after registers are restored, a new vector is fetched, and the stack pointer is adjusted to point at the CCR value that was just recovered (SP = SP – 9). This makes it appear that the registers have been stacked again. After the SP is adjusted, three program words are fetched to refill the instruction queue, starting at the address the vector points to. Processing then continues with execution of the instruction that is now at the head of the queue.

7.6 Unimplemented Opcode Trap

The CPU12 has opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the 202 unused opcodes on page 2, an unimplemented opcode trap occurs. The 202 unimplemented opcodes are essentially interrupts that share a common interrupt vector, $FFF8:$FFF9.

The CPU12 uses the next address after an unimplemented page 2 opcode as a return address. This differs from the M68HC11 illegal opcode interrupt, which uses the address of an illegal opcode as the return address. In the CPU12, the stacked return address can be used to calculate the address of the unimplemented opcode for software-controlled traps.

7.7 Software Interrupt Instruction (SWI)

Execution of the SWI instruction causes an interrupt without an interrupt service request. SWI is not inhibited by the global mask bits in the CCR, and execution of SWI sets the I mask bit. Once an SWI interrupt begins, maskable interrupts are inhibited until the I bit in the CCR is cleared. This typically occurs when an RTI instruction at the end of the SWI service routine restores context.

7.8 Exception Processing Flow

The first cycle in the exception processing flow for all CPU12 exceptions is the same, regardless of the source of the exception. Between the first and second cycles of execution, the CPU chooses one of three alternative paths. The first path is for resets, the second path is for pending X or I interrupts, and the third path is used for software interrupts (SWI) and trapping unimplemented opcodes. The last two paths are virtually identical, differing only in the details of calculating the return address. Refer to **Figure 7-1** for the following discussion.

7.8.1 Vector Fetch

The first cycle of all exception processing, regardless of the cause, is a vector fetch. The vector points to the address where exception processing will continue. Exception vectors are stored in a table located at the top of the memory map ($FFxx). The CPU cannot use the fetched vector until the third cycle of the exception processing sequence.

During the vector fetch cycle, the CPU issues a signal that tells the interrupt module to drive the vector address of the highest priority, pending exception onto the system address bus (the CPU does not provide this address).

After the vector fetch, the CPU selects one of the three alternate execution paths, depending upon the cause of the exception.

7.8.2 Reset Exception Processing

If reset caused the exception, processing continues to cycle 2.0. This cycle sets the S, X, and I bits in the CCR. Cycles 3.0 through 5.0 are program word fetches that refill the instruction queue. Fetches start at the address pointed to by the reset vector. When the fetches are completed, exception processing ends, and the CPU starts executing the instruction at the head of the instruction queue.

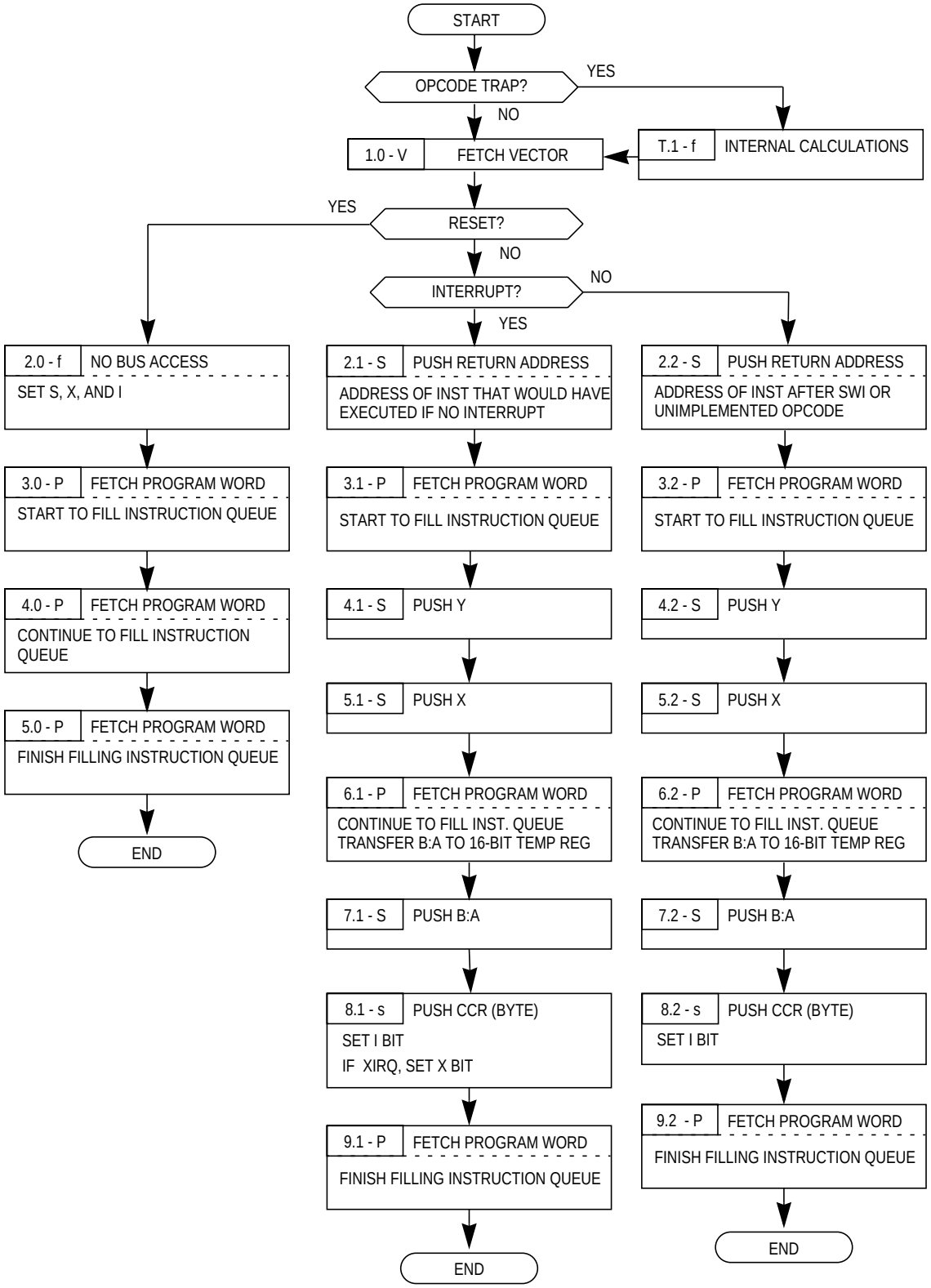


Figure 7-1. Exception Processing Flow Diagram

7.8.3 Interrupt and Unimplemented Opcode Trap Exception Processing

If an exception was not caused by a reset, a return address is calculated.

- Cycles 2.1and 2.2 are both S cycles (stack a 16-bit word), but the CPU12 performs different return address calculations for each type of exception.
  - When an X- or I-related interrupt causes the exception, the return address points to the next instruction that would have been executed had processing not been interrupted.
  - When an exception is caused by an SWI opcode or by an unimplemented opcode (see **7.6 Unimplemented Opcode Trap**), the return address points to the next address after the opcode.
- Once calculated, the return address is pushed onto the stack.
- Cycles 3.1 through 9.1 are identical to cycles 3.2 through 9.2 for the rest of the sequence, except for optional setting of the X mask bit performed in cycle 8.1 (see below).
- Cycle 3.1/3.2 is the first of three program word fetches that refill the instruction queue.
- Cycle 4.1/4.2 pushes Y onto the stack.
- Cycle 5.1/5.2 pushes X onto the stack.
- Cycle 6.1/6.2 is the second of three program word fetches that refill the instruction queue. During this cycle, the contents of the A and B accumulators are concatenated into a 16-bit word in the order B:A. This makes register order in the stack frame the same as that of the M68HC11, M6801, and the M6800.
- Cycle 7.1/7.2 pushes the 16-bit word containing B:A onto the stack.
- Cycle 8.1/8.2 pushes the 8-bit CCR onto the stack, then updates the mask bits.
  - When an $\overline{\text{XIRQ}}$ interrupt causes an exception, both X and I are set, which inhibits further interrupts during exception processing.
  - When any other interrupt causes an exception, the I bit is set, but the X bit is not changed.

- Cycle 9.1/9.2 is the third of three program word fetches that refill the instruction queue. It is the last cycle of exception processing. After this cycle the CPU starts executing the first cycle of the instruction at the head of the instruction queue.

Section 8. Instruction Queue

8.1 Introduction

This section describes development and debug support features related to the central processor unit (CPU12). Topics include:

- Single-wire background debug interface
- Hardware breakpoint system
- Instruction queue operation and reconstruction
- Instruction tagging

1 = Valid Data

TRACE — Trace Flag

Indicates when tracing is enabled. Firmware in the BDM ROM sets TRACE in response to a TRACE1 command and TRACE is cleared upon completion of the TRACE1 command. Do not attempt to write TRACE directly with WRITE_BD_BYTE commands.

0 = Tracing not enabled
1 = TRACE1 command in progress

8.2 External Reconstruction of the Queue

The CPU12 uses an instruction queue to buffer program information and increase instruction throughput. The HCS12 implements the queue somewhat differently from the original M68HC12. The HCS12 queue consists of three 16-bit stages while the M68HC12 queue consists of two 16-bit stages, plus a 16-bit holding latch. Program information is always fetched in aligned 16-bit words. At least three bytes of program information are available to the CPU when instruction execution begins. The holding latch in the M68HC12 is used when a word of program information arrives before the queue can advance.

Because of the queue, program information is fetched a few cycles before it is used by the CPU. Internally, the microcontroller unit (MCU) only needs to buffer the fetched data. But, in order to monitor cycle-by-cycle CPU activity externally, it is necessary to capture data and address to discern what is happening in the instruction queue.

Two external pins, IPIPE1 and IPIPE0, provide time-multiplexed information about data movement in the queue and instruction execution. The instruction queue and cycle-by-cycle activity can be reconstructed in real time or from trace history captured by a logic analyzer. However, neither scheme can be used to stop the CPU12 at a specific instruction. By the time an operation is visible outside the MCU, the instruction has already begun execution. A separate instruction tagging mechanism is provided for this purpose. A tag follows the information in the queue as the queue is advanced. During debugging, the CPU enters active background debug mode when a tagged instruction reaches the head of the queue, rather than executing the tagged instruction. For more information about tagging, refer to **8.6 Instruction Tagging**.

8.3 Instruction Queue Status Signals

The IPIPE1 and IPIPE0 signals carry time-multiplexed information about data movement and instruction execution during normal CPU operation. The signals are available on two multifunctional device pins. During reset, the pins are used as mode-select input signals MODA and MODB.

To reconstruct the queue, the information carried by the status signals must be captured externally. In general, data movement and execution start information are considered to be distinct 2-bit values, with the low-order bit on IPIPE0 and the high-order bit on IPIPE1.

8.3.1 HCS12 Timing Detail

In the HCS12, data-movement information is available when E clock is high or on falling edges of the E clock; execution-start information is available when E clock is low or on rising edges of the E clock, as shown in **Figure 8-1**. Data-movement information refers to data on the bus. Execution-start information refers to the bus cycle that starts with that E-low time and continues through the following E-high time. **Table 8-1** summarizes the information encoded on the IPIPE1 and IPIPE0 pins.

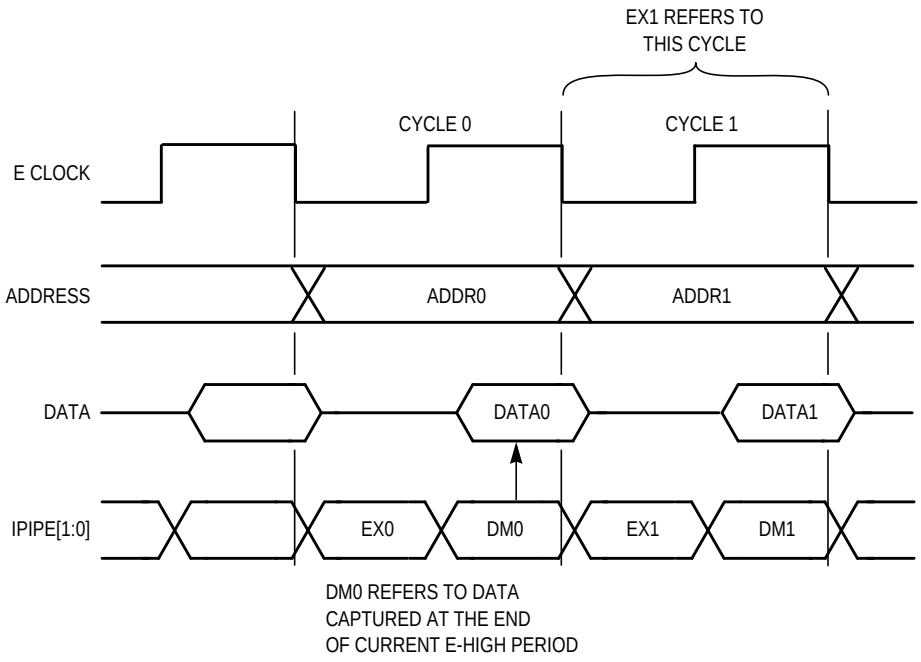


Figure 8-1. Queue Status Signal Timing (HCS12)

8.3.2 M68HC12 Timing Detail

In the M68HC12, data movement information is available on rising edges of the E clock; execution start information is available on falling edges of the E clock, as shown in **Figure 8-2**. Data movement information refers to data on the bus at the previous falling edge of E. Execution information refers to the bus cycle from the current falling edge to the next falling edge of E. **Table 8-1** summarizes the information encoded on the IPIPE1 and IPIPE0 pins.

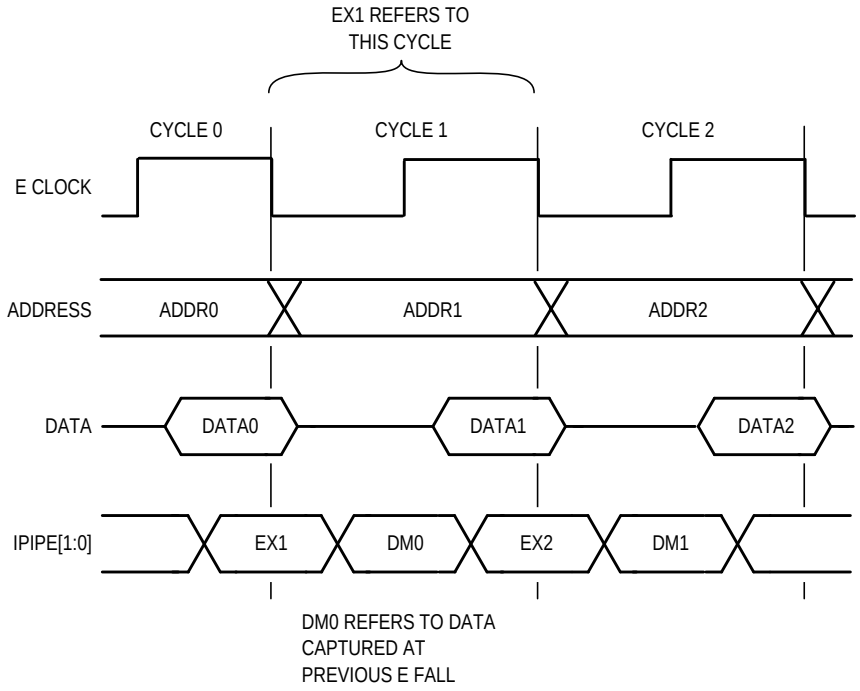


Figure 8-2. Queue Status Signal Timing (M68HC12)

Table 8-1. IPIPE1 and IPIPE0 Decoding (HCS12 and M68HC12)

	Mnemonic	Meaning
Data Movement	**Capture at E Fall in HCS12 (E Rise in M68HC12)**	
0:0	—	No movement
0:1	LAT[1]	Latch data from bus
1:0	ALD	Advance queue and load from bus
1:1	ALL[1]	Advance queue and load from latch
Execution Start	**Capture at E Rise in HCS12 (E Fall in M68HC12)**	
0:0	—	No start
0:1	INT	Start interrupt sequence
1:0	SEV	Start even instruction
1:1	SOD	Start odd instruction

1. The HCS12 implementation does not include a holding latch, so these data movement codes are used only in the original M68HC12.

8.3.3 Null (Code 0:0)

The 0:0 data movement state indicates that there was no data movement in the instruction queue; the 0:0 execution start state indicates continuation of an instruction or interrupt sequence (no new instruction or interrupt start).

8.3.4 LAT — Latch Data from Bus (Code 0:1)

This code is not used in the HCS12. In the M68HC12, fetched program information has arrived, but the queue is not ready to advance. The information is latched into a buffer. Later, when the queue does advance, stage 1 is refilled from the buffer or from the data bus if the buffer is empty. In some instruction sequences, there can be several latch cycles before the queue advances. In these cases, the buffer is filled on the first latch event and additional latch requests are ignored.

8.3.5 ALD — Advance and Load from Data Bus (Code 1:0)

The instruction queue is advanced by one word and stage one is refilled with a word of program information from the data bus. The CPU requested the information two bus cycles earlier but, due to access delays, the information was not available until the E cycle referred to by the ALD code.

8.3.6 ALL — Advance and Load from Latch (Code 1:1)

This code is not used in the HCS12. In the M68HC12, the 2-stage instruction queue is advanced by one word and stage one is refilled with a word of program information from the buffer. The information was latched from the data bus at the falling edge of a previous E cycle because the instruction queue was not ready to advance when it arrived.

8.3.7 INT — Interrupt Sequence Start (Code 0:1)

The E cycle associated with this code is the first cycle of an interrupt sequence. Normally, this cycle is a read of the interrupt vector. However, in systems that have interrupt vectors in external memory and an 8-bit data bus, this cycle reads the upper byte of the 16-bit interrupt vector.

8.3.8 SEV — Start Instruction on Even Address (Code 1:0)

The E cycle associated with this code is the first cycle of the instruction in the even (high order) half of the word at the head of the instruction queue. The queue treats the $18 prebyte for instructions on page 2 of the opcode map as a special 1-byte, 1-cycle instruction, except that interrupts are not recognized at the boundary between the prebyte and the rest of the instruction.

8.3.9 SOD — Start Instruction on Odd Address (Code 1:1)

The E cycle associated with this code is the first cycle of the instruction in the odd (low order) half of the word at the head of the instruction queue. The queue treats the $18 prebyte for instructions on page 2 of the opcode map as a special 1-byte, 1-cycle instruction, except that interrupts are not recognized at the boundary between the prebyte and the rest of the instruction.

8.4 Queue Reconstruction (for HCS12)

The raw signals required for queue reconstruction are the address bus (ADDR), the data bus (DATA), the system clock (E), and the queue status signals (IPIPE1 and IPIPE2). An ALD data movement implies a read; therefore, it is not necessary to capture the R/$\overline{W}$ signal. An E clock cycle begins at a falling edge of E. Addresses and execution status must be captured at the rising E edge in the middle of the cycle. Data and data-movement status must be captured at the falling edge of E at the end of the cycle. These captures can then be organized into records with one record per E clock cycle.

Implementation details depend on the type of MCU and the mode of operation. For instance, the data bus can be eight bits or 16 bits wide, and nonmultiplexed or multiplexed. In all cases, the externally reconstructed queue must use 16-bit words. Demultiplexing and assembly of 8-bit data into 16-bit words is done before program information enters the real queue, so it must also be done for the external reconstruction.

An example:

Systems with an 8-bit data bus and a program stored in external memory require two cycles for each program word fetch. MCU bus-control logic freezes the CPU clocks long enough to do two 8-bit accesses rather than a single 16-bit access, so the CPU sees only 16-bit words of program information. To recover the 16-bit program words externally, latch the data bus state at the falling edge of E when ADDR0 = 0, and gate the outputs of the latch onto DATA[15:8] when an ALD cycle occurs. Since the 8-bit data bus is connected to DATA[7:0], the 16-bit word on the data lines corresponds to the ALD during the last half of the second 8-bit fetch, which is always to an odd address. IPIPE[1:0] status signals indicate 0:0 for the second half of the E cycle corresponding to the first 8-bit fetch.

Some MCUs have address lines to support memory expansion beyond the standard 64-Kbyte address space. When memory expansion is used, expanded addresses must also be captured and maintained.

8.4.1 Queue Reconstruction Registers (for HCS12)

Queue reconstruction requires the following registers, which can be implemented as software variables when previously captured trace data is used, or as hardware latches in real time.

8.4.1.1 fetch_add Register

This register buffers the fetch address.

8.4.1.2 st1_add, st1_dat Registers

These registers contain address and data for the first stage of the reconstructed instruction queue.

8.4.1.3 st2_add, st2_dat Registers

These registers contain address and data for the middle stage of the reconstructed instruction queue.

8.4.1.4 st3_add, st3_dat Registers

These registers contain address and data for the final stage of the reconstructed instruction queue. When the IPIPE[1:0] signals indicate the execution status, the address and opcode can be found in these registers.

8.4.2 Reconstruction Algorithm (for HCS12)

This section describes how to use IPIPE[1:0] signals and queue reconstruction registers to reconstruct the queue.

Typically, the first few cycles of raw capture data are not useful because it takes several cycles before an instruction propagates to the head of the queue. During these first raw cycles, the only meaningful information available is data movement signals. Information on the external address and data buses during this setup time is still captured and propagated through the reconstructed queue, but the information reflects the actions of instructions that were fetched before data collection started.

In the special case of a reset, there is a five-cycle sequence (VfPPP) during which the reset vector is fetched and the instruction queue is filled, before execution of the first instruction begins. Due to the timing of the switchover of the IPIPE[1:0] pins from their alternate function as mode-select inputs, the status information on these two pins may be erroneous during the first cycle or two after the release of reset. This is not a problem because the status is correct in time for queue reconstruction logic to correctly replicate the queue.

On an advance-and-load-from-data-bus (ALD) cycle, the information in the instruction queue must advance by one stage. Whatever was in stage three of the queue simply disappears. The previous contents of stage two go to stage three, the previous contents of stage one go to stage two, and the contents of fetch_add and data from the current cycle go to stage one.

Figure 8-3 shows the reset sequence and illustrates the relationship between instruction cycle codes (VfPPP) and pipe status signals. One cycle of the data bus is shown to indicate the relationship between the ALD data movement code and the data value it refers to. The SEV execution start code indicates that the reset vector pointed to an even address in this example.

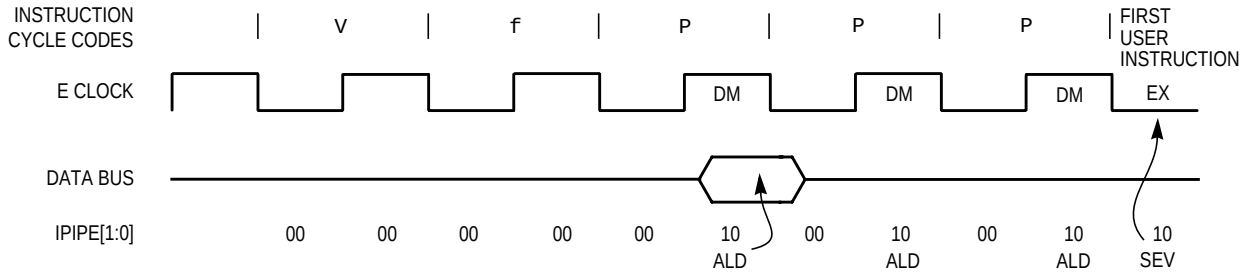


Figure 8-3. Reset Sequence for HCS12

8.5 Queue Reconstruction (for M68HC12)

The raw signals required for queue reconstruction are the address bus (ADDR), the data bus (DATA), the system clock (E), and the queue status signals (IPIPE1 and IPIPE0). An E-clock cycle begins after an E fall. Addresses and data movement status must be captured at the E rise in the middle of the cycle. Data and execution start status must be captured at the E fall at the end of the cycle. These captures can then be organized into records with one record per E clock cycle.

Implementation details depend upon the type of device and the mode of operation. For instance, the data bus can be eight bits or 16 bits wide, and non-multiplexed or multiplexed. In all cases, the externally reconstructed queue must use 16-bit words. Demultiplexing and assembly of 8-bit data into 16-bit words is done before program information enters the real queue, so it must also be done for the external reconstruction.

An example:

Systems with an 8-bit data bus and a program stored in external memory require two cycles for each program word fetch. MCU bus control logic freezes the CPU clocks long enough to do two 8-bit accesses rather than a single 16-bit access, so the CPU sees only 16-bit words of program information. To recover the 16-bit program words externally, latch the data bus state at the falling edge of E when ADDR0 = 0, and gate the outputs of the latch onto DATA[15:8] when a LAT or ALD cycle occurs. Since the 8-bit data bus is connected to DATA[7:0], the 16-bit word on the data lines corresponds to the ALD or LAT status indication at the E rise after the second 8-bit fetch,

which is always to an odd address. IPIPE1 and IPIPE0 status signals indicate 0:0 at the beginning (E fall) and middle (E rise) of the first 8-bit fetch.

Some M68HC12 devices have address lines to support memory expansion beyond the standard 64-Kbyte address space. When memory expansion is used, expanded addresses must also be captured and maintained.

8.5.1 Queue Reconstruction Registers (for M68HC12)

Queue reconstruction requires these registers, which can be implemented as software variables when previously captured trace data is used or as hardware latches in real time.

8.5.1.1 in_add, in_dat Registers

These registers contain the address and data from the previous external bus cycle. Depending on how records are read and processed from the raw capture information, it may be possible to simply read this information from the raw capture data file when needed.

8.5.1.2 fetch_add, fetch_dat Registers

These registers buffer address and data for information that was fetched before the queue was ready to advance.

8.5.1.3 st1_add, st1_dat Registers

These registers contain address and data for the first stage of the reconstructed instruction queue.

8.5.1.4 st2_add, st2_dat Registers

These registers contain address and data for the final stage of the reconstructed instruction queue. When the IPIPE1 and IPIPE0 signals indicate that an instruction is starting to execute, the address and opcode can be found in these registers.

8.5.2 Reconstruction Algorithm (for M68HC12)

This subsection describes in detail how to use IPIPE1 and IPIPE0 signals and queue reconstruction registers to reconstruct the queue. An “is_full” flag is used to indicate when the fetch_add and fetch_dat buffer registers contain information. The use of the flag is explained more fully in subsequent paragraphs.

Typically, the first few cycles of raw capture data are not useful because it takes several cycles before an instruction propagates to the head of the queue. During these first raw cycles, the only meaningful information available are data movement signals. Information on the external address and data buses during this setup time reflects the actions of instructions that were fetched before data collection started.

In the special case of a reset, there is a 5-cycle sequence (VfPPP) during which the reset vector is fetched and the instruction queue is filled, before execution of the first instruction begins. Due to the timing of the switchover of the IPIPE1 and IPIPE0 pins from their alternate function as mode select inputs, the status information on these two pins may be erroneous during the first cycle or two after the release of reset. This is not a problem because the status is correct in time for queue reconstruction logic to correctly replicate the queue.

Before starting to reconstruct the queue, clear the is_full flag to indicate that there is no meaningful information in the fetch_add and fetch_dat buffers. Further movement of information in the instruction queue is based on the decoded status on the IPIPE1 and IPIPE0 signals at the rising edges of E.

8.5.2.1 LAT Decoding

On a latch cycle (LAT), check the is_full flag. If and only if is_full = 0, transfer the address and data from the previous bus cycle (in_add and in_dat) into the fetch_add and fetch_dat registers, respectively. Then, set the is_full flag. The usual reason for a latch request instead of an advance request is that the previous instruction ended with a single aligned byte of program information in the last stage of the instruction queue. Since the odd half of this word still holds the opcode for the next instruction, the queue cannot advance on this cycle. However, the cycle to fetch the next word of program information has already started and the data is on its way.

8.5.2.2 ALD Decoding

On an advance-and-load-from-data-bus (ALD) cycle, the information in the instruction queue must advance by one stage. Whatever was in stage 2 of the queue is simply thrown away. The previous contents of stage 1 are moved to stage 2, and the address and data from the previous cycle (in_add and in_dat) are transferred into stage 1 of the instruction queue. Finally, clear the is_full flag to indicate the buffer latch is ready for new data. Usually, there would be no useful information in the fetch buffer when an ALD cycle was encountered, but in the case of a change-of-flow, any data that was there needs to be flushed out (by clearing the is_full flag).

8.5.2.3 ALL Decoding

On an advance-and-load-from-latch (ALL) cycle, the information in the instruction queue must advance by one stage. Whatever was in stage 2 of the queue is simply thrown away. The previous contents of stage 1 are moved to stage 2, and the contents of the fetch buffer latch are transferred into stage 1 of the instruction queue. One or more cycles preceding the ALL cycle will have been a LAT cycle. After updating the instruction queue, clear the is_full flag to indicate the fetch buffer is ready for new information.

Figure 8-4 shows the reset sequence and illustrates the relationship between instruction cycle codes (VfPPP) and pipe status signals. One cycle of the data bus is shown to indicate the relationship between the ALD data movement code and the data value it refers to. The SEV execution start code indicates that the reset vector pointed to an even address in this example.

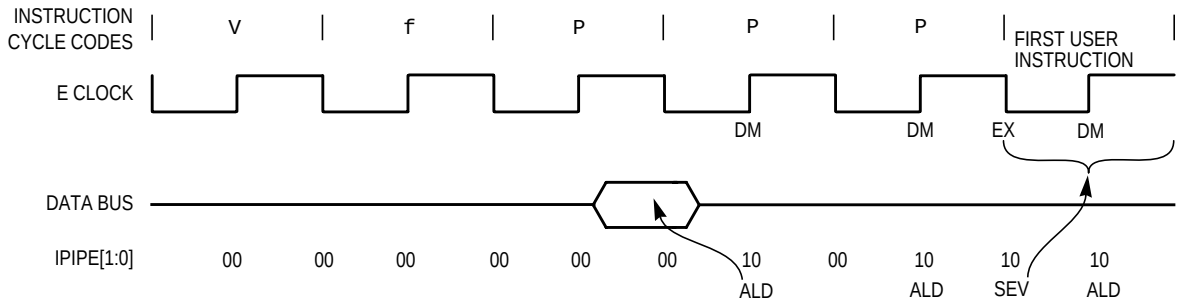


Figure 8-4. Reset Sequence for M68HC12

8.6 Instruction Tagging

The instruction queue and cycle-by-cycle CPU activity can be reconstructed in real time or from trace history that was captured by a logic analyzer. However, the reconstructed queue cannot be used to stop the CPU at a specific instruction, because execution has already begun by the time an operation is visible outside the MCU. A separate instruction tagging mechanism is provided for this purpose.

Executing the BDM TAGGO command configures two MCU pins for tagging. The $\overline{\text{TAGLO}}$ signal shares a pin with the $\overline{\text{LSTRB}}$ signal, and the $\overline{\text{TAGHI}}$ signal shares the BKGD pin. Tagging information is latched on the falling edge of ECLK, as shown in **Figure 8-5**.

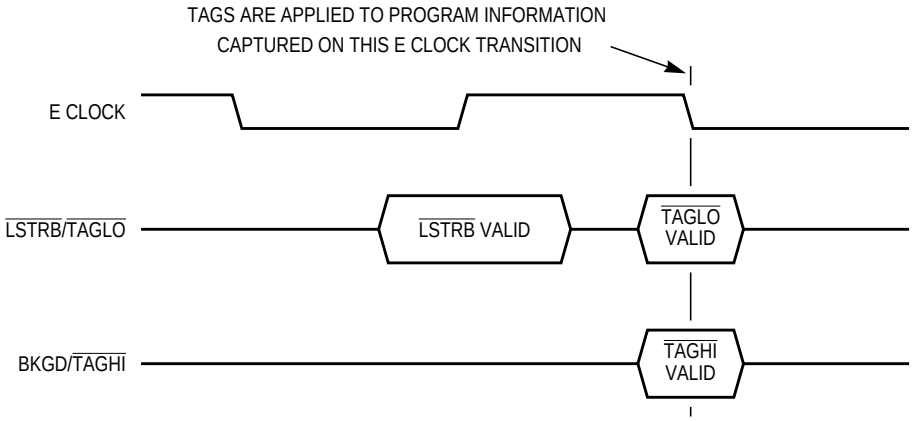


Figure 8-5. Tag Input Timing

Table 8-2 shows the functions of the two independent tagging pins. The presence of logic level 0 on either pin at the fall of ECLK tags (marks) the associated byte of program information as it is read into the instruction queue. Tagging is allowed in all modes. Tagging is disabled when BDM becomes active.

Table 8-2. Tag Pin Function

$\overline{\text{TAGHI}}$	$\overline{\text{TAGLO}}$	Tag
1	1	No tag
1	0	Low byte
0	1	High byte
0	0	Both bytes

In HCS12 and M68HC12 derivatives that have hardware breakpoint capability, the breakpoint control logic and BDM control logic use the same internal signals for instruction tagging. The CPU does not differentiate between the two kinds of tags.

The tag follows program information as it advances through the queue. When a tagged instruction reaches the head of the queue, the CPU enters active background debug mode rather than executing the instruction.

Section 9. Fuzzy Logic Support

9.1 Introduction

The instruction set of the central processor unit (CPU12) is the first instruction set to specifically address the needs of fuzzy logic. This section describes the use of fuzzy logic in control systems, discusses the CPU12 fuzzy logic instructions, and provides examples of fuzzy logic programs.

The CPU12 includes four instructions that perform specific fuzzy logic tasks. In addition, several other instructions are especially useful in fuzzy logic programs. The overall C-friendliness of the instruction set also aids development of efficient fuzzy logic programs.

This section explains the basic fuzzy logic algorithm for which the four fuzzy logic instructions are intended. Each of the fuzzy logic instructions are then explained in detail. Finally, other custom fuzzy logic algorithms are discussed, with emphasis on use of other CPU12 instructions.

The four fuzzy logic instructions are:

- MEM (determine grade of membership), which evaluates trapezoidal membership functions
- REV (fuzzy logic rule evaluation) and REVW (fuzzy logic rule evaluation weighted), which perform unweighted or weighted MIN-MAX rule evaluation
- WAV (weighted average), which performs weighted average defuzzification on singleton output membership functions.

Other instructions that are useful for custom fuzzy logic programs include:

- MINA (place smaller of two unsigned 8-bit values in accumulator A)
- EMIND (place smaller of two unsigned 16-bit values in accumulator D)
- MAXM (place larger of two unsigned 8-bit values in memory)

- EMAXM (place larger of two unsigned 16-bit values in memory)
- TBL (table lookup and interpolate)
- ETBL (extended table lookup and interpolate)
- EMACS (extended multiply and accumulate signed 16-bit by 16-bit to 32-bit)

For higher resolution fuzzy programs, the fast extended precision math instructions in the CPU12 are also beneficial. Flexible indexed addressing modes help simplify access to fuzzy logic data structures stored as lists or tabular data structures in memory.

The actual logic additions required to implement fuzzy logic support in the CPU12 are quite small, so there is no appreciable increase in cost for the typical user. A fuzzy inference kernel for the CPU12 requires one-fifth as much code space and executes almost 50 times faster than a comparable kernel implemented on a typical midrange microcontroller. By incorporating fuzzy logic support into a high-volume, general-purpose microcontroller product family, Motorola has made fuzzy logic available for a huge base of applications.

9.2 Fuzzy Logic Basics

This is an overview of basic fuzzy logic concepts. It can serve as a general introduction to the subject, but that is not the main purpose. There are a number of fuzzy logic programming strategies. This discussion concentrates on the methods implemented in the CPU12 fuzzy logic instructions. The primary goal is to provide a background for a detailed explanation of the CPU12 fuzzy logic instructions.

In general, fuzzy logic provides for set definitions that have fuzzy boundaries rather than the crisp boundaries of Aristotelian logic. These sets can overlap so that, for a specific input value, one or more sets associated with linguistic labels may be true to a degree at the same time. As the input varies from the range of one set into the range of an adjacent set, the first set becomes progressively less true while the second set becomes progressively more true.

Fuzzy logic has membership functions which emulate human concepts like “temperature is warm”; that is, conditions are perceived to have gradual boundaries. This concept seems to be a key element of the human ability to solve certain types of complex problems that have eluded traditional control methods.

Fuzzy sets provide a means of using linguistic expressions like “temperature is warm” in rules which can then be evaluated with a high degree of numerical precision and repeatability. This directly contradicts the common misperception that fuzzy logic produces approximate results — a specific set of input conditions always produces the same result, just as a conventional control system does.

A microcontroller-based fuzzy logic control system has two parts:

- A fuzzy inference kernel which is executed periodically to determine system outputs based on current system inputs
- A knowledge base which contains membership functions and rules

Figure 9-1 is a block diagram of this kind of fuzzy logic system.

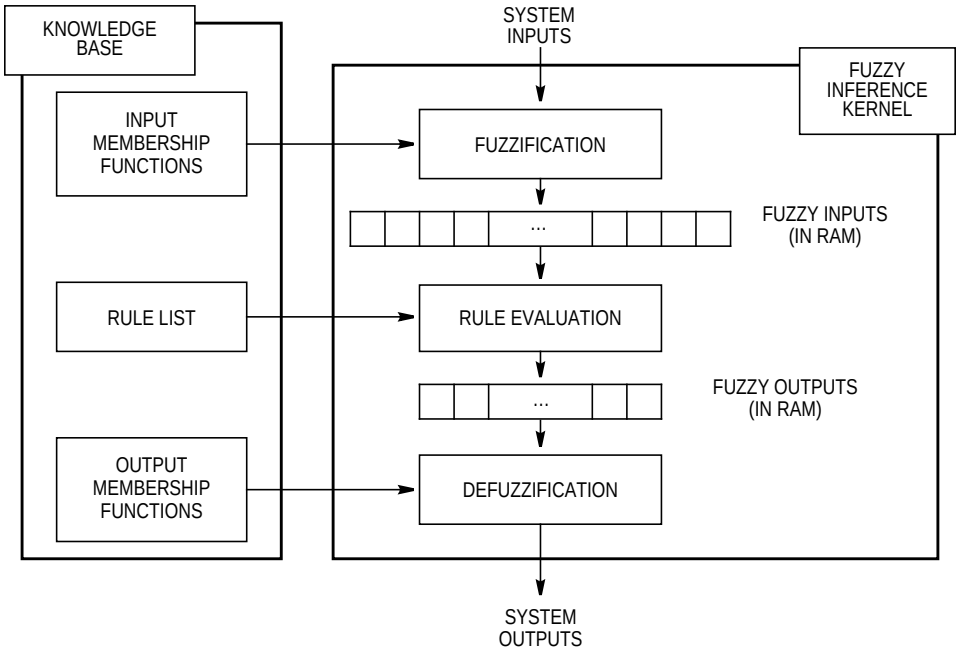


Figure 9-1. Block Diagram of a Fuzzy Logic System

The knowledge base can be developed by an application expert without any microcontroller programming experience. Membership functions are simply expressions of the expert’s understanding of the linguistic terms that describe the system to be controlled. Rules are ordinary language statements that describe the actions a human expert would take to solve the application problem.

Rules and membership functions can be reduced to relatively simple data structures (the knowledge base) stored in non-volatile memory. A fuzzy inference kernel can be written by a programmer who does not know how the application system works. The only thing the programmer needs to do with knowledge base information is store it in the memory locations used by the kernel.

One execution pass through the fuzzy inference kernel generates system output signals in response to current input conditions. The kernel is executed as often as needed to maintain control. If the kernel is executed more often than needed, processor bandwidth and power are wasted; delaying too long between passes can cause the system to get too far out of control. Choosing a periodic rate for a fuzzy control system is the same as it would be for a conventional control system.

9.2.1 Fuzzification (MEM)

During the fuzzification step, the current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y-value for the current input value on a trapezoidal membership function for each label of each system input. The MEM instruction in the CPU12 performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure.

Figure 9-2 shows a system of three input membership functions, one for each label of the system input. The x-axis of all three membership functions represents the range of possible values of the system input. The vertical line through all three membership functions represents a specific system input value. The y-axis represents degree of truth and varies from completely false ($00 or 0 percent) to completely true ($FF or 100 percent). The y-value where the vertical line intersects each of the membership functions, is the degree to which the current input value matches the associated label for this system input. For example, the expression “temperature is warm” is 25 percent true ($40). The value $40 is stored to a random-access memory (RAM) location and is called a fuzzy input (in this case, the fuzzy input for “temperature is warm”). There is a RAM location for each fuzzy input (for each label of each system input).

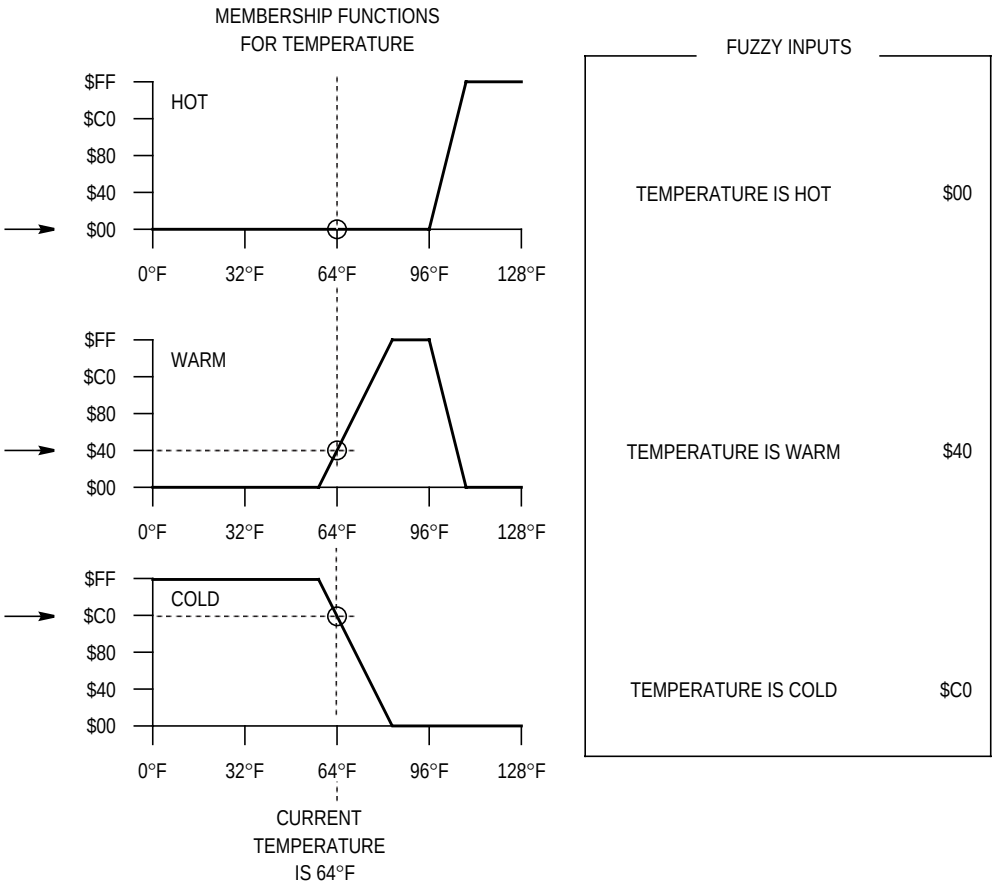


Figure 9-2. Fuzzification Using Membership Functions

When the fuzzification step begins, the current value of the system input is in an accumulator of the CPU12, one index register points to the first membership function definition in the knowledge base, and a second index register points to the first fuzzy input in RAM. As each fuzzy input is calculated by executing a MEM instruction, the result is stored to the fuzzy input and both pointers are updated automatically to point to the locations associated with the next fuzzy input. The MEM instruction takes care of everything except counting the number of labels per system input and loading the current value of any subsequent system inputs.

The end result of the fuzzification step is a table of fuzzy inputs representing current system conditions.

9.2.2 Rule Evaluation (REV and REVW)

Rule evaluation is the central element of a fuzzy logic inference program. This step processes a list of rules from the knowledge base using current fuzzy input values from RAM to produce a list of fuzzy outputs in RAM. These fuzzy outputs can be thought of as raw suggestions for what the system output should be in response to the current input conditions. Before the results can be applied, the fuzzy outputs must be further processed, or defuzzified, to produce a single output value that represents the combined effect of all of the fuzzy outputs.

The CPU12 offers two variations of rule evaluation instructions. The REV instruction provides for unweighted rules (all rules are considered to be equally important). The REVW instruction is similar but allows each rule to have a separate weighting factor which is stored in a separate parallel data structure in the knowledge base. In addition to the weights, the two rule evaluation instructions also differ in the way rules are encoded into the knowledge base.

An understanding of the structure and syntax of rules is needed to understand how a microcontroller performs the rule evaluation task. An example of a typical rule is:

> If temperature is warm and pressure is high, then heat is (should be) off.

At first glance, it seems that encoding this rule in a compact form understandable to the microcontroller would be difficult, but it is actually simple to reduce the rule to a small list of memory pointers. The antecedent portion of the rule is a statement of input conditions and the consequent portion of the rule is a statement of output actions.

The antecedent portion of a rule is made up of one or more (in this case two) antecedents connected by a fuzzy *and* operator. Each antecedent expression consists of the name of a system input, followed by *is*, followed by a label name. The label must be defined by a membership function in the knowledge base. Each antecedent expression corresponds to one of the fuzzy inputs in RAM. Since *and* is the only operator allowed to connect antecedent expressions, there is no need to include these in the encoded rule. The antecedents can be encoded as a simple list of pointers to (or addresses of) the fuzzy inputs to which they refer.

The consequent portion of a rule is made up of one or more (in this case one) consequents. Each consequent expression consists of the name of a system output, followed by *is*, followed by a label name. Each consequent expression corresponds to a specific fuzzy output in RAM. Consequents for a rule can be encoded as a simple list of pointers to (or addresses of) the fuzzy outputs to which they refer.

The complete rules are stored in the knowledge base as a list of pointers or addresses of fuzzy inputs and fuzzy outputs. For the rule evaluation logic to work, there must be some means of knowing which pointers refer to fuzzy inputs and which refer to fuzzy outputs. There also must be a way to know when the last rule in the system has been reached.

- One method of organization is to have a fixed number of rules with a specific number of antecedents and consequents.
- A second method, employed in Motorola Freeware M68HC11 kernels, is to mark the end of the rule list with a reserved value, and use a bit in the pointers to distinguish antecedents from consequents.
- A third method of organization, used in the CPU12, is to mark the end of the rule list with a reserved value, and separate antecedents and consequents with another reserved value. This permits any number of rules, and allows each rule to have any number of antecedents and consequents, subject to the limits imposed by availability of system memory.

Each rule is evaluated sequentially, but the rules as a group are treated as if they were all evaluated simultaneously. Two mathematical operations take place during rule evaluation. The fuzzy *and* operator corresponds to the mathematical minimum operation and the fuzzy *or* operation corresponds to the mathematical maximum operation. The fuzzy *and* is used to connect antecedents within a rule. The fuzzy *or* is implied between successive rules. Before evaluating any rules, all fuzzy outputs are set to zero (meaning not true at all). As each rule is evaluated, the smallest (minimum) antecedent is taken to be the overall truth of the rule. This rule truth value is applied to each consequent of the rule (by storing this value to the corresponding fuzzy output) unless the fuzzy output is already larger (maximum). If two rules affect the same fuzzy output, the rule that is most true governs the value in the fuzzy output because the rules are connected by an implied fuzzy *or*.

In the case of rule weighting, the truth value for a rule is determined as usual by finding the smallest rule antecedent. Before applying this truth value to the consequents for the rule, the value is multiplied by a fraction from zero (rule disabled) to one (rule fully enabled). The resulting modified truth value is then applied to the fuzzy outputs.

The end result of the rule evaluation step is a table of suggested or “raw” fuzzy outputs in RAM. These values were obtained by plugging current conditions (fuzzy input values) into the system rules in the knowledge base. The raw results cannot be supplied directly to the system outputs because they may be ambiguous. For instance, one raw output can indicate that the system output should be medium with a degree of truth of 50 percent while, at the same time, another indicates that the system output should be low with a degree of truth of 25 percent. The defuzzification step resolves these ambiguities.

9.2.3 Defuzzification (WAV)

The final step in the fuzzy logic program combines the raw fuzzy outputs into a composite system output. Unlike the trapezoidal shapes used for inputs, the CPU12 typically uses singletons for output membership functions. As with the inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function.

The WAV instruction calculates the numerator and denominator sums for weighted average of the fuzzy outputs according to the formula:

$$\text{System Output} = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

Where n is the number of labels of a system output, S_i are the singleton positions from the knowledge base, and F_i are fuzzy outputs from RAM. For a common fuzzy logic program on the CPU12, n is eight or less (though this instruction can handle any value to 255) and S_i and F_i are 8-bit values. The final divide is performed with a separate EDIV instruction placed immediately after the WAV instruction.

Before executing WAV, an accumulator must be loaded with the number of iterations (n), one index register must be pointed at the list of singleton positions in the knowledge base, and a second index register must be pointed at the list of fuzzy outputs in RAM. If the system has more than one system output, the WAV instruction is executed once for each system output.

9.3 Example Inference Kernel

Figure 9-3 is a complete fuzzy inference kernel written in CPU12 assembly language. Numbers in square brackets are cycle counts for an HCS12 device. The kernel uses two system inputs with seven labels each and one system output with seven labels. The program assembles to 57 bytes. It executes in about 20 µs at an 25-MHz bus rate. The basic structure can easily be extended to a general-purpose system with a larger number of inputs and outputs.

```
   *
01 [2]   FUZZIFY     LDX   #INPUT_MFS     ;Point at MF definitions
02 [2]               LDY   #FUZ_INS       ;Point at fuzzy input table
03 [3]               LDAA  CURRENT_INS    ;Get first input value
04 [1]               LDAB  #7             ;7 labels per input
05 [5]   GRAD_LOOP   MEM                  ;Evaluate one MF
06 [3]               DBNE  B,GRAD_LOOP    ;For 7 labels of 1 input
07 [3]               LDAA  CURRENT_INS+1  ;Get second input value
08 [1]               LDAB  #7             ;7 labels per input
09 [5]   GRAD_LOOP1  MEM                  ;Evaluate one MF
10 [3]               DBNE  B,GRAD_LOOP1   ;For 7 labels of 1 input

11 [1]               LDAB  #7             ;Loop count
12 [2]   RULE_EVAL   CLR   1,Y+           ;Clr a fuzzy out & inc ptr
13 [3]               DBNE  b,RULE_EVAL    ;Loop to clr all fuzzy outs
14 [2]               LDX   #RULE_START    ;Point at first rule element
15 [2]               LDY   #FUZ_INS       ;Point at fuzzy ins and outs
16 [1]               LDAA  #$FF           ;Init A (and clears V-bit)
17 [3n+4]            REV                  ;Process rule list

18 [2]   DEFUZ       LDY   #FUZ_OUT       ;Point at fuzzy outputs
19 [2]               LDX   #SGLTN_POS     ;Point at singleton positions
20 [1]               LDAB  #7             ;7 fuzzy outs per COG output
21 [7b+4]            WAV                  ;Calculate sums for wtd av
22 [11]              EDIV                 ;Final divide for wtd av
23 [1]               TFR   Y,D            ;Move result to A:B
24 [3]               STAB  COG_OUT        ;Store system output
   *
   ***** End
```

Figure 9-3. Fuzzy Inference Engine

Lines 1 to 3 set up pointers and load the system input value into the A accumulator.

Line 4 sets the loop count for the loop in lines 5 and 6.

Lines 5 and 6 make up the fuzzification loop for seven labels of one system input. The MEM instruction finds the y-value on a trapezoidal membership function for the current input value, for one label of the current input, and then stores the result to the corresponding fuzzy input. Pointers in X and Y are automatically updated by four and one so they point at the next membership function and fuzzy input respectively.

Line 7 loads the current value of the next system input. Pointers in X and Y already point to the right places as a result of the automatic update function of the MEM instruction in line 5.

Line 8 reloads a loop count.

Lines 9 and 10 form a loop to fuzzify the seven labels of the second system input. When the program drops to line 11, the Y index register is pointing at the next location after the last fuzzy input, which is the first fuzzy output in this system.

Line 11 sets the loop count to clear seven fuzzy outputs.

Lines 12 and 13 form a loop to clear all fuzzy outputs before rule evaluation starts.

Line 14 initializes the X index register to point at the first element in the rule list for the REV instruction.

Line 15 initializes the Y index register to point at the fuzzy inputs and outputs in the system. The rule list (for REV) consists of 8-bit offsets from this base address to particular fuzzy inputs or fuzzy outputs. The special value $FE is interpreted by REV as a marker between rule antecedents and consequents.

Line 16 initializes the A accumulator to the highest 8-bit value in preparation for finding the smallest fuzzy input referenced by a rule antecedent. The LDAA #$FF instruction also clears the V-bit in the CPU12’s condition code register so the REV instruction knows it is processing antecedents. During rule list processing, the V bit is toggled each time an $FE is detected in the list. The V bit indicates whether REV is processing antecedents or consequents.

Line 17 is the REV instruction, a self-contained loop to process successive elements in the rule list until an $FF character is found. For a system of 17 rules with two antecedents and one consequent each, the REV instruction takes 259 cycles, but it is interruptible so it does not cause a long interrupt latency.

Lines 18 through 20 set up pointers and an iteration count for the WAV instruction.

Line 21 is the beginning of defuzzification. The WAV instruction calculates a sum-of-products and a sum-of-weights.

Line 22 completes defuzzification. The EDIV instruction performs a 32-bit by 16-bit divide on the intermediate results from WAV to get the weighted average.

Line 23 moves the EDIV result into the double accumulator.

Line 24 stores the low 8-bits of the defuzzification result.

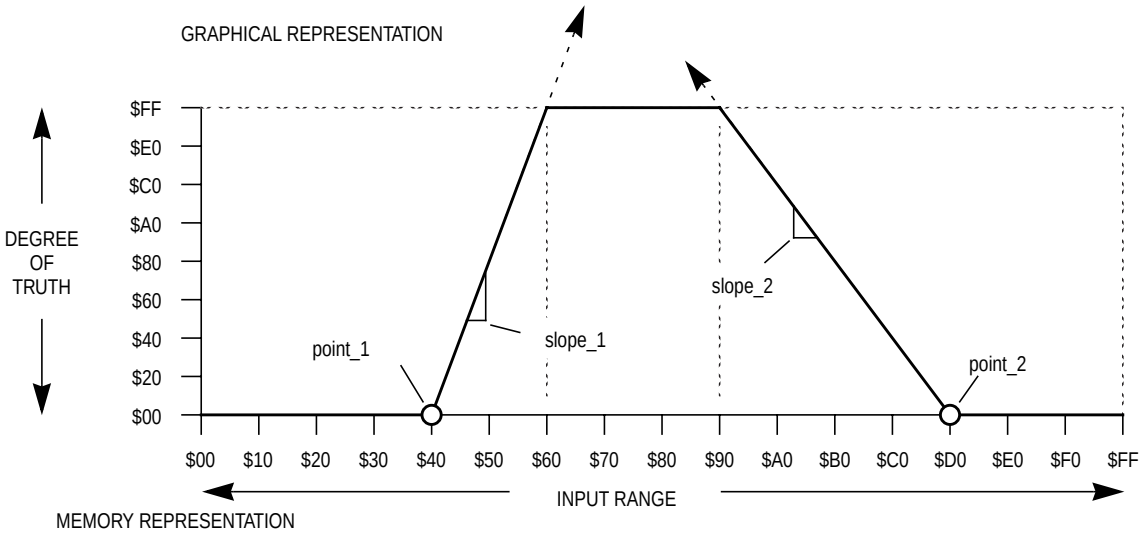
This example inference program shows how easy it is to incorporate fuzzy logic into general applications using the CPU12. Code space and execution time are no longer serious factors in the decision to use fuzzy logic. The next section begins a much more detailed look at the fuzzy logic instructions of the CPU12.

9.4 MEM Instruction Details

This section provides a more detailed explanation of the membership function evaluation instruction (MEM), including details about abnormal special cases for improperly defined membership functions.

9.4.1 Membership Function Definitions

Figure 9-4 shows how a normal membership function is specified in the CPU12. Typically, a software tool is used to input membership functions graphically, and the tool generates data structures for the target processor and software kernel. Alternatively, points and slopes for the membership functions can be determined and stored in memory with define-constant assembler directives.



MEMORY REPRESENTATION

ADDR	$40	X-POSITION OF point_1
ADDR+1	$D0	X-POSITION OF point_2
ADDR+2	$08	slope_1 ($FF/(X-POS OF SATURATION – point_1))
ADDR+3	$04	slope_2 ($FF/(point_2 – X-POS OF SATURATION))

Figure 9-4. Defining a Normal Membership Function

An internal CPU algorithm calculates the y-value where the current input intersects a membership function. This algorithm assumes the membership function obeys some common-sense rules. If the membership function definition is improper, the results may be unusual. See **9.4.2 Abnormal Membership Function Definitions** for a discussion of these cases.

These rules apply to normal membership functions.

- $00 ≤ point1 < $FF
- $00 < point2 ≤ $FF
- point1 < point2
- The sloping sides of the trapezoid meet at or above $FF.

Each system input such as temperature has several labels such as cold, cool, normal, warm, and hot. Each label of each system input must have a membership function to describe its meaning in an unambiguous numerical way. Typically, there are three to seven labels per system input, but there is no practical restriction on this number as far as the fuzzification step is concerned.

9.4.2 Abnormal Membership Function Definitions

In the CPU12, it is possible (and proper) to define “crisp” membership functions. A crisp membership function has one or both sides vertical (infinite slope). Since the slope value $00 is not used otherwise, it is assigned to mean infinite slope to the MEM instruction in the CPU12.

Although a good fuzzy development tool will not allow the user to specify an improper membership function, it is possible to have program errors or memory errors which result in erroneous abnormal membership functions. Although these abnormal shapes do not correspond to any working systems, understanding how the CPU12 treats these cases can be helpful for debugging.

A close examination of the MEM instruction algorithm will show how such membership functions are evaluated. **Figure 9-5** is a complete flow diagram for the execution of a MEM instruction. Each rectangular box represents one CPU bus cycle. The number in the upper left corner corresponds to the cycle number and the letter corresponds to the cycle type (refer to **Section 6. Instruction Glossary** for details). The upper portion of the box includes information about bus activity during this cycle (if any). The lower portion of the box, which is separated by a dashed line, includes information about internal CPU processes. It is common for several internal functions to take place during a single CPU cycle (for example, in cycle 2, two 8-bit subtractions take place and a flag is set based on the results).

Consider 4a: If (((Slope_2 = 0) or (Grade_2 > $FF)) and (flag_d12n = 0)).

The flag_d12n is zero as long as the input value (in accumulator A) is within the trapezoid. Everywhere outside the trapezoid, one or the other delta term will be negative, and the flag will equal one. Slope_2 equals zero indicates the right side of the trapezoid has infinite slope, so the resulting grade should be $FF everywhere in the trapezoid, including at point_2, as far as this side is concerned. The term grade_2 greater than $FF means the value is far enough into the trapezoid that the right sloping side of the trapezoid has crossed above the $FF cutoff level and the resulting grade should be $FF as far as the right sloping side is concerned. 4a decides if the value is left of the right sloping side (Grade = $FF), or on the sloping portion of the right side of the trapezoid (Grade = Grade_2). 4b could still override this tentative value in grade.

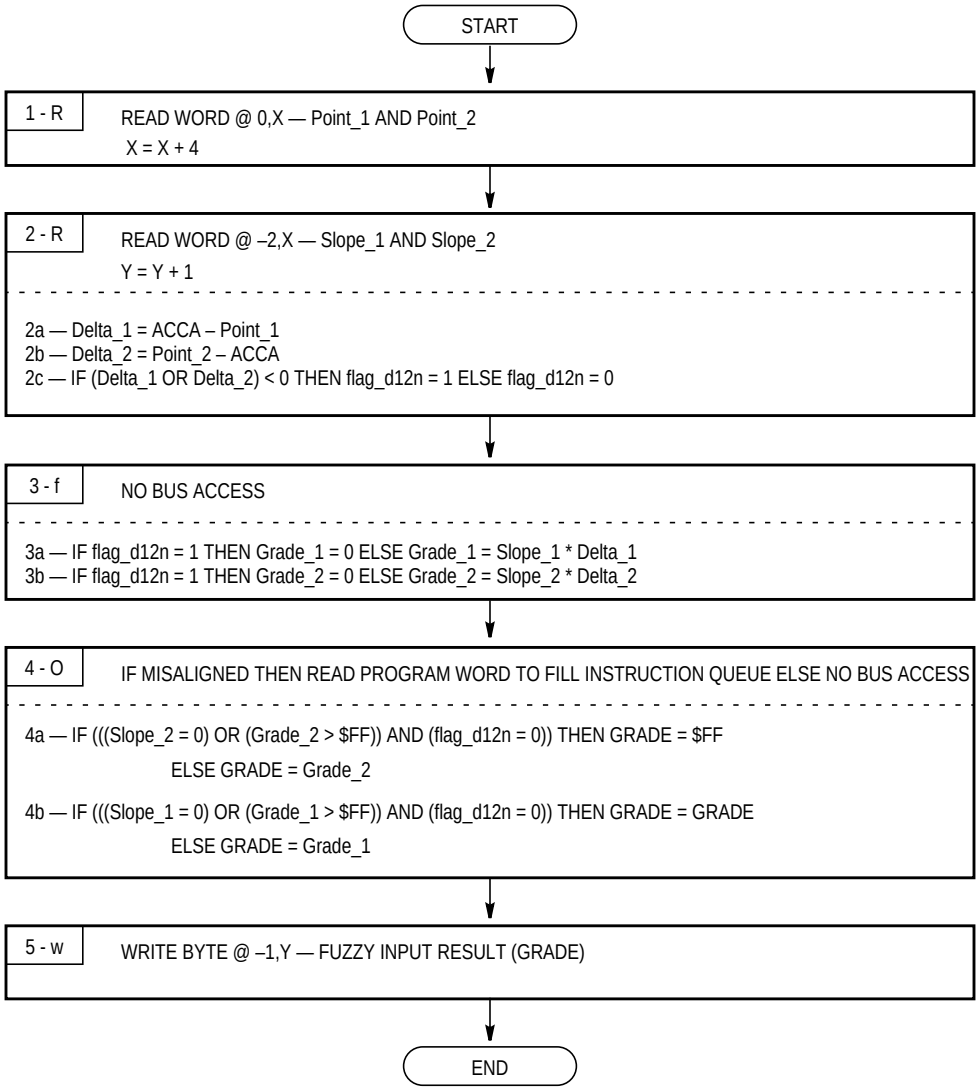


Figure 9-5. MEM Instruction Flow Diagram

In 4b, slope_1 is zero if the left side of the trapezoid has infinite slope (vertical). If so, the result (grade) should be $FF at and to the right of point_1 everywhere within the trapezoid as far as the left side is concerned. The grade_1 greater than $FF term corresponds to the input being to the right of where the left sloping side passes the $FF cutoff level. If either of these conditions is true, the result (grade) is left at the value it got from 4a. The “else” condition in 4b corresponds to the input falling on the sloping portion of the left side of the trapezoid (or possibly outside the trapezoid), so the result is grade equal grade_1. If the input was outside the trapezoid, flag_d12n would be one and grade_1 and

grade_2 would have been forced to $00 in cycle 3. The else condition of 4b would set the result to $00.

The special cases shown here represent abnormal membership function definitions. The explanations describe how the specific algorithm in the CPU12 resolves these unusual cases. The results are not all intuitively obvious, but rather fall out from the specific algorithm. Remember, these cases should not occur in a normal system.

9.4.2.1 Abnormal Membership Function Case 1

This membership function is abnormal because the sloping sides cross below the $FF cutoff level. The flag_d12n signal forces the membership function to evaluate to $00 everywhere except from point_1 to point_2. Within this interval, the tentative values for grade_1 and grade_2 calculated in cycle 3 fall on the crossed sloping sides. In step 4a, grade gets set to the grade_2 value, but in 4b this is overridden by the grade_1 value, which ends up as the result of the MEM instruction. One way to say this is that the result follows the left sloping side until the input passes point_2, where the result goes to $00.

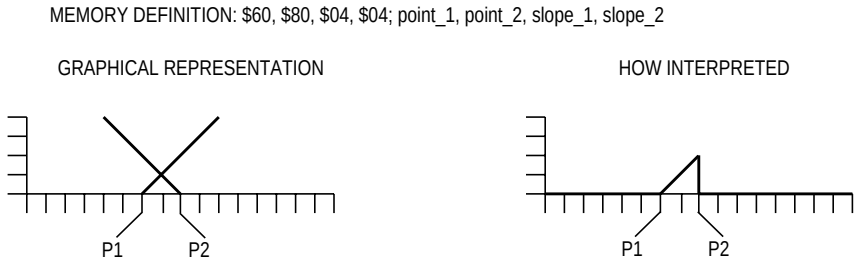


Figure 9-6. Abnormal Membership Function Case 1

If point_1 was to the right of point_2, flag_d12n would force the result to be $00 for all input values. In fact, flag_d12n always limits the region of interest to the space greater than or equal to point_1 and less than or equal to point_2.

9.4.2.2 Abnormal Membership Function Case 2

Like the previous example, the membership function in case 2 is abnormal because the sloping sides cross below the $FF cutoff level, but the left sloping side reaches the $FF cutoff level before the input gets to point_2. In this case, the result follows the left sloping side until it reaches the $FF cutoff level. At this point, the (grade_1 > $FF) term of 4b kicks in, making the expression true so grade equals grade (no overwrite). The result from here to point_2 becomes controlled by the “else” part of 4a (grade = grade_2), and the result follows the right sloping side.

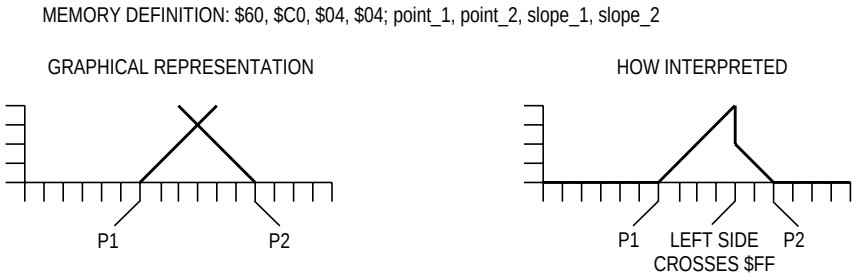


Figure 9-7. Abnormal Membership Function Case 2

9.4.2.3 Abnormal Membership Function Case 3

The membership function in case 3 is abnormal because the sloping sides cross below the $FF cutoff level, and the left sloping side has infinite slope. In this case, 4a is not true, so grade equals grade_2. 4b is true because slope_1 is zero, so 4b does not overwrite grade.

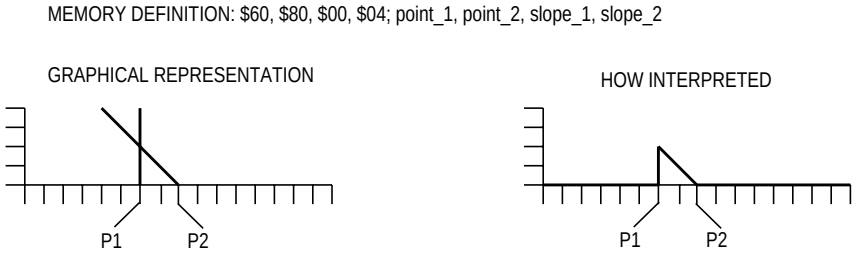


Figure 9-8. Abnormal Membership Function Case 3

9.5 REV and REVW Instruction Details

This section provides a more detailed explanation of the rule evaluation instructions (REV and REVW). The data structures used to specify rules are somewhat different for the weighted versus unweighted versions of the instruction. One uses 8-bit offsets in the encoded rules, while the other uses full 16-bit addresses. This affects the size of the rule data structure and execution time.

9.5.1 Unweighted Rule Evaluation (REV)

This instruction implements basic min-max rule evaluation. CPU registers are used for pointers and intermediate calculation results.

Since the REV instruction is essentially a list-processing instruction, execution time is dependent on the number of elements in the rule list. The REV instruction is interruptible (typically within three bus cycles), so it does not adversely affect worst case interrupt latency. Since all intermediate results and instruction status are held in stacked CPU registers, the interrupt service code can even include independent REV and REVW instructions.

9.5.1.1 Set Up Prior to Executing REV

Some CPU registers and memory locations need to be set up prior to executing the REV instruction. X and Y index registers are used as index pointers to the rule list and the fuzzy inputs and outputs. The A accumulator is used for intermediate calculation results and needs to be set to $FF initially. The V condition code bit is used as an instruction status indicator to show whether antecedents or consequents are being processed. Initially, the V bit is cleared to zero to indicate antecedents are being processed. The fuzzy outputs (working RAM locations) need to be cleared to $00. If these values are not initialized before executing the REV instruction, results will be erroneous.

The X index register is set to the address of the first element in the rule list (in the knowledge base). The REV instruction automatically updates this pointer so that the instruction can resume correctly if it is interrupted. After the REV instruction finishes, X will point at the next address past the $FF separator character that marks the end of the rule list.

The Y index register is set to the base address for the fuzzy inputs and outputs (in working RAM). Each rule antecedent is an unsigned 8-bit offset from this base address to the referenced fuzzy input. Each rule consequent is an unsigned 8-bit offset from this base address to the referenced fuzzy output. The Y index register remains constant throughout execution of the REV instruction.

The 8-bit A accumulator is used to hold intermediate calculation results during execution of the REV instruction. During antecedent processing, A starts out at $FF and is replaced by any smaller fuzzy input that is referenced by a rule antecedent (MIN). During consequent processing, A holds the truth value for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (MAX).

Before starting to execute REV, A must be set to $FF (the largest 8-bit value) because rule evaluation always starts with processing of the antecedents of the first rule. For subsequent rules in the list, A is automatically set to $FF when the instruction detects the $FE marker character between the last consequent of the previous rule and the first antecedent of a new rule.

The instruction LDAA #$FF clears the V bit at the same time it initializes A to $FF. This satisfies the REV setup requirement to clear the V bit as well as the requirement to initialize A to $FF. Once the REV instruction starts, the value in the V bit is automatically maintained as $FE separator characters are detected.

The final requirement to clear all fuzzy outputs to $00 is part of the MAX algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output.

After REV finishes, A will hold the truth value for the last rule in the rule list. The V condition code bit should be one because the last element before the $FF end marker should have been a rule consequent. If V is zero after executing REV, it indicates the rule list was structured incorrectly.

9.5.1.2 Interrupt Details

The REV instruction includes a 3-cycle processing loop for each byte in the rule list (including antecedents, consequents, and special separator characters). Within this loop, a check is performed to see if any qualified interrupt request is pending. If an interrupt is detected, the current CPU registers are stacked and the interrupt is honored. When the interrupt service routine finishes, an RTI instruction causes the CPU to recover its previous context from the stack, and the REV instruction is resumed as if it had not been interrupted.

The stacked value of the program counter (PC), in case of an interrupted REV instruction, points to the REV instruction rather than the instruction that follows. This causes the CPU to try to execute a new REV instruction upon return from the interrupt. Since the CPU registers (including the V bit in the condition codes register) indicate the current status of the interrupted REV instruction, this effectively causes the rule evaluation operation to resume from where it left off.

9.5.1.3 Cycle-by-Cycle Details for REV

The central element of the REV instruction is a 3-cycle loop that is executed once for each byte in the rule list. There is a small amount of housekeeping activity to get this loop started as REV begins and a small sequence to end the instruction. If an interrupt comes, there is a special small sequence to save CPU status on the stack before honoring the requested interrupt.

Figure 9-9 is a REV instruction flow diagram. Each rectangular box represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of each bold box correspond to execution cycle codes (refer to **Section 6. Instruction Glossary** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit or no data is transferred.

When a value is read from memory, it cannot be used by the CPU until the second cycle after the read takes place. This is due to access and propagation delays.

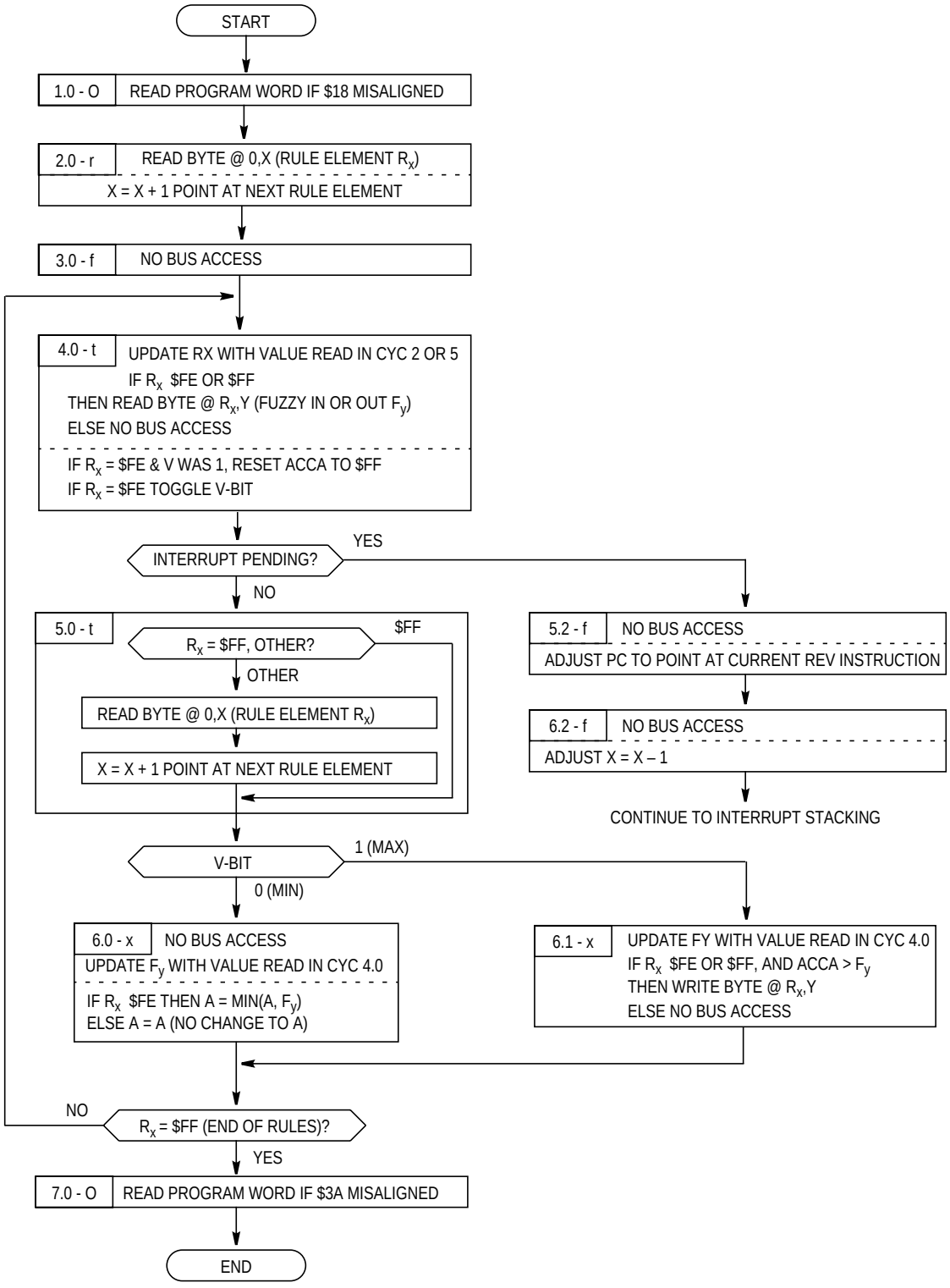


Figure 9-9. REV Instruction Flow Diagram

Since there is more than one flow path through the REV instruction, cycle numbers have a decimal place. This decimal place indicates which of several possible paths is being used. The CPU normally moves forward by one digit at a time within the same flow (flow number is indicated after the decimal point in the cycle number). There are two exceptions possible to this orderly sequence through an instruction. The first is a branch back to an earlier cycle number to form a loop as in 6.0 to 4.0. The second type of sequence change is from one flow to a parallel flow within the same instruction such as 4.0 to 5.2, which occurs if the REV instruction senses an interrupt. In this second type of sequence branch, the whole number advances by one and the flow number changes to a new value (the digit after the decimal point).

In cycle 1.0, the CPU12 does an optional program word access to replace the $18 prebyte of the REV instruction. Notice that cycle 7.0 is also an O type cycle. One or the other of these will be a program word fetch, while the other will be a free cycle where the CPU does not access the bus. Although the $18 page prebyte is a required part of the REV instruction, it is treated by the CPU12 as a somewhat separate single cycle instruction.

Rule evaluation begins at cycle 2.0 with a byte read of the first element in the rule list. Usually this would be the first antecedent of the first rule, but the REV instruction can be interrupted, so this could be a read of any byte in the rule list. The X index register is incremented so it points to the next element in the rule list. Cycle 3.0 is needed to satisfy the required delay between a read and when data is valid to the CPU. Some internal CPU housekeeping activity takes place during this cycle, but there is no bus activity. By cycle 4.0, the rule element that was read in cycle 2.0 is available to the CPU.

Cycle 4.0 is the first cycle of the main three cycle rule evaluation loop. Depending upon whether rule antecedents or consequents are being processed, the loop will consist of cycles 4.0, 5.0, 6.0, or the sequence 4.0, 5.0, 6.1. This loop is executed once for every byte in the rule list, including the $FE separators and the $FF end-of-rules marker.

At each cycle 4.0, a fuzzy input or fuzzy output is read, except during the loop passes associated with the $FE and $FF marker bytes, where no bus access takes place during cycle 4.0. The read access uses the Y index register as the base address and the previously read rule byte (R_X) as an unsigned offset from Y. The fuzzy input or output value read here

will be used during the next cycle 6.0 or 6.1. Besides being used as the offset from Y for this read, the previously read R_X is checked to see if it is a separator character ($FE). If R_X was $FE and the V bit was one, this indicates a switch from processing consequents of one rule to starting to process antecedents of the next rule. At this transition, the A accumulator is initialized to $FF to prepare for the min operation to find the smallest fuzzy input. Also, if Rx is $FE, the V bit is toggled to indicate the change from antecedents to consequents, or consequents to antecedents.

During cycle 5.0, a new rule byte is read unless this is the last loop pass, and R_X is $FF (marking the end of the rule list). This new rule byte will not be used until cycle 4.0 of the next pass through the loop.

Between cycle 5.0 and 6.x, the V-bit is used to decide which of two paths to take. If V is zero, antecedents are being processed and the CPU progresses to cycle 6.0. If V is one, consequents are being processed and the CPU goes to cycle 6.1.

During cycle 6.0, the current value in the A accumulator is compared to the fuzzy input that was read in the previous cycle 4.0, and the lower value is placed in the A accumulator (min operation). If Rx is $FE, this is the transition between rule antecedents and rule consequents, and this min operation is skipped (although the cycle is still used). No bus access takes place during cycle 6.0 but cycle 6.x is considered an x type cycle because it could be a byte write (cycle 6.1) or a free cycle (cycle 6.0 or 6.1 with Rx = $FE or $FF).

If an interrupt arrives while the REV instruction is executing, REV can break between cycles 4.0 and 5.0 in an orderly fashion so that the rule evaluation operation can resume after the interrupt has been serviced. Cycles 5.2 and 6.2 are needed to adjust the PC and X index register so the REV operation can recover after the interrupt. PC is adjusted backward in cycle 5.2 so it points to the currently running REV instruction. After the interrupt, rule evaluation will resume, but the values that were stored on the stack for index registers, accumulator A, and CCR will cause the operation to pick up where it left off. In cycle 6.2, the X index register is adjusted backward by one because the last rule byte needs to be re-fetched when the REV instruction resumes.

After cycle 6.2, the REV instruction is finished, and execution would continue to the normal interrupt processing flow.

9.5.2 Weighted Rule Evaluation (REVW)

This instruction implements a weighted variation of min-max rule evaluation. The weighting factors are stored in a table with one 8-bit entry per rule. The weight is used to multiply the truth value of the rule (minimum of all antecedents) by a value from zero to one to get the weighted result. This weighted result is then applied to the consequents, just as it would be for unweighted rule evaluation.

Since the REVW instruction is essentially a list-processing instruction, execution time is dependent on the number of rules and the number of elements in the rule list. The REVW instruction is interruptible (typically within three to five bus cycles), so it does not adversely affect worst case interrupt latency. Since all intermediate results and instruction status are held in stacked CPU registers, the interrupt service code can even include independent REV and REVW instructions.

The rule structure is different for REVW than for REV. For REVW, the rule list is made up of 16-bit elements rather than 8-bit elements. Each antecedent is represented by the full 16-bit address of the corresponding fuzzy input. Each rule consequent is represented by the full address of the corresponding fuzzy output.

The markers separating antecedents from consequents are the reserved 16-bit value $FFFE, and the end of the last rule is marked by the reserved 16-bit value $FFFF. Since $FFFE and $FFFF correspond to the addresses of the reset vector, there would never be a fuzzy input or output at either of these locations.

9.5.2.1 Set Up Prior to Executing REVW

Some CPU registers and memory locations need to be set up prior to executing the REVW instruction. X and Y index registers are used as index pointers to the rule list and the list of rule weights. The A accumulator is used for intermediate calculation results and needs to be set to $FF initially. The V condition code bit is used as an instruction status indicator that shows whether antecedents or consequents are being processed. Initially the V bit is cleared to zero to indicate antecedents are being processed. The C condition code bit is used to indicate whether rule weights are to be used (1) or not (0). The fuzzy outputs (working RAM locations) need to be cleared to $00. If these values are not initialized before executing the REVW instruction, results will be erroneous.

The X index register is set to the address of the first element in the rule list (in the knowledge base). The REVW instruction automatically updates this pointer so that the instruction can resume correctly if it is interrupted. After the REVW instruction finishes, X will point at the next address past the $FFFF separator word that marks the end of the rule list.

The Y index register is set to the starting address of the list of rule weights. Each rule weight is an 8-bit value. The weighted result is the truncated upper eight bits of the 16-bit result, which is derived by multiplying the minimum rule antecedent value ($00–$FF) by the weight plus one ($001–$100). This method of weighting rules allows an 8-bit weighting factor to represent a value between zero and one inclusive.

The 8-bit A accumulator is used to hold intermediate calculation results during execution of the REVW instruction. During antecedent processing, A starts out at $FF and is replaced by any smaller fuzzy input that is referenced by a rule antecedent. If rule weights are enabled by the C condition code bit equal one, the rule truth value is multiplied by the rule weight just before consequent processing starts. During consequent processing, A holds the truth value (possibly weighted) for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (MAX).

Before starting to execute REVW, A must be set to $FF (the largest 8-bit value) because rule evaluation always starts with processing of the antecedents of the first rule. For subsequent rules in the list, A is automatically set to $FF when the instruction detects the $FFFE marker word between the last consequent of the previous rule, and the first antecedent of a new rule.

Both the C and V condition code bits must be set up prior to starting a REVW instruction. Once the REVW instruction starts, the C bit remains constant and the value in the V bit is automatically maintained as $FFFE separator words are detected.

The final requirement to clear all fuzzy outputs to $00 is part of the MAX algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value (weighted) for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output.

After REVW finishes, A will hold the truth value (weighted) for the last rule in the rule list. The V condition code bit should be one because the last element before the $FFFF end marker should have been a rule consequent. If V is zero after executing REVW, it indicates the rule list was structured incorrectly.

9.5.2.2 Interrupt Details

The REVW instruction includes a 3-cycle processing loop for each word in the rule list (this loop expands to five cycles between antecedents and consequents to allow time for the multiplication with the rule weight). Within this loop, a check is performed to see if any qualified interrupt request is pending. If an interrupt is detected, the current CPU registers are stacked and the interrupt is honored. When the interrupt service routine finishes, an RTI instruction causes the CPU to recover its previous context from the stack, and the REVW instruction is resumed as if it had not been interrupted.

The stacked value of the program counter (PC), in case of an interrupted REVW instruction, points to the REVW instruction rather than the instruction that follows. This causes the CPU to try to execute a new REVW instruction upon return from the interrupt. Since the CPU registers (including the C bit and V bit in the condition codes register) indicate the current status of the interrupted REVW instruction, this effectively causes the rule evaluation operation to resume from where it left off.

9.5.2.3 Cycle-by-Cycle Details for REVW

The central element of the REVW instruction is a 3-cycle loop that is executed once for each word in the rule list. For the special case pass (where the $FFFE separator word is read between the rule antecedents and the rule consequents, and weights are enabled by the C bit equal one), this loop takes five cycles. There is a small amount of housekeeping activity to get this loop started as REVW begins and a small sequence to end the instruction. If an interrupt comes, there is a special small sequence to save CPU status on the stack before the interrupt is serviced.

Figure 9-10 is a detailed flow diagram for the REVW instruction. Each rectangular box represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of each bold box correspond to the execution cycle codes (refer to **Section 6. Instruction Glossary** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit data could be transferred.

In cycle 2.0, the first element of the rule list (a 16-bit address) is read from memory. Due to propagation delays, this value cannot be used for calculations until two cycles later (cycle 4.0). The X index register, which is used to access information from the rule list, is incremented by two to point at the next element of the rule list.

The operations performed in cycle 4.0 depend on the value of the word read from the rule list. $FFFE is a special token that indicates a transition from antecedents to consequents or from consequents to antecedents of a new rule. The V bit can be used to decide which transition is taking place, and V is toggled each time the $FFFE token is detected. If V was zero, a change from antecedents to consequents is taking place, and it is time to apply weighting (provided it is enabled by the C bit equal one). The address in TMP2 (derived from Y) is used to read the weight byte from memory. In this case, there is no bus access in cycle 5.0, but the index into the rule list is updated to point to the next rule element.

The old value of X (X_0) is temporarily held on internal nodes, so it can be used to access a rule word in cycle 7.2. The read of the rule word is timed to start two cycles before it will be used in cycle 4.0 of the next loop pass. The actual multiply takes place in cycles 6.2 through 8.2. The 8-bit weight from memory is incremented (possibly overflowing to $100) before the multiply, and the upper eight bits of the 16-bit internal result is used as the weighted result. By using weight+1, the result can range from 0.0 times A to 1.0 times A. After 8.2, flow continues to the next loop pass at cycle 4.0.

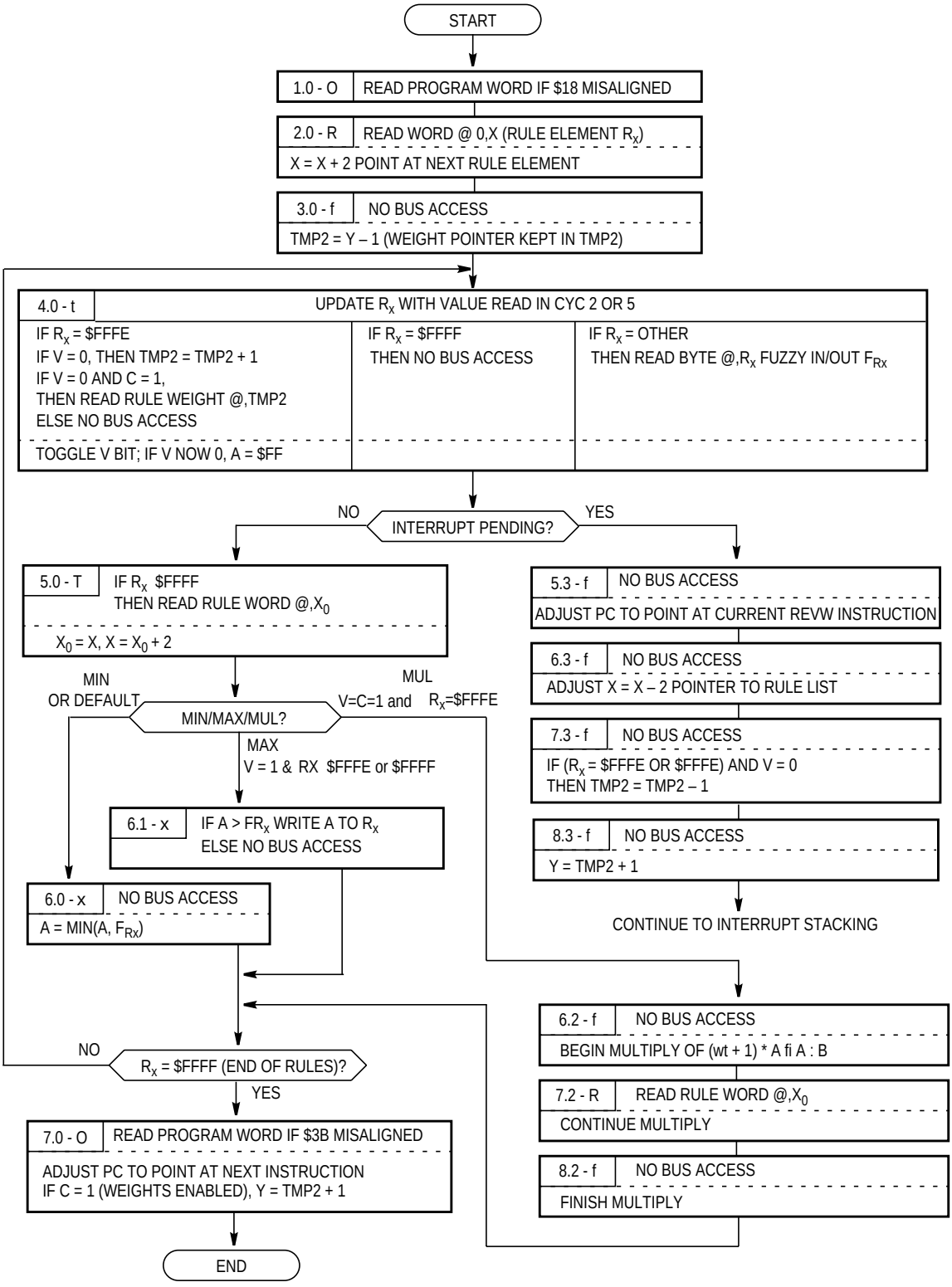


Figure 9-10. REVW Instruction Flow Diagram

At cycle 4.0, if R_x is $FFFE and V was one, a change from consequents to antecedents of a new rule is taking place, so accumulator A must be reinitialized to $FF. During processing of rule antecedents, A is updated with the smaller of A, or the current fuzzy input (cycle 6.0). Cycle 5.0 is usually used to read the next rule word and update the pointer in X. This read is skipped if the current R_x is $FFFF (end of rules mark). If this is a weight multiply pass, the read is delayed until cycle 7.2. During processing of consequents, cycle 6.1 is used to optionally update a fuzzy output if the value in accumulator A is larger.

After all rules have been processed, cycle 7.0 is used to update the PC to point at the next instruction. If weights were enabled, Y is updated to point at the location that immediately follows the last rule weight.

9.6 WAV Instruction Details

The WAV instruction performs weighted average calculations used in defuzzification. The pseudo-instruction wavr is used to resume an interrupted weighted average operation. WAV calculates the numerator and denominator sums using:

$$\text{System Output} = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

Where n is the number of labels of a system output, S_i are the singleton positions from the knowledge base, and F_i are fuzzy outputs from RAM. S_i and F_i are 8-bit values. The 8-bit B accumulator holds the iteration count n. Internal temporary registers hold intermediate sums, 24 bits for the numerator and 16 bits for the denominator. This makes this instruction suitable for n values up to 255 although eight is a more typical value. The final long division is performed with a separate EDIV instruction immediately after the WAV instruction. The WAV instruction returns the numerator and denominator sums in the correct registers for the EDIV. (EDIV performs the unsigned division Y = Y : D / X; remainder in D.)

Execution time for this instruction depends on the number of iterations (labels for the system output). WAV is interruptible so that worst case interrupt latency is not affected by the execution time for the complete weighted average operation. WAV includes initialization for the 24-bit and 16-bit partial sums so the first entry into WAV looks different than a resume from interrupt operation. The CPU12 handles this difficulty with a pseudo-instruction (wavr), which is specifically intended to resume an interrupted weighted average calculation. Refer to **9.6.3 Cycle-by-Cycle Details for WAV and wavr** for more detail.

9.6.1 Set Up Prior to Executing WAV

Before executing the WAV instruction, index registers X and Y and accumulator B must be set up. Index register X is a pointer to the S_i singleton list. X must have the address of the first singleton value in the knowledge base. Index register Y is a pointer to the fuzzy outputs F_i. Y must have the address of the first fuzzy output for this system output. B is the iteration count n. The B accumulator must be set to the number of labels for this system output.

9.6.2 WAV Interrupt Details

The WAV instruction includes a 7-cycle processing loop for each label of the system output (8 cycles in M68HC12). Within this loop, the CPU checks whether a qualified interrupt request is pending. If an interrupt is detected, the current values of the internal temporary registers for the 24-bit and 16-bit sums are stacked, the CPU registers are stacked, and the interrupt is serviced.

A special processing sequence is executed when an interrupt is detected during a weighted average calculation. This exit sequence adjusts the PC so that it points to the second byte of the WAV object code ($3C), before the PC is stacked. Upon return from the interrupt, the $3C value is interpreted as a wavr pseudo-instruction. The wavr pseudo-instruction causes the CPU to execute a special WAV resumption sequence. The wavr recovery sequence adjusts the PC so that it looks like it did during execution of the original WAV instruction, then jumps back into the WAV processing loop. If another interrupt occurs before the weighted average calculation finishes, the PC is adjusted again as it was for the first interrupt. WAV can be interrupted any number of times, and additional WAV instructions can be executed while a WAV instruction is interrupted.

9.6.3 Cycle-by-Cycle Details for WAV and wavr

The WAV instruction is unusual in that the logic flow has two separate entry points. The first entry point is the normal start of a WAV instruction. The second entry point is used to resume the weighted average operation after a WAV instruction has been interrupted. This recovery operation is called the wavr pseudo-instruction.

Figure 9-12 is a flow diagram of the WAV instruction in the HCS12, including the wavr pseudo-instruction. **Figure 9-12** is a flow diagram of the WAV instruction in the M68HC12, including the wavr pseudo-instruction. Each rectangular box in these figures represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of the boxes correspond to execution cycle codes (refer to **Section 6. Instruction Glossary** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit data could be transferred.

The cycle-by-cycle description provided here refers to the HCS12 flow in **Figure 9-11**. In terms of cycle-by-cycle bus activity, the $18 page select prebyte is treated as a special 1-byte instruction. In cycle 1.0 of the WAV instruction, one word of program information will be fetched into the instruction queue if the $18 is located at an odd address. If the $18 is at an even address, the instruction queue cannot advance so there is no bus access in this cycle.

In cycle 2.0, three internal 16-bit temporary registers are cleared in preparation for summation operations, but there is no bus access. The WAV instruction maintains a 32-bit sum-of-products in TMP1 : TMP2 and a 16-bit sum-of-weights in TMP3. By keeping these sums inside the CPU, bus accesses are reduced and the WAV operation is optimized for high speed.

Cycles 3.0 through 9.0 form the 7-cycle main loop for WAV. The value in the 8-bit B accumulator is used to count the number of loop iterations. B is decremented at the top of the loop in cycle 3.0, and the test for zero is located at the bottom of the loop after cycle 9.0. Cycle 4.0 and 5.0 are used to fetch the 8-bit operands for one iteration of the loop. X and Y index registers are used to access these operands. The index registers are incremented as the operands are fetched. Cycle 6.0 is used to accumulate the current fuzzy output into TMP3. Cycles 7.0 through 9.0 are used to perform the eight by eight multiply of F_i times S_i, and

accumulate this result into TMP1 : TMP2. Even though the sum-of-products will not exceed 24 bits, the sum is maintained in the 32-bit combined TMP1 : TMP2 register because it is easier to use existing 16-bit operations than it would be to create a new smaller operation to handle the high order bits of this sum.

Since the weighted average operation could be quite long, it is made to be interruptible. The usual longest latency path is from very early in cycle 6.0, through cycle 9.0, to the top of the loop to cycle 3.0, through cycle 5.0 to the interrupt check.

If the WAV instruction is interrupted, the internal temporary registers TMP3, TMP2, and TMP1 need to be stored on the stack so the operation can be resumed. Since the WAV instruction included initialization in cycle 2.0, the recovery path after an interrupt needs to be different. The wavr pseudo-instruction has the same opcode as WAV, but it is on the first page of the opcode map so there is no page prebyte ($18) like there is for WAV. When WAV is interrupted, the PC is adjusted to point at the second byte of the WAV object code, so that it will be interpreted as the wavr pseudo-instruction on return from the interrupt, rather than the WAV instruction. During the recovery sequence, the PC is readjusted in case another interrupt comes before the weighted average operation finishes.

The resume sequence includes recovery of the temporary registers from the stack (1.1 through 3.1), and reads to get the operands for the current iteration. The normal WAV flow is then rejoined at cycle 6.0.

Upon normal completion of the instruction (cycle 10.0), the PC is adjusted so it points to the next instruction. The results are transferred from the TMP registers into CPU registers in such a way that the EDIV instruction can be used to divide the sum-of-products by the sum-of-weights. TMP1 : TMP2 is transferred into Y : D and TMP3 is transferred into X.

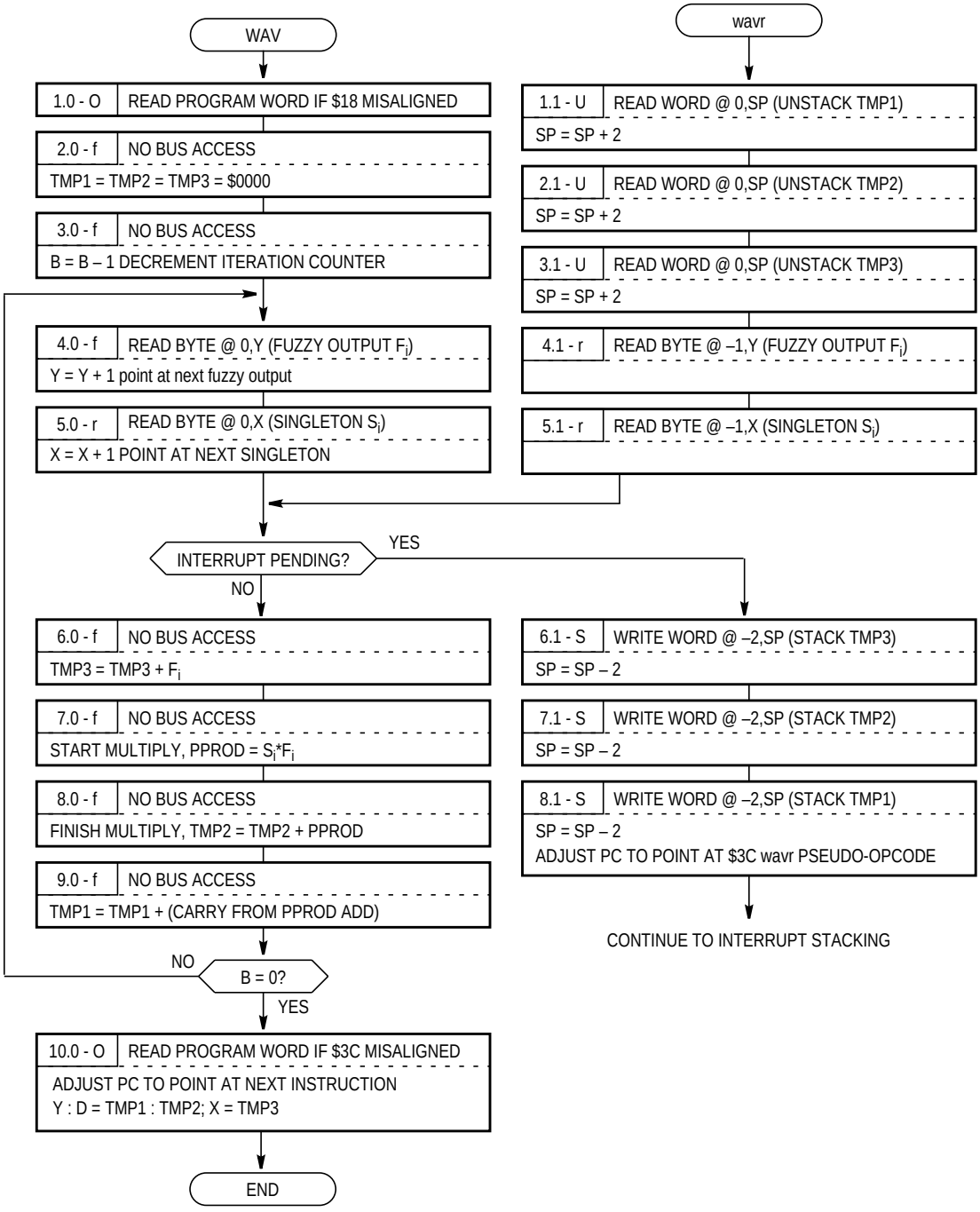


Figure 9-11. WAV and wavr Instruction Flow Diagram (for HCS12)

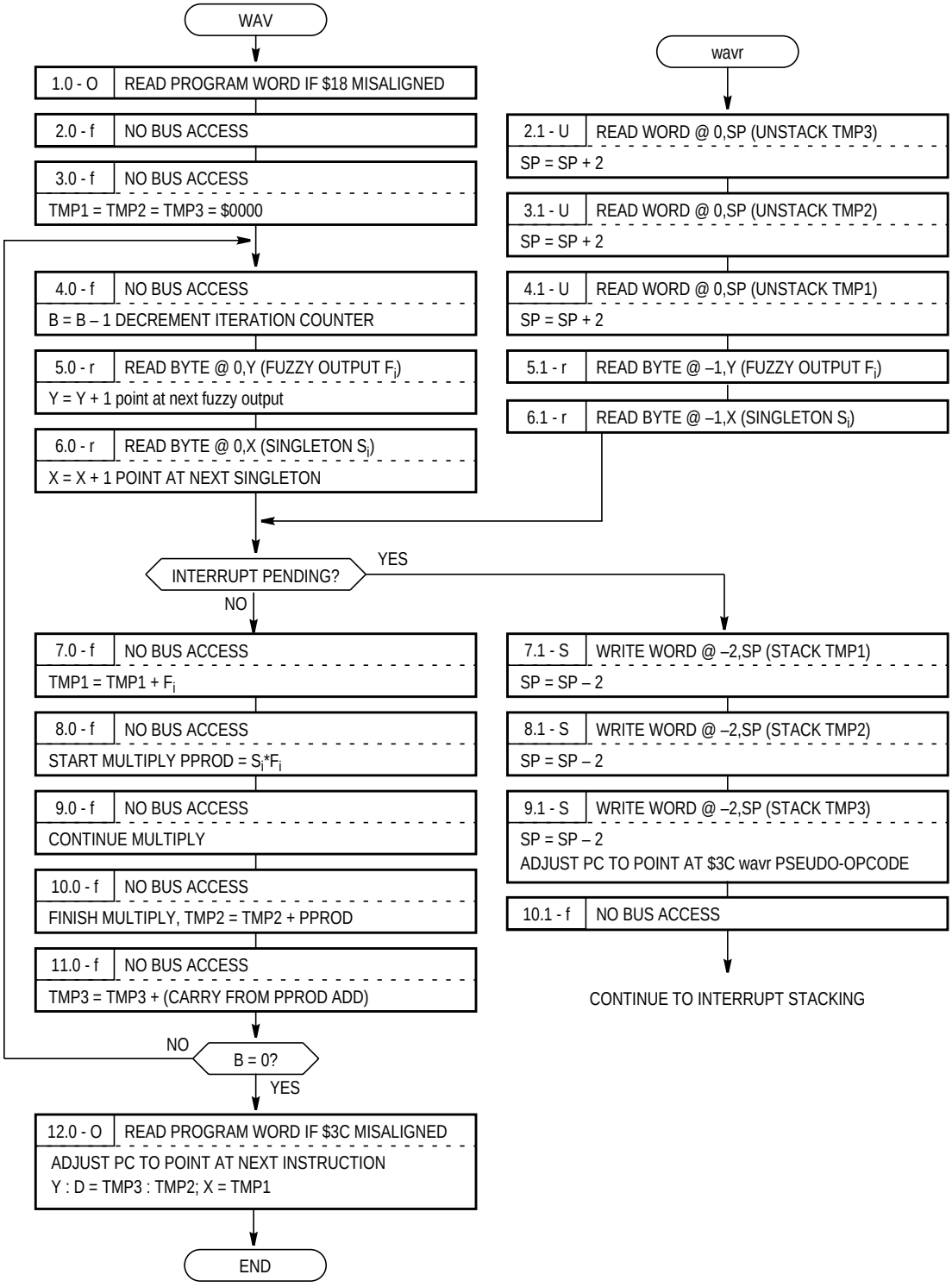


Figure 9-12. WAV and wavr Instruction Flow Diagram (for M68HC12)

9.7 Custom Fuzzy Logic Programming

The basic fuzzy logic inference techniques described earlier are suitable for a broad range of applications, but some systems may require customization. The built-in fuzzy instructions use 8-bit resolution and some systems may require finer resolution. The rule evaluation instructions only support variations of MIN-MAX rule evaluation and other methods have been discussed in fuzzy logic literature. The weighted average of singletons is not the only defuzzification technique. The CPU12 has several instructions and addressing modes that can be helpful when developing custom fuzzy logic systems.

9.7.1 Fuzzification Variations

The MEM instruction supports trapezoidal membership functions and several other varieties, including membership functions with vertical sides (infinite slope sides). Triangular membership functions are a subset of trapezoidal functions. Some practitioners refer to s-, z-, and π-shaped membership functions. These refer to a trapezoid butted against the right end of the x-axis, a trapezoid butted against the left end of the x-axis, and a trapezoidal membership function that isn't butted against either end of the x-axis, respectively. Many other membership function shapes are possible, if memory space and processing bandwidth are sufficient.

Tabular membership functions offer complete flexibility in shape and very fast evaluation time. However, tables take a very large amount of memory space (as many as 256 bytes per label of one system input). The excessive size to specify tabular membership functions makes them impractical for most microcontroller-based fuzzy systems. The CPU12 instruction set includes two instructions (TBL and ETBL) for lookup and interpolation of compressed tables.

The TBL instruction uses 8-bit table entries (y-values) and returns an 8-bit result. The ETBL instruction uses 16-bit table entries (y-values) and returns a 16-bit result. A flexible indexed addressing mode is used to identify the effective address of the data point at the beginning of the line segment, and the data value for the end point of the line segment is the next consecutive memory location (byte for TBL and word for ETBL). In both cases, the B accumulator represents the ratio of (the x-distance from the beginning of the line segment to the lookup point) to (the

x-distance from the beginning of the line segment to the end of the line segment). B is treated as an 8-bit binary fraction with radix point left of the MSB, so each line segment can effectively be divided into 256 pieces. During execution of the TBL or ETBL instruction, the difference between the end point y-value and the beginning point y-value (a signed byte-TBL or word-ETBL) is multiplied by the B accumulator to get an intermediate delta-y term. The result is the y-value of the beginning point, plus this signed intermediate delta-y value.

Because indexed addressing mode is used to identify the starting point of the line segment of interest, there is a great deal of flexibility in constructing tables. A common method is to break the x-axis range into 256 equal width segments and store the y value for each of the resulting 257 endpoints. The 16-bit D accumulator is then used as the x input to the table. The upper eight bits (A) is used as a coarse lookup to find the line segment of interest, and the lower eight bits (B) is used to interpolate within this line segment.

In the program sequence

```
LDX     #TBL_START
LDD     DATA_IN
TBL     A,X
```

The notation A,X causes the TBL instruction to use the A^{th} line segment in the table. The low-order half of D (B) is used by TBL to calculate the exact data value from this line segment. This type of table uses only 257 entries to approximate a table with 16 bits of resolution. This type of table has the disadvantage of equal width line segments, which means just as many points are needed to describe a flat portion of the desired function as are needed for the most active portions.

Another type of table stores x:y coordinate pairs for the endpoints of each linear segment. This type of table may reduce the table storage space compared to the previous fixed-width segments because flat areas of the functions can be specified with a single pair of endpoints. This type of table is a little harder to use with the CPU12 TBL and ETBL instructions because the table instructions expect y-values for segment endpoints to be in consecutive memory locations.

Consider a table made up of an arbitrary number of x:y coordinate pairs, where all values are eight bits. The table is entered with the x-coordinate of the desired point to lookup in the A accumulator. When the table is exited, the corresponding y-value is in the A accumulator. **Figure 9-13** shows one way to work with this type of table.

```
BEGIN       LDY     #TABLE_START-2    ;setup initial table pointer
FIND_LOOP   CMPA    2,+Y              ;find first Xn > XL
                                      ;(auto pre-inc Y by 2)
            BLS     FIND_LOOP         ;loop if XL .le. Xn
* on fall thru, XB@-2,Y YB@-1,Y XE@0,Y and YE@1,Y
            TFR     D,X               ;save XL in high half of X
            CLRA                      ;zero upper half of D
            LDAB    0,Y               ;D = 0:XE
            SUBB    -2,Y              ;D = 0:(XE-XB)
            EXG     D,X               ;X = (XE-XB).. D = XL:junk
            SUBA    -2,Y              ;A = (XL-XB)
            EXG     A,D               ;D = 0:(XL-XB), uses trick of EXG
            FDIV                      ;X reg = (XL-XB)/(XE-XB)
            EXG     D,X               ;move fractional result to A:B
            EXG     A,B               ;byte swap - need result in B
            TSTA                      ;check for rounding
            BPL     NO_ROUND
            INCB                      ;round B up by 1
NO_ROUND    LDAA    1,Y               ;YE
            PSHA                      ;put on stack for TBL later
            LDAA    -1,Y              ;YB
            PSHA                      ;now YB@0,SP and YE@1,SP
            TBL     2,SP+             ;interpolate and deallocate
                                      ;stack temps
```

Figure 9-13. Endpoint Table Handling

The basic idea is to find the segment of interest, temporarily build a 1-segment table of the correct format on the stack, then use TBL with stack relative indexed addressing to interpolate. The most difficult part of the routine is calculating the proportional distance from the beginning of the segment to the lookup point versus the width of the segment ((XL–XB)/(XE–XB)). With this type of table, this calculation must be done at run time. In the previous type of table, this proportional term is an inherent part (the lowest order bits) of the data input to the table.

Some fuzzy theorists have suggested membership functions should be shaped like normal distribution curves or other mathematical functions. This may be correct, but the processing requirements to solve for an intercept on such a function would be unacceptable for most microcontroller-based fuzzy systems. Such a function could be encoded into a table of one of the previously described types.

For many common systems, the thing that is most important about membership function shape is that there is a gradual transition from non-membership to membership as the system input value approaches the central range of the membership function.

Examine the human problem of stopping a car at an intersection. Rules such as “If intersection is close and speed is fast, apply brakes” might be used. The meaning (reflected in membership function shape and position) of the labels “close” and “fast” will be different for a teenager than they are for a grandmother, but both can accomplish the goal of stopping. It makes intuitive sense that the exact shape of a membership function is much less important than the fact that it has gradual boundaries.

9.7.2 Rule Evaluation Variations

The REV and REVW instructions expect fuzzy input and fuzzy output values to be 8-bit values. In a custom fuzzy inference program, higher resolution may be desirable (although this is not a common requirement). The CPU12 includes variations of minimum and maximum operations that work with the fuzzy MIN-MAX inference algorithm. The problem with the fuzzy inference algorithm is that the min and max operations need to store their results differently, so the min and max instructions must work differently or more than one variation of these instructions is needed.

The CPU12 has MIN and MAX instructions for 8- or 16-bit operands, where one operand is in an accumulator and the other is a referenced memory location. There are separate variations that replace the accumulator or the memory location with the result. While processing rule antecedents in a fuzzy inference program, a reference value must be compared to each of the referenced fuzzy inputs, and the smallest input must end up in an accumulator. The instruction

```
EMIND     2,X+    ;process one rule antecedent
```

automates the central operations needed to process rule antecedents. The E stands for extended, so this instruction compares 16-bit operands. The D at the end of the mnemonic stands for the D accumulator, which is both the first operand for the comparison and the destination of the result. The 2,X+ is an indexed addressing specification that says X points to the second operand for the comparison and it will be post-incremented by 2 to point at the next rule antecedent.

When processing rule consequents, the operand in the accumulator must remain constant (in case there is more than one consequent in the rule), and the result of the comparison must replace the referenced fuzzy output in RAM. To do this, use the instruction

```
EMAXM     2,X+    ;process one rule consequent
```

The M at the end of the mnemonic indicates that the result will replace the referenced memory operand. Again, indexed addressing is used. These two instructions would form the working part of a 16-bit resolution fuzzy inference routine.

There are many other methods of performing inference, but none of these are as widely used as the min-max method. Since the CPU12 is a general-purpose microcontroller, the programmer has complete freedom to program any algorithm desired. A custom programmed algorithm would typically take more code space and execution time than a routine that used the built-in REV or REVW instructions.

9.7.3 Defuzzification Variations

Other CPU12 instructions can help with custom defuzzification routines in two main areas:

- The first case is working with operands that are more than eight bits.
- The second case involves using an entirely different approach than weighted average of singletons.

The primary part of the WAV instruction is a multiply and accumulate operation to get the numerator for the weighted average calculation. When working with operands as large as 16 bits, the EMACS instruction could at least be used to automate the multiply and accumulate function. The CPU12 has extended math capabilities, including the EMACS instruction which uses 16-bit input operands and accumulates the sum to a 32-bit memory location and 32-bit by 16-bit divide instructions.

One benefit of the WAV instruction is that both a sum of products and a sum of weights are maintained, while the fuzzy output operand is only accessed from memory once. Since memory access time is such a significant part of execution time, this provides a speed advantage compared to conventional instructions.

The weighted average of singletons is the most commonly used technique in microcontrollers because it is computationally less difficult than most other methods. The simplest method is called max defuzzification, which simply uses the largest fuzzy output as the system result. However, this approach does not take into account any other fuzzy outputs, even when they are almost as true as the chosen max output. Max defuzzification is not a good general choice because it only works for a subset of fuzzy logic applications.

The CPU12 is well suited for more computationally challenging algorithms than weighted average. A 32-bit by 16-bit divide instruction takes 11 or 12 25-MHz cycles for unsigned or signed variations. A 16-bit by 16-bit multiply with a 32-bit result takes only three 25-MHz cycles. The EMACS instruction uses 16-bit operands and accumulates the result in a 32-bit memory location, taking only 12 25-MHz cycles per iteration, including accessing all operands from memory and storing the result to memory.

Fuzzy Logic Support

Appendix A. Instruction Reference

A.1 Introduction

This appendix provides quick references for the instruction set, opcode map, and encoding.

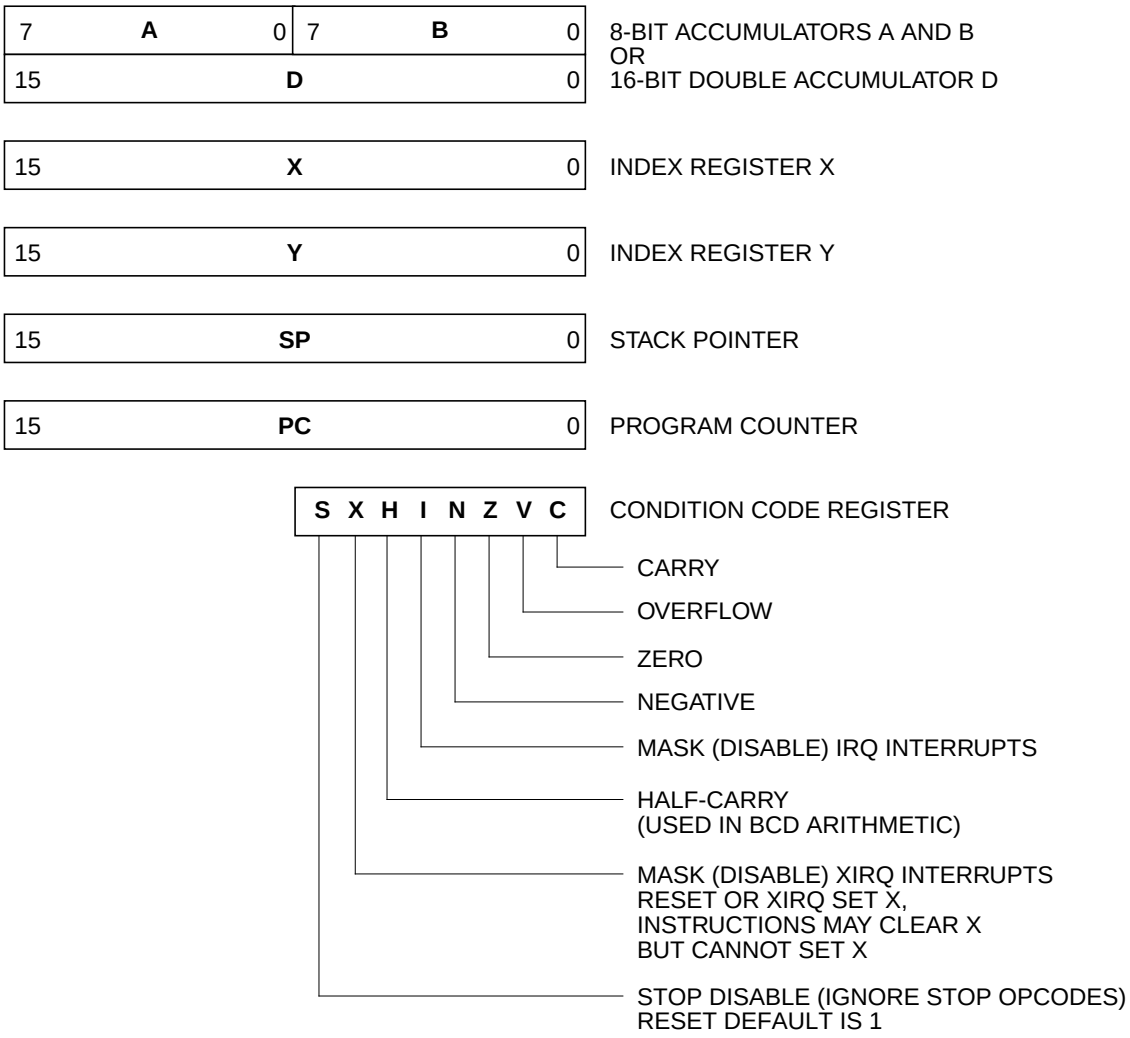
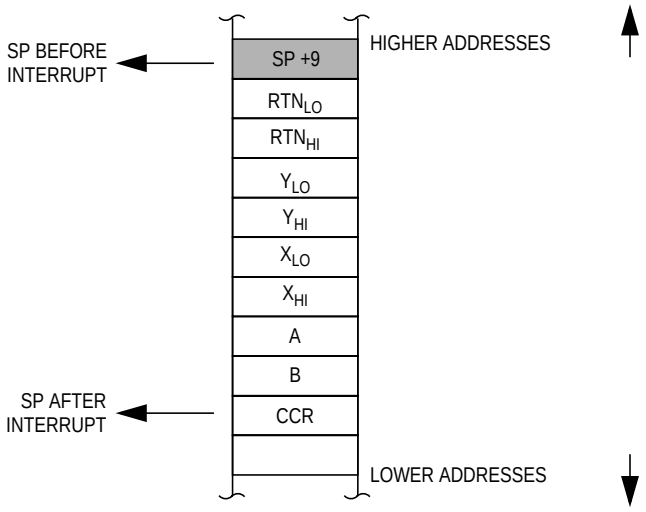


Figure A-1. Programming Model

A.2 Stack and Memory Layout



STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS ODD BEFORE INTERRUPT

SP +8	RTN_{LO}		SP +9
SP +6	Y_{LO}	RTN_{HI}	SP +7
SP +4	X_{LO}	Y_{HI}	SP +5
SP +2	A	X_{HI}	SP +3
SP	CCR	B	SP +1
SP –2			SP –1

STACK UPON ENTRY TO SERVICE ROUTINE
IF SP WAS EVEN BEFORE INTERRUPT

SP +9			SP +10
SP +7	RTN_{HI}	RTN_{LO}	SP +8
SP +5	Y_{HI}	Y_{LO}	SP +6
SP +4	X_{HI}	X_{LO}	SP +4
SP +1	B	A	SP +2
SP –1		CCR	SP

A.3 Interrupt Vector Locations

$FFFE, $FFFF	Power-On (POR) or External Reset
$FFFC, $FFFD	Clock Monitor Reset
$FFFA, $FFFB	Computer Operating Properly (COP Watchdog Reset
$FFF8, $FFF9	Unimplemented Opcode Trap
$FFF6, $FFF7	Software Interrupt Instruction (SWI)
$FFF4, $FFF5	XIRQ
$FFF2, $FFF3	IRQ
$FFC0–$FFF1 (M68HC12)	Device-Specific Interrupt Sources
$FF00–$FFF1 (HCS12)	Device-Specific Interrupt Sources

A.4 Notation Used in Instruction Set Summary

CPU Register Notation

Accumulator A — A or a	Index Register Y — Y or y
Accumulator B — B or b	Stack Pointer — SP, sp, or s
Accumulator D — D or d	Program Counter — PC, pc, or p
Index Register X — X or x	Condition Code Register — CCR or c

Explanation of Italic Expressions in Source Form Column

abc — A or B or CCR
abcdxys — A or B or CCR or D or X or Y or SP. Some assemblers also allow T2 or T3.
abd — A or B or D
abdxys — A or B or D or X or Y or SP
dxys — D or X or Y or SP
msk8 — 8-bit mask, some assemblers require # symbol before value
opr8i — 8-bit immediate value
opr16i — 16-bit immediate value
opr8a — 8-bit address used with direct address mode
opr16a — 16-bit address value
oprx0_xysp — Indexed addressing postbyte code:
- *oprx3*,–*xys* Predecrement X or Y or SP by 1 . . . 8
- *oprx3*,+*xys* Preincrement X or Y or SP by 1 . . . 8
- *oprx3*,*xys*– Postdecrement X or Y or SP by 1 . . . 8
- *oprx3*,*xys*+ Postincrement X or Y or SP by 1 . . . 8
- *oprx5,xysp* 5-bit constant offset from X or Y or SP or PC
- *abd*,*xysp* Accumulator A or B or D offset from X or Y or SP or PC

oprx3 — Any positive integer 1 . . . 8 for pre/post increment/decrement
oprx5 — Any integer in the range –16 . . . +15
oprx9 — Any integer in the range –256 . . . +255
oprx16 — Any integer in the range –32,768 . . . 65,535
page — 8-bit value for PPAGE, some assemblers require # symbol before this value
rel8 — Label of branch destination within –128 to +127 locations
rel9 — Label of branch destination within –256 to +255 locations
rel16 — Any label within 64K memory space
trapnum — Any 8-bit integer in the range $30-$39 or $40-$FF
xys — X or Y or SP
xysp — X or Y or SP or PC

Operators

\+ — Addition
– — Subtraction
• — Logical AND
+ — Logical OR (inclusive)

Continued on next page

Operators (continued)

- ⊕ — Logical exclusive OR
- × — Multiplication
- ÷ — Division
- $\overline{M}$ — Negation. One's complement (invert each bit of M)
- : — Concatenate
  Example: A : B means the 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B.
  A is in the high-order position.
- ⇒ — Transfer
  Example: (A) ⇒ M means the content of accumulator A is transferred to memory location M.
- ⇔ — Exchange
  Example: D ⇔ X means exchange the contents of D with those of X.

Address Mode Notation

- INH — Inherent; no operands in object code
- IMM — Immediate; operand in object code
- DIR — Direct; operand is the lower byte of an address from $0000 to $00FF
- EXT — Operand is a 16-bit address
- REL — Two's complement relative offset; for branch instructions
- IDX — Indexed (no extension bytes); includes:
  5-bit constant offset from X, Y, SP, or PC
  Pre/post increment/decrement by 1 . . . 8
  Accumulator A, B, or D offset
- IDX1 — 9-bit signed offset from X, Y, SP, or PC; 1 extension byte
- IDX2 — 16-bit signed offset from X, Y, SP, or PC; 2 extension bytes
- [IDX2] — Indexed-indirect; 16-bit offset from X, Y, SP, or PC
- [D, IDX] — Indexed-indirect; accumulator D offset from X, Y, SP, or PC

Machine Coding

- `dd` — 8-bit direct address $0000 to $00FF. (High byte assumed to be $00).
- `ee` — High-order byte of a 16-bit constant offset for indexed addressing.
- `eb` — Exchange/Transfer post-byte. See **Table A-5** on page 405.
- `ff` — Low-order eight bits of a 9-bit signed constant offset for indexed addressing, or low-order byte of a 16-bit constant offset for indexed addressing.
- `hh` — High-order byte of a 16-bit extended address.
- `ii` — 8-bit immediate data value.
- `jj` — High-order byte of a 16-bit immediate data value.
- `kk` — Low-order byte of a 16-bit immediate data value.
- `lb` — Loop primitive (DBNE) post-byte. See **Table A-6** on page 406.
- `ll` — Low-order byte of a 16-bit extended address.

mm — 8-bit immediate mask value for bit manipulation instructions. Set bits indicate bits to be affected.

pg — Program page (bank) number used in CALL instruction.

qq — High-order byte of a 16-bit relative offset for long branches.

tn — Trap number $30–$39 or $40–$FF.

rr — Signed relative offset $80 (–128) to $7F (+127). Offset relative to the byte following the relative offset byte, or low-order byte of a 16-bit relative offset for long branches.

xb — Indexed addressing post-byte. See **Table A-3** on page 403 and **Table A-4** on page 404.

Access Detail

Each code letter except (,), and comma equals one CPU cycle. Uppercase = 16-bit operation and lowercase = 8-bit operation. For complex sequences see the *CPU12 Reference Manual* (CPU12RM/AD) for more detailed information.

f — Free cycle, CPU doesn't use bus

g — Read PPAGE internally

I — Read indirect pointer (indexed indirect)

i — Read indirect PPAGE value (CALL indirect only)

n — Write PPAGE internally

O — Optional program word fetch (P) if instruction is misaligned and has an odd number of bytes of object code — otherwise, appears as a free cycle (f); Page 2 prebyte treated as a separate 1-byte instruction

P — Program word fetch (always an aligned-word read)

r — 8-bit data read

R — 16-bit data read

s — 8-bit stack write

S — 16-bit stack write

w — 8-bit data write

W — 16-bit data write

u — 8-bit stack read

U — 16-bit stack read

V — 16-bit vector fetch (always an aligned-word read)

t — 8-bit conditional read (or free cycle)

T — 16-bit conditional read (or free cycle)

x — 8-bit conditional write (or free cycle)

( ) — Indicate a microcode loop

, — Indicates where an interrupt could be honored

Special Cases

PPP/P — Short branch, PPP if branch taken, P if not

OPPP/OPO — Long branch, OPPP if branch taken, OPO if not

Condition Codes Columns

- – — Status bit not affected by operation.
- 0 — Status bit cleared by operation.
- 1 — Status bit set by operation.
- Δ — Status bit affected by operation.
- fl — Status bit may be cleared or remain set, but is not set by operation.
- ⇑ — Status bit may be set or remain cleared, but is not cleared by operation.
- ? — Status bit may be changed by operation but the final state is not defined.
- ! — Status bit used for a special purpose.

Table A-1. Instruction Set Summary (Sheet 1 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
ABA	(A) + (B) ⇒ A Add Accumulators A and B	INH	18 06	OO	OO	– – Δ –	Δ Δ Δ Δ
ABX	(B) + (X) ⇒ X *Translates to* LEAX B,X	IDX	1A E5	Pf	PP[1]	– – – –	– – – –
ABY	(B) + (Y) ⇒ Y *Translates to* LEAY B,Y	IDX	19 ED	Pf	PP[1]	– – – –	– – – –
ADCA *#opr8i* ADCA *opr8a* ADCA *opr16a* ADCA *oprx0_xysp* ADCA *oprx9,xysp* ADCA *oprx16,xysp* ADCA [D,*xysp*] ADCA [*oprx16,xysp*]	(A) + (M) + C ⇒ A Add with Carry to A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	89 ii 99 dd B9 hh ll A9 xb A9 xb ff A9 xb ee ff A9 xb A9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – Δ –	Δ Δ Δ Δ
ADCB *#opr8i* ADCB *opr8a* ADCB *opr16a* ADCB *oprx0_xysp* ADCB *oprx9,xysp* ADCB *oprx16,xysp* ADCB [D,*xysp*] ADCB [*oprx16,xysp*]	(B) + (M) + C ⇒ B Add with Carry to B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C9 ii D9 dd F9 hh ll E9 xb E9 xb ff E9 xb ee ff E9 xb E9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – Δ –	Δ Δ Δ Δ

ADDA *#opr8i*
ADDA *opr8a*
ADDA *opr16a*
ADDA *oprx0_xysp*
ADDA *oprx9,xysp*
ADDA *oprx16,xysp*
ADDA [D,*xysp*]
ADDA [*oprx16,xysp*]
	(A) + (M) ⇒ A
	Add without Carry to A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			8B ii
			9B dd
			BB hh ll
			AB xb
			AB xb ff
			AB xb ee ff
			AB xb
			AB xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						– – Δ –	Δ Δ Δ Δ

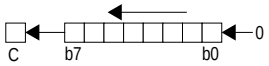
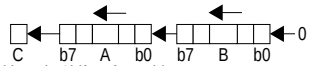
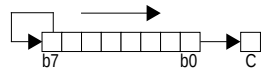
ADDB *#opr8i*
ADDB *opr8a*
ADDB *opr16a*
ADDB *oprx0_xysp*
ADDB *oprx9,xysp*
ADDB *oprx16,xysp*
ADDB [D,*xysp*]
ADDB [*oprx16,xysp*]
	(B) + (M) ⇒ B
	Add without Carry to B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CB ii
			DB dd
			FB hh ll
			EB xb
			EB xb ff
			EB xb ee ff
			EB xb
			EB xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						– – Δ –	Δ Δ Δ Δ

ADDD *#opr16i*
ADDD *opr8a*
ADDD *opr16a*
ADDD *oprx0_xysp*
ADDD *oprx9,xysp*
ADDD *oprx16,xysp*
ADDD [D,*xysp*]
ADDD [*oprx16,xysp*]
	(A:B) + (M:M+1) ⇒ A:B
	Add 16-Bit to D (A:B)
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C3 jj kk
			D3 dd
			F3 hh ll
			E3 xb
			E3 xb ff
			E3 xb ee ff
			E3 xb
			E3 xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						– – – –	Δ Δ Δ Δ

ANDA *#opr8i* ANDA *opr8a* ANDA *opr16a* ANDA *oprx0_xysp* ANDA *oprx9,xysp* ANDA *oprx16,xysp* ANDA [D,*xysp*] ANDA [*oprx16,xysp*]	(A) • (M) ⇒ A Logical AND A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh ll A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –
ANDB *#opr8i* ANDB *opr8a* ANDB *opr16a* ANDB *oprx0_xysp* ANDB *oprx9,xysp* ANDB *oprx16,xysp* ANDB [D,*xysp*] ANDB [*oprx16,xysp*]	(B) • (M) ⇒ B Logical AND B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh ll E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –
ANDCC *#opr8i*	(CCR) • (M) ⇒ CCR Logical AND CCR with Memory	IMM	10 ii	P	P	⇓ ⇓ ⇓ ⇓	⇓ ⇓ ⇓ ⇓

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Table A-1. Instruction Set Summary (Sheet 2 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
ASL *opr16a* ASL *oprx0_xysp* ASL *oprx9,xysp* ASL *oprx16,xysp* ASL [D,*xysp*] ASL [*oprx16,xysp*] ASLA ASLB	 Arithmetic Shift Left Arithmetic Shift Left Accumulator A Arithmetic Shift Left Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	---- 	ΔΔΔΔ
ASLD	 Arithmetic Shift Left Double	INH	59	O	O	----	ΔΔΔΔ
ASR *opr16a* ASR *oprx0_xysp* ASR *oprx9,xysp* ASR *oprx16,xysp* ASR [D,*xysp*] ASR [*oprx16,xysp*] ASRA ASRB	 Arithmetic Shift Right Arithmetic Shift Right Accumulator A Arithmetic Shift Right Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	77 hh ll 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff 47 57	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	---- 	ΔΔΔΔ
BCC *rel8*	Branch if Carry Clear (if C = 0)	REL	24 rr	PPP/P[1]	PPP/P[1]	----	----
BCLR *opr8a*, *msk8* BCLR *opr16a*, *msk8* BCLR *oprx0_xysp*, *msk8* BCLR *oprx9,xysp*, *msk8* BCLR *oprx16,xysp*, *msk8*	(M) • ($\overline{mm}$) ⇒ M Clear Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4D dd mm 1D hh ll mm 0D xb mm 0D xb ff mm 0D xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	rPOw rPPw rPOw rPwP frPwOP	---- 	ΔΔ0-
BCS *rel8*	Branch if Carry Set (if C = 1)	REL	25 rr	PPP/P[1]	PPP/P[1]	----	----
BEQ *rel8*	Branch if Equal (if Z = 1)	REL	27 rr	PPP/P[1]	PPP/P[1]	----	----
BGE *rel8*	Branch if Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	2C rr	PPP/P[1]	PPP/P[1]	----	----
BGND	Place CPU in Background Mode see *CPU12 Reference Manual*	INH	00	VfPPP	VfPPP	----	----
BGT *rel8*	Branch if Greater Than (if Z ┼ (N ⊕ V) = 0) (signed)	REL	2E rr	PPP/P[1]	PPP/P[1]	----	----
BHI *rel8*	Branch if Higher (if C ┼ Z = 0) (unsigned)	REL	22 rr	PPP/P[1]	PPP/P[1]	----	----
BHS *rel8*	Branch if Higher or Same (if C = 0) (unsigned) same function as BCC	REL	24 rr	PPP/P[1]	PPP/P[1]	----	----
BITA #*opr8i* BITA *opr8a* BITA *opr16a* BITA *oprx0_xysp* BITA *oprx9,xysp* BITA *oprx16,xysp* BITA [D,*xysp*] BITA [*oprx16,xysp*]	(A) • (M) Logical AND A with Memory Does not change Accumulator or Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	85 ii 95 dd B5 hh ll A5 xb A5 xb ff A5 xb ee ff A5 xb A5 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	---- 	ΔΔ0-
BITB #*opr8i* BITB *opr8a* BITB *opr16a* BITB *oprx0_xysp* BITB *oprx9,xysp* BITB *oprx16,xysp* BITB [D,*xysp*] BITB [*oprx16,xysp*]	(B) • (M) Logical AND B with Memory Does not change Accumulator or Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C5 ii D5 dd F5 hh ll E5 xb E5 xb ff E5 xb ee ff E5 xb E5 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	---- 	ΔΔ0-
BLE *rel8*	Branch if Less Than or Equal (if Z ┼ (N ⊕ V) = 1) (signed)	REL	2F rr	PPP/P[1]	PPP/P[1]	----	----
BLO *rel8*	Branch if Lower (if C = 1) (unsigned) same function as BCS	REL	25 rr	PPP/P[1]	PPP/P[1]	----	----

Note 1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Table A-1. Instruction Set Summary (Sheet 3 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
BLS *rel8*	Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	23 rr	PPP/P[1]	PPP/P[1]	----	----
BLT *rel8*	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	PPP/P[1]	PPP/P[1]	----	----
BMI *rel8*	Branch if Minus (if N = 1)	REL	2B rr	PPP/P[1]	PPP/P[1]	----	----
BNE *rel8*	Branch if Not Equal (if Z = 0)	REL	26 rr	PPP/P[1]	PPP/P[1]	----	----
BPL *rel8*	Branch if Plus (if N = 0)	REL	2A rr	PPP/P[1]	PPP/P[1]	----	----
BRA *rel8*	Branch Always (if 1 = 1)	REL	20 rr	PPP	PPP	----	----
BRCLR *opr8a, msk8, rel8* BRCLR *opr16a, msk8, rel8* BRCLR *oprx0_xysp, msk8, rel8* BRCLR *oprx9,xysp, msk8, rel8* BRCLR *oprx16,xysp, msk8, rel8*	Branch if (M) • (mm) = 0 (if All Selected Bit(s) Clear)	DIR EXT IDX IDX1 IDX2	4F dd mm rr 1F hh ll mm rr 0F xb mm rr 0F xb ff mm rr 0F xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	rPPP rfPPP rPPP rffPPP frPffPPP	----	----
BRN *rel8*	Branch Never (if 1 = 0)	REL	21 rr	P	P	----	----
BRSET *opr8, msk8, rel8* BRSET *opr16a, msk8, rel8* BRSET *oprx0_xysp, msk8, rel8* BRSET *oprx9,xysp, msk8, rel8* BRSET *oprx16,xysp, msk8, rel8*	Branch if ($\overline{M}$) • (mm) = 0 (if All Selected Bit(s) Set)	DIR EXT IDX IDX1 IDX2	4E dd mm rr 1E hh ll mm rr 0E xb mm rr 0E xb ff mm rr 0E xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	rPPP rfPPP rPPP rffPPP frPffPPP	----	----
BSET *opr8, msk8* BSET *opr16a, msk8* BSET *oprx0_xysp, msk8* BSET *oprx9,xysp, msk8* BSET *oprx16,xysp, msk8*	(M) + (mm) ⇒ M Set Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4C dd mm 1C hh ll mm 0C xb mm 0C xb ff mm 0C xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	rPOw rPPw rPOw rPwP frPwOP	----	ΔΔ0-
BSR *rel8*	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Subroutine address ⇒ PC Branch to Subroutine	REL	07 rr	SPPP	PPPS	----	----
BVC *rel8*	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	PPP/P[1]	PPP/P[1]	----	----
BVS *rel8*	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	PPP/P[1]	PPP/P[1]	----	----
CALL *opr16a, page* CALL *oprx0_xysp, page* CALL *oprx9,xysp, page* CALL *oprx16,xysp, page* CALL [D,*xysp*] CALL [*oprx16, xysp*]	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$ (SP) – 1 ⇒ SP; (PPG) ⇒ $M_{(SP)}$; pg ⇒ PPAGE register; Program address ⇒ PC Call subroutine in extended memory (Program may be located on another expansion memory page.) Indirect modes get program address and new pg value based on pointer.	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg 4B xb 4B xb ee ff	gnSsPPP gnSsPPP gnSsPPP fgnSsPPP fIignSsPPP fIignSsPPP	gnfSsPPP gnfSsPPP gnfSsPPP fgnfSsPPP fIignSsPPP fIignSsPPP	----	----
CBA	(A) – (B) Compare 8-Bit Accumulators	INH	18 17	OO	OO	----	ΔΔΔΔ
CLC	0 ⇒ C *Translates to* ANDCC #$FE	IMM	10 FE	P	P	----	---0
CLI	0 ⇒ I *Translates to* ANDCC #$EF (enables I-bit interrupts)	IMM	10 EF	P	P	---0	----
CLR *oprx16a* CLR *oprx0_xysp* CLR *oprx9,xysp* CLR *oprx16,xysp* CLR [D,*xysp*] CLR [*oprx16,xysp*] CLRA CLRB	0 ⇒ M Clear Memory Location 0 ⇒ A Clear Accumulator A 0 ⇒ B Clear Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff 87 C7	PwO Pw PwO PwP PIfw PIPw O O	wOP Pw PwO PwP PIfPw PIPPw O O	----	0100
CLV	0 ⇒ V *Translates to* ANDCC #$FD	IMM	10 FD	P	P	----	--0-

Note 1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Table A-1. Instruction Set Summary (Sheet 4 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C

CMPA #*oprx8i*
CMPA *oprx8a*
CMPA *oprx16a*
CMPA *oprx0_xysp*
CMPA *oprx9,xysp*
CMPA *oprx16,xysp*
CMPA [D,*xysp*]
CMPA [*oprx16,xysp*]
	(A) – (M)
	Compare Accumulator A with Memory
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			81 ii
			91 dd
			B1 hh ll
			A1 xb
			A1 xb ff
			A1 xb ee ff
			A1 xb
			A1 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						----	Δ Δ Δ Δ

CMPB #*oprx8i*
CMPB *oprx8a*
CMPB *oprx16a*
CMPB *oprx0_xysp*
CMPB *oprx9,xysp*
CMPB *oprx16,xysp*
CMPB [D,*xysp*]
CMPB [*oprx16,xysp*]
	(B) – (M)
	Compare Accumulator B with Memory
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C1 ii
			D1 dd
			F1 hh ll
			E1 xb
			E1 xb ff
			E1 xb ee ff
			E1 xb
			E1 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						----	Δ Δ Δ Δ

COM *opr16a* COM *oprx0_xysp* COM *oprx9,xysp* COM *oprx16,xysp* COM [D,*xysp*] COM [*oprx16,xysp*] COMA COMB	($\overline{M}$) ⇒ M *equivalent to* $FF – (M) ⇒ M 1's Complement Memory Location ($\overline{A}$) ⇒ A Complement Accumulator A ($\overline{B}$) ⇒ B Complement Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	71 hh ll 61 xb 61 xb ff 61 xb ee ff 61 xb 61 xb ee ff 41 51	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	----	Δ Δ 0 1
CPD #*opr16i* CPD *opr8a* CPD *opr16a* CPD *oprx0_xysp* CPD *oprx9,xysp* CPD *oprx16,xysp* CPD [D,*xysp*] CPD [*oprx16,xysp*]	(A:B) – (M:M+1) Compare D to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8C jj kk 9C dd BC hh ll AC xb AC xb ff AC xb ee ff AC xb AC xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ Δ Δ
CPS #*opr16i* CPS *opr8a* CPS *opr16a* CPS *oprx0_xysp* CPS *oprx9,xysp* CPS *oprx16,xysp* CPS [D,*xysp*] CPS [*oprx16,xysp*]	(SP) – (M:M+1) Compare SP to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8F jj kk 9F dd BF hh ll AF xb AF xb ff AF xb ee ff AF xb AF xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ Δ Δ
CPX #*oprx16i* CPX *oprx8a* CPX *oprx16a* CPX *oprx0_xysp* CPX *oprx9,xysp* CPX *oprx16,xysp* CPX [D,*xysp*] CPX [*oprx16,xysp*]	(X) – (M:M+1) Compare X to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8E jj kk 9E dd BE hh ll AE xb AE xb ff AE xb ee ff AE xb AE xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ Δ Δ
CPY #*oprx16i* CPY *oprx8a* CPY *oprx16a* CPY *oprx0_xysp* CPY *oprx9,xysp* CPY *oprx16,xysp* CPY [D,*xysp*] CPY [*oprx16,xysp*]	(Y) – (M:M+1) Compare Y to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8D jj kk 9D dd BD hh ll AD xb AD xb ff AD xb ee ff AD xb AD xb ee ff	PO RPf RPO RPf RPO fRPP fIfRPf fIPRPf	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ Δ Δ
DAA	Adjust Sum to BCD Decimal Adjust Accumulator A	INH	18 07	OfO	OfO	----	Δ Δ ? Δ
DBEQ *abdxys, rel9*	(cntr) – 1⇒ cntr if (cntr) = 0, then Branch else Continue to next instruction Decrement Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	----	----

Table A-1. Instruction Set Summary (Sheet 5 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
DBNE *abdxys, rel9*	(cntr) – 1 ⇒ cntr If (cntr) not = 0, then Branch; else Continue to next instruction Decrement Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	– – – –	– – – –
DEC *opr16a* DEC *oprx0_xysp* DEC *oprx9,xysp* DEC *oprx16,xysp* DEC [D,*xysp*] DEC [*oprx16,xysp*]	(M) – $01 ⇒ M Decrement Memory Location	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	73 hh ll 63 xb 63 xb ff 63 xb ee ff 63 xb 63 xb ee ff	rPwO rPw rPwO frPwP fIfrPw fIPrPw	rOPw rPw rPOw frPPw fIfrPw fIPrPw	– – – –	Δ Δ Δ –
DECA DECB	(A) – $01 ⇒ A Decrement A (B) – $01 ⇒ B Decrement B	INH INH	43 53	O O	O O	– – – –	– – – –
DES	(SP) – $0001 ⇒ SP *Translates to* LEAS –1,SP	IDX	1B 9F	Pf	PP[1]	– – – –	– – – –
DEX	(X) – $0001 ⇒ X Decrement Index Register X	INH	09	O	O	– – – –	– Δ – –
DEY	(Y) – $0001 ⇒ Y Decrement Index Register Y	INH	03	O	O	– – – –	– Δ – –
EDIV	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 by 16 Bit ⇒ 16 Bit Divide (unsigned)	INH	11	fffffffffO	fffffffffO	– – – –	Δ Δ Δ Δ
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 by 16 Bit ⇒ 16 Bit Divide (signed)	INH	18 14	OfffffffffO	OfffffffffO	– – – –	Δ Δ Δ Δ
EMACS *opr16a* [2]	($M_{(X)}$:$M_{(X+1)}$) × ($M_{(Y)}$:$M_{(Y+1)}$) + (M~M+3) ⇒ M~M+3 16 by 16 Bit ⇒ 32 Bit Multiply and Accumulate (signed)	Special	18 12 hh ll	ORROfffRRfWWP	ORROfffRRfWWP	– – – –	Δ Δ Δ Δ
EMAXD *oprx0_xysp* EMAXD *oprx9,xysp* EMAXD *oprx16,xysp* EMAXD [D,*xysp*] EMAXD [*oprx16,xysp*]	MAX((D), (M:M+1)) ⇒ D MAX of 2 Unsigned 16-Bit Values N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1A xb 18 1A xb ff 18 1A xb ee ff 18 1A xb 18 1A xb ee ff	ORPf ORPO OfRPP OfIfRPf OfIPRPf	ORfP ORPO OfRPP OfIfRfP OfIPRfP	– – – –	Δ Δ Δ Δ
EMAXM *oprx0_xysp* EMAXM *oprx9,xysp* EMAXM *oprx16,xysp* EMAXM [D,*xysp*] EMAXM [*oprx16,xysp*]	MAX((D), (M:M+1)) ⇒ M:M+1 MAX of 2 Unsigned 16-Bit Values N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1E xb 18 1E xb ff 18 1E xb ee ff 18 1E xb 18 1E xb ee ff	ORPW ORPWO OfRPWP OfIfRPW OfIPRPW	ORPW ORPWO OfRPWP OfIfRPW OfIPRPW	– – – –	Δ Δ Δ Δ
EMIND *oprx0_xysp* EMIND *oprx9,xysp* EMIND *oprx16,xysp* EMIND [D,*xysp*] EMIND [*oprx16,xysp*]	MIN((D), (M:M+1)) ⇒ D MIN of 2 Unsigned 16-Bit Values N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1B xb 18 1B xb ff 18 1B xb ee ff 18 1B xb 18 1B xb ee ff	ORPf ORPO OfRPP OfIfRPf OfIPRPf	ORfP ORPO OfRPP OfIfRfP OfIPRfP	– – – –	Δ Δ Δ Δ
EMINM *oprx0_xysp* EMINM *oprx9,xysp* EMINM *oprx16,xysp* EMINM [D,*xysp*] EMINM [*oprx16,xysp*]	MIN((D), (M:M+1)) ⇒ M:M+1 MIN of 2 Unsigned 16-Bit Values N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1F xb 18 1F xb ff 18 1F xb ee ff 18 1F xb 18 1F xb ee ff	ORPW ORPWO OfRPWP OfIfRPW OfIPRPW	ORPW ORPWO OfRPWP OfIfRPW OfIPRPW	– – – –	Δ Δ Δ Δ
EMUL	(D) × (Y) ⇒ Y:D 16 by 16 Bit Multiply (unsigned)	INH	13	ffO	ffO	– – – –	Δ Δ – Δ
EMULS	(D) × (Y) ⇒ Y:D 16 by 16 Bit Multiply (signed)	INH	18 13	OfO (if followed by page 2 instruction) OffO	OfO OfO	– – – –	Δ Δ – Δ
EORA #*opr8i* EORA *opr8a* EORA *opr16a* EORA *oprx0_xysp* EORA *oprx9,xysp* EORA *oprx16,xysp* EORA [D,*xysp*] EORA [*oprx16,xysp*]	(A) ⊕ (M) ⇒ A Exclusive-OR A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	88 ii 98 dd B8 hh ll A8 xb A8 xb ff A8 xb ee ff A8 xb A8 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –

Notes:
1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.
2. *opr16a* is an extended address specification. Both X and Y point to source operands.

Table A-1. Instruction Set Summary (Sheet 6 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
EORB *#oprx8i* EORB *oprx8a* EORB *oprx16a* EORB *oprx0_xysp* EORB *oprx9,xysp* EORB *oprx16,xysp* EORB [D,*xysp*] EORB [*oprx16,xysp*]	(B) ⊕ (M) ⇒ B Exclusive-OR B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C8 ii D8 dd F8 hh ll E8 xb E8 xb ff E8 xb ee ff E8 xb E8 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –
ETBL *oprx0_xysp*	(M:M+1)+ [(B)×((M+2:M+3) – (M:M+1))] ⇒ D 16-Bit Table Lookup and Interpolate Initialize B, and index before ETBL. <ea> points at first table entry (M:M+1) and B is fractional part of lookup value (no indirect addr. modes or extensions allowed)	IDX	18 3F xb	ORRfffffP	ORRfffffP	– – – –	Δ Δ – Δ ? C Bit is undefined in HC12
EXG *abcdxys,abcdxys*	(r1) ⇔ (r2) (if r1 and r2 same size) *or* $00:(r1) ⇒ r2 (if r1=8-bit; r2=16-bit) *or* ($r1_{low}$) ⇔ (r2) (if r1=16-bit; r2=8-bit) r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	P	P	– – – –	– – – –
FDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Fractional Divide	INH	18 11	Offffffffff0	Offffffffff0	– – – –	– Δ Δ Δ
IBEQ *abdxys, rel9*	(cntr) + 1⇒ cntr If (cntr) = 0, then Branch else Continue to next instruction Increment Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	– – – –	– – – –
IBNE *abdxys, rel9*	(cntr) + 1⇒ cntr if (cntr) not = 0, then Branch; else Continue to next instruction Increment Counter and Branch if ≠ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	– – – –	– – – –
IDIV	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Integer Divide (unsigned)	INH	18 10	Offffffffff0	Offffffffff0	– – – –	– Δ 0 Δ
IDIVS	(D) ÷ (X) ⇒ X; Remainder ⇒ D 16 by 16 Bit Integer Divide (signed)	INH	18 15	Offffffffff0	Offffffffff0	– – – –	Δ Δ Δ Δ
INC *oprx16a* INC *oprx0_xysp* INC *oprx9,xysp* INC *oprx16,xysp* INC [D,*xysp*] INC [*oprx16,xysp*] INCA INCB	(M) + $01 ⇒ M Increment Memory Byte (A) + $01 ⇒ A Increment Acc. A (B) + $01 ⇒ B Increment Acc. B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	72 hh ll 62 xb 62 xb ff 62 xb ee ff 62 xb 62 xb ee ff 42 52	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	– – – –	Δ Δ Δ –
INS	(SP) + $0001 ⇒ SP *Translates to* LEAS 1,SP	IDX	1B 81	Pf	PP[1]	– – – –	– – – –
INX	(X) + $0001 ⇒ X Increment Index Register X	INH	08	O	O	– – – –	– Δ – –
INY	(Y) + $0001 ⇒ Y Increment Index Register Y	INH	02	O	O	– – – –	– Δ – –
JMP *oprx16a* JMP *oprx0_xysp* JMP *oprx9,xysp* JMP *oprx16,xysp* JMP [D,*xysp*] JMP [*oprx16,xysp*]	Routine address ⇒ PC Jump	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	06 hh ll 05 xb 05 xb ff 05 xb ee ff 05 xb 05 xb ee ff	PPP PPP PPP fPPP fIfPPP fIfPPP	PPP PPP PPP fPPP fIfPPP fIfPPP	– – – –	– – – –

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Table A-1. Instruction Set Summary (Sheet 7 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
JSR *opr8a* JSR *opr16a* JSR *oprx0_xysp* JSR *oprx9,xysp* JSR *oprx16,xysp* JSR [D,*xysp*] JSR [*oprx16,xysp*]	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; Subroutine address ⇒ PC Jump to Subroutine	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	17 dd 16 hh ll 15 xb 15 xb ff 15 xb ee ff 15 xb 15 xb ee ff	SPPP SPPP PPPS PPPS fPPPS fIfPPPS fIfPPPS	PPPS PPPS PPPS PPPS fPPPS fIfPPPS fIfPPPS	----	----
LBCC *rel16*	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBCS *rel16*	Long Branch if Carry Set (if C = 1)	REL	18 25 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBEQ *rel16*	Long Branch if Equal (if Z = 1)	REL	18 27 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBGE *rel16*	Long Branch Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	18 2C qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBGT *rel16*	Long Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	18 2E qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBHI *rel16*	Long Branch if Higher (if C + Z = 0) (unsigned)	REL	18 22 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBHS *rel16*	Long Branch if Higher or Same (if C = 0) (unsigned) same function as LBCC	REL	18 24 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBLE *rel16*	Long Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	18 2F qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBLO *rel16*	Long Branch if Lower (if C = 1) (unsigned) *same function as LBCS*	REL	18 25 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBLS *rel16*	Long Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	18 23 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBLT *rel16*	Long Branch if Less Than (if N ⊕ V = 1) (signed)	REL	18 2D qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBMI *rel16*	Long Branch if Minus (if N = 1)	REL	18 2B qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBNE *rel16*	Long Branch if Not Equal (if Z = 0)	REL	18 26 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBPL *rel16*	Long Branch if Plus (if N = 0)	REL	18 2A qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBRA *rel16*	Long Branch Always (if 1=1)	REL	18 20 qq rr	OPPP	OPPP	----	----
LBRN *rel16*	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	OPO	OPO	----	----
LBVC *rel16*	Long Branch if Overflow Bit Clear (if V=0)	REL	18 28 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----
LBVS *rel16*	Long Branch if Overflow Bit Set (if V = 1)	REL	18 29 qq rr	OPPP/OPO[1]	OPPP/OPO[1]	----	----

LDAA #*opr8i*
LDAA *opr8a*
LDAA *opr16a*
LDAA *oprx0_xysp*
LDAA *oprx9,xysp*
LDAA *oprx16,xysp*
LDAA [D,*xysp*]
LDAA [*oprx16,xysp*]
	(M) ⇒ A
	Load Accumulator A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			86 ii
			96 dd
			B6 hh ll
			A6 xb
			A6 xb ff
			A6 xb ee ff
			A6 xb
			A6 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						----	Δ Δ 0 –

LDAB #*opr8i*
LDAB *opr8a*
LDAB *opr16a*
LDAB *oprx0_xysp*
LDAB *oprx9,xysp*
LDAB *oprx16,xysp*
LDAB [D,*xysp*]
LDAB [*oprx16,xysp*]
	(M) ⇒ B
	Load Accumulator B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C6 ii
			D6 dd
			F6 hh ll
			E6 xb
			E6 xb ff
			E6 xb ee ff
			E6 xb
			E6 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						----	Δ Δ 0 –

LDD #*opr16i*
LDD *opr8a*
LDD *opr16a*
LDD *oprx0_xysp*
LDD *oprx9,xysp*
LDD *oprx16,xysp*
LDD [D,*xysp*]
LDD [*oprx16,xysp*]
	(M:M+1) ⇒ A:B
	Load Double Accumulator D (A:B)
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CC jj kk
			DC dd
			FC hh ll
			EC xb
			EC xb ff
			EC xb ee ff
			EC xb
			EC xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						----	Δ Δ 0 –

Note 1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

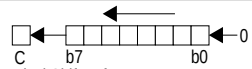
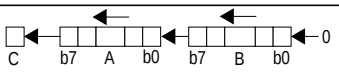
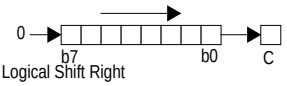
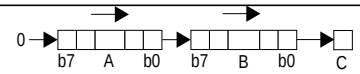
Table A-1. Instruction Set Summary (Sheet 8 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C

LDS #*oprx16i*
LDS *opr8a*
LDS *opr16a*
LDS *oprx0_xysp*
LDS *oprx9,xysp*
LDS *oprx16,xysp*
LDS [D,*xysp*]
LDS [*oprx16,xysp*]
	(M:M+1) ⇒ SP
	Load Stack Pointer
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CF jj kk
			DF dd
			FF hh ll
			EF xb
			EF xb ff
			EF xb ee ff
			EF xb
			EF xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						– – – –	Δ Δ 0 –

LDX #*oprx16i*
LDX *opr8a*
LDX *opr16a*
LDX *oprx0_xysp*
LDX *oprx9,xysp*
LDX *oprx16,xysp*
LDX [D,*xysp*]
LDX [*oprx16,xysp*]
	(M:M+1) ⇒ X
	Load Index Register X
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CE jj kk
			DE dd
			FE hh ll
			EE xb
			EE xb ff
			EE xb ee ff
			EE xb
			EE xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						– – – –	Δ Δ 0 –

LDY #*oprx16i*
LDY *opr8a*
LDY *opr16a*
LDY *oprx0_xysp*
LDY *oprx9,xysp*
LDY *oprx16,xysp*
LDY [D,*xysp*]
LDY [*oprx16,xysp*]
	(M:M+1) ⇒ Y
	Load Index Register Y
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			CD jj kk
			DD dd
			FD hh ll
			ED xb
			ED xb ff
			ED xb ee ff
			ED xb
			ED xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						– – – –	Δ Δ 0 –

LEAS *oprx0_xysp* LEAS *oprx9,xysp* LEAS *oprx16,xysp*	Effective Address ⇒ SP Load Effective Address into SP	IDX IDX1 IDX2	1B xb 1B xb ff 1B xb ee ff	Pf PO PP	PP[1] PO PP	– – – –	– – – –
LEAX *oprx0_xysp* LEAX *oprx9,xysp* LEAX *oprx16,xysp*	Effective Address ⇒ X Load Effective Address into X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	Pf PO PP	PP[1] PO PP	– – – –	– – – –
LEAY *oprx0_xysp* LEAY *oprx9,xysp* LEAY *oprx16,xysp*	Effective Address ⇒ Y Load Effective Address into Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	Pf PO PP	PP[1] PO PP	– – – –	– – – –
LSL *opr16a* LSL *oprx0_xysp* LSL *oprx9,xysp* LSL *oprx16,xysp* LSL [D,*xysp*] LSL [*oprx16,xysp*] LSLA LSLB	 Logical Shift Left same function as ASL Logical Shift Accumulator A to Left Logical Shift Accumulator B to Left	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rPwO rPw rPwO frPPw fIfrPw fIPrPw 0 0	rOPw rPw rPOw frPPw fIfrPw fIPrPw 0 0	– – – –	Δ Δ Δ Δ
LSLD	 Logical Shift Left D Accumulator same function as ASLD	INH	59	0	0	– – – –	Δ Δ Δ Δ
LSR *opr16a* LSR *oprx0_xysp* LSR *oprx9,xysp* LSR *oprx16,xysp* LSR [D,*xysp*] LSR [*oprx16,xysp*] LSRA LSRB	 Logical Shift Right Logical Shift Accumulator A to Right Logical Shift Accumulator B to Right	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	74 hh ll 64 xb 64 xb ff 64 xb ee ff 64 xb 64 xb ee ff 44 54	rPwO rPw rPwO frPwP fIfrPw fIPrPw 0 0	rOPw rPw rPOw frPPw fIfrPw fIPrPw 0 0	– – – –	0 Δ Δ Δ
LSRD	 Logical Shift Right D Accumulator	INH	49	0	0	– – – –	0 Δ Δ Δ
MAXA *oprx0_xysp* MAXA *oprx9,xysp* MAXA *oprx16,xysp* MAXA [D,*xysp*] MAXA [*oprx16,xysp*]	MAX((A), (M)) ⇒ A MAX of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 18 xb 18 18 xb ff 18 18 xb ee ff 18 18 xb 18 18 xb ee ff	OrPf OrPO OfrPP OfIfrPf OfIPrPf	OrfP OrPO OfrPP OfIfrfP OfIPrfP	– – – –	Δ Δ Δ Δ

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Table A-1. Instruction Set Summary (Sheet 9 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
MAXM *oprx0_xysp* MAXM *oprx9,xysp* MAXM *oprx16,xysp* MAXM [D,*xysp*] MAXM [*oprx16,xysp*]	MAX((A), (M)) ⇒ M MAX of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1C xb 18 1C xb ff 18 1C xb ee ff 18 1C xb 18 1C xb ee ff	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	– – – –	Δ Δ Δ Δ
MEM	μ (grade) ⇒ $M_{(Y)}$; (X) + 4 ⇒ X; (Y) + 1 ⇒ Y; A unchanged if (A) < P1 or (A) > P2 then μ = 0, else μ = MIN[((A) – P1)×S1, (P2 – (A))×S2, $FF] where: A = current crisp input value; X points at 4-byte data structure that describes a trapezoidal membership function (P1, P2, S1, S2); Y points at fuzzy input (RAM location). See *CPU12 Reference Manual* for special cases.	Special	01	RRfOw	RRfOw	– – ? –	? ? ? ?
MINA *oprx0_xysp* MINA *oprx9,xysp* MINA *oprx16,xysp* MINA [D,*xysp*] MINA [*oprx16,xysp*]	MIN((A), (M)) ⇒ A MIN of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 19 xb 18 19 xb ff 18 19 xb ee ff 18 19 xb 18 19 xb ee ff	OrPf OrPO OfrPP OfIfrPf OfIPrPf	OrfP OrPO OfrPP OfIfrfP OfIPrfP	– – – –	Δ Δ Δ Δ
MINM *oprx0_xysp* MINM *oprx9,xysp* MINM *oprx16,xysp* MINM [D,*xysp*] MINM [*oprx16,xysp*]	MIN((A), (M)) ⇒ M MIN of 2 Unsigned 8-Bit Values N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1D xb 18 1D xb ff 18 1D xb ee ff 18 1D xb 18 1D xb ee ff	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	OrPw OrPwO OfrPwP OfIfrPw OfIPrPw	– – – –	Δ Δ Δ Δ

MOVB *#opr8, opr16a*[1]
MOVB *#opr8i, oprx0_xysp*[1]
MOVB *opr16a, opr16a*[1]
MOVB *opr16a, oprx0_xysp*[1]
MOVB *oprx0_xysp, opr16a*[1]
MOVB *oprx0_xysp, oprx0_xysp*[1]
	$(M_1) \Rightarrow M_2$
	Memory to Memory Byte-Move (8-Bit)
		IMM-EXT
		IMM-IDX
		EXT-EXT
		EXT-IDX
		IDX-EXT
		IDX-IDX
			18 0B ii hh ll
			18 08 xb ii
			18 0C hh ll hh ll
			18 09 xb hh ll
			18 0D xb hh ll
			18 0A xb xb
				OPwP
				OPwO
				OrPwPO
				OPrPw
				OrPwP
				OrPwO
					OPwP
					OPwO
					OrPwPO
					OPrPw
					OrPwP
					OrPwO
						– – – –	– – – –

MOVW *#oprx16, opr16a*[1]
MOVW *#opr16i, oprx0_xysp*[1]
MOVW *opr16a, opr16a*[1]
MOVW *opr16a, oprx0_xysp*[1]
MOVW *oprx0_xysp, opr16a*[1]
MOVW *oprx0_xysp, oprx0_xysp*[1]
	$(M:M+1_1) \Rightarrow M:M+1_2$
	Memory to Memory Word-Move (16-Bit)
		IMM-EXT
		IMM-IDX
		EXT-EXT
		EXT-IDX
		IDX-EXT
		IDX-IDX
			18 03 jj kk hh ll
			18 00 xb jj kk
			18 04 hh ll hh ll
			18 01 xb hh ll
			18 05 xb hh ll
			18 02 xb xb
				OPWPO
				OPPW
				ORPWPO
				OPRPW
				ORPWP
				ORPWO
					OPWPO
					OPPW
					ORPWPO
					OPRPW
					ORPWP
					ORPWO
						– – – –	– – – –

MUL	(A) × (B) ⇒ A:B 8 by 8 Unsigned Multiply	INH	12	O	ffO	– – – –	– – – Δ
NEG *opr16a* NEG *oprx0_xysp* NEG *oprx9,xysp* NEG *oprx16,xysp* NEG [D,*xysp*] NEG [*oprx16,xysp*] NEGA NEGB	0 – (M) ⇒ M *equivalent to* $(\overline{M}) + 1 \Rightarrow M$ Two's Complement Negate 0 – (A) ⇒ A *equivalent to* $(\overline{A}) + 1 \Rightarrow A$ Negate Accumulator A 0 – (B) ⇒ B *equivalent to* $(\overline{B}) + 1 \Rightarrow B$ Negate Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	70 hh ll 60 xb 60 xb ff 60 xb ee ff 60 xb 60 xb ee ff 40 50	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	– – – –	Δ Δ Δ Δ
NOP	No Operation	INH	A7	O	O	– – – –	– – – –
ORAA *#opr8i* ORAA *opr8a* ORAA *opr16a* ORAA *oprx0_xysp* ORAA *oprx9,xysp* ORAA *oprx16,xysp* ORAA [D,*xysp*] ORAA [*oprx16,xysp*]	(A) + (M) ⇒ A Logical OR A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8A ii 9A dd BA hh ll AA xb AA xb ff AA xb ee ff AA xb AA xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –

Note 1. The first operand in the source code statement specifies the source for the move.

Table A-1. Instruction Set Summary (Sheet 10 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
ORAB *#opr8i* ORAB *opr8a* ORAB *opr16a* ORAB *oprx0_xysp* ORAB *oprx9,xysp* ORAB *oprx16,xysp* ORAB [D,*xysp*] ORAB [*oprx16,xysp*]	(B) + (M) ⇒ B Logical OR B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CA ii DA dd FA hh ll EA xb EA xb ff EA xb ee ff EA xb EA xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ 0 –
ORCC *#opr8i*	(CCR) + M ⇒ CCR Logical OR CCR with Memory	IMM	14 ii	P	P	⇑ – ⇑ ⇑	⇑ ⇑ ⇑ ⇑
PSHA	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$ Push Accumulator A onto Stack	INH	36	Os	Os	– – – –	– – – –
PSHB	(SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$ Push Accumulator B onto Stack	INH	37	Os	Os	– – – –	– – – –
PSHC	(SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$ Push CCR onto Stack	INH	39	Os	Os	– – – –	– – – –
PSHD	(SP) – 2 ⇒ SP; (A:B) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push D Accumulator onto Stack	INH	3B	OS	OS	– – – –	– – – –
PSHX	(SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push Index Register X onto Stack	INH	34	OS	OS	– – – –	– – – –
PSHY	(SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$ Push Index Register Y onto Stack	INH	35	OS	OS	– – – –	– – – –
PULA	($M_{(SP)}$) ⇒ A; (SP) + 1 ⇒ SP Pull Accumulator A from Stack	INH	32	ufO	ufO	– – – –	– – – –
PULB	($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP Pull Accumulator B from Stack	INH	33	ufO	ufO	– – – –	– – – –
PULC	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP Pull CCR from Stack	INH	38	ufO	ufO	Δ ⇓ Δ Δ	Δ Δ Δ Δ
PULD	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ A:B; (SP) + 2 ⇒ SP Pull D from Stack	INH	3A	UfO	UfO	– – – –	– – – –
PULX	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 2 ⇒ SP Pull Index Register X from Stack	INH	30	UfO	UfO	– – – –	– – – –
PULY	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 2 ⇒ SP Pull Index Register Y from Stack	INH	31	UfO	UfO	– – – –	– – – –
REV	MIN-MAX rule evaluation Find smallest rule input (MIN). Store to rule outputs unless fuzzy output is already larger (MAX). For rule weights see REVW. Each rule input is an 8-bit offset from the base address in Y. Each rule output is an 8-bit offset from the base address in Y. $FE separates rule inputs from rule outputs. $FF terminates the rule list. REV may be interrupted.	Special	18 3A	Orf(t,tx)O (exit + re-entry replaces comma above if interrupted) ff + Orf(t,	Orf(t,tx)O ff + Orf(t,	– – ? –	? ? Δ ?
REVW	MIN-MAX rule evaluation Find smallest rule input (MIN), Store to rule outputs unless fuzzy output is already larger (MAX). Rule weights supported, optional. Each rule input is the 16-bit address of a fuzzy input. Each rule output is the 16-bit address of a fuzzy output. The value $FFFE separates rule inputs from rule outputs. $FFFF terminates the rule list. REVW may be interrupted.	Special	18 3B	ORf(t,Tx)O (loop to read weight if enabled) (r,RfRf) (exit + re-entry replaces comma above if interrupted) ffff + ORf(t,	ORf(t,Tx)O (r,RfRf) fff + ORf(t,	– – ? –	? ? Δ !

Table A-1. Instruction Set Summary (Sheet 11 of 14)

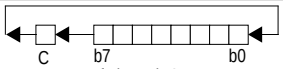
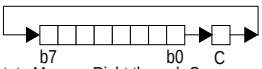
Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
ROL *opr16a* ROL *oprx0_xysp* ROL *oprx9,xysp* ROL *oprx16,xysp* ROL [D,*xysp*] ROL [*oprx16,xysp*] ROLA ROLB	 Rotate Memory Left through Carry Rotate A Left through Carry Rotate B Left through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	75 hh ll 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff 45 55	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	– – – –	Δ Δ Δ Δ
ROR *opr16a* ROR *oprx0_xysp* ROR *oprx9,xysp* ROR *oprx16,xysp* ROR [D,*xysp*] ROR [*oprx16,xysp*] RORA RORB	 Rotate Memory Right through Carry Rotate A Right through Carry Rotate B Right through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	76 hh ll 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff 46 56	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	– – – –	Δ Δ Δ Δ
RTC	($M_{(SP)}$) ⇒ PPAGE; (SP) + 1 ⇒ SP; ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) + 2 ⇒ SP Return from Call	INH	0A	uUnfPPP	uUnPPP	– – – –	– – – –
RTI	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ B:A; (SP) + 2 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ X_H:X_L; (SP) + 4 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) – 2 ⇒ SP ($M_{(SP)}$:$M_{(SP+1)}$) ⇒ Y_H:Y_L; (SP) + 4 ⇒ SP Return from Interrupt	INH	0B	uUUUUPPP (with interrupt pending) uUUUUVfPPP	uUUUUPPP uUUUUfVfPPP	Δ ⇓ Δ Δ	Δ Δ Δ Δ
RTS	($M_{(SP)}$:$M_{(SP+1)}$) ⇒ PC_H:PC_L; (SP) + 2 ⇒ SP Return from Subroutine	INH	3D	UfPPP	UfPPP	– – – –	– – – –
SBA	(A) – (B) ⇒ A Subtract B from A	INH	18 16	OO	OO	– – – –	Δ Δ Δ Δ
SBCA #*opr8i* SBCA *opr8a* SBCA *opr16a* SBCA *oprx0_xysp* SBCA *oprx9,xysp* SBCA *oprx16,xysp* SBCA [D,*xysp*] SBCA [*oprx16,xysp*]	(A) – (M) – C ⇒ A Subtract with Borrow from A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	82 ii 92 dd B2 hh ll A2 xb A2 xb ff A2 xb ee ff A2 xb A2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ Δ Δ
SBCB #*opr8i* SBCB *opr8a* SBCB *opr16a* SBCB *oprx0_xysp* SBCB *oprx9,xysp* SBCB *oprx16,xysp* SBCB [D,*xysp*] SBCB [*oprx16,xysp*]	(B) – (M) – C ⇒ B Subtract with Borrow from B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C2 ii D2 dd F2 hh ll E2 xb E2 xb ff E2 xb ee ff E2 xb E2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	– – – –	Δ Δ Δ Δ
SEC	1 ⇒ C *Translates to* ORCC #$01	IMM	14 01	P	P	– – – –	– – – 1
SEI	1 ⇒ I; (inhibit I interrupts) *Translates to* ORCC #$10	IMM	14 10	P	P	– – – 1	– – – –
SEV	1 ⇒ V *Translates to* ORCC #$02	IMM	14 02	P	P	– – – –	– – 1 –
SEX *abc,dxys*	$00:(r1) ⇒ r2 if r1, bit 7 is 0 *or* $FF:(r1) ⇒ r2 if r1, bit 7 is 1 Sign Extend 8-bit r1 to 16-bit r2 r1 may be A, B, or CCR r2 may be D, X, Y, or SP *Alternate mnemonic for* TFR r1, r2	INH	B7 eb	P	P	– – – –	– – – –

Table A-1. Instruction Set Summary (Sheet 12 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C
STAA *oprx0_xysp* STAA *oprx9,xysp* STAA *oprx16,xysp* STAA [D,*xysp*] STAA [*oprx16,xysp*]	(A) ⇒ M Store Accumulator A to Memory	IDX IDX1 IDX2 [D,IDX] [IDX2]	6A xb 6A xb ff 6A xb ee ff 6A xb 6A xb ee ff	Pw PwO PwP PIfw PIPw	Pw wOP PwO PwP PIfPw PIPPw	– – – –	Δ Δ 0 –
STAB *oprx0_xysp* STAB *oprx9,xysp* STAB *oprx16,xysp* STAB [D,*xysp*] STAB [*oprx16,xysp*]	(B) ⇒ M Store Accumulator B to Memory	IDX IDX1 IDX2 [D,IDX] [IDX2]	6B xb 6B xb ff 6B xb ee ff 6B xb 6B xb ee ff	Pw PwO PwP PIfw PIPw	Pw wOP PwO PwP PIfPw PIPPw	– – – –	Δ Δ 0 –
STD *oprx0_xysp* STD *oprx9,xysp* STD *oprx16,xysp* STD [D,*xysp*] STD [*oprx16,xysp*]	(A) ⇒ M, (B) ⇒ M+1 Store Double Accumulator	IDX IDX1 IDX2 [D,IDX] [IDX2]	6C xb 6C xb ff 6C xb ee ff 6C xb 6C xb ee ff	PW PWO PWP PIfW PIPW	PW WOP PWO PWP PIfPW PIPPW	– – – –	Δ Δ 0 –
STOP	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (B:A) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$; STOP All Clocks Registers stacked to allow quicker recovery by interrupt. If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP.	INH	18 3E	(entering STOP) 00SSSSsf (exiting STOP) fVfPPP (continue) ff (if STOP disabled) 00	(entering STOP) 00SSSfSs (exiting STOP) fVfPPP (continue) f0 (if STOP disabled) 00	– – – –	– – – –
STS *oprx0_xysp* STS *oprx9,xysp* STS *oprx16,xysp* STS [D,*xysp*] STS [*oprx16,xysp*]	(SP_H:SP_L) ⇒ M:M+1 Store Stack Pointer	IDX IDX1 IDX2 [D,IDX] [IDX2]	6F xb 6F xb ff 6F xb ee ff 6F xb 6F xb ee ff	PW PWO PWP PIfW PIPW	PW WOP PWO PWP PIfPW PIPPW	– – – –	Δ Δ 0 –
STX *oprx0_xysp* STX *oprx9,xysp* STX *oprx16,xysp* STX [D,*xysp*] STX [*oprx16,xysp*]	(X_H:X_L) ⇒ M:M+1 Store Index Register X	IDX IDX1 IDX2 [D,IDX] [IDX2]	6E xb 6E xb ff 6E xb ee ff 6E xb 6E xb ee ff	PW PWO PWP PIfW PIPW	PW WOP PWO PWP PIfPW PIPPW	– – – –	Δ Δ 0 –
STY *oprx0_xysp* STY *oprx9,xysp* STY *oprx16,xysp* STY [D,*xysp*] STY [*oprx16,xysp*]	(Y_H:Y_L) ⇒ M:M+1 Store Index Register Y	IDX IDX1 IDX2 [D,IDX] [IDX2]	6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	PW PWO PWP PIfW PIPW	PW WOP PWO PWP PIfPW PIPPW	– – – –	Δ Δ 0 –

SUBA #*oprx0_xysp*
SUBA *oprx9,xysp*
SUBA *oprx16,xysp*
SUBA [D,*xysp*]
SUBA [*oprx16,xysp*]
	(A) – (M) ⇒ A
	Subtract Memory from Accumulator A
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			80 ii
			90 dd
			B0 hh ll
			A0 xb
			A0 xb ff
			A0 xb ee ff
			A0 xb
			A0 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						– – – –	Δ Δ Δ Δ

Table A-1. Instruction Set Summary (Sheet 13 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12	Access Detail M68HC12	S X H I	N Z V C

SUBB *#oprx8i*
SUBB *oprx8a*
SUBB *oprx16a*
SUBB *oprx0_xysp*
SUBB *oprx9,xysp*
SUBB *oprx16,xysp*
SUBB [D,*xysp*]
SUBB [*oprx16,xysp*]
	(B) – (M) ⇒ B
	Subtract Memory from Accumulator B
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			C0 ii
			D0 dd
			F0 hh ll
			E0 xb
			E0 xb ff
			E0 xb ee ff
			E0 xb
			E0 xb ee ff
				P
				rPf
				rPO
				rPf
				rPO
				frPP
				fIfrPf
				fIPrPf
					P
					rfP
					rOP
					rfP
					rPO
					frPP
					fIfrfP
					fIPrfP
						-----	ΔΔΔΔ

SUBD *#oprx16i*
SUBD *oprx8a*
SUBD *oprx16a*
SUBD *oprx0_xysp*
SUBD *oprx9,xysp*
SUBD *oprx16,xysp*
SUBD [D,*xysp*]
SUBD [*oprx16,xysp*]
	(D) – (M:M+1) ⇒ D
	Subtract Memory from D (A:B)
		IMM
		DIR
		EXT
		IDX
		IDX1
		IDX2
		[D,IDX]
		[IDX2]
			83 jj kk
			93 dd
			B3 hh ll
			A3 xb
			A3 xb ff
			A3 xb ee ff
			A3 xb
			A3 xb ee ff
				PO
				RPf
				RPO
				RPf
				RPO
				fRPP
				fIfRPf
				fIPRPf
					OP
					RfP
					ROP
					RfP
					RPO
					fRPP
					fIfRfP
					fIPRfP
						-----	ΔΔΔΔ

SWI	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (B:A) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$ 1 ⇒ I; (SWI Vector) ⇒ PC Software Interrupt	INH	3F	VSPSSPSsP* (for Reset) VfPPP	VSPSSPSsP* VfPPP	---1 11–1	----- -----
*The CPU also uses the SWI microcode sequence for hardware interrupts and unimplemented opcode traps. Reset uses the VfPPP variation of this sequence.							
TAB	(A) ⇒ B Transfer A to B	INH	18 0E	OO	OO	-----	ΔΔ0–
TAP	(A) ⇒ CCR *Translates to* TFR A , CCR	INH	B7 02	P	P	Δ⇓ΔΔ	ΔΔΔΔ
TBA	(B) ⇒ A Transfer B to A	INH	18 0F	OO	OO	-----	ΔΔ0–
TBEQ *abdxys,rel9*	If (cntr) = 0, then Branch; else Continue to next instruction Test Counter and Branch if Zero (cntr = A, B, D, X,Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	-----	-----
TBL *oprx0_xysp*	(M) + [(B) × ((M+1) – (M))] ⇒ A 8-Bit Table Lookup and Interpolate Initialize B, and index before TBL. <ea> points at first 8-bit table entry (M) and B is fractional part of lookup value. (no indirect addressing modes or extensions allowed)	IDX	18 3D xb	ORfffP	Orrfffff	----- C Bit is undefined in HC12	ΔΔ–Δ ?
TBNE *abdxys,rel9*	If (cntr) not = 0, then Branch; else Continue to next instruction Test Counter and Branch if Not Zero (cntr = A, B, D, X,Y, or SP)	REL (9-bit)	04 lb rr	PPP (branch) PPO (no branch)	PPP	-----	-----
TFR *abcdxys,abcdxys*	(r1) ⇒ r2 *or* $00:(r1) ⇒ r2 *or* (r1[7:0]) ⇒ r2 Transfer Register to Register r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	P	P	----- or Δ⇓ΔΔ	----- or ΔΔΔΔ
TPA	(CCR) ⇒ A *Translates to* TFR CCR ,A	INH	B7 20	P	P	-----	-----

Table A-1. Instruction Set Summary (Sheet 14 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12 / M68HC12	S X H I	N Z V C
TRAP *trapnum*	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (B:A) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$ 1 ⇒ I; (TRAP Vector) ⇒ PC Unimplemented opcode trap	INH	18 tn tn = $30–$39 or $40–$FF	OVSPSSPSsP / OfVSPSSPSsP	– – – 1	– – – –
TST *opr16a* TST *oprx0_xysp* TST *oprx9,xysp* TST *oprx16,xysp* TST [D,*xysp*] TST [*oprx16,xysp*]	(M) – 0 Test Memory for Zero or Minus	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb E7 xb ee ff	rPO / rOP rPf / rfP rPO / rPO frPP / frPP fIfrPf / fIfrfP fIPrPf / fIPrfP	– – – –	Δ Δ 0 0
TSTA TSTB	(A) – 0 Test A for Zero or Minus (B) – 0 Test B for Zero or Minus	INH INH	97 D7	O / O O / O		
TSX	(SP) ⇒ X *Translates to* TFR SP,X	INH	B7 75	P / P	– – – –	– – – –
TSY	(SP) ⇒ Y *Translates to* TFR SP,Y	INH	B7 76	P / P	– – – –	– – – –
TXS	(X) ⇒ SP *Translates to* TFR X,SP	INH	B7 57	P / P	– – – –	– – – –
TYS	(Y) ⇒ SP *Translates to* TFR Y,SP	INH	B7 67	P / P	– – – –	– – – –
WAI	(SP) – 2 ⇒ SP; RTN_H:RTN_L ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (Y_H:Y_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (X_H:X_L) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 2 ⇒ SP; (B:A) ⇒ $M_{(SP)}$:$M_{(SP+1)}$; (SP) – 1 ⇒ SP; (CCR) ⇒ $M_{(SP)}$; WAIT for interrupt	INH	3E	OSSSSsf / OSSSfSsf (after interrupt) fVfPPP / VfPPP	– – – –	– – – –
					– – – 1	– – – –
					– 1 – 1	– – – –
WAV	$\sum_{i=1}^{B} S_i F_i \Rightarrow Y{:}D$ and $\sum_{i=1}^{B} F_i \Rightarrow X$ Calculate Sum of Products and Sum of Weights for Weighted Average Calculation Initialize B, X, and Y before WAV. B specifies number of elements. X points at first element in S_i list. Y points at first element in F_i list. All S_i and F_i elements are 8-bits. If interrupted, six extra bytes of stack used for intermediate values	Special	18 3C	Of(frr,ffff)O / Off(frr,fffff)O (add if interrupt) SSS + UUUrr, / SSSf + UUUrr	– – ? –	? Δ ? ?
wavr pseudo-instruction	*see* WAV Resume executing an interrupted WAV instruction (recover intermediate results from stack rather than initializing them to zero)	Special	3C	UUUrr,ffff / UUUrrffffff (frr,ffff)O / (frr,fffff)O (exit + re-entry replaces comma above if interrupted) SSS + UUUrr, / SSSf + UUUrr	– – ? –	? Δ ? ?
XGDX	(D) ⇔ (X) *Translates to* EXG D, X	INH	B7 C5	P / P	– – – –	– – – –
XGDY	(D) ⇔ (Y) *Translates to* EXG D, Y	INH	B7 C6	P / P	– – – –	– – – –

Table A-2. CPU12 Opcode Map (Sheet 1 of 2)

00 †5 BGND IH 1	10 1 ANDCC IM 2	20 3 BRA RL 2	30 3 PULX IH 1	40 1 NEGA IH 1	50 1 NEGB IH 1	60 3-6 NEG ID 2-4	70 4 NEG EX 3	80 1 SUBA IM 2	90 3 SUBA DI 2	A0 3-6 SUBA ID 2-4	B0 3 SUBA EX 3	C0 1 SUBB IM 2	D0 3 SUBB DI 2	E0 3-6 SUBB ID 2-4	F0 3 SUBB EX 3
01 5 MEM IH 1	11 11 EDIV IH 1	21 1 BRN RL 2	31 3 PULY IH 1	41 1 COMA IH 1	51 1 COMB IH 1	61 3-6 COM ID 2-4	71 4 COM EX 3	81 1 CMPA IM 2	91 3 CMPA DI 2	A1 3-6 CMPA ID 2-4	B1 3 CMPA EX 3	C1 1 CMPB IM 2	D1 3 CMPB DI 2	E1 3-6 CMPB ID 2-4	F1 3 CMPB EX 3
02 1 INY IH 1	12 ‡1 MUL IH 1	22 3/1 BHI RL 2	32 3 PULA IH 1	42 1 INCA IH 1	52 1 INCB IH 1	62 3-6 INC ID 2-4	72 4 INC EX 3	82 1 SBCA IM 2	92 3 SBCA DI 2	A2 3-6 SBCA ID 2-4	B2 3 SBCA EX 3	C2 1 SBCB IM 2	D2 3 SBCB DI 2	E2 3-6 SBCB ID 2-4	F2 3 SBCB EX 3
03 1 DEY IH 1	13 3 EMUL IH 1	23 3/1 BLS RL 2	33 3 PULB IH 1	43 1 DECA IH 1	53 1 DECB IH 1	63 3-6 DEC ID 2-4	73 4 DEC EX 3	83 2 SUBD IM 3	93 3 SUBD DI 2	A3 3-6 SUBD ID 2-4	B3 3 SUBD EX 3	C3 2 ADDD IM 3	D3 3 ADDD DI 2	E3 3-6 ADDD ID 2-4	F3 3 ADDD EX 3
04 3 loop* RL 3	14 1 ORCC IM 2	24 3/1 BCC RL 2	34 2 PSHX IH 1	44 1 LSRA IH 1	54 1 LSRB IH 1	64 3-6 LSR ID 2-4	74 4 LSR EX 3	84 1 ANDA IM 2	94 3 ANDA DI 2	A4 3-6 ANDA ID 2-4	B4 3 ANDA EX 3	C4 1 ANDB IM 2	D4 3 ANDB DI 2	E4 3-6 ANDB ID 2-4	F4 3 ANDB EX 3
05 3-6 JMP ID 2-4	15 4-7 JSR ID 2-4	25 3/1 BCS RL 2	35 2 PSHY IH 1	45 1 ROLA IH 1	55 1 ROLB IH 1	65 3-6 ROL ID 2-4	75 4 ROL EX 3	85 1 BITA IM 2	95 3 BITA DI 2	A5 3-6 BITA ID 2-4	B5 3 BITA EX 3	C5 1 BITB IM 2	D5 3 BITB DI 2	E5 3-6 BITB ID 2-4	F5 3 BITB EX 3
06 3 JMP EX 3	16 4 JSR EX 3	26 3/1 BNE RL 2	36 2 PSHA IH 1	46 1 RORA IH 1	56 1 RORB IH 1	66 3-6 ROR ID 2-4	76 4 ROR EX 3	86 1 LDAA IM 2	96 3 LDAA DI 2	A6 3-6 LDAA ID 2-4	B6 3 LDAA EX 3	C6 1 LDAB IM 2	D6 3 LDAB DI 2	E6 3-6 LDAB ID 2-4	F6 3 LDAB EX 3
07 4 BSR RL 2	17 4 JSR DI 2	27 3/1 BEQ RL 2	37 2 PSHB IH 1	47 1 ASRA IH 1	57 1 ASRB IH 1	67 3-6 ASR ID 2-4	77 4 ASR EX 3	87 1 CLRA IH 1	97 1 TSTA IH 1	A7 1 NOP IH 1	B7 1 TFR/EXG IH 2	C7 1 CLRB IH 1	D7 1 TSTB IH 1	E7 3-6 TST ID 2-4	F7 3 TST EX 3
08 1 INX IH 1	18 - Page 2 - -	28 3/1 BVC RL 2	38 3 PULC IH 1	48 1 ASLA IH 1	58 1 ASLB IH 1	68 3-6 ASL ID 2-4	78 4 ASL EX 3	88 1 EORA IM 2	98 3 EORA DI 2	A8 3-6 EORA ID 2-4	B8 3 EORA EX 3	C8 1 EORB IM 2	D8 3 EORB DI 2	E8 3-6 EORB ID 2-4	F8 3 EORB EX 3
09 1 DEX IH 1	19 2 LEAY ID 2-4	29 3/1 BVS RL 2	39 2 PSHC IH 1	49 1 LSRD IH 1	59 1 ASLD IH 1	69 ‡2-4 CLR ID 2-4	79 3 CLR EX 3	89 1 ADCA IM 2	99 3 ADCA DI 2	A9 3-6 ADCA ID 2-4	B9 3 ADCA EX 3	C9 1 ADCB IM 2	D9 3 ADCB DI 2	E9 3-6 ADCB ID 2-4	F9 3 ADCB EX 3
0A ‡7 RTC IH 1	1A 2 LEAX ID 2-4	2A 3/1 BPL RL 2	3A 3 PULD IH 1	4A ‡7 CALL EX 4	5A 2 STAA DI 2	6A ‡2-4 STAA ID 2-4	7A 3 STAA EX 3	8A 1 ORAA IM 2	9A 3 ORAA DI 2	AA 3-6 ORAA ID 2-4	BA 3 ORAA EX 3	CA 1 ORAB IM 2	DA 3 ORAB DI 2	EA 3-6 ORAB ID 2-4	FA 3 ORAB EX 3
0B †8 RTI IH 1	1B 2 LEAS ID 2-4	2B 3/1 BMI RL 2	3B 2 PSHD IH 1	4B ‡7-10 CALL ID 2-5	5B 2 STAB DI 2	6B ‡2-4 STAB ID 2-4	7B 3 STAB EX 3	8B 1 ADDA IM 2	9B 3 ADDA DI 2	AB 3-6 ADDA ID 2-4	BB 3 ADDA EX 3	CB 1 ADDB IM 2	DB 3 ADDB DI 2	EB 3-6 ADDB ID 2-4	FB 3 ADDB EX 3
0C 4-6 BSET ID 3-5	1C 4 BSET EX 4	2C 3/1 BGE RL 2	3C ‡+5 wavr SP 1	4C 4 BSET DI 3	5C 2 STD DI 2	6C ‡2-4 STD ID 2-4	7C 3 STD EX 3	8C 2 CPD IM 3	9C 3 CPD DI 2	AC 3-6 CPD ID 2-4	BC 3 CPD EX 3	CC 2 LDD IM 3	DC 3 LDD DI 2	EC 3-6 LDD ID 2-4	FC 3 LDD EX 3
0D 4-6 BCLR ID 3-5	1D 4 BCLR EX 4	2D 3/1 BLT RL 2	3D 5 RTS IH 1	4D 4 BCLR DI 3	5D 2 STY DI 2	6D ‡2-4 STY ID 2-4	7D 3 STY EX 3	8D 2 CPY IM 3	9D 3 CPY DI 2	AD 3-6 CPY ID 2-4	BD 3 CPY EX 3	CD 2 LDY IM 3	DD 3 LDY DI 2	ED 3-6 LDY ID 2-4	FD 3 LDY EX 3
0E ‡4-6 BRSET ID 4-6	1E 5 BRSET EX 5	2E 3/1 BGT RL 2	3E ‡†7 WAI IH 1	4E 4 BRSET DI 4	5E 2 STX DI 2	6E ‡2-4 STX ID 2-4	7E 3 STX EX 3	8E 2 CPX IM 3	9E 3 CPX DI 2	AE 3-6 CPX ID 2-4	BE 3 CPX EX 3	CE 2 LDX IM 3	DE 3 LDX DI 2	EE 3-6 LDX ID 2-4	FE 3 LDX EX 3
0F ‡4-6 BRCLR ID 4-6	1F 5 BRCLR EX 5	2F 3/1 BLE RL 2	3F 9 SWI IH 1	4F 4 BRCLR DI 4	5F 2 STS DI 2	6F ‡2-4 STS ID 2-4	7F 3 STS EX 3	8F 2 CPS IM 3	9F 3 CPS DI 2	AF 3-6 CPS ID 2-4	BF 3 CPS EX 3	CF 2 LDS IM 3	DF 3 LDS DI 2	EF 3-6 LDS ID 2-4	FF 3 LDS EX 3

Key to Table A-2

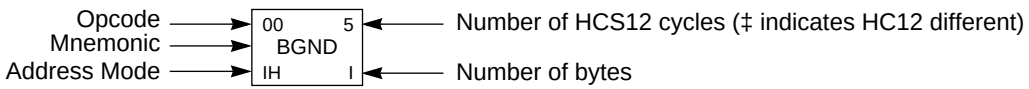


Table A-2. CPU12 Opcode Map (Sheet 2 of 2)

00 4 MOVW IM-ID 5	10 12 IDIV IH 2	20 4 LBRA RL 4	30 10 TRAP IH 2	40 10 TRAP IH 2	50 10 TRAP IH 2	60 10 TRAP IH 2	70 10 TRAP IH 2	80 10 TRAP IH 2	90 10 TRAP IH 2	A0 10 TRAP IH 2	B0 10 TRAP IH 2	C0 10 TRAP IH 2	D0 10 TRAP IH 2	E0 10 TRAP IH 2	F0 10 TRAP IH 2
01 5 MOVW EX-ID 5	11 12 FDIV IH 2	21 3 LBRN RL 4	31 10 TRAP IH 2	41 10 TRAP IH 2	51 10 TRAP IH 2	61 10 TRAP IH 2	71 10 TRAP IH 2	81 10 TRAP IH 2	91 10 TRAP IH 2	A1 10 TRAP IH 2	B1 10 TRAP IH 2	C1 10 TRAP IH 2	D1 10 TRAP IH 2	E1 10 TRAP IH 2	F1 10 TRAP IH 2
02 5 MOVW ID-ID 4	12 13 EMACS SP 4	22 4/3 LBHI RL 4	32 10 TRAP IH 2	42 10 TRAP IH 2	52 10 TRAP IH 2	62 10 TRAP IH 2	72 10 TRAP IH 2	82 10 TRAP IH 2	92 10 TRAP IH 2	A2 10 TRAP IH 2	B2 10 TRAP IH 2	C2 10 TRAP IH 2	D2 10 TRAP IH 2	E2 10 TRAP IH 2	F2 10 TRAP IH 2
03 5 MOVW IM-EX 6	13 3 EMULS IH 2	23 4/3 LBLS RL 4	33 10 TRAP IH 2	43 10 TRAP IH 2	53 10 TRAP IH 2	63 10 TRAP IH 2	73 10 TRAP IH 2	83 10 TRAP IH 2	93 10 TRAP IH 2	A3 10 TRAP IH 2	B3 10 TRAP IH 2	C3 10 TRAP IH 2	D3 10 TRAP IH 2	E3 10 TRAP IH 2	F3 10 TRAP IH 2
04 6 MOVW EX-EX 6	14 12 EDIVS IH 2	24 4/3 LBCC RL 4	34 10 TRAP IH 2	44 10 TRAP IH 2	54 10 TRAP IH 2	64 10 TRAP IH 2	74 10 TRAP IH 2	84 10 TRAP IH 2	94 10 TRAP IH 2	A4 10 TRAP IH 2	B4 10 TRAP IH 2	C4 10 TRAP IH 2	D4 10 TRAP IH 2	E4 10 TRAP IH 2	F4 10 TRAP IH 2
05 5 MOVW ID-EX 5	15 12 IDIVS IH 2	25 4/3 LBCS RL 4	35 10 TRAP IH 2	45 10 TRAP IH 2	55 10 TRAP IH 2	65 10 TRAP IH 2	75 10 TRAP IH 2	85 10 TRAP IH 2	95 10 TRAP IH 2	A5 10 TRAP IH 2	B5 10 TRAP IH 2	C5 10 TRAP IH 2	D5 10 TRAP IH 2	E5 10 TRAP IH 2	F5 10 TRAP IH 2
06 2 ABA IH 2	16 2 SBA IH 2	26 4/3 LBNE RL 4	36 10 TRAP IH 2	46 10 TRAP IH 2	56 10 TRAP IH 2	66 10 TRAP IH 2	76 10 TRAP IH 2	86 10 TRAP IH 2	96 10 TRAP IH 2	A6 10 TRAP IH 2	B6 10 TRAP IH 2	C6 10 TRAP IH 2	D6 10 TRAP IH 2	E6 10 TRAP IH 2	F6 10 TRAP IH 2
07 3 DAA IH 2	17 2 CBA IH 2	27 4/3 LBEQ RL 4	37 10 TRAP IH 2	47 10 TRAP IH 2	57 10 TRAP IH 2	67 10 TRAP IH 2	77 10 TRAP IH 2	87 10 TRAP IH 2	97 10 TRAP IH 2	A7 10 TRAP IH 2	B7 10 TRAP IH 2	C7 10 TRAP IH 2	D7 10 TRAP IH 2	E7 10 TRAP IH 2	F7 10 TRAP IH 2
08 4 MOVB IM-ID 4	18 4-7 MAXA ID 3-5	28 4/3 LBVC RL 4	38 10 TRAP IH 2	48 10 TRAP IH 2	58 10 TRAP IH 2	68 10 TRAP IH 2	78 10 TRAP IH 2	88 10 TRAP IH 2	98 10 TRAP IH 2	A8 10 TRAP IH 2	B8 10 TRAP IH 2	C8 10 TRAP IH 2	D8 10 TRAP IH 2	E8 10 TRAP IH 2	F8 10 TRAP IH 2
09 5 MOVB EX-ID 5	19 4-7 MINA ID 3-5	29 4/3 LBVS RL 4	39 10 TRAP IH 2	49 10 TRAP IH 2	59 10 TRAP IH 2	69 10 TRAP IH 2	79 10 TRAP IH 2	89 10 TRAP IH 2	99 10 TRAP IH 2	A9 10 TRAP IH 2	B9 10 TRAP IH 2	C9 10 TRAP IH 2	D9 10 TRAP IH 2	E9 10 TRAP IH 2	F9 10 TRAP IH 2
0A 5 MOVB ID-ID 4	1A 4-7 EMAXD ID 3-5	2A 4/3 LBPL RL 4	3A †3n REV SP 2	4A 10 TRAP IH 2	5A 10 TRAP IH 2	6A 10 TRAP IH 2	7A 10 TRAP IH 2	8A 10 TRAP IH 2	9A 10 TRAP IH 2	AA 10 TRAP IH 2	BA 10 TRAP IH 2	CA 10 TRAP IH 2	DA 10 TRAP IH 2	EA 10 TRAP IH 2	FA 10 TRAP IH 2
0B 4 MOVB IM-EX 5	1B 4-7 EMIND ID 3-5	2B 4/3 LBMI RL 4	3B †5n/3n REVW SP 2	4B 10 TRAP IH 2	5B 10 TRAP IH 2	6B 10 TRAP IH 2	7B 10 TRAP IH 2	8B 10 TRAP IH 2	9B 10 TRAP IH 2	AB 10 TRAP IH 2	BB 10 TRAP IH 2	CB 10 TRAP IH 2	DB 10 TRAP IH 2	EB 10 TRAP IH 2	FB 10 TRAP IH 2
0C 6 MOVB EX-EX 6	1C 4-7 MAXM ID 3-5	2C 4/3 LBGE RL 4	3C ‡†7B WAV SP 2	4C 10 TRAP IH 2	5C 10 TRAP IH 2	6C 10 TRAP IH 2	7C 10 TRAP IH 2	8C 10 TRAP IH 2	9C 10 TRAP IH 2	AC 10 TRAP IH 2	BC 10 TRAP IH 2	CC 10 TRAP IH 2	DC 10 TRAP IH 2	EC 10 TRAP IH 2	FC 10 TRAP IH 2
0D 5 MOVB ID-EX 5	1D D4-7 MINM ID 3-5	2D 4/3 LBLT RL 4	3D ‡6 TBL ID 3	4D 10 TRAP IH 2	5D 10 TRAP IH 2	6D 10 TRAP IH 2	7D 10 TRAP IH 2	8D 10 TRAP IH 2	9D 10 TRAP IH 2	AD 10 TRAP IH 2	BD 10 TRAP IH 2	CD 10 TRAP IH 2	DD 10 TRAP IH 2	ED 10 TRAP IH 2	FD 10 TRAP IH 2
0E 2 TAB IH 2	1E 4-7 EMAXM ID 3-5	2E 4/3 LBGT RL 4	3E ‡8 STOP IH 2	4E 10 TRAP IH 2	5E 10 TRAP IH 2	6E 10 TRAP IH 2	7E 10 TRAP IH 2	8E 10 TRAP IH 2	9E 10 TRAP IH 2	AE 10 TRAP IH 2	BE 10 TRAP IH 2	CE 10 TRAP IH 2	DE 10 TRAP IH 2	EE 10 TRAP IH 2	FE 10 TRAP IH 2
0F 2 TBA IH 2	1F 4-7 EMINM ID 3-5	2F 4/3 LBLE RL 4	3F 10 ETBL ID 3	4F 10 TRAP IH 2	5F 10 TRAP IH 2	6F 10 TRAP IH 2	7F 10 TRAP IH 2	8F 10 TRAP IH 2	9F 10 TRAP IH 2	AF 10 TRAP IH 2	BF 10 TRAP IH 2	CF 10 TRAP IH 2	DF 10 TRAP IH 2	EF 10 TRAP IH 2	FF 10 TRAP IH 2

* The opcode $04 (on sheet 1 of 2) corresponds to one of the loop primitive instructions DBEQ, DBNE, IBEQ, IBNE, TBEQ, or TBNE.

† Refer to instruction summary for more information.

‡ Refer to instruction summary for different HC12 cycle count.

Page 2: When the CPU encounters a page 2 opcode ($18 on page 1 of the opcode map), it treats the next byte of object code as a page 2 instruction opcode.

Table A-3. Indexed Addressing Mode Postbyte Encoding (xb)

00 0,X 5b const	10 –16,X 5b const	20 1,+X pre-inc	30 1,X+ post-inc	40 0,Y 5b const	50 –16,Y 5b const	60 1,+Y pre-inc	70 1,Y+ post-inc	80 0,SP 5b const	90 –16,SP 5b const	A0 1,+SP pre-inc	B0 1,SP+ post-inc	C0 0,PC 5b const	D0 –16,PC 5b const	E0 n,X 9b const	F0 n,SP 9b const
01 1,X 5b const	11 –15,X 5b const	21 2,+X pre-inc	31 2,X+ post-inc	41 1,Y 5b const	51 –15,Y 5b const	61 2,+Y pre-inc	71 2,Y+ post-inc	81 1,SP 5b const	91 –15,SP 5b const	A1 2,+SP pre-inc	B1 2,SP+ post-inc	C1 1,PC 5b const	D1 –15,PC 5b const	E1 –n,X 9b const	F1 –n,SP 9b const
02 2,X 5b const	12 –14,X 5b const	22 3,+X pre-inc	32 3,X+ post-inc	42 2,Y 5b const	52 –14,Y 5b const	62 3,+Y pre-inc	72 3,Y+ post-inc	82 2,SP 5b const	92 –14,SP 5b const	A2 3,+SP pre-inc	B2 3,SP+ post-inc	C2 2,PC 5b const	D2 –14,PC 5b const	E2 n,X 16b const	F2 n,SP 16b const
03 3,X 5b const	13 –13,X 5b const	23 4,+X pre-inc	33 4,X+ post-inc	43 3,Y 5b const	53 –13,Y 5b const	63 4,+Y pre-inc	73 4,Y+ post-inc	83 3,SP 5b const	93 –13,SP 5b const	A3 4,+SP pre-inc	B3 4,SP+ post-inc	C3 3,PC 5b const	D3 –13,PC 5b const	E3 [n,X] 16b indr	F3 [n,SP] 16b indr
04 4,X 5b const	14 –12,X 5b const	24 5,+X pre-inc	34 5,X+ post-inc	44 4,Y 5b const	54 –12,Y 5b const	64 5,+Y pre-inc	74 5,Y+ post-inc	84 4,SP 5b const	94 –12,SP 5b const	A4 5,+SP pre-inc	B4 5,SP+ post-inc	C4 4,PC 5b const	D4 –12,PC 5b const	E4 A,X A offset	F4 A,SP A offset
05 5,X 5b const	15 –11,X 5b const	25 6,+X pre-inc	35 6,X+ post-inc	45 5,Y 5b const	55 –11,Y 5b const	65 6,+Y pre-inc	75 6,Y+ post-inc	85 5,SP 5b const	95 –11,SP 5b const	A5 6,+SP pre-inc	B5 6,SP+ post-inc	C5 5,PC 5b const	D5 –11,PC 5b const	E5 B,X B offset	F5 B,SP B offset
06 6,X 5b const	16 –10,X 5b const	26 7,+X pre-inc	36 7,X+ post-inc	46 6,Y 5b const	56 –10,Y 5b const	66 7,+Y pre-inc	76 7,Y+ post-inc	86 6,SP 5b const	96 –10,SP 5b const	A6 7,+SP pre-inc	B6 7,SP+ post-inc	C6 6,PC 5b const	D6 –10,PC 5b const	E6 D,X D offset	F6 D,SP D offset
07 7,X 5b const	17 –9,X 5b const	27 8,+X pre-inc	37 8,X+ post-inc	47 7,Y 5b const	57 –9,Y 5b const	67 8,+Y pre-inc	77 8,Y+ post-inc	87 7,SP 5b const	97 –9,SP 5b const	A7 8,+SP pre-inc	B7 8,SP+ post-inc	C7 7,PC 5b const	D7 –9,PC 5b const	E7 [D,X] D indirect	F7 [D,SP] D indirect
08 8,X 5b const	18 –8,X 5b const	28 8,–X pre-dec	38 8,X– post-dec	48 8,Y 5b const	58 –8,Y 5b const	68 8,–Y pre-dec	78 8,Y– post-dec	88 8,SP 5b const	98 –8,SP 5b const	A8 8,–SP pre-dec	B8 8,SP– post-dec	C8 8,PC 5b const	D8 –8,PC 5b const	E8 n,Y 9b const	F8 n,PC 9b const
09 9,X 5b const	19 –7,X 5b const	29 7,–X pre-dec	39 7,X– post-dec	49 9,Y 5b const	59 –7,Y 5b const	69 7,–Y pre-dec	79 7,Y– post-dec	89 9,SP 5b const	99 –7,SP 5b const	A9 7,–SP pre-dec	B9 7,SP– post-dec	C9 9,PC 5b const	D9 –7,PC 5b const	E9 –n,Y 9b const	F9 –n,PC 9b const
0A 10,X 5b const	1A –6,X 5b const	2A 6,–X pre-dec	3A 6,X– post-dec	4A 10,Y 5b const	5A –6,Y 5b const	6A 6,–Y pre-dec	7A 6,Y– post-dec	8A 10,SP 5b const	9A –6,SP 5b const	AA 6,–SP pre-dec	BA 6,SP– post-dec	CA 10,PC 5b const	DA –6,PC 5b const	EA n,Y 16b const	FA n,PC 16b const
0B 11,X 5b const	1B –5,X 5b const	2B 5,–X pre-dec	3B 5,X– post-dec	4B 11,Y 5b const	5B –5,Y 5b const	6B 5,–Y pre-dec	7B 5,Y– post-dec	8B 11,SP 5b const	9B –5,SP 5b const	AB 5,–SP pre-dec	BB 5,SP– post-dec	CB 11,PC 5b const	DB –5,PC 5b const	EB [n,Y] 16b indr	FB [n,PC] 16b indr
0C 12,X 5b const	1C –4,X 5b const	2C 4,–X pre-dec	3C 4,X– post-dec	4C 12,Y 5b const	5C –4,Y 5b const	6C 4,–Y pre-dec	7C 4,Y– post-dec	8C 12,SP 5b const	9C –4,SP 5b const	AC 4,–SP pre-dec	BC 4,SP– post-dec	CC 12,PC 5b const	DC –4,PC 5b const	EC A,Y A offset	FC A,PC A offset
0D 13,X 5b const	1D –3,X 5b const	2D 3,–X pre-dec	3D 3,X– post-dec	4D 13,Y 5b const	5D –3,Y 5b const	6D 3,–Y pre-dec	7D 3,Y– post-dec	8D 13,SP 5b const	9D –3,SP 5b const	AD 3,–SP pre-dec	BD 3,SP– post-dec	CD 13,PC 5b const	DD –3,PC 5b const	ED B,Y B offset	FD B,PC B offset
0E 14,X 5b const	1E –2,X 5b const	2E 2,–X pre-dec	3E 2,X– post-dec	4E 14,Y 5b const	5E –2,Y 5b const	6E 2,–Y pre-dec	7E 2,Y– post-dec	8E 14,SP 5b const	9E –2,SP 5b const	AE 2,–SP pre-dec	BE 2,SP– post-dec	CE 14,PC 5b const	DE –2,PC 5b const	EE D,Y D offset	FE D,PC D offset
0F 15,X 5b const	1F –1,X 5b const	2F 1,–X pre-dec	3F 1,X– post-dec	4F 15,Y 5b const	5F –1,Y 5b const	6F 1,–Y pre-dec	7F 1,Y– post-dec	8F 15,SP 5b const	9F –1,SP 5b const	AF 1,–SP pre-dec	BF 1,SP– post-dec	CF 15,PC 5b const	DF –1,PC 5b const	EF [D,Y] D indirect	FF [D,PC] D indirect

Key to Table A-3

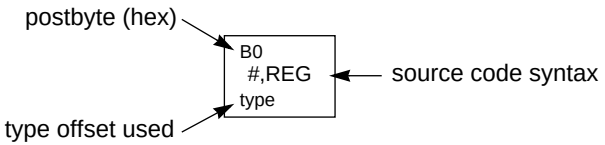


Table A-4. Indexed Addressing Mode Summary

Postbyte Code (xb)	Operand Syntax	Comments
rr0nnnnn	,r n,r –n,r	**5-bit constant offset** n = –16 to +15 rr can specify X, Y, SP, or PC
111rr0zs	n,r –n,r	**Constant offset** (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte (s) 1 = 16-bit if z = s = 1, 16-bit offset indexed-indirect (see below) rr can specify X, Y, SP, or PC
rr1pnnnn	n,–r n,+r n,r– n,r+	**Auto predecrement, preincrement, postdecrement, or postincrement**; p = pre-(0) or post-(1), n = –8 to –1, +1 to +8 rr can specify X, Y, or SP (PC not a valid choice)
111rr1aa	A,r B,r D,r	**Accumulator offset** (unsigned 8-bit or 16-bit) aa - 00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect rr can specify X, Y, SP, or PC
111rr011	[n,r]	**16-bit offset indexed-indirect** rr can specify X, Y, SP, or PC
111rr111	[D,r]	**Accumulator D offset indexed-indirect** rr can specify X, Y, SP, or PC

Table A-5. Transfer and Exchange Postbyte Encoding

TRANSFERS									
⇓LS	MS⇒	0	1	2	3	4	5	6	7
0		A ⇒ A	B ⇒ A	CCR ⇒ A	$TMP3_L$ ⇒ A	B ⇒ A	X_L ⇒ A	Y_L ⇒ A	SP_L ⇒ A
1		A ⇒ B	B ⇒ B	CCR ⇒ B	$TMP3_L$ ⇒ B	B ⇒ B	X_L ⇒ B	Y_L ⇒ B	SP_L ⇒ B
2		A ⇒ CCR	B ⇒ CCR	CCR ⇒ CCR	$TMP3_L$ ⇒ CCR	B ⇒ CCR	X_L ⇒ CCR	Y_L ⇒ CCR	SP_L ⇒ CCR
3		sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2	TMP3 ⇒ TMP2	D ⇒ TMP2	X ⇒ TMP2	Y ⇒ TMP2	SP ⇒ TMP2
4		sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D	TMP3 ⇒ D	D ⇒ D	X ⇒ D	Y ⇒ D	SP ⇒ D
5		sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X	TMP3 ⇒ X	D ⇒ X	X ⇒ X	Y ⇒ X	SP ⇒ X
6		sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y	TMP3 ⇒ Y	D ⇒ Y	X ⇒ Y	Y ⇒ Y	SP ⇒ Y
7		sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP	TMP3 ⇒ SP	D ⇒ SP	X ⇒ SP	Y ⇒ SP	SP ⇒ SP
EXCHANGES									
⇓LS	MS⇒	8	9	A	B	C	D	E	F
0		A ⇔ A	B ⇔ A	CCR ⇔ A	$TMP3_L$ ⇒ A $00:A ⇒ TMP3	B ⇒ A A ⇒ B	X_L ⇒ A $00:A ⇒ X	Y_L ⇒ A $00:A ⇒ Y	SP_L ⇒ A $00:A ⇒ SP
1		A ⇔ B	B ⇔ B	CCR ⇔ B	$TMP3_L$ ⇒ B $FF:B ⇒ TMP3	B ⇒ B $FF ⇒ A	X_L ⇒ B $FF:B ⇒ X	Y_L ⇒ B $FF:B ⇒ Y	SP_L ⇒ B $FF:B ⇒ SP
2		A ⇔ CCR	B ⇔ CCR	CCR ⇔ CCR	$TMP3_L$ ⇒ CCR $FF:CCR ⇒ TMP3	B ⇒ CCR $FF:CCR ⇒ D	X_L ⇒ CCR $FF:CCR ⇒ X	Y_L ⇒ CCR $FF:CCR ⇒ Y	SP_L ⇒ CCR $FF:CCR ⇒ SP
3		$00:A ⇒ TMP2 $TMP2_L$ ⇒ A	$00:B ⇒ TMP2 $TMP2_L$ ⇒ B	$00:CCR ⇒ TMP2 $TMP2_L$ ⇒ CCR	TMP3 ⇔ TMP2	D ⇔ TMP2	X ⇔ TMP2	Y ⇔ TMP2	SP ⇔ TMP2
4		$00:A ⇒ D	$00:B ⇒ D	$00:CCR ⇒ D B ⇒ CCR	TMP3 ⇔ D	D ⇔ D	X ⇔ D	Y ⇔ D	SP ⇔ D
5		$00:A ⇒ X X_L ⇒ A	$00:B ⇒ X X_L ⇒ B	$00:CCR ⇒ X X_L ⇒ CCR	TMP3 ⇔ X	D ⇔ X	X ⇔ X	Y ⇔ X	SP ⇔ X
6		$00:A ⇒ Y Y_L ⇒ A	$00:B ⇒ Y Y_L ⇒ B	$00:CCR ⇒ Y Y_L ⇒ CCR	TMP3 ⇔ Y	D ⇔ Y	X ⇔ Y	Y ⇔ Y	SP ⇔ Y
7		$00:A ⇒ SP SP_L ⇒ A	$00:B ⇒ SP SP_L ⇒ B	$00:CCR ⇒ SP SP_L ⇒ CCR	TMP3 ⇔ SP	D ⇔ SP	X ⇔ SP	Y ⇔ SP	SP ⇔ SP

TMP2 and TMP3 registers are for factory use only.

Table A-6. Loop Primitive Postbyte Encoding (lb)

00 A DBEQ (+)	10 A DBEQ (–)	20 A DBNE (+)	30 A DBNE (–)	40 A TBEQ (+)	50 A TBEQ (–)	60 A TBNE (+)	70 A TBNE (–)	80 A IBEQ (+)	90 A IBEQ (–)	A0 A IBNE (+)	B0 A IBNE (–)
01 B DBEQ (+)	11 B DBEQ (–)	21 B DBNE (+)	31 B DBNE (–)	41 B TBEQ (+)	51 B TBEQ (–)	61 B TBNE (+)	71 B TBNE (–)	81 B IBEQ (+)	91 B IBEQ (–)	A1 B IBNE (+)	B1 B IBNE (–)
02 —	12 —	22 —	32 —	42 —	52 —	62 —	72 —	82 —	92 —	A2 —	B2 —
03 —	13 —	23 —	33 —	43 —	53 —	63 —	73 —	83 —	93 —	A3 —	B3 —
04 D DBEQ (+)	14 D DBEQ (–)	24 D DBNE (+)	34 D DBNE (–)	44 D TBEQ (+)	54 D TBEQ (–)	64 D TBNE (+)	74 D TBNE (–)	84 D IBEQ (+)	94 D IBEQ (–)	A4 D IBNE (+)	B4 D IBNE (–)
05 X DBEQ (+)	15 X DBEQ (–)	25 X DBNE (+)	35 X DBNE (–)	45 X TBEQ (+)	55 X TBEQ (–)	65 X TBNE (+)	75 X TBNE (–)	85 X IBEQ (+)	95 X IBEQ (–)	A5 X IBNE (+)	B5 X IBNE (–)
06 Y DBEQ (+)	16 Y DBEQ (–)	26 Y DBNE (+)	36 Y DBNE (–)	46 Y TBEQ (+)	56 Y TBEQ (–)	66 Y TBNE (+)	76 Y TBNE (–)	86 Y IBEQ (+)	96 Y IBEQ (–)	A6 Y IBNE (+)	B6 Y IBNE (–)
07 SP DBEQ (+)	17 SP DBEQ (–)	27 SP DBNE (+)	37 SP DBNE (–)	47 SP TBEQ (+)	57 SP TBEQ (–)	67 SP TBNE (+)	77 SP TBNE (–)	87 SP IBEQ (+)	97 SP IBEQ (–)	A7 SP IBNE (+)	B7 SP IBNE (–)

Key to Table A-6

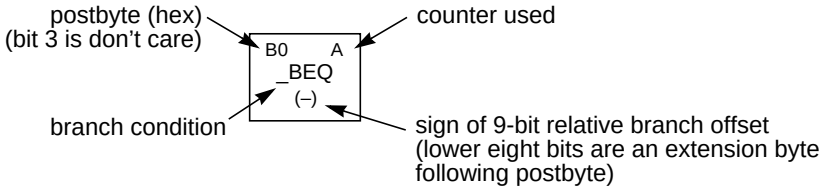


Table A-7. Branch/Complementary Branch

Branch				Complementary Branch			
Test	**Mnemonic**	**Opcode**	**Boolean**	**Test**	**Mnemonic**	**Opcode**	**Comment**
r>m	BGT	2E	Z + (N ⊕ V) = 0	r≤m	BLE	2F	Signed
r≥m	BGE	2C	N ⊕ V = 0	r<m	BLT	2D	Signed
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Signed
r≤m	BLE	2F	Z + (N ⊕ V) = 1	r>m	BGT	2E	Signed
r<m	BLT	2D	N ⊕ V = 1	r≥m	BGE	2C	Signed
r>m	BHI	22	C + Z = 0	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	C = 0	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	Z = 1	r≠m	BNE	26	Unsigned
r≤m	BLS	23	C + Z = 1	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	C = 1	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	C = 1	No Carry	BCC	24	Simple
Negative	BMI	2B	N = 1	Plus	BPL	2A	Simple
Overflow	BVS	29	V = 1	No Overflow	BVC	28	Simple
r=0	BEQ	27	Z = 1	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

For 16-bit offset long branches precede opcode with a $18 page prebyte.

Table A-8. Hexadecimal to ASCII Conversion

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
$00	NUL	$20	SP *space*	$40	@	$60	` *grave*
$01	SOH	$21	!	$41	A	$61	a
$02	STX	$22	“ quote	$42	B	$62	b
$03	ETX	$23	#	$43	C	$63	c
$04	EOT	$24	$	$44	D	$64	d
$05	ENQ	$25	%	$45	E	$65	e
$06	ACK	$26	&	$46	F	$66	f
$07	BEL *beep*	$27	‘ *apost.*	$47	G	$67	g
$08	BS *back sp*	$28	(	$48	H	$68	h
$09	HT *tab*	$29	)	$49	I	$69	i
$0A	LF *linefeed*	$2A	*	$4A	J	$6A	j
$0B	VT	$2B	+	$4B	K	$6B	k
$0C	FF	$2C	, *comma*	$4C	L	$6C	l
$0D	CR *return*	$2D	- *dash*	$4D	M	$6D	m
$0E	SO	$2E	. *period*	$4E	N	$6E	n
$0F	SI	$2F	/	$4F	O	$6F	o
$10	DLE	$30	0	$50	P	$70	p
$11	DC1	$31	1	$51	Q	$71	q
$12	DC2	$32	2	$52	R	$72	r
$13	DC3	$33	3	$53	S	$73	s
$14	DC4	$34	4	$54	T	$74	t
$15	NAK	$35	5	$55	U	$75	u
$16	SYN	$36	6	$56	V	$76	v
$17	ETB	$37	7	$57	W	$77	w
$18	CAN	$38	8	$58	X	$78	x
$19	EM	$39	9	$59	Y	$79	y
$1A	SUB	$3A	:	$5A	Z	$7A	z
$1B	ESCAPE	$3B	;	$5B	[	$7B	{
$1C	FS	$3C	<	$5C	\	$7C	|
$1D	GS	$3D	=	$5D	]	$7D	}
$1E	RS	$3E	>	$5E	^	$7E	~
$1F	US	$3F	?	$5F	_ *under*	$7F	DEL *delete*

A.5 Hexadecimal to Decimal Conversion

To convert a hexadecimal number (up to four hexadecimal digits) to decimal, look up the decimal equivalent of each hexadecimal digit in **Table A-9**. The decimal equivalent of the original hexadecimal number is the sum of the weights found in the table for all hexadecimal digits.

Table A-9. Hexadecimal to/from Decimal Conversion

15 Bit 8				7 Bit 0			
15 12		11 8		7 4		3 0	
4th Hex Digit		3rd Hex Digit		2nd Hex Digit		1st Hex Digit	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,484	E	224	E	14
F	61,440	F	3,840	F	240	F	15

A.6 Decimal to Hexadecimal Conversion

To convert a decimal number (up to $65{,}535_{10}$) to hexadecimal, find the largest decimal number in **Table A-9** that is less than or equal to the number you are converting. The corresponding hexadecimal digit is the most significant hexadecimal digit of the result. Subtract the decimal number found from the original decimal number to get the *remaining decimal value*. Repeat the procedure using the remaining decimal value for each subsequent hexadecimal digit.

Appendix B. M68HC11 to CPU12 Upgrade Path

B.1 Introduction

This appendix discusses similarities and differences between the CPU12 and the M68HC11 CPU. In general, the CPU12 is a proper superset of the M68HC11. Significant changes have been made to improve the efficiency and capabilities of the CPU12 without eliminating compatibility and familiarity for the large community of M68HC11 programmers.

B.2 CPU12 Design Goals

The primary goals of the CPU12 design were:

- Absolute source code compatibility with the M68HC11
- Same programming model
- Same stacking operations
- Upgrade to 16-bit architecture
- Eliminate extra byte/extra cycle penalty for using index register Y
- Improve performance
- Improve compatibility with high-level languages

B.3 Source Code Compatibility

Every M68HC11 instruction mnemonic and source code statement can be assembled directly with a CPU12 assembler with no modifications.

The CPU12 supports all M68HC11 addressing modes and includes several new variations of indexed addressing mode. CPU12 instructions affect condition code bits in the same way as M68HC11 instructions.

CPU12 object code is similar to but not identical to M68HC11 object code. Some primary objectives, such as the elimination of the penalty for using Y, could not be achieved without object code differences. While the object code has been changed, the majority of the opcodes are identical to those of the M6800, which was developed more than 20 years earlier.

The CPU12 assembler automatically translates a few M68HC11 instruction mnemonics into functionally equivalent CPU12 instructions. For example, the CPU12 does not have an increment stack pointer (INS) instruction, so the INS mnemonic is translated to LEAS 1,S. The CPU12 does provide single-byte DEX, DEY, INX, and INY instructions because the LEAX and LEAY instructions do not affect the condition codes, while the M68HC11 instructions update the Z bit according to the result of the decrement or increment.

Table B-1 shows M68HC11 instruction mnemonics that are automatically translated into equivalent CPU12 instructions. This translation is performed by the assembler so there is no need to modify an old M68HC11 program to assemble it for the CPU12. In fact, the M68HC11 mnemonics can be used in new CPU12 programs.

Table B-1. Translated M68HC11 Mnemonics

M68HC11 Mnemonic	Equivalent CPU12 Instruction	Comments
ABX ABY	LEAX B,X LEAY B,Y	Since CPU12 has accumulator offset indexing, ABX and ABY are rarely used in new CPU12 programs. ABX is one byte on M68HC11 but ABY is two bytes. The LEA substitutes are two bytes.

Continued on next page

Table B-1. Translated M68HC11 Mnemonics (Continued)

M68HC11 Mnemonic	Equivalent CPU12 Instruction	Comments
CLC CLI CLV SEC SEI SEV	ANDCC #$FE ANDCC #$EF ANDCC #$FD ORCC #$01 ORCC #$10 ORCC #$02	ANDCC and ORCC now allow more control over the CCR, including the ability to set or clear multiple bits in a single instruction. These instructions take one byte each on M68HC11 while the ANDCC and ORCC equivalents take two bytes each.
DES INS	LEAS –1,S LEAS 1,S	Unlike DEX and INX, DES and INS did not affect CCR bits in the M68HC11, so the LEAS equivalents in CPU12 duplicate the function of DES and INS. These instructions are one byte on M68HC11 and two bytes on CPU12.
TAP TPA TSX TSY TXS TYS XGDX XGDY	TFR A,CCR TFR CCR,A TFR S,X TFR S,Y TFR X,S TFR Y,S EXG D,X EXG D,Y	The M68HC11 has a small collection of specific transfer and exchange instructions. CPU12 expanded this to allow transfer or exchange between any two CPU registers. For all but TSY and TYS (which take two bytes on either CPU), the CPU12 transfer/exchange costs one extra byte compared to the M68HC11. The substitute instructions execute in one cycle rather than two.

All of the translations produce the same amount of or slightly more object code than the original M68HC11 instructions. However, there are offsetting savings in other instructions. Y-indexed instructions in particular assemble into one byte less object code than the same M68HC11 instruction.

The CPU12 has a 2-page opcode map, rather than the 4-page M68HC11 map. This is largely due to redesign of the indexed addressing modes. Most of pages 2, 3, and 4 of the M68HC11 opcode map are required because Y-indexed instructions use different opcodes than X-indexed instructions. Approximately two-thirds of the M68HC11 page 1 opcodes are unchanged in CPU12, and some M68HC11 opcodes have been moved to page 1 of the CPU12 opcode map. Object code for each of the moved instructions is one byte smaller than object code for the equivalent M68HC11 instruction. **Table B-2** shows instructions that assemble to one byte less object code on the CPU12.

Table B-2. Instructions with Smaller Object Code

Instruction	Comments
DEY INY	Page 2 opcodes in M68HC11 but page 1 in CPU12
INST n,Y	For values of n less than 16 (the majority of cases). Were on page 2, now are on page 1. Applies to BSET, BCLR, BRSET, BRCLR, NEG, COM, LSR, ROR, ASR, ASL, ROL, DEC, INC, TST, JMP, CLR, SUB, CMP, SBC, SUBD, ADDD, AND, BIT, LDA, STA, EOR, ADC, ORA, ADD, JSR, LDS, and STS. If X is the index reference and the offset is greater than 15 (much less frequent than offsets of 0, 1, and 2), the CPU12 instruction assembles to one byte more of object code than the equivalent M68HC11 instruction.
PSHY PULY	Were on page 2, now are on page 1
LDY STY CPY	Were on page 2, now are on page 1
CPY n,Y LDY n,Y STY n,Y	For values of n less than 16 (the majority of cases); were on page 3, now are on page 1
CPD	Was on page 2, 3, or 4, now on page 1. In the case of indexed with offset greater than 15, CPU12 and M68HC11 object code are the same size.

Instruction set changes offset each other to a certain extent. Programming style also affects the rate at which instructions appear. As a test, the BUFFALO monitor, an 8-Kbyte M68HC11 assembly code program, was reassembled for the CPU12. The resulting object code is six bytes smaller than the M68HC11 code. It is fair to conclude that M68HC11 code can be reassembled with very little change in size.

The relative size of code for M68HC11 vs. code for CPU12 has also been tested by rewriting several smaller programs from scratch. In these cases, the CPU12 code is typically about 30 percent smaller. These savings are mostly due to improved indexed addressing.

It seems useful to mention the results of size comparisons done on C programs. A C program compiled for the CPU12 is about 30 percent smaller than the same program compiled for the M68HC11. The savings are largely due to better indexing.

B.4 Programmer’s Model and Stacking

The CPU12 programming model and stacking order are identical to those of the M68HC11.

B.5 True 16-Bit Architecture

The M68HC11 is a direct descendant of the M6800, one of the first microprocessors, which was introduced in 1974. The M6800 was strictly an 8-bit machine, with 8-bit data buses and 8-bit instructions. As Motorola devices evolved from the M6800 to the M68HC11, a number of 16-bit instructions were added, but the data buses remained eight bits wide, so these instructions were performed as sequences of 8-bit operations. The CPU12 is a true 16-bit implementation, but it retains the ability to work with the mostly 8-bit M68HC11 instruction set. The larger arithmetic logic unit (ALU) of the CPU12 (it can perform some 20-bit operations) is used to calculate 16-bit pointers and to speed up math operations.

B.5.1 Bus Structures

The CPU12 is a 16-bit processor with 16-bit data paths. Typical HCS12 and M68HC12 devices have internal and external 16-bit data paths, but some derivatives incorporate operating modes that allow for an 8-bit data bus, so that a system can be built with low-cost 8-bit program memory. HCS12 and M68HC12 MCUs include an on-chip integration module that manages the external bus interface. When the CPU makes a 16-bit access to a resource that is served by an 8-bit bus, the integration module performs two 8-bit accesses, freezes the CPU clocks for part of the sequence, and assembles the data into a 16-bit word. As far as the CPU is concerned, there is no difference between this access and a 16-bit access to an internal resource via the 16-bit data bus. This is similar to the way an M68HC11 can stretch clock cycles to accommodate slow peripherals.

B.5.2 Instruction Queue

The CPU12 has a 2-word instruction queue and a 16-bit holding buffer, which sometimes acts as a third word for queueing program information. All program information is fetched from memory as aligned 16-bit words, even though there is no requirement for instructions to begin or end on even word boundaries. There is no penalty for misaligned instructions. If a program begins on an odd boundary (if the reset vector is an odd address), program information is fetched to fill the instruction queue, beginning with the aligned word at the next address below the misaligned reset vector. The instruction queue logic starts execution with the opcode in the low-order half of this word.

The instruction queue causes three bytes of program information (starting with the instruction opcode) to be directly available to the CPU at the beginning of every instruction. As it executes, each instruction performs enough additional program fetches to refill the space it took up in the queue. Alignment information is maintained by the logic in the instruction queue. The CPU provides signals that tell the queue logic when to advance a word of program information and when to toggle the alignment status.

The CPU is not aware of instruction alignment. The queue logic includes a multiplexer that sorts out the information in the queue to present the opcode and the next two bytes of information as CPU inputs. The multiplexer determines whether the opcode is in the even or odd half of the word at the head of the queue. Alignment status is also available to the ALU for address calculations. The execution sequence for all instructions is independent of the alignment of the instruction.

The only situation where alignment can affect the number of cycles an instruction takes occurs in devices that have a narrow (8-bit) external data bus and is related to optional program fetch cycles (O type cycles). O cycles are always performed, but serve different purposes determined by instruction size and alignment.

Each instruction includes one program fetch cycle for every two bytes of object code. Instructions with an odd number of bytes can use an O cycle to fetch an extra word of object code. If the queue is aligned at the start of an instruction with an odd byte count, the last byte of object code shares a queue word with the opcode of the next instruction. Since this word holds part of the next instruction, the queue cannot advance after

the odd byte executes because the first byte of the next instruction would be lost. In this case, the O cycle appears as a free cycle since the queue is not ready to accept the next word of program information. If this same instruction had been misaligned, the queue would be ready to advance and the O cycle would be used to perform a program word fetch.

In a single-chip system or in a system with the program in 16-bit memory, both the free cycle and the program fetch cycle take one bus cycle. In a system with the program in an external 8-bit memory, the O cycle takes one bus cycle when it appears as a free cycle, but it takes two bus cycles when used to perform a program fetch. In this case, the on-chip integration module freezes the CPU clocks long enough to perform the cycle as two smaller accesses. The CPU handles only 16-bit data, and is not aware that the 16-bit program access is split into two 8-bit accesses.

To allow development systems to track events in the CPU12 instruction queue, two status signals (IPIPE[1:0]) provide information about data movement in the queue and about the start of instruction execution. A development system can use this information along with address and data information to externally reconstruct the queue. This representation of the queue can also track both the data and address buses.

B.5.3 Stack Function

Both the M68HC11 and the CPU12 stack nine bytes for interrupts. Since this is an odd number of bytes, there is no practical way to ensure that the stack will stay aligned. To ensure that instructions take a fixed number of cycles regardless of stack alignment, the internal RAM in M68HC12 MCUs is designed to allow single cycle 16-bit accesses to misaligned addresses. As long as the stack is located in this special RAM, stacking and unstacking operations take the same amount of execution time, regardless of stack alignment. If the stack is located in an external 16-bit RAM, a PSHX instruction can take two or three cycles depending on the alignment of the stack. This extra access time is transparent to the CPU because the integration module freezes the CPU clocks while it performs the extra 8-bit bus cycle required for a misaligned stack operation.

The CPU12 has a “last-used” stack rather than a “next-available” stack like the M68HC11 CPU. That is, the stack pointer points to the last 16-bit stack address used, rather than to the address of the next available

stack location. This generally has very little effect, because it is very unusual to access stacked information using absolute addressing. The change allows a 16-bit word of data to be removed from the stack without changing the value of the SP twice.

To illustrate, consider the operation of a PULX instruction. With the next-available M68HC11 stack, if the SP = $01F0 when execution begins, the sequence of operations is: SP = SP + 1; load X from $01F1:01F2; SP = SP + 1; and the SP ends up at $01F2. With the last-used CPU12 stack, if the SP = $01F0 when execution begins, the sequence is: load X from $01F0:01F1; SP = SP + 2; and the SP again ends up at $01F2. The second sequence requires one less stack pointer adjustment.

The stack pointer change also affects operation of the TSX and TXS instructions. In the M68HC11, TSX increments the SP by one during the transfer. This adjustment causes the X index to point to the last stack location used. The TXS instruction operates similarly, except that it decrements the SP by one during the transfer. CPU12 TSX and TXS instructions are ordinary transfers — the CPU12 stack requires no adjustment.

For ordinary use of the stack, such as pushes, pulls, and even manipulations involving TSX and TXS, there are no differences in the way the M68HC11 and the CPU12 stacks look to a programmer. However, the stack change can affect a program algorithm in two subtle ways.

The LDS #$xxxx instruction is normally used to initialize the stack pointer at the start of a program. In the M68HC11, the address specified in the LDS instruction is the first stack location used. In the CPU12, however, the first stack location used is one address lower than the address specified in the LDS instruction. Since the stack builds downward, M68HC11 programs reassembled for the CPU12 operate normally, but the program stack is one physical address lower in memory.

In very uncommon situations, such as test programs used to verify CPU operation, a program could initialize the SP, stack data, and then read the stack via an extended mode read (it is normally improper to read stack data from an absolute extended address). To make an M68HC11 source program that contains such a sequence work on the CPU12, change either the initial LDS #$xxxx or the absolute extended address used to read the stack.

B.6 Improved Indexing

The CPU12 has significantly improved indexed addressing capability, yet retains compatibility with the M68HC11. The one cycle and one byte cost of doing Y-related indexing in the M68HC11 has been eliminated. In addition, high-level language requirements, including stack relative indexing and the ability to perform pointer arithmetic directly in the index registers, have been accommodated.

The M68HC11 has one variation of indexed addressing that works from X or Y as the reference pointer. For X indexed addressing, an 8-bit unsigned offset in the instruction is added to the index pointer to arrive at the address of the operand for the instruction. A load accumulator instruction assembles into two bytes of object code, the opcode and a 1-byte offset. Using Y as the reference, the same instruction assembles into three bytes (a page prebyte, the opcode, and a 1-byte offset.) Analysis of M68HC11 source code indicates that the offset is most frequently zero and seldom greater than four.

The CPU12 indexed addressing scheme uses a postbyte plus 0, 1, or 2 extension bytes after the instruction opcode. These bytes specify which index register is used, determine whether an accumulator is used as the offset, implement automatic pre/post increment/decrement of indices, and allow a choice of 5-, 9-, or 16-bit signed offsets. This approach eliminates the differences between X and Y register use and dramatically enhances indexed addressing capabilities.

Major improvements that result from this new approach are:

- Stack pointer can be used as an index register in all indexed operations (very important for C compilers)
- Program counter can be used as index register in all but auto inc/dec modes
- Accumulator offsets allowed using A, B, or D accumulators
- Automatic pre- or post- increment or decrement by –8 to +8
- 5-bit, 9-bit, or 16-bit signed constant offsets (M68HC11 only supported positive unsigned 8-bit offsets)
- 16-bit offset indexed-indirect and accumulator D offset indexed-indirect

The change completely eliminates pages three and four of the M68HC11 opcode map and eliminates almost all instructions from page two of the opcode map. For offsets of 0 to +15 from the X index register, the object code is the same size as it was for the M68HC11. For offsets of 0 to +15 from the Y index register, the object code is one byte smaller than it was for the M68HC11.

Table A-3 and **Table A-4** summarize CPU12 indexed addressing mode capabilities. **Table A-6** shows how the postbyte is encoded.

B.6.1 Constant Offset Indexing

The CPU12 offers three variations of constant offset indexing to optimize the efficiency of object code generation.

The most common constant offset is 0. Offsets of 1, 2, 3, 4 are used fairly often, but with less frequency than 0.

The 5-bit constant offset variation covers the most frequent indexing requirements by including the offset in the postbyte. This reduces a load accumulator indexed instruction to two bytes of object code, and matches the object code size of the smallest M68HC11 indexed instructions, which can only use X as the index register. The CPU12 can use X, Y, SP, or PC as the index reference with no additional object code size cost.

The signed 9-bit constant offset indexing mode covers the same positive range as the M68HC11 8-bit unsigned offset. The size was increased to nine bits with the sign bit (ninth bit) included in the postbyte, and the remaining 8 bits of the offset in a single extension byte.

The 16-bit constant offset indexing mode allows indexed access to the entire normal 64-Kbyte address space. Since the address consists of 16 bits, the 16-bit offset can be regarded as a signed (–32,768 to +32,767) or unsigned (0 to 65,535) value. In 16-bit constant offset mode, the offset is supplied in two extension bytes after the opcode and postbyte.

B.6.2 Auto-Increment Indexing

The CPU12 provides greatly enhanced auto increment and decrement modes of indexed addressing. In the CPU12, the index modification may be specified for before the index is used (pre-), or after the index is used (post-), and the index can be incremented or decremented by any amount from one to eight, independent of the size of the operand that was accessed. X, Y, and SP can be used as the index reference, but this mode does not allow PC to be the index reference (this would interfere with proper program execution).

This addressing mode can be used to implement a software stack structure or to manipulate data structures in lists or tables, rather than manipulating bytes or words of data. Anywhere an M68HC11 program has an increment or decrement index register operation near an indexed mode instruction, the increment or decrement operation can be combined with the indexed instruction with no cost in object code size, as shown in the following code comparison.

18 A6 00 18 08 18 08	LDAA 0,Y INY INY	A6 71	LDAA 2,Y+

The M68HC11 object code requires seven bytes, while the CPU12 requires only two bytes to accomplish the same functions. Three bytes of M68HC11 code were due to the page prebyte for each Y-related instruction ($18). CPU12 post-increment indexing capability allowed the two INY instructions to be absorbed into the LDAA indexed instruction. The replacement code is not identical to the original 3-instruction sequence because the Z condition code bit is affected by the M68HC11 INY instructions, while the Z bit in the CPU12 would be determined by the value loaded into A.

B.6.3 Accumulator Offset Indexing

This indexed addressing variation allows the programmer to use either an 8-bit accumulator (A or B) or the 16-bit D accumulator as the offset for indexed addressing. This allows for a program-generated offset, which is more difficult to achieve in the M68HC11. The following code compares the M68HC11 and CPU12 operations.

```
C6 05       LDAB     #$5     [2]
CE 10 00    LOOP LDX #$1000  [3]
3A          ABX              [3]
A6 00       LDAA     0,X     [4]
             |
5A          DECB             [2]
26 F7       BNE      LOOP    [3]
```

```
C6 05       LDAB  #$5        [1]
CE 10 00    LDX  #$1000      [2]
A6 E5       LOOP LDAA  B,X   [3]
             |
04 31 FB    DBNE  B,LOOP     [3]
```

The CPU12 object code is only one byte smaller, but the LDX # instruction is outside the loop. It is not necessary to reload the base address in the index register on each pass through the loop because the LDAA B,X instruction does not alter the index register. This reduces the loop execution time from 15 cycles to six cycles. This reduction, combined with the 25-MHz bus speed of the HCS12 (M68HC12) Family, can have significant effects.

B.6.4 Indirect Indexing

The CPU12 allows some forms of indexed indirect addressing where the instruction points to a location in memory where the address of the operand is stored. This is an extra level of indirection compared to ordinary indexed addressing. The two forms of indexed indirect addressing are 16-bit constant offset indexed indirect and D accumulator indexed indirect. The reference index register can be X, Y, SP, or PC as in other CPU12 indexed addressing modes. PC-relative indirect addressing is one of the more common uses of indexed indirect addressing. The indirect variations of indexed addressing help in the implementation of pointers. D accumulator indexed indirect addressing can be used to implement a runtime computed GOTO function. Indirect addressing is also useful in high-level language compilers. For instance, PC-relative indirect indexing can be used to efficiently implement some C case statements.

B.7 Improved Performance

The HCS12 uses a system-on-a-chip (SoC) design methodology and is normally implemented in a 0.25µ FLASH process. HCS12 devices can operate at up to 25 MHz and are designed to be migrated easily to faster, smaller silicon process technologies as they are developed.

The M68HC12 improves on M68HC11 performance in several ways. M68HC12 devices are designed using sub-micron design rules and fabricated using advanced semiconductor processing, the same methods used to manufacture the M68HC16 and M68300 Families of modular microcontrollers. M68HC12 devices have a base bus speed of 8 MHz and are designed to operate over a wide range of supply voltages.

The 16-bit wide architecture of the CPU12 also increases performance. Beyond these obvious improvements, the CPU12 uses a reduced number of cycles for many of its instructions, and a 20-bit ALU makes certain CPU12 math operations much faster.

B.7.1 Reduced Cycle Counts

No M68HC11 instruction takes less than two cycles, but the CPU12 has more than 50 opcodes that take only one cycle. Some of the reduction comes from the instruction queue, which ensures that several program bytes are available at the start of each instruction. Other cycle reductions occur because the CPU12 can fetch 16 bits of information at a time, rather than eight bits at a time.

B.7.2 Fast Math

The CPU12 has some of the fastest math ever designed into a Motorola general-purpose MCU. Much of the speed is due to a 20-bit ALU that can perform two smaller operations simultaneously. The ALU can also perform two operations in a single bus cycle in certain cases.

Table B-3 compares the speed of CPU12 and M68HC11 math instructions. The CPU12 requires fewer cycles to perform an operation, and the cycle time is considerably faster than that of the M68HC11.

Table B-3. Comparison of Math Instruction Speeds

Instruction Mnemonic	Math Operation	M68HC11 1 Cycle = 250 ns	M68HC11 With Coprocessor 1 Cycle = 250 ns	CPU12 1 Cycle = 40 ns (125 ns in M68HC12)
MUL	$8 \times 8 = 16$ (signed)	10 cycles	—	3 cycles
EMUL	$16 \times 16 = 32$ (unsigned)	—	20 cycles	3 cycles
EMULS	$16 \times 16 = 32$ (signed)	—	20 cycles	3 cycles
IDIV	$16 \div 16 = 16$ (unsigned)	41 cycles	—	12 cycles
FDIV	$16 \div 16 = 16$ (fractional)	41 cycles	—	12 cycles
EDIV	$32 \div 16 = 16$ (unsigned)	—	33 cycles	11 cycles
EDIVS	$32 \div 16 = 16$ (signed)	—	37 cycles	12 cycles
IDIVS	$16 \div 16 = 16$ (signed)	—	—	12 cycles
EMACS	$32 \times (16 \times 16) \Rightarrow 32$ (signed MAC)	—	20 cycles	12 cycles

The IDIVS instruction is included specifically for C compilers, where word-sized operands are divided to produce a word-sized result (unlike the $32 \div 16 = 16$ EDIV). The EMUL and EMULS instructions place the result in registers so a C compiler can choose to use only 16 bits of the 32-bit result.

B.7.3 Code Size Reduction

CPU12 assembly language programs written from scratch tend to be 30 percent smaller than equivalent programs written for the M68HC11. This figure has been independently qualified by Motorola programmers and an independent C compiler vendor. The major contributors to the reduction appear to be improved indexed addressing and the universal transfer/exchange instruction.

In some specialized areas, the reduction is much greater. A fuzzy logic inference kernel requires about 250 bytes in the M68HC11, and the same program for the CPU12 requires about 50 bytes. The CPU12 fuzzy logic instructions replace whole subroutines in the M68HC11 version. Table lookup instructions also greatly reduce code space.

Other CPU12 code space reductions are more subtle. Memory-to-memory moves are one example. The CPU12 move instruction requires almost as many bytes as an equivalent sequence of M68HC11 instructions, but the move operations themselves do not require the use of an accumulator. This means that the accumulator often need not be saved and restored, which saves instructions.

Arithmetic operations on index pointers are another example. The M68HC11 usually requires that the content of the index register be moved into accumulator D, where calculations are performed, then back to the index register before indexing can take place. In the CPU12, the LEAS, LEAX, and LEAY instructions perform arithmetic operations directly on the index pointers. The pre-/post-increment/decrement variations of indexed addressing also allow index modification to be incorporated into an existing indexed instruction rather than performing the index modification as a separate operation.

Transfer and exchange operations often allow register contents to be temporarily saved in another register rather than having to save the contents in memory. Some CPU12 instructions such as MIN and MAX combine the actions of several M68HC11 instructions into a single operation.

B.8 Additional Functions

The CPU12 incorporates a number of new instructions that provide added functionality and code efficiency. Among other capabilities, these new instructions allow efficient processing for fuzzy logic applications and support subroutine processing in extended memory beyond the standard 64-Kbyte address map for M68HC12 devices incorporating this feature. **Table B-4** is a summary of these new instructions. Subsequent paragraphs discuss significant enhancements.

Table B-4. New M68HC12 Instructions (Sheet 1 of 2)

Mnemonic	Addressing Modes	Brief Functional Description
ANDCC	Immediate	AND CCR with mask (replaces CLC, CLI, and CLV)
BCLR	Extended	Bit(s) clear (added extended mode)
BGND	Inherent	Enter background debug mode, if enabled
BRCLR	Extended	Branch if bit(s) clear (added extended mode)
BRSET	Extended	Branch if bit(s) set (added extended mode)
BSET	Extended	Bit(s) set (added extended mode)
CALL	Extended, indexed	Similar to JSR except also stacks PPAGE value; with RTC instruction, allows easy access to >64-Kbyte space
CPS	Immediate, direct, extended, and indexed	Compare stack pointer
DBNE	Relative	Decrement and branch if equal to zero (looping primitive)
DBEQ	Relative	Decrement and branch if not equal to zero (looping primitive)
EDIV	Inherent	Extended divide Y:D/X = Y(Q) and D(R) (unsigned)
EDIVS	Inherent	Extended divide Y:D/X = Y(Q) and D(R) (signed)
EMACS	Special	Multiply and accumulate $16 \times 16 \Rightarrow 32$ (signed)
EMAXD	Indexed	Maximum of two unsigned 16-bit values
EMAXM	Indexed	Maximum of two unsigned 16-bit values
EMIND	Indexed	Minimum of two unsigned 16-bit values
EMINM	Indexed	Minimum of two unsigned 16-bit values
EMUL	Special	Extended multiply $16 \times 16 \Rightarrow 32$; M(idx) $*$ D $\Rightarrow$ Y:D
EMULS	Special	Extended multiply $16 \times 16 \Rightarrow 32$ (signed); M(idx) $*$ D $\Rightarrow$ Y:D
ETBL	Special	Table lookup and interpolate (16-bit entries)
EXG	Inherent	Exchange register contents
IBEQ	Relative	Increment and branch if equal to zero (looping primitive)
IBNE	Relative	Increment and branch if not equal to zero (looping primitive)
IDIVS	Inherent	Signed integer divide D/X $\Rightarrow$ X(Q) and D(R) (signed)
LBCC	Relative	Long branch if carry clear (same as LBHS)
LBCS	Relative	Long branch if carry set (same as LBLO)
LBEQ	Relative	Long branch if equal (Z=1)
LBGE	Relative	Long branch if greater than or equal to zero
LBGT	Relative	Long branch if greater than zero
LBHI	Relative	Long branch if higher
LBHS	Relative	Long branch if higher or same (same as LBCC)
LBLE	Relative	Long branch if less than or equal to zero
LBLO	Relative	Long branch if lower (same as LBCS)

Table B-4. New M68HC12 Instructions (Sheet 2 of 2)

Mnemonic	Addressing Modes	Brief Functional Description
LBLS	Relative	Long branch if lower or same
LBLT	Relative	Long branch if less than zero
LBMI	Relative	Long branch if minus
LBNE	Relative	Long branch if not equal to zero
LBPL	Relative	Long branch if plus
LBRA	Relative	Long branch always
LBRN	Relative	Long branch never
LBVC	Relative	Long branch if overflow clear
LBVS	Relative	Long branch if overflow set
LEAS	Indexed	Load stack pointer with effective address
LEAX	Indexed	Load X index register with effective address
LEAY	Indexed	Load Y index register with effective address
MAXA	Indexed	Maximum of two unsigned 8-bit values
MAXM	Indexed	Maximum of two unsigned 8-bit values
MEM	Special	Determine grade of fuzzy membership
MINA	Indexed	Minimum of two unsigned 8-bit values
MINM	Indexed	Minimum of two unsigned 8-bit values
MOVB(W)	Combinations of immediate, extended, and indexed	Move data from one memory location to another
ORCC	Immediate	OR CCR with mask (replaces SEC, SEI, and SEV)
PSHC	Inherent	Push CCR onto stack
PSHD	Inherent	Push double accumulator onto stack
PULC	Inherent	Pull CCR contents from stack
PULD	Inherent	Pull double accumulator from stack
REV	Special	Fuzzy logic rule evaluation
REVW	Special	Fuzzy logic rule evaluation with weights
RTC	Inherent	Restore program page and return address from stack used with CALL instruction, allows easy access to >64-Kbyte space
SEX	Inherent	Sign extend 8-bit register into 16-bit register
TBEQ	Relative	Test and branch if equal to zero (looping primitive)
TBL	Inherent	Table lookup and interpolate (8-bit entries)
TBNE	Relative	Test register and branch if not equal to zero (looping primitive)
TFR	Inherent	Transfer register contents to another register
WAV	Special	Weighted average (fuzzy logic support)

B.8.1 Memory-to-Memory Moves

The CPU12 has both 8- and 16-bit variations of memory-to-memory move instructions. The source address can be specified with immediate, extended, or indexed addressing modes. The destination address can be specified by extended or indexed addressing mode. The indexed addressing mode for move instructions is limited to modes that require no extension bytes (9- and 16-bit constant offsets are not allowed), and indirect indexing is not allowed for moves. This leaves 5-bit signed constant offsets, accumulator offsets, and the automatic increment/decrement modes. The following simple loop is a block move routine capable of moving up to 256 words of information from one memory area to another.

```
LOOP   MOVW   2,X+ , 2,Y+  ;move a word and update pointers
DBNE   B,LOOP              ;repeat B times
```

The move immediate to extended is a convenient way to initialize a register without using an accumulator or affecting condition codes.

B.8.2 Universal Transfer and Exchange

The M68HC11 has only eight transfer instructions and two exchange instructions. The CPU12 has a universal transfer/exchange instruction that can be used to transfer or exchange data between any two CPU registers. The operation is obvious when the two registers are the same size, but some of the other combinations provide very useful results. For example when an 8-bit register is transferred to a 16-bit register, a sign-extend operation is performed. Other combinations can be used to perform a zero-extend operation.

These instructions are used often in CPU12 assembly language programs. Transfers can be used to make extra copies of data in another register, and exchanges can be used to temporarily save data during a call to a routine that expects data in a specific register. This is sometimes faster and produces more compact object code than saving data to memory with pushes or stores.

B.8.3 Loop Construct

The CPU12 instruction set includes a new family of six loop primitive instructions. These instructions decrement, increment, or test a loop count in a CPU register and then branch based on a zero or non-zero test result. The CPU registers that can be used for the loop count are A, B, D, X, Y, or SP. The branch range is a 9-bit signed value (–512 to +511) which gives these instructions twice the range of a short branch instruction.

B.8.4 Long Branches

All of the branch instructions from the M68HC11 are also available with 16-bit offsets which allows them to reach any location in the 64-Kbyte address space.

B.8.5 Minimum and Maximum Instructions

Control programs often need to restrict data values within upper and lower limits. The CPU12 facilitates this function with 8- and 16-bit versions of MIN and MAX instructions. Each of these instructions has a version that stores the result in either the accumulator or in memory.

For example, in a fuzzy logic inference program, rule evaluation consists of a series of MIN and MAX operations. The min operation is used to determine the smallest rule input (the running result is held in an accumulator), and the max operation is used to store the largest rule truth value (in an accumulator) or the previous fuzzy output value (in a RAM location) to the fuzzy output in RAM. The following code demonstrates how MIN and MAX instructions can be used to evaluate a rule with four inputs and two outputs.

```
LDY     #OUT1     ;Point at first output
LDX     #IN1      ;Point at first input value
LDAA    #$FF      ;start with largest 8-bit number in A
MINA    1,X+      ;A=MIN(A,IN1)
MINA    1,X+      ;A=MIN(A,IN2)
MINA    1,X+      ;A=MIN(A,IN3)
MINA    1,X+      ;A=MIN(A,IN4) so A holds smallest input
MAXM    1,Y+      ;OUT1=MAX(A,OUT1) and A is unchanged
MAXM    1,Y+      ;OUT1=MAX(A,OUT2) A still has min input
```

Before this sequence is executed, the fuzzy outputs must be cleared to zeros (not shown). M68HC11 MIN or MAX operations are performed by

executing a compare followed by a conditional branch around a load or store operation.

These instructions can also be used to limit a data value prior to using it as an input to a table lookup or other routine. Suppose a table is valid for input values between $20 and $7F. An arbitrary input value can be tested against these limits and be replaced by the largest legal value if it is too big, or the smallest legal value if too small using the following two CPU12 instructions.

```
HILIMIT  FCB    $7F           ;comparison value needs to be in mem
LOWLIMIT FCB    $20           ;so it can be referenced via indexed
         MINA   HILIMIT,PCR   ;A=MIN(A,$7F)
         MAXA   LOWLIMIT,PCR  ;A=MAX(A,$20)
                              ;A now within the legal range $20 to $7F
```

The “,PCR” notation is also new for the CPU12. This notation indicates the programmer wants an appropriate offset from the PC reference to the memory location (HILIMIT or LOWLIMIT in this example), and then to assemble this instruction into a PC-relative indexed MIN or MAX instruction.

B.8.6 Fuzzy Logic Support

The CPU12 includes four instructions (MEM, REV, REVW, and WAV) specifically designed to support fuzzy logic programs. These instructions have a very small impact on the size of the CPU and even less impact on the cost of a complete MCU. At the same time, these instructions dramatically reduce the object code size and execution time for a fuzzy logic inference program. A kernel written for the M68HC11 required about 250 bytes and executed in about 750 milliseconds. The CPU12 kernel uses about 50 bytes and executes in about 16 microseconds (in a 25-MHz HCS12).

B.8.7 Table Lookup and Interpolation

The CPU12 instruction set includes two instructions (TBL and ETBL) for lookup and interpolation of compressed tables. Consecutive table values are assumed to be the x coordinates of the endpoints of a line segment. The TBL instruction uses 8-bit table entries (y-values) and returns an 8-bit result. The ETBL instruction uses 16-bit table entries (y-values) and returns a 16-bit result.

An indexed addressing mode is used to identify the effective address of the data point at the beginning of the line segment, and the data value for the end point of the line segment is the next consecutive memory location (byte for TBL and word for ETBL). In both cases, the B accumulator represents the ratio of (the x-distance from the beginning of the line segment to the lookup point) to (the x-distance from the beginning of the line segment to the end of the line segment). B is treated as an 8-bit binary fraction with radix point left of the MSB, so each line segment is effectively divided into 256 pieces. During execution of the TBL or ETBL instruction, the difference between the end point y-value and the beginning point y-value (a signed byte for TBL or a signed word for ETBL) is multiplied by the B accumulator to get an intermediate delta-y term. The result is the y-value of the beginning point, plus this signed intermediate delta-y value.

B.8.8 Extended Bit Manipulation

The M68HC11 CPU allows only direct or indexed addressing. This typically causes the programmer to dedicate an index register to point at some memory area such as the on-chip registers. The CPU12 allows all bit manipulation instructions to work with direct, extended, or indexed addressing modes.

B.8.9 Push and Pull D and CCR

The CPU12 includes instructions to push and pull the D accumulator and the CCR. It is interesting to note that the order in which 8-bit accumulators A and B are stacked for interrupts is the opposite of what would be expected for the upper and lower bytes of the 16-bit D accumulator. The order used originated in the M6800, an 8-bit microprocessor developed long before anyone thought 16-bit single-chip devices would be made. The interrupt stacking order for accumulators A and B is retained for code compatibility.

B.8.10 Compare SP

This instruction was added to the CPU12 instruction set to improve orthogonality and high-level language support. One of the most important requirements for C high-level language support is the ability to

do arithmetic on the stack pointer for such things as allocating local variable space on the stack. The LEAS –5,SP instruction is an example of how the compiler could easily allocate five bytes on the stack for local variables. LDX 5,SP+ loads X with the value on the bottom of the stack and deallocates five bytes from the stack in a single operation that takes only two bytes of object code.

B.8.11 Support for Memory Expansion

Bank switching is a common method of expanding memory beyond the 64-Kbyte limit of a CPU with a 64-Kbyte address space, but there are some known difficulties associated with bank switching. One problem is that interrupts cannot take place during the bank switching operation. This increases worst case interrupt latency and requires extra programming space and execution time.

Some HCS12 and M68HC12 variants include a built-in bank switching scheme that eliminates many of the problems associated with external switching logic. The CPU12 includes CALL and return-from-call (RTC) instructions that manage the interface to the bank-switching system. These instructions are analogous to the JSR and RTS instructions, except that the bank page number is saved and restored automatically during execution. Since the page change operation is part of an uninterruptable instruction, many of the difficulties associated with bank switching are eliminated. On HCS12 and M68HC12 derivatives with expanded memory capability, bank numbers are specified by on-chip control registers. Since the addresses of these control registers may not be the same in all derivatives, the CPU12 has a dedicated control line to the on-chip integration module that indicates when a memory-expansion register is being read or written. This allows the CPU to access the PPAGE register without knowing the register address.

The indexed indirect versions of the CALL instruction access the address of the called routine and the destination page value indirectly. For other addressing mode variations of the CALL instruction, the destination page value is provided as immediate data in the instruction object code. CALL and RTC execute correctly in the normal 64-Kbyte address space, thus providing for portable code.

Appendix C. High-Level Language Support

C.1 Introduction

Many programmers are turning to high-level languages such as C as an alternative to coding in native assembly languages. High-level language (HLL) programming can improve productivity and produce code that is more easily maintained than assembly language programs. The most serious drawback to the use of HLL in MCUs has been the relatively large size of programs written in HLL. Larger program ROM size requirements translate into increased system costs.

Motorola solicited the cooperation of third-party software developers to assure that the CPU12 instruction set would meet the needs of a more efficient generation of compilers. Several features of the CPU12 were specifically designed to improve the efficiency of compiled HLL, and thus minimize cost.

This appendix identifies CPU12 instructions and addressing modes that provide improved support for high-level language. C language examples are provided to demonstrate how these features support efficient HLL structures and concepts. Since the CPU12 instruction set is a superset of the M68HC11 instruction set, some of the discussions use the M68HC11 as a basis for comparison.

C.2 Data Types

The CPU12 supports the bit-sized data type with bit manipulation instructions which are available in extended, direct, and indexed variations. The char data type is a simple 8-bit value that is commonly used to specify variables in a small microcontroller system because it requires less memory space than a 16-bit integer (provided the variable has a range small enough to fit into eight bits). The 16-bit CPU12 can easily handle 16-bit integer types and the available set of conditional branches (including long branches) allow branching based on signed or

unsigned arithmetic results. Some of the higher math functions allow for division and multiplication involving 32-bit values, although it is somewhat less common to use such long values in a microcontroller system.

The CPU12 has special sign extension instructions to allow easy type-casting from smaller data types to larger ones, such as from char to integer. This sign extension is automatically performed when an 8-bit value is transferred to a 16-bit register.

C.3 Parameters and Variables

High-level languages make extensive use of the stack, both to pass variables and for temporary and local storage. It follows that there should be easy ways to push and pull each CPU register, stack pointer based indexing should be allowed, and that direct arithmetic manipulation of the stack pointer value should be allowed. The CPU12 instruction set provided for all of these needs with improved indexed addressing, the addition of an LEAS instruction, and the addition of push and pull instructions for the D accumulator and the CCR.

C.3.1 Register Pushes and Pulls

The M68HC11 has push and pull instructions for A, B, X, and Y, but requires separate 8-bit pushes and pulls of accumulators A and B to stack or unstack the 16-bit D accumulator (the concatenated combination of A:B). The PSHD and PULD instructions allow directly stacking the D accumulator in the expected 16-bit order.

Adding PSHC and PULC improved orthogonality by completing the set of stacking instructions so that any of the CPU registers can be pushed or pulled. These instructions are also useful for preserving the CCR value during a function call subroutine.

C.3.2 Allocating and Deallocating Stack Space

The LEAS instruction can be used to allocate or deallocate space on the stack for temporary variables:

```
LEAS   –10,S   ;Allocate space for 5 16-bit integers
LEAS   10,S    ;Deallocate space for 5 16-bit ints
```

The (de)allocation can even be combined with a register push or pull as in this example:

```
LDX    8,S+    ;Load return value and deallocate
```

X is loaded with the 16-bit integer value at the top of the stack, and the stack pointer is adjusted up by eight to deallocate space for eight bytes worth of temporary storage. Post-increment indexed addressing is used in this example, but all four combinations of pre/post increment/decrement are available (offsets from –8 to +8 inclusive, from X, Y, or SP). This form of indexing can often be used to get an index (or stack pointer) adjustment for free during an indexed operation (the instruction requires no more code space or cycles than a zero-offset indexed instruction).

C.3.3 Frame Pointer

In the C language, it is common to have a frame pointer in addition to the CPU stack pointer. The frame is an area of memory within the system stack which is used for parameters and local storage of variables used within a function subroutine. The following is a description of how a frame pointer can be set up and used.

First, parameters (typically values in CPU registers) are pushed onto the system stack prior to using a JSR or CALL to get to the function subroutine. At the beginning of the called subroutine, the frame pointer of the calling program is pushed onto the stack. Typically, an index register, such as X, is used as the frame pointer, so a PSHX instruction would save the frame pointer from the calling program.

Next, the called subroutine establishes a new frame pointer by executing a TFR S,X. Space is allocated for local variables by executing an LEAS –n,S, where n is the number of bytes needed for local variables.

Notice that parameters are at positive offsets from the frame pointer while locals are at negative offsets. In the M68HC11, the indexed addressing mode uses only positive offsets, so the frame pointer always

points to the lowest address of any parameter or local. After the function subroutine finishes, calculations are required to restore the stack pointer to the mid-frame position between the locals and the parameters before returning to the calling program. The CPU12 only requires execution of TFR X,S to deallocate the local storage and return.

The concept of a frame pointer is supported in the CPU12 through a combination of improved indexed addressing, universal transfer/exchange, and the LEA instruction. These instructions work together to achieve more efficient handling of frame pointers. It is important to consider the complete instruction set as a complex system with subtle interrelationships rather than simply examining individual instructions when trying to improve an instruction set. Adding or removing a single instruction can have unexpected consequences.

C.4 Increment and Decrement Operators

In C, the notation $++$i or i$--$ is often used to form loop counters. Within limited constraints, the CPU12 loop primitives can be used to speed up the loop count and branch function.

The CPU12 includes a set of six basic loop control instructions which decrement, increment, or test a loop count register, and then branch if it is either equal to zero or not equal to zero. The loop count register can be A, B, D, X, Y, or SP. A or B could be used if the loop count fits in an 8-bit char variable; the other choices are all 16-bit registers. The relative offset for the loop branch is a 9-bit signed value, so these instructions can be used with loops as long as 256 bytes.

In some cases, the pre- or post-increment operation can be combined with an indexed instruction to eliminate the cost of the increment operation. This is typically done by post-compile optimization because the indexed instruction that could absorb the increment/decrement operation may not be apparent at compile time.

C.5 Higher Math Functions

In the CPU12, subtle characteristics of higher math operations such as IDIVS and EMUL are arranged so a compiler can handle inputs and outputs more efficiently.

The most apparent case is the IDIVS instruction, which divides two 16-bit signed numbers to produce a 16-bit result. While the same function can be accomplished with the EDIVS instruction (a 32 by 16 divide), doing so is much less efficient because extra steps are required to prepare inputs to the EDIVS, and because EDIVS uses the Y index register. EDIVS uses a 32-bit signed numerator and the C compiler would typically want to use a 16-bit value (the size of an integer data type). The 16-bit C value would need to be sign-extended into the upper 16 bits of the 32-bit EDIVS numerator before the divide operation.

Operand size is also a potential problem in the extended multiply operations but the difficulty can be minimized by putting the results in CPU registers. Having higher precision math instructions is not necessarily a requirement for supporting high-level language because these functions can be performed as library functions. However, if an application requires these functions, the code is much more efficient if the MCU can use native instructions instead of relatively large, slow routines.

C.6 Conditional If Constructs

In the CPU12 instruction set, most arithmetic and data manipulation instructions automatically update the condition code register, unlike other architectures that only change condition codes during a few specific compare instructions. The CPU12 includes branch instructions that perform conditional branching based on the state of the indicators in the condition codes register. Short branches use a single byte relative offset that allows branching to a destination within about ±128 locations from the branch. Long branches use a 16-bit relative offset that allows conditional branching to any location in the 64-Kbyte map.

C.7 Case and Switch Statements

Case and switch statements (and computed GOTOs) can use PC-relative indirect addressing to determine which path to take. Depending upon the situation, cases can use either the constant offset variation or the accumulator D offset variation of indirect indexed addressing.

C.8 Pointers

The CPU12 supports pointers by allowing direct arithmetic operations on the 16-bit index registers (LEAS, LEAX, and LEAY instructions) and by allowing indexed indirect addressing modes.

C.9 Function Calls

Bank switching is a fairly common way of adapting a CPU with a 16-bit address bus to accommodate more than 64 Kbytes of program memory space. One of the most significant drawbacks of this technique has been the requirement to mask (disable) interrupts while the bank page value was being changed. Another problem is that the physical location of the bank page register can change from one MCU derivative to another (or even due to a change to mapping controls by a user program). In these situations, an operating system program has to keep track of the physical location of the page register. The CPU12 addresses both of these problems with the uninterruptible CALL and return-from-call (RTC) instructions.

The CALL instruction is similar to a JSR instruction, except that the programmer supplies a destination page value as part of the instruction. When CALL executes, the old page value is saved on the stack and the new page value is written to the bank page register. Since the CALL instruction is uninterruptible, this eliminates the need to separately mask off interrupts during the context switch.

The CPU12 has dedicated signal lines that allow the CPU to access the bank page register without having to use an address in the normal 64-Kbyte address space. This eliminates the need for the program to know where the page register is physically located.

The RTC instruction is similar to the RTS instruction, except that RTC uses the byte of information that was saved on the stack by the corresponding CALL instruction to restore the bank page register to its old value. Although a CALL/RTC pair can be used to access any function subroutine regardless of the location of the called routine (on the current bank page or a different page), it is most efficient to access some subroutines with JSR/RTS instructions when the called subroutine is on the current page or in an area of memory that is always visible in the 64-Kbyte map regardless of the bank page selection.

Push and pull instructions can be used to stack some or all the CPU registers during a function call. The CPU12 can push and pull any of the CPU registers A, B, CCR, D, X, Y, or SP.

C.10 Instruction Set Orthogonality

One helpful aspect of the CPU12 instruction set, orthogonality, is difficult to quantify in terms of direct benefit to an HLL compiler. Orthogonality refers to the regularity of the instruction set. A completely orthogonal instruction set would allow any instruction to operate in any addressing mode, would have identical code sizes and execution times for similar operations on different registers, and would include both signed and unsigned versions of all mathematical instructions. Greater regularity of the instruction set makes it possible to implement compilers more efficiently, because operation is more consistent, and fewer special cases must be handled.

High-Level Language Support

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

JAPAN:
Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

S12CPUV2/D
Rev. 0
7/2003

CRG
Block User Guide
V04.05

Original Release Date: 29 Feb. 2000
Revised: 2 August 2002

Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
1.0	02/16/00	02/16/00		Initial Release
2.0	11/09/00	10/18/00		Initial SRS2.0 compliant release
2.01	03/03/01	03/03/01		Added RSBCK bit.
2.02	03/22/01			Make the content SRS compliant (usable as a customer document)
3.00	10 May 2001	10 May 2001		Modified according to new Pierce Oscillator feature and different PLLSEL bit write conditions
3.01	13 July 01	13 July 01		Minor corrections
3.02	30 July 01	30 July 01		Enhanced Block diagram (VREG, POR), Corrected register figure CRGINT to read and write
V03.03	2 Aug. 01	2 Aug. 01		Enhanced Clock Quality Check Diagram. Corrections in Reset section
V03.04	27 Aug. 01	27 Aug. 01		Improved description of exiting Wait and Pseudo Stop Mode. Enhanced SCME Bit description.
V03.05	5 Sept. 01	5 Sept. 01		Corrected C_{DC} position in Colpitts Oscillator Connections diagram.
V03.06	1 Oct. 01	1 Oct. 01		Motorola internal: spec tagging
V04.00	10 Oct. 01	10 Oct. 01		Added Low Voltage Reset feature
V04.01	21 Nov. 01	21 Nov. 01		Added Interrupts to Block Diagram. Corrected LVRF flag description.
V04.02	22 Nov. 01	22 Nov. 01		Re-Corrected LVRF flag description.
V04.03	11 Mar 02	11 Mar 02		Removed document number from all pages except cover page Replaced f_{VCOMIN} by $f_{SCM.}$ Added Bus Clock formulas close to PLLCLK formulas. Spelling improvements
V04.04	03 May 02	03 May 02		Corrected in COPCTL register description: RSBCK is write once EXTAL signal description: mentioning pull-down resistor which is active in full stop mode
V04.05	2 Aug.02	2 Aug. 02		Removed oscillator specific information as separate Oscillator Block Guide is available now.

Table of Contents

Section 1 Introduction

Section 2 Signal Description

Section 3 Memory Map and Registers

Section 4 Functional Description

Section 5 Resets

Section 6 Interrupts

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

This specification describes the function of the Clocks and Reset Generator (CRG).

1.2 Features

The main features of this block are:

- Phase Locked Loop (PLL) frequency multiplier
  - Reference divider
  - Automatic bandwidth control mode for low-jitter operation
  - Automatic frequency lock detector
  - CPU interrupt on entry or exit from locked condition
  - Self Clock Mode in absence of reference clock
- System Clock Generator
  - Clock Quality Check
  - Clock switch for either Oscillator or PLL based system clocks
  - User selectable disabling of clocks during Wait Mode for reduced power consumption.
- Computer Operating Properly (COP) watchdog timer with time-out clear window.
- System Reset generation from the following possible sources:
  - Power on reset
  - Low voltage reset
    Refer to device specification for availability of this feature.
  - COP reset
  - Loss of clock reset
  - External pin reset
- Real-Time Interrupt (RTI)

1.3 Modes of Operation

This subsection lists and briefly describes all operating modes supported by the CRG.

- Run Mode

All functional parts of the CRG are running during normal Run Mode. If RTI or COP functionality is required the individual bits of the associated rate select registers (COPCTL, RTICTL) have to be set to a non zero value.

- Wait Mode

  This mode allows to disable the system and core clocks depending on the configuration of the individual bits in the CLKSEL register.

- Stop Mode

  Depending on the setting of the PSTP bit Stop Mode can be differentiated between Full Stop Mode (PSTP=0) and Pseudo Stop Mode (PSTP=1).

  – Full Stop Mode

    The oscillator is disabled and thus all system and core clocks are stopped. The COP and the RTI remain frozen.

  – Pseudo Stop Mode

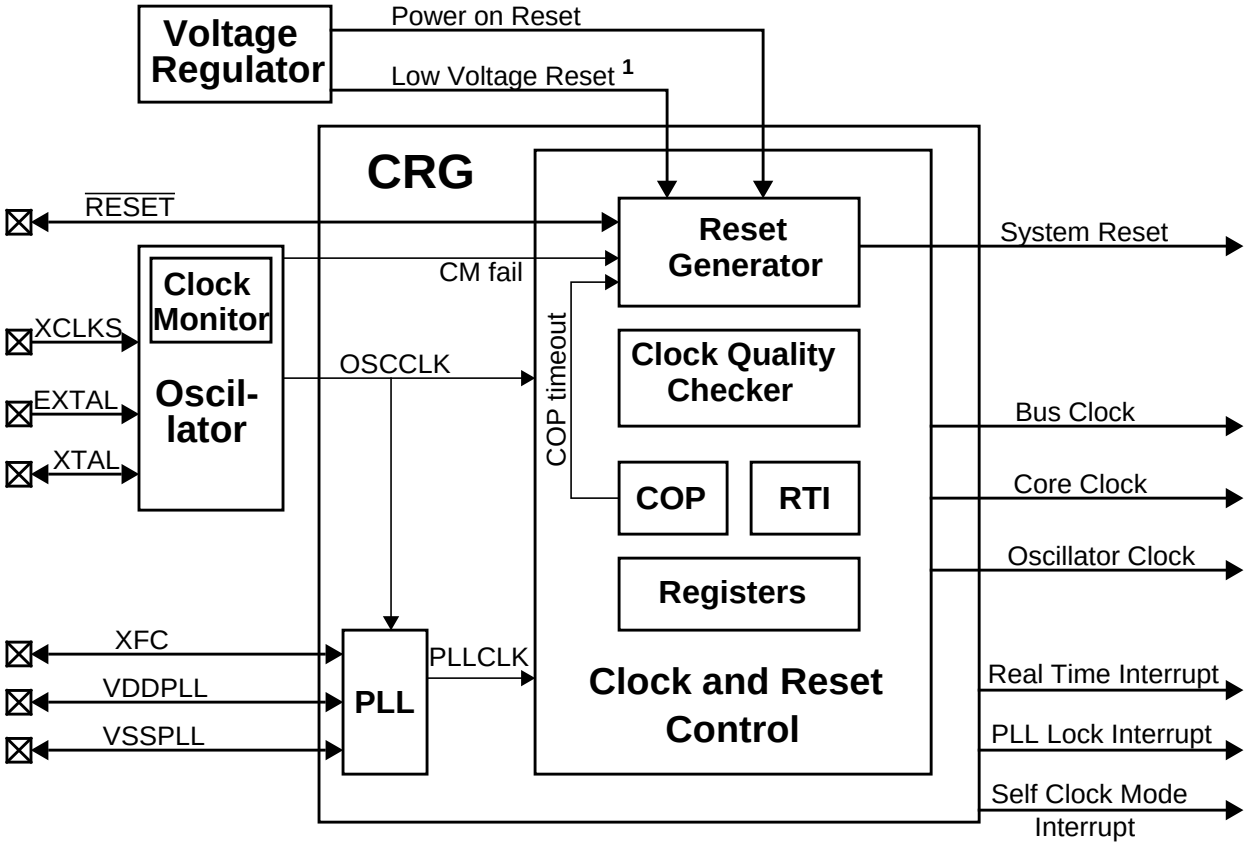
    The oscillator continues to run and most of the system and core clocks are stopped. If the respective enable bits are set the COP and RTI will continue to run, else they remain frozen.

- Self Clock Mode

  Self Clock Mode will be entered if the Clock Monitor Enable Bit (CME) and the Self Clock Mode Enable Bit (SCME) are both asserted and the clock monitor in the oscillator block detects a loss of clock. As soon as Self Clock Mode is entered the CRG starts to perform a clock quality check. Self Clock Mode remains active until the clock quality check indicates that the required quality of the incoming clock signal is met (frequency and amplitude). Self Clock Mode should be used for safety purposes only. It provides reduced functionality to the MCU in case a loss of clock is causing severe system conditions.

1.4 Block Diagram

Figure 1-1 shows a block diagram of the CRG.



1) Refer to device specification for availability of the low voltage reset feature.

Figure 1-1 Block diagram of CRG

Section 2 Signal Description

2.1 Overview

This section lists and describes the signals that connect off chip.

2.2 Detailed Signal Descriptions

2.2.1 VDDPLL, VSSPLL

These pins provides operating voltage (VDDPLL) and ground (VSSPLL) for the PLL circuitry. This allows the supply voltage to the PLL to be independently bypassed. Even if PLL usage is not required VDDPLL and VSSPLL must be connected to properly.

2.2.2 XFC

A passive external loop filter must be placed on the XFC pin. The filter is a second-order, low-pass filter to eliminate the VCO input ripple. The value of the external filter network and the reference frequency determines the speed of the corrections and the stability of the PLL. **Refer to device specification for calculation of PLL Loop Filter (XFC) components.** If PLL usage is not required the XFC pin must be tied to VDDPLL.

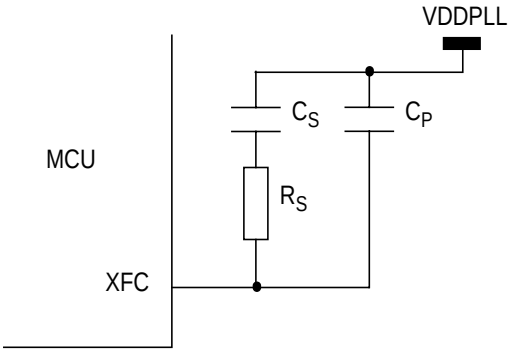


Figure 2-1 PLL Loop Filter Connections

2.2.3 $\overline{\text{RESET}}$

$\overline{\text{RESET}}$ is an active low bidirectional reset pin. As an input it initializes the MCU asynchronously to a known start-up state. As an open-drain output it indicates that an system reset (internal to MCU) has been triggered.

Section 3 Memory Map and Registers

3.1 Overview

This section provides a detailed description of all registers accessible in the CRG.

3.2 Module Memory Map

Table 3-1 gives an overview on all CRG registers.

Table 3-1 CRG Memory Map

Address Offset	Use	Access
$_00	CRG Synthesizer Register (SYNR)	R/W
$_01	CRG Reference Divider Register (REFDV)	R/W
$_02	CRG Test Flags Register (CTFLG)[1]	R/W
$_03	CRG Flags Register (CRGFLG)	R/W
$_04	CRG Interrupt Enable Register (CRGINT)	R/W
$_05	CRG Clock Select Register (CLKSEL)	R/W
$_06	CRG PLL Control Register (PLLCTL)	R/W
$_07	CRG RTI Control Register (RTICTL)	R/W
$_08	CRG COP Control Register (COPCTL)	R/W
$_09	CRG Force and Bypass Test Register (FORBYP)[2]	R/W
$_0A	CRG Test Control Register (CTCTL)[3]	R/W
$_0B	CRG COP Arm/Timer Reset (ARMCOP)	R/W

NOTES:
1. CTFLG is intended for factory test purposes only.
2. FORBYP is intended for factory test purposes only.
3. CTCTL is intended for factory test purposes only.

NOTE: *Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.*

3.3 Register Descriptions

This section describes in address order all the CRG registers and their individual bits.

3.3.1 CRG Synthesizer Register (SYNR)

The SYNR register controls the multiplication factor of the PLL. If the PLL is on, the count in the loop divider (SYNR) register effectively multiplies up the PLL clock (PLLCLK) from the reference frequency by 2 x (SYNR+1). PLLCLK will not be below the minimum VCO frequency (f_{SCM}).

$$PLLCLK = 2 \times OSCCLK \times \frac{(SYNR + 1)}{(REFDV + 1)}$$

NOTE: *If PLL is selected (PLLSEL=1), Bus Clock = PLLCLK / 2*
Bus Clock must not exceed the maximum operating system frequency.

Address Offset: $_00

	7	6	5	4	3	2	1	0
R	0	0	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-1 CRG Synthesizer Register (SYNR)

Read: anytime

Write: anytime except if PLLSEL = 1

NOTE: *Write to this register initializes the lock detector bit and the track detector bit.*

3.3.2 CRG Reference Divider Register (REFDV)

The REFDV register provides a finer granularity for the PLL multiplier steps. The count in the reference divider divides OSCCLK frequency by REFDV+1.

Address Offset: $_01

	7	6	5	4	3	2	1	0
R	0	0	0	0	REFDV3	REFDV2	REFDV1	REFDV0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 CRG Reference Divider Register (REFDV)

Read: anytime

Write: anytime except when PLLSEL = 1

NOTE: *Write to this register initializes the lock detector bit and the track detector bit.*

3.3.3 Reserved Register (CTFLG)

This register is reserved for factory testing of the CRG module and is not available in normal modes.

Address Offset: $_02

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 Reserved Register (CTFLG)

Read: always reads $00 in normal modes

Write: unimplemented in normal modes

NOTE: *Writing to this register when in special mode can alter the CRG fucntionality.*

3.3.4 CRG Flags Register (CRGFLG)

This register provides CRG status bits and flags.

Address Offset: $_03

	7	6	5	4	3	2	1	0
R	RTIF	PORF	LVRF	LOCKIF	LOCK	TRACK	SCMIF	SCM
W								
RESET:	0	1	2	0	0	0	0	0

= Unimplemented or Reserved

NOTES:
1. PORF is set to 1 when a power on reset occurs. Unaffected by system reset.
2. LVRF is set to 1 when a low voltage reset occurs. Unaffected by system reset.

Figure 3-4 CRG Flags Register (CRGFLG)

Read: anytime

Write: refer to each bit for individual write conditions

RTIF — Real Time Interrupt Flag

RTIF is set to 1 at the end of the RTI period. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (RTIE=1), RTIF causes an interrupt request.

1 = RTI time-out has occurred.

0 = RTI time-out has not yet occurred.

PORF — Power on Reset Flag

PORF is set to 1 when a power on reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect.

1 = Power on reset has occurred.
0 = Power on reset has not occurred.

LVRF — Low Voltage Reset Flag

If low voltage reset feature is not available (see device specification) LVRF always reads 0.
LVRF is set to 1 when a low voltage reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect.

1 = Low voltage reset has occurred.
0 = Low voltage reset has not occurred.

LOCKIF — PLL Lock Interrupt Flag

LOCKIF is set to 1 when LOCK status bit changes. This flag can only be cleared by writing a 1. Writing a 0 has no effect.If enabled (LOCKIE=1), LOCKIF causes an interrupt request.

1 = LOCK bit has changed.
0 = No change in LOCK bit.

LOCK — Lock Status Bit

LOCK reflects the current state of PLL lock condition. This bit is cleared in Self Clock Mode. Writes have no effect.

1 = PLL VCO is within the desired tolerance of the target frequency.
0 = PLL VCO is not within the desired tolerance of the target frequency.

TRACK — Track Status Bit

TRACK reflects the current state of PLL track condition. This bit is cleared in Self Clock Mode. Writes have no effect.

1 = Tracking mode status.
0 = Acquisition mode status.

SCMIF — Self Clock Mode Interrupt Flag

SCMIF is set to 1 when SCM status bit changes. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (SCMIE=1), SCMIF causes an interrupt request.

1 = SCM bit has changed.
0 = No change in SCM bit.

SCM — Self Clock Mode Status Bit

SCM reflects the current clocking mode. Writes have no effect.

1 = MCU is operating in Self Clock Mode with OSCCLK in an unknown state. All clocks are derived from PLLCLK running at its minimum frequency f_{SCM}.
0 = MCU is operating normally with OSCCLK available.

3.3.5 CRG Interrupt Enable Register (CRGINT)

This register enables CRG interrupt requests.

Address Offset: $_04

	7	6	5	4	3	2	1	0
R	RTIE	0	0	LOCKIE	0	0	SCMIE	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-5 CRG Interrupt Enable Register (CRGINT)

Read: anytime

Write: anytime

RTIE — Real Time Interrupt Enable Bit.
1 = Interrupt will be requested whenever RTIF is set.
0 = Interrupt requests from RTI are disabled.

LOCKIE — Lock Interrupt Enable Bit
1 = Interrupt will be requested whenever LOCKIF is set.
0 = LOCK interrupt requests are disabled.

SCMIE — Self Clock Mode Interrupt Enable Bit
1 = Interrupt will be requested whenever SCMIF is set.
0 = SCM interrupt requests are disabled.

3.3.6 CRG Clock Select Register (CLKSEL)

This register controls CRG clock selection. Refer to **Figure 4-2 System Clocks Generator** for more details on the effect of each bit.

Address Offset: $_05

	7	6	5	4	3	2	1	0
R	PLLSEL	PSTP	SYSWAI	ROAWAI	PLLWAI	CWAI	RTIWAI	COPWAI
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 CRG Clock Select Register (CLKSEL)

Read: anytime

Write: refer to each bit for individual write conditions

PLLSEL — PLL Select Bit

Write anytime. Writing a one when LOCK=0 and AUTO=1, or TRACK=0 and AUTO=0 has no effect This prevents the selection of an unstable PLLCLK as SYSCLK. PLLSEL bit is cleared when the MCU enters Self Clock Mode, Stop Mode or Wait Mode with PLLWAI bit set.

1 = System clocks are derived from PLLCLK (Bus Clock = PLLCLK / 2).
0 = System clocks are derived from OSCCLK (Bus Clock = OSCCLK / 2).

PSTP — Pseudo Stop Bit

Write: anytime

This bit controls the functionality of the oscillator during Stop Mode.

1 = Oscillator continues to run in Stop Mode (Pseudo Stop). The oscillator amplitude is reduced.
Refer to oscillator block description for availability of a reduced oscillator amplitude.
0 = Oscillator is disabled in Stop Mode.

NOTE: *Pseudo-STOP allows for faster STOP recovery and reduces the mechanical stress and aging of the resonator in case of frequent STOP conditions at the expense of a slightly increased power consumption.*

Lower oscillator amplitude exhibits lower power consumption but could have adverse effects during any Electro-Magnetic Susceptibility (EMS) tests.

SYSWAI — System clocks stop in Wait Mode Bit

Write: anytime

1 = In Wait Mode the system clocks stop.
0 = In Wait Mode the system clocks continue to run.

NOTE: *RTI and COP are not affected by SYSWAI bit.*

ROAWAI — Reduced Oscillator Amplitude in Wait Mode Bit.

Refer to oscillator block description for availability of a reduced oscillator amplitude. If no such feature exists in the oscillator block then setting this bit to one will not have any effect on power consumption!

Write: anytime

1 = Reduced oscillator amplitude in Wait Mode.
0 = Normal oscillator amplitude in Wait Mode.

NOTE: *Lower oscillator amplitude exhibits lower power consumption but could have adverse effects during any Electro-Magnetic Susceptibility (EMS) tests.*

PLLWAI — PLL stops in Wait Mode Bit

Write: anytime

If PLLWAI is set, the CRG will clear the PLLSEL bit before entering Wait Mode. The PLLON bit remains set during Wait Mode but the PLL is powered down. Upon exiting Wait Mode, the PLLSEL bit has to be set manually if PLL clock is required.

While the PLLWAI bit is set the AUTO bit is set to 1 in order to allow the PLL to automatically lock on the selected target frequency after exiting Wait Mode.

1 = PLL stops in Wait Mode.
0 = PLL keeps running in Wait Mode.

CWAI — Core stops in Wait Mode Bit

Write: anytime

1 = Core clock stops in Wait Mode.
0 = Core clock keeps running in Wait Mode.

RTIWAI — RTI stops in Wait Mode Bit

Write: anytime

1 = RTI stops and initializes the RTI dividers whenever the part goes into Wait Mode.
0 = RTI keeps running in Wait Mode.

COPWAI — COP stops in Wait Mode Bit

Normal modes: Write once

Special modes: Write anytime

1 = COP stops and initializes the COP dividers whenever the part goes into Wait Mode.
0 = COP keeps running in Wait Mode.

3.3.7 CRG PLL Control Register (PLLCTL)

This register controls the PLL functionality.

Address Offset: $_06

	7	6	5	4	3	2	1	0
R	CME	PLLON	AUTO	ACQ	0	PRE	PCE	SCME
W								
RESET:	1	1	1	1	0	0	0	1

= Unimplemented or Reserved

Figure 3-7 CRG PLL Control Register (PLLCTL)

Read: anytime

Write: refer to each bit for individual write conditions

CME — Clock Monitor Enable Bit

CME enables the clock monitor. Write anytime except when SCM = 1.

1 = Clock monitor is enabled. Slow or stopped clocks will cause a clock monitor reset sequence or Self Clock Mode.
0 = Clock monitor is disabled.

NOTE: *Operating with CME=0 will not detect any loss of clock. In case of poor clock quality this could cause unpredictable operation of the MCU!*

In Stop Mode (PSTP=0) the clock monitor is disabled independently of the CME bit setting and any loss of clock will not be detected.

PLLON — Phase Lock Loop On Bit

PLLON turns on the PLL circuitry. In Self Clock Mode, the PLL is turned on, but the PLLON bit reads the last latched value. Write anytime except when PLLSEL = 1.

1 = PLL is turned on. If AUTO bit is set, the PLL will lock automatically.
0 = PLL is turned off.

AUTO — Automatic Bandwidth Control Bit

AUTO selects either the high bandwidth (acquisition) mode or the low bandwidth (tracking) mode depending on how close to the desired frequency the VCO is running. Write anytime except when PLLWAI=1, because PLLWAI sets the AUTO bit to 1.

1 = Automatic Mode Control is enabled and ACQ bit has no effect.
0 = Automatic Mode Control is disabled and the PLL is under software control, using ACQ bit.

ACQ — Acquisition Bit

Write anytime. If AUTO=1 this bit has no effect.

1 = High bandwidth filter is selected.
0 = Low bandwidth filter is selected.

PRE — RTI Enable during Pseudo Stop Bit

PRE enables the RTI during Pseudo Stop Mode. Write anytime.

1 = RTI continues running during Pseudo Stop Mode.
0 = RTI stops running during Pseudo Stop Mode.

NOTE: *If the PRE bit is cleared the RTI dividers will go static while Pseudo-Stop Mode is active. The RTI dividers will not initialize like in Wait Mode with RTIWAI bit set.*

PCE — COP Enable during Pseudo Stop Bit

PCE enables the COP during Pseudo Stop Mode. Write anytime.

1 = COP continues running during Pseudo Stop Mode
0 = COP stops running during Pseudo Stop Mode

NOTE: *If the PCE bit is cleared the COP dividers will go static while Pseudo-Stop Mode is active. The COP dividers will not initialize like in Wait Mode with COPWAI bit set.*

SCME — Self Clock Mode Enable Bit

Normal modes: Write once

Special modes: Write anytime

SCME can not be cleared while operating in Self Clock Mode (SCM=1).

0 = Detection of crystal clock failure causes clock monitor reset (see **5.2.1 Clock Monitor Reset**).

1 = Detection of crystal clock failure forces the MCU in Self Clock Mode (see **4.2.2 Self Clock Mode**).

3.3.8 CRG RTI Control Register (RTICTL)

This register selects the timeout period for the Real Time Interrupt.

Address Offset: $_07

	7	6	5	4	3	2	1	0
R	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-8 CRG RTI Control Register (RTICTL)

Read: anytime

Write: anytime

NOTE: *A write to this register initializes the RTI counter.*

RTR[6:4] — Real Time Interrupt Prescale Rate Select Bits

These bits select the prescale rate for the RTI. See **Table 3-2**.

RTR[3:0] — Real Time Interrupt Modulus Counter Select Bits

These bits select the modulus counter target value to provide additional granularity. **Table 3-2** shows all possible divide values selectable by the RTICTL register. The source clock for the RTI is OSCCLK.

Table 3-2 RTI Frequency Divide Rates

RTR[3:0]	RTR[6:4] =							
	000 (OFF)	**001 (2^{10})**	**010 (2^{11})**	**011 (2^{12})**	**100 (2^{13})**	**101 (2^{14})**	**110 (2^{15})**	**111 (2^{16})**
0000 (÷1)	OFF*	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
0001 (÷2)	OFF*	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010 (÷3)	OFF*	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011 (÷4)	OFF*	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100 (÷5)	OFF*	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101 (÷6)	OFF*	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$

Table 3-2 RTI Frequency Divide Rates

RTR[3:0]	RTR[6:4] =							
0110 (÷7)	OFF*	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111 (÷8)	OFF*	$8x2^{10}$	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000 (÷9)	OFF*	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001 (÷10)	OFF*	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$
1010 (÷11)	OFF*	$11x2^{10}$	$11x2^{11}$	$11x2^{12}$	$11x2^{13}$	$11x2^{14}$	$11x2^{15}$	$11x2^{16}$
1011 (÷12)	OFF*	$12x2^{10}$	$12x2^{11}$	$12x2^{12}$	$12x2^{13}$	$12x2^{14}$	$12x2^{15}$	$12x2^{16}$
1100 (÷ 13)	OFF*	$13x2^{10}$	$13x2^{11}$	$13x2^{12}$	$13x2^{13}$	$13x2^{14}$	$13x2^{15}$	$13x2^{16}$
1101 (÷14)	OFF*	$14x2^{10}$	$14x2^{11}$	$14x2^{12}$	$14x2^{13}$	$14x2^{14}$	$14x2^{15}$	$14x2^{16}$
1110 (÷15)	OFF*	$15x2^{10}$	$15x2^{11}$	$15x2^{12}$	$15x2^{13}$	$15x2^{14}$	$15x2^{15}$	$15x2^{16}$
1111 (÷ 16)	OFF*	$16x2^{10}$	$16x2^{11}$	$16x2^{12}$	$16x2^{13}$	$16x2^{14}$	$16x2^{15}$	$16x2^{16}$

* Denotes the default value out of reset.This value should be used to disable the RTI to ensure future backwards compatibility.

3.3.9 CRG COP Control Register (COPCTL)

This register controls the COP (Computer Operating Properly) watchdog.

Address Offset: $_08

	7	6	5	4	3	2	1	0
R	WCOP	RSBCK	0	0	0	CR2	CR1	CR0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-9 CRG COP Control Register (COPCTL)

Read: anytime

Write: WCOP, CR2, CR1, CR0: once in user mode, anytime in special mode

Write: RSBCK: once

WCOP — Window COP Mode Bit

When set, a write to the ARMCOP register must occur in the last 25% of the selected period. A write during the first 75% of the selected period will reset the part. As long as all writes occur during this window, $55 can be written as often as desired. Once $AA is written after the $55, the time-out logic restarts and the user must wait until the next window before writing to ARMCOP. **Table 3-3** shows the exact duration of this window for the seven available COP rates.

1 = Window COP operation
0 = Normal COP operation

RSBCK — COP and RTI stop in Active BDM mode Bit

1 = Stops the COP and RTI counters whenever the part is in Active BDM mode.
0 = Allows the COP and RTI to keep running in Active BDM mode.

CR[2:0] — COP Watchdog Timer Rate select

These bits select the COP time-out rate (see **Table 3-3**). The COP time-out period is OSCCLK period divided by CR[2:0] value. Writing a nonzero value to CR[2:0] enables the COP counter and starts the time-out period. A COP counter time-out causes a system reset. This can be avoided by periodically (before time-out) reinitializing the COP counter via the ARMCOP register.

Table 3-3 COP Watchdog Rates[1]

CR2	CR1	CR0	OSCCLK cycles to time-out
0	0	0	COP disabled
0	0	1	2^{14}
0	1	0	2^{16}
0	1	1	2^{18}
1	0	0	2^{20}
1	0	1	2^{22}
1	1	0	2^{23}
1	1	1	2^{24}

NOTES:
1. OSCCLK cycles are referenced from the previous COP time-out reset (writing $55/$AA to the ARMCOP register)

3.3.10 Reserved Register (FORBYP)

NOTE: *This reserved register is designed for factory test purposes only, and is not intended for general user access. Writing to this register when in special modes can alter the CRG's functionality.*

Address Offset: $_09

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-10 Reserved Register (FORBYP)

Read: always read $00 except in special modes

Write: only in special modes

3.3.11 Reserved Register (CTCTL)

NOTE: *This reserved register is designed for factory test purposes only, and is not intended for general user access. Writing to this register when in special test modes can alter the CRG's functionality.*

Address Offset: $_0A

	7	6	5	4	3	2	1	0
R	1	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-11 Reserved Register (CTCTL)

Read: always read $80 except in special modes

Write: only in special modes

3.3.12 CRG COP Timer Arm/Reset Register (ARMCOP)

This register is used to restart the COP time-out period.

Address Offset: $_0B

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-12 ARMCOP Register Diagram

Read: always reads $00

Write: anytime

When the COP is disabled (CR[2:0] = “000”) writing to this register has no effect.

When the COP is enabled by setting CR[2:0] nonzero, the following applies:

Writing any value other than $55 or $AA causes a COP reset. To restart the COP time-out period you must write $55 followed by a write of $AA. Other instructions may be executed between these writes but the sequence ($55, $AA) must be completed prior to COP end of time-out period to avoid a COP reset. Sequences of $55 writes or sequences of $AA writes are allowed. When the WCOP bit is set, $55 and $AA writes must be done in the last 25% of the selected time-out period; writing any value in the first 75% of the selected period will cause a COP reset.

Section 4 Functional Description

4.1 Functional Blocks

4.1.1 Phase Locked Loop (PLL)

The PLL is used to run the MCU from a different time base than the incoming OSCCLK. For increased flexibility, OSCCLK can be divided in a range of 1 to 16 to generate the reference frequency. This offers a finer multiplication granularity. The PLL can multiply this reference clock by a multiple of 2, 4, 6,... 126,128 based on the SYNR register.

$$PLLCLK = 2 \times OSCCLK \times \frac{[SYNR + 1]}{[REFDV + 1]}$$

CAUTION: *Although it is possible to set the two dividers to command a very high clock frequency, do not exceed the specified bus frequency limit for the MCU. If (PLLSEL=1), Bus Clock = PLLCLK / 2*

The PLL is a frequency generator that operates in either acquisition mode or tracking mode, depending on the difference between the output frequency and the target frequency. The PLL can change between acquisition and tracking modes either automatically or manually.

The VCO has a minimum operating frequency, which corresponds to the self clock mode frequency f_{SCM}.

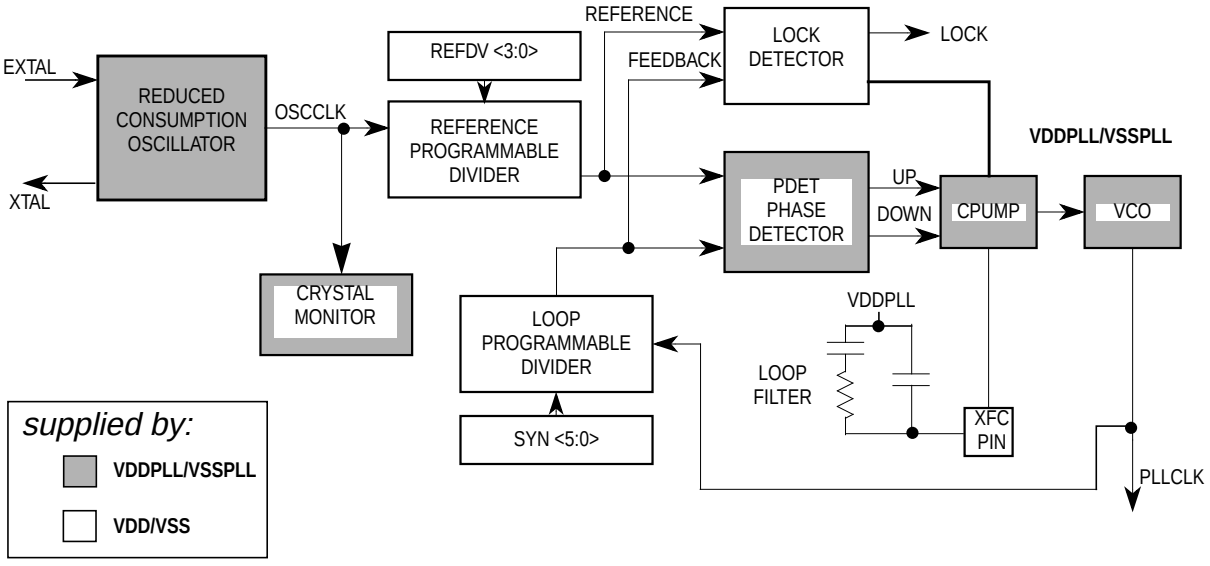


Figure 4-1 PLL Functional Diagram

4.1.1.1 PLL Operation

The oscillator output clock signal (OSCCLK) is fed through the reference programmable divider and is divided in a range of 1 to 16 (REFDV+1) to output the REFERENCE clock. The VCO output clock, (PLLCLK) is fed back through the programmable loop divider and is divided in a range of 2 to 128 in increments of [2 x (SYNR +1)] to output the FEEDBACK clock. See **Figure 4-1**.

The phase detector then compares the FEEDBACK clock, with the REFERENCE clock. Correction pulses are generated based on the phase difference between the two signals. The loop filter then slightly alters the DC voltage on the external filter capacitor connected to XFC pin, based on the width and direction of the correction pulse. The filter can make fast or slow corrections depending on its mode, as described in the next subsection. The values of the external filter network and the reference frequency determine the speed of the corrections and the stability of the PLL.

4.1.1.2 Acquisition and Tracking Modes

The lock detector compares the frequencies of the FEEDBACK clock, and the REFERENCE clock. Therefore, the speed of the lock detector is directly proportional to the final reference frequency. The circuit determines the mode of the PLL and the lock condition based on this comparison.

The PLL filter can be manually or automatically configured into one of two possible operating modes:

- Acquisition mode

  In acquisition mode, the filter can make large frequency corrections to the VCO. This mode is used at PLL start-up or when the PLL has suffered a severe noise hit and the VCO frequency is far off the desired frequency. When in acquisition mode, the TRACK status bit is cleared in the CRGFLG register.

- Tracking mode

  In tracking mode, the filter makes only small corrections to the frequency of the VCO. PLL jitter is much lower in tracking mode, but the response to noise is also slower. The PLL enters tracking mode when the VCO frequency is nearly correct and the TRACK bit is set in the CRGFLG register.

The PLL can change the bandwidth or operational mode of the loop filter manually or automatically.

In automatic bandwidth control mode (AUTO = 1), the lock detector automatically switches between acquisition and tracking modes. Automatic bandwidth control mode also is used to determine when the PLL clock (PLLCLK) is safe to use as the source for the system and core clocks. If PLL LOCK interrupt requests are enabled, the software can wait for an interrupt request and then check the LOCK bit. If CPU interrupts are disabled, software can poll the LOCK bit continuously (during PLL start-up, usually) or at periodic intervals. In either case, only when the LOCK bit is set, is the PLLCLK clock safe to use as the source for the system and core clocks. If the PLL is selected as the source for the system and core clocks and the LOCK bit is clear, the PLL has suffered a severe noise hit and the software must take appropriate action, depending on the application.

The following conditions apply when the PLL is in automatic bandwidth control mode (AUTO=1):

- The TRACK bit is a read-only indicator of the mode of the filter.

- The TRACK bit is set when the VCO frequency is within a certain tolerance, Δ_{trk}, and is clear when the VCO frequency is out of a certain tolerance, Δ_{unt}.
- The LOCK bit is a read-only indicator of the locked state of the PLL.
- The LOCK bit is set when the VCO frequency is within a certain tolerance, Δ_{Lock}, and is cleared when the VCO frequency is out of a certain tolerance, Δ_{unl}.
- CPU interrupts can occur if enabled (LOCKIE = 1) when the lock condition changes, toggling the LOCK bit.

The PLL can also operate in manual mode (AUTO = 0). Manual mode is used by systems that do not require an indicator of the lock condition for proper operation. Such systems typically operate well below the maximum system frequency (f_{sys}) and require fast start-up. The following conditions apply when in manual mode:

- ACQ is a writable control bit that controls the mode of the filter. Before turning on the PLL in manual mode, the ACQ bit should be asserted to configure the filter in acquisition mode.
- After turning on the PLL by setting the PLLON bit software must wait a given time (t_{acq}) before entering tracking mode (ACQ = 0).
- After entering tracking mode software must wait a given time (t_{al}) before selecting the PLLCLK as the source for system and core clocks (PLLSEL = 1).

4.1.2 System Clocks Generator

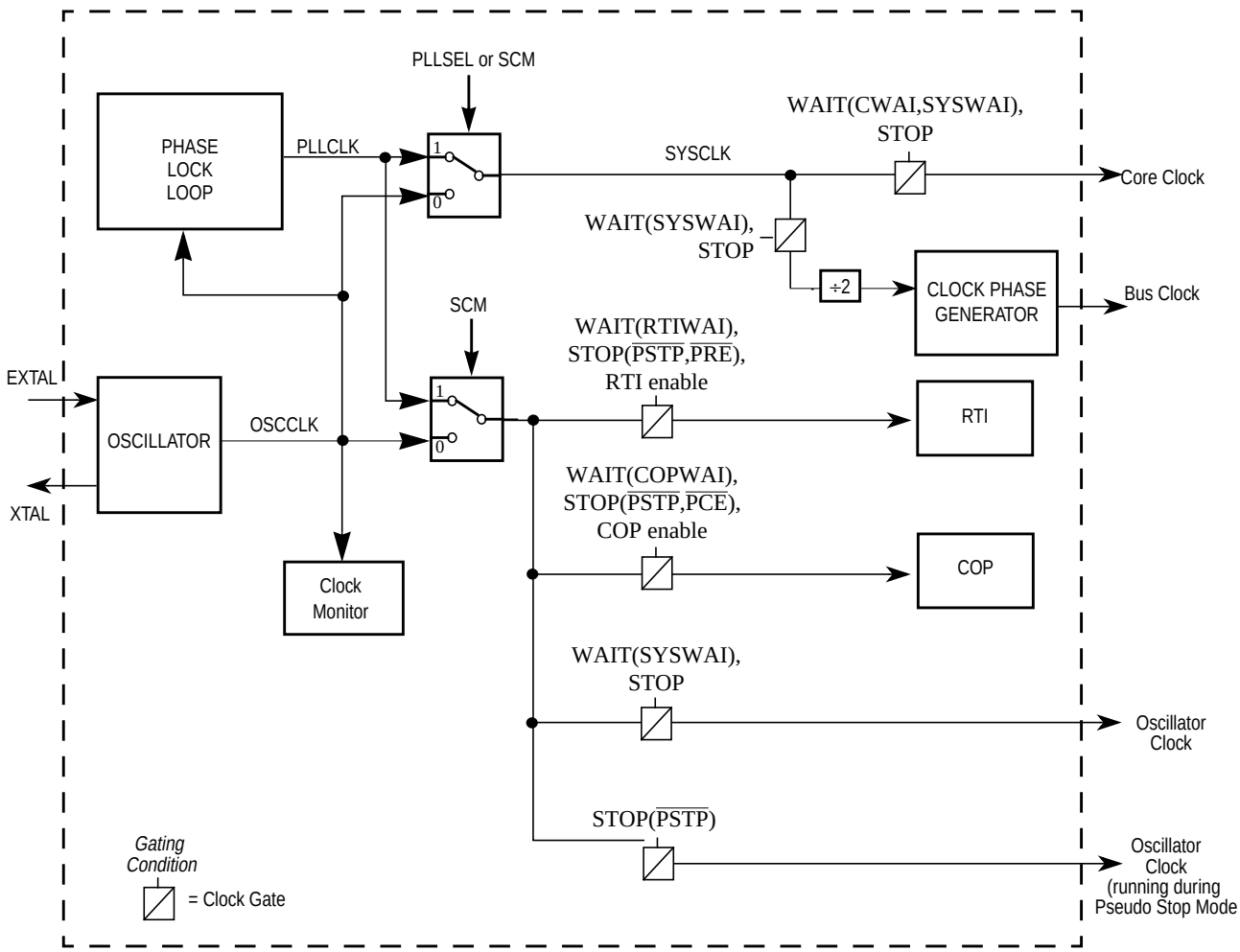


Figure 4-2 System Clocks Generator

The clock generator creates the clocks used in the MCU (see **Figure 4-2**). The gating condition placed on top of the individual clock gates indicates the dependencies of different modes (STOP, WAIT) and the setting of the respective configuration bits.

The peripheral modules use the Bus Clock. Some peripheral modules also use the Oscillator Clock. The memory blocks use the Bus Clock. If the MCU enters Self Clock Mode (see **4.2.2 Self Clock Mode**) Oscillator clock source is switched to PLLCLK running at its minimum frequency f_{SCM}. The Bus Clock is used to generate the clock visible at the ECLK pin. The Core Clock signal is the clock for the CPU. The Core Clock is twice the Bus Clock as shown in **Figure 4-3**. But note that a CPU cycle corresponds to one Bus Clock.

PLL clock mode is selected with PLLSEL bit in the CLKSEL register. When selected, the PLL output clock drives SYSCLK for the main system including the CPU and peripherals. The PLL cannot be turned off by clearing the PLLON bit, if the PLL clock is selected. When PLLSEL is changed, it takes a maximum

of 4 OSCCLK plus 4 PLLCLK cycles to make the transition. During the transition, all clocks freeze and CPU activity ceases.

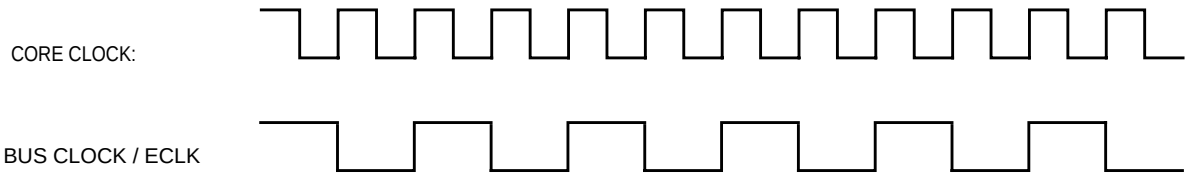


Figure 4-3 Core Clock and Bus Clock relationship

4.1.3 Clock Monitor (CM)

If no OSCCLK edges are detected within a certain time, the clock monitor within the oscillator block generates a clock monitor fail event. The CRG then asserts self clock mode or generates a system reset depending on the state of SCME bit. If the clock monitor is disabled or the presence of clocks is detected no failure is indicated by the oscillator block. The clock monitor function is enabled/disabled by the CME control bit.

4.1.4 Clock Quality Checker

The clock monitor performs a coarse check on the incoming clock signal. The clock quality checker provides a more accurate check in addition to the clock monitor.

A clock quality check is triggered by any of the following events:

- Power on reset (*POR*)
- Low voltage reset (*LVR*)
- Wake-up from Full Stop Mode (*exit full stop*)
- Clock Monitor fail indication (*CM fail*)

A time window of 50000 VCO clock cycles[1] is called *check window*.

A number greater equal than 4096 rising OSCCLK edges within a *check window* is called *osc ok*. Note that *osc ok* immediately terminates the current *check window*. See **Figure 4-4** as an example.

NOTES:
1. VCO clock cycles are generated by the PLL when running at minimum frequency f_{SCM}.

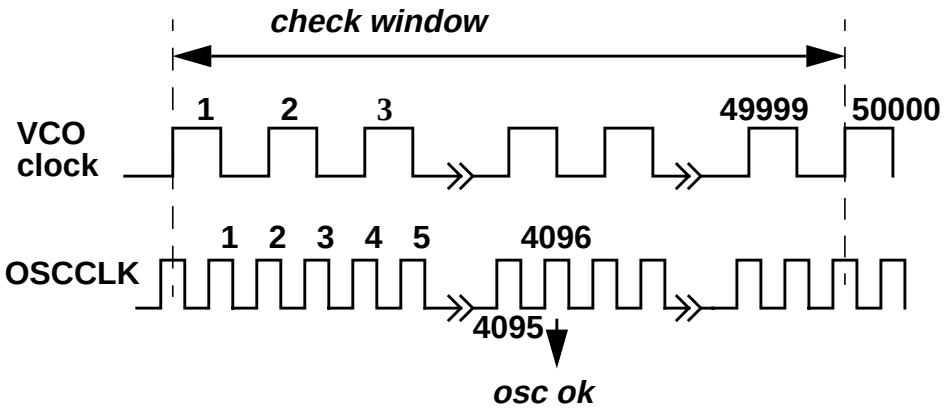


Figure 4-4 Check Window Example

The Sequence for clock quality check is shown in **Figure 4-5**.

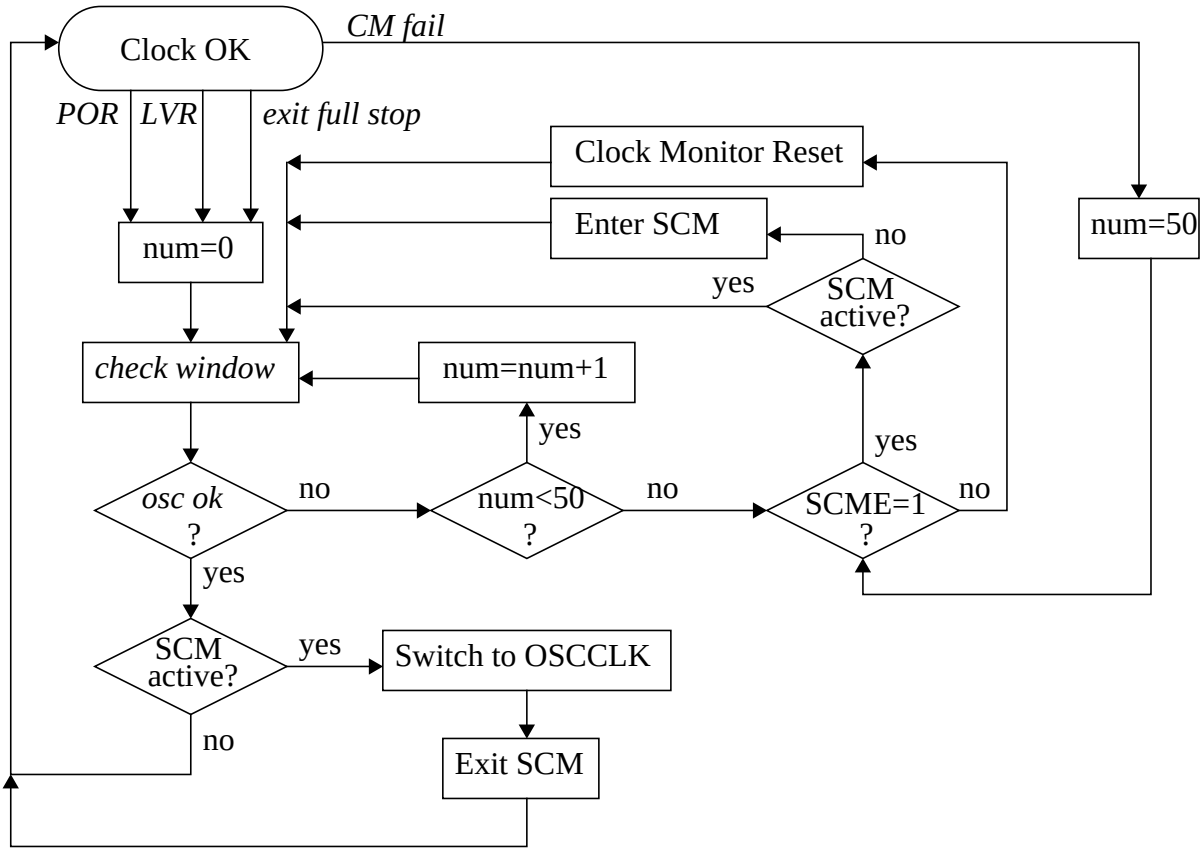


Figure 4-5 Sequence for Clock Quality Check

NOTE: *Remember that in parallel to additional actions caused by Self Clock Mode or Clock Monitor Reset[1] handling the clock quality checker* ***continues*** *to check the OSCCLK signal.*

NOTE: *The Clock Quality Checker enables the PLL and the voltage regulator (VREG) anytime a clock check has to be performed. An ongoing clock quality check could also cause a running PLL (f_{SCM}) and an active VREG during Pseudo-Stop Mode or Wait Mode.*

4.1.5 Computer Operating Properly Watchdog (COP)

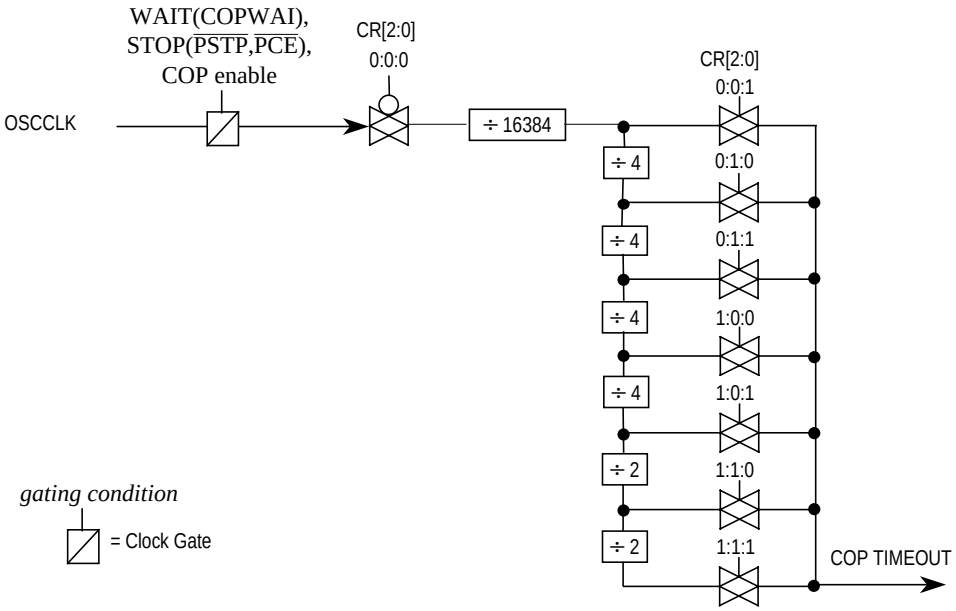


Figure 4-6 Clock Chain for COP

The COP (free running watchdog timer) enables the user to check that a program is running and sequencing properly. The COP is disabled out of reset. When the COP is being used, software is responsible for keeping the COP from timing out. If the COP times out it is an indication that the software is no longer being executed in the intended sequence; thus a system reset is initiated (see **5.2.2 Computer Operating Properly Watchdog (COP) Reset**). The COP runs with a gated OSCCLK (see **Figure 4-6 Clock Chain for COP**). Three control bits in the COPCTL register allow selection of seven COP time-out periods.

When COP is enabled, the program must write $55 and $AA (in this order) to the ARMCOP register during the selected time-out period. Once this is done, the COP time-out period is restarted. If the program

NOTES:
1. A Clock Monitor Reset will always set the SCME bit to logical'1'

fails to do this and the COP times out, the part will reset. Also, if any value other than $55 or $AA is written, the part is immediately reset.

Windowed COP operation is enabled by setting WCOP in the COPCTL register. In this mode, writes to the ARMCOP register to clear the COP timer must occur in the last 25% of the selected time-out period. A premature write will immediately reset the part.

If PCE bit is set, the COP will continue to run in Pseudo-Stop Mode.

4.1.6 Real Time Interrupt (RTI)

The RTI can be used to generate a hardware interrupt at a fixed periodic rate. If enabled (by setting RTIE=1), this interrupt will occur at the rate selected by the RTICTL register. The RTI runs with a gated OSCCLK (see **Figure 4-7 Clock Chain for RTI**). At the end of the RTI time-out period the RTIF flag is set to one and a new RTI time-out period starts immediately.

A write to the RTICTL register restarts the RTI time-out period.

If the PRE bit is set, the RTI will continue to run in Pseudo-Stop Mode.

.

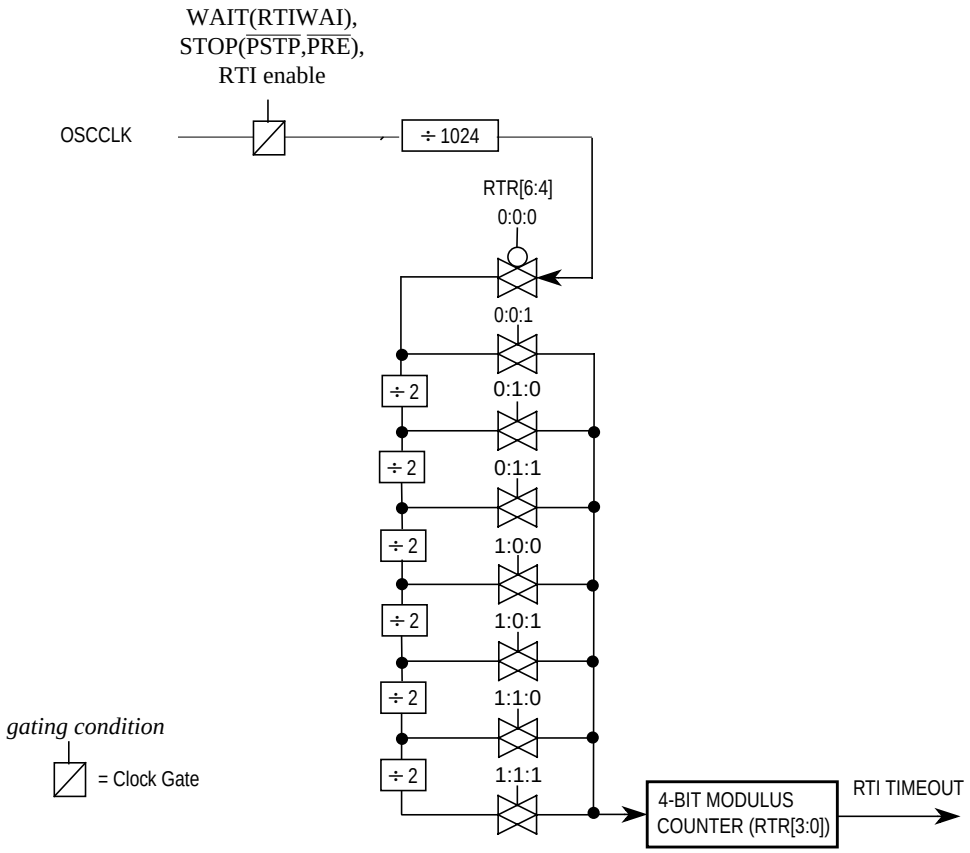


Figure 4-7 Clock Chain for RTI

4.2 Operation Modes

4.2.1 Normal Mode

The CRG block behaves as described within this specification in all normal modes.

4.2.2 Self Clock Mode

The VCO has a minimum operating frequency, f_{SCM}. If the external clock frequency is not available due to a failure or due to long crystal start-up time, the Bus Clock and the Core Clock are derived from the VCO running at minimum operating frequency; this mode of operation is called Self Clock Mode. This requires CME=1 and SCME=1. If the MCU was clocked by the PLL clock prior to entering Self Clock Mode, the PLLSEL bit will be cleared. If the external clock signal has stabilized again, the CRG will automatically select OSCCLK to be the system clock and return to normal mode. See **4.1.4 Clock Quality Checker** for more information on entering and leaving Self Clock Mode.

NOTE: *In order to detect a potential clock loss the CME bit should be always enabled (CME=1)!*

If CME bit is disabled and the MCU is configured to run on PLL clock (PLLCLK), a loss of external clock (OSCCLK) will not be detected and will cause the system clock to drift towards the VCO's minimum frequency f_{SCM}. As soon as the external clock is available again the system clock ramps up to its PLL target frequency. If the MCU is running on external clock any loss of clock will cause the system to go static.

4.3 Low Power Options

This section summarizes the low power options available in the CRG.

4.3.1 Run Mode

The RTI can be stopped by setting the associated rate select bits to zero.

The COP can be stopped by setting the associated rate select bits to zero.

4.3.2 Wait Mode

The WAI instruction puts the MCU in a low power consumption stand-by mode depending on setting of the individual bits in the CLKSEL register. All individual Wait Mode configuration bits can be superposed. This provides enhanced granularity in reducing the level of power consumption during Wait

Mode. **Table 4-1** lists the individual configuration bits and the parts of the MCU that are affected in Wait Mode.

Table 4-1 MCU configuration during Wait Mode

	PLLWAI	CWAI	SYSWAI	RTIWAI	COPWAI	ROAWAI
PLL	stopped	-	-	-	-	-
Core	-	stopped	stopped	-	-	-
System	-	-	stopped	-	-	-
RTI	-	-	-	stopped	-	-
COP	-	-	-	-	stopped	-
Oscillator	-	-	-	-	-	reduced[1]

NOTES:
1. Refer to oscillator block description for availability of a reduced oscillator amplitude!

After executing the WAI instruction the core requests the CRG to switch MCU into Wait Mode. The CRG then checks whether the PLLWAI, CWAI and SYSWAI bits are asserted (see **Figure 4-8 Wait Mode Entry/Exit Sequence**). Depending on the configuration the CRG switches the system and core clocks to OSCCLK by clearing the PLLSEL bit, disables the PLL, disables the core clocks and finally disables the remaining system clocks. As soon as all clocks are switched off Wait Mode is active.

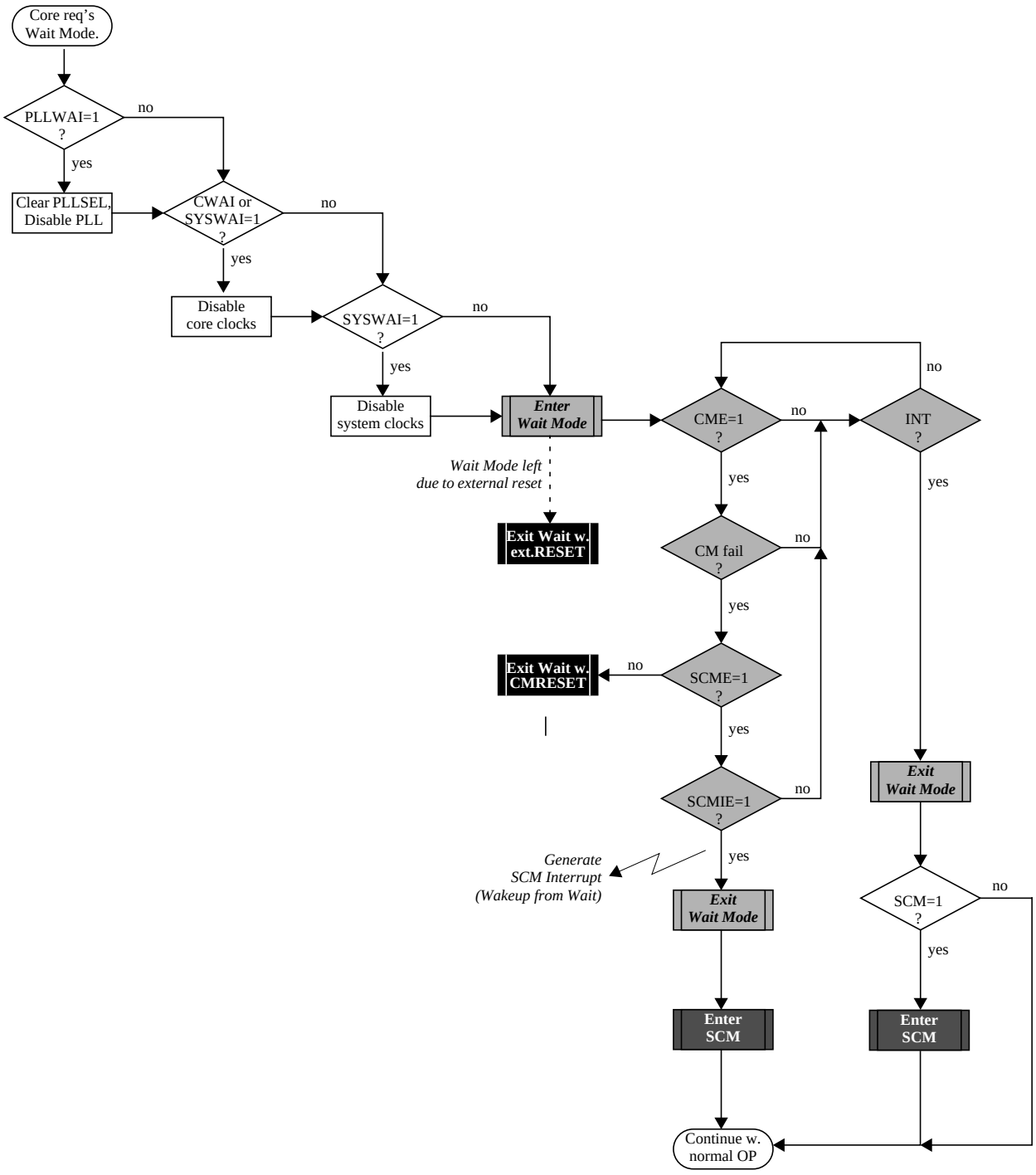


Figure 4-8 Wait Mode Entry/Exit Sequence

There are five different scenarios for the CRG to restart the MCU from Wait Mode:

- External Reset

- Clock Monitor Reset
- COP Reset
- Self Clock Mode Interrupt
- Real Time Interrupt (RTI)

If the MCU gets an external reset during Wait Mode active, the CRG asynchronously restores all configuration bits in the register space to its default settings and starts the reset generator. After completing the reset sequence processing begins by fetching the normal reset vector. Wait Mode is left and the MCU is in Run Mode again.

If the clock monitor is enabled (CME=1) the MCU is able to leave Wait-Mode when loss of oscillator/external clock is detected by a clock monitor fail. If the SCME bit is not asserted the CRG generates a clock monitor fail reset (CMRESET). The CRG's behavior for CMRESET is the same compared to external reset, but another reset vector is fetched after completion of the reset sequence. If the SCME bit is asserted the CRG generates a SCM interrupt if enabled (SCMIE=1). After generating the interrupt the CRG enters Self-Clock Mode and starts the clock quality checker (see **4.1.4 Clock Quality Checker**). Then the MCU continues with normal operation.If the SCM interrupt is blocked by SCMIE=0, the SCMIF flag will be asserted and clock quality checks will be performed but the MCU will not wake-up from Wait-Mode.

If any other interrupt source (e.g. RTI) triggers exit from Wait Mode the MCU immediately continues with normal operation.If the PLL has been powered-down during Wait-Mode the PLLSEL bit is cleared and the MCU runs on OSCCLK after leaving Wait-Mode. The software must manually set the PLLSEL bit again, in order to switch system and core clocks to the PLLCLK.

If Wait Mode is entered from Self-Clock Mode the CRG will continue to check the clock quality until clock check is successful. The PLL and voltage regulator (VREG) will remain enabled.

Table 4-2 summarizes the outcome of a clock loss while in Wait Mode.

Table 4-2 Outcome of Clock Loss in Wait Mode

CME	SCME	SCMIE	CRG Actions
0	X	X	Clock failure --> No action, clock loss not detected.
1	0	X	Clock failure --> CRG performs Clock Monitor Reset immediately

Table 4-2 Outcome of Clock Loss in Wait Mode

CME	SCME	SCMIE	CRG Actions
1	1	0	Clock failure --> Scenario 1: OSCCLK **recovers** prior to exiting Wait Mode. – MCU remains in Wait Mode, – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – Set SCMIF interrupt flag. *Some time later OSCCLK recovers.* – CM no longer indicates a failure, – 4096 OSCCLK cycles later Clock Quality Check indicates clock o.k., – SCM deactivated, – PLL disabled depending on PLLWAI, – VREG remains enabled *(never gets disabled in Wait Mode)*. – MCU remains in Wait Mode. S*ome time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Wait Mode using OSCCLK as system clock (SYSCLK), – Continue normal operation. *or an External Reset is applied.* – Exit Wait Mode using OSCCLK as system clock, – Start reset sequence. Scenario 2: OSCCLK **does not recover** prior to exiting Wait Mode. – MCU remains in Wait Mode, – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – Set SCMIF interrupt flag, – Keep performing Clock Quality Checks (could continue infinitely) while in Wait Mode. S*ome time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Wait Mode in SCM using PLL clock (f_{SCM}) as system clock, – Continue to perform additional Clock Quality Checks until OSCCLK is o.k. again. *or an External RESET is applied.* – Exit Wait Mode in SCM using PLL clock (f_{SCM}) as system clock, – Start reset sequence, – Continue to perform additional Clock Quality Checks until OSCCLK is o.k.again.

Table 4-2 Outcome of Clock Loss in Wait Mode

CME	SCME	SCMIE	CRG Actions
1	1	1	Clock failure --> – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – SCMIF set. *SCMIF generates Self Clock Mode wakeup interrupt.* – Exit Wait Mode in SCM using PLL clock (f_{SCM}) as system clock, – Continue to perform a additional Clock Quality Checks until OSCCLK is o.k. again.

4.3.3 CPU Stop Mode

All clocks are stopped in STOP mode, dependent of the setting of the PCE, PRE and PSTP bit. The oscillator is disabled in STOP mode unless the PSTP bit is set. All counters and dividers remain frozen but do not initialize. If the PRE or PCE bits are set, the RTI or COP continues to run in Pseudo-Stop Mode. In addition to disabling system and core clocks the CRG requests other functional units of the MCU (e.g. voltage-regulator) to enter their individual powersaving modes (if available). This is the main difference between Pseudo-Stop Mode and Wait Mode.

After executing the STOP instruction the core requests the CRG to switch the MCU into Stop Mode. If the PLLSEL bit is still set when entering Stop-Mode, the CRG will switch the system and core clocks to OSCCLK by clearing the PLLSEL bit. Then the CRG disables the PLL, disables the core clock and finally disables the remaining system clocks. As soon as all clocks are switched off Stop-Mode is active.

If Pseudo-Stop Mode (PSTP=1) is entered from Self-Clock Mode the CRG will continue to check the clock quality until clock check is successful. The PLL and the voltage regulator (VREG) will remain enabled. If Full-Stop Mode (PSTP=0) is entered from Self-Clock Mode an ongoing clock quality check will be stopped. A complete timeout window check will be started when Stop Mode is left again.

Wake-up from Stop-Mode also depends on the setting of the PSTP bit.

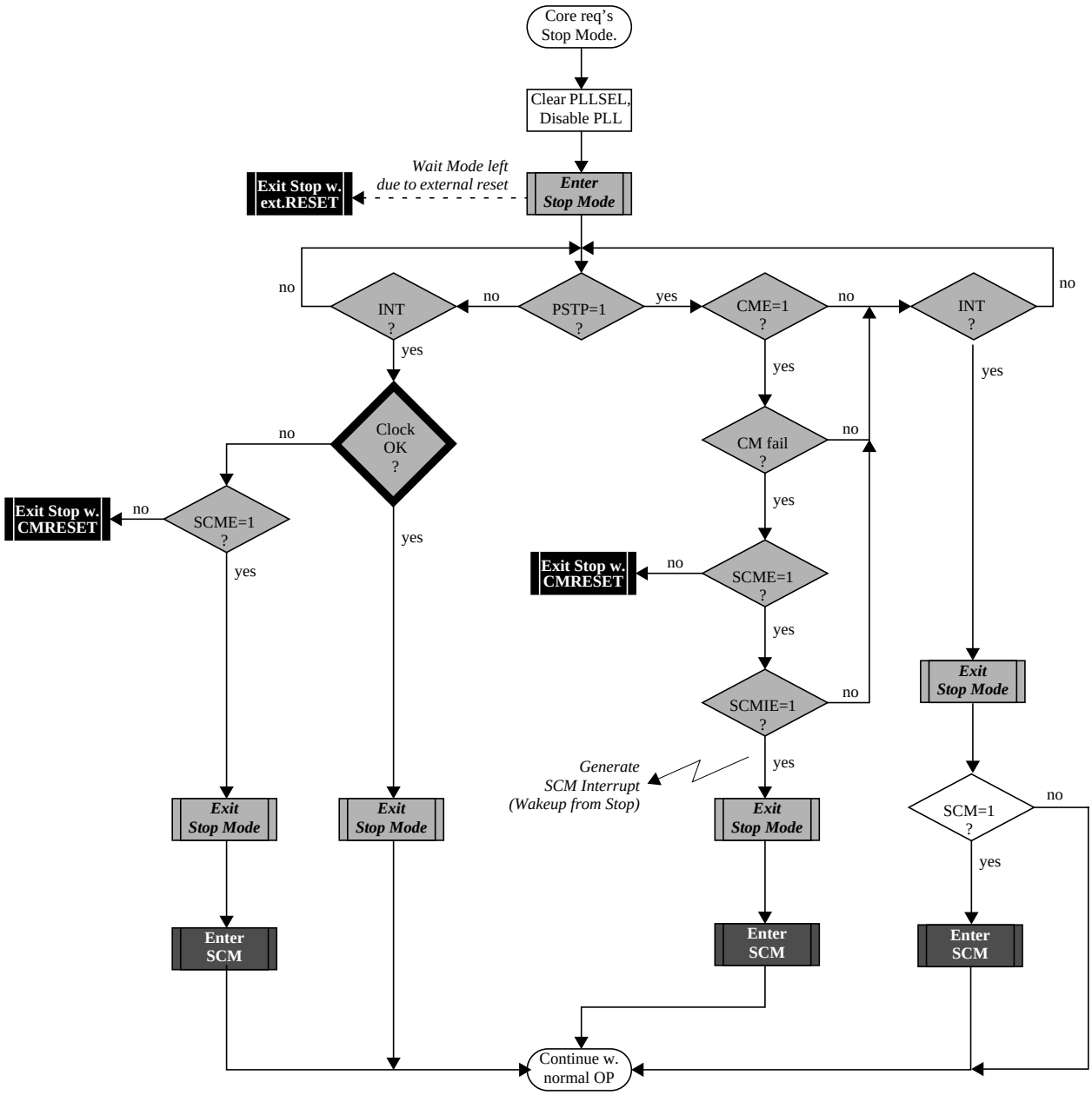


Figure 4-9 Stop Mode Entry/Exit Sequence

4.3.3.1 Wake-up from Pseudo-Stop (PSTP=1)

Wake-up from Pseudo-Stop is the same as wake-up from Wait-Mode. There are also three different scenarios for the CRG to restart the MCU from Pseudo-Stop Mode:

- External Reset
- Clock Monitor Fail
- Wake-up Interrupt

If the MCU gets an external reset during Pseudo-Stop Mode active, the CRG asynchronously restores all configuration bits in the register space to its default settings and starts the reset generator. After completing the reset sequence processing begins by fetching the normal reset vector. Pseudo-Stop Mode is left and the MCU is in Run Mode again.

If the clock monitor is enabled (CME=1) the MCU is able to leave Pseudo-Stop Mode when loss of oscillator/external clock is detected by a clock monitor fail. If the SCME bit is not asserted the CRG generates a clock monitor fail reset (CMRESET). The CRG's behavior for CMRESET is the same compared to external reset, but another reset vector is fetched after completion of the reset sequence. If the SCME bit is asserted the CRG generates a SCM interrupt if enabled (SCMIE=1). After generating the interrupt the CRG enters Self-Clock Mode and starts the clock quality checker (see **4.1.4 Clock Quality Checker**). Then the MCU continues with normal operation. If the SCM interrupt is blocked by SCMIE=0, the SCMIF flag will be asserted but the CRG will not wake-up from Pseudo-Stop Mode.

If any other interrupt source (e.g. RTI) triggers exit from Pseudo-Stop Mode the MCU immediately continues with normal operation. Because the PLL has been powered-down during Stop-Mode the PLLSEL bit is cleared and the MCU runs on OSCCLK after leaving Stop-Mode. The software must set the PLLSEL bit again, in order to switch system and core clocks to the PLLCLK.

Table 4-3 summarizes the outcome of a clock loss while in Pseudo-Stop Mode.

Table 4-3 Outcome of Clock Loss in Pseudo-Stop Mode

CME	SCME	SCMIE	CRG Actions
0	X	X	Clock failure --> No action, clock loss not detected.
1	0	X	Clock failure --> CRG performs Clock Monitor Reset immediately

Table 4-3 Outcome of Clock Loss in Pseudo-Stop Mode

CME	SCME	SCMIE	CRG Actions
1	1	0	Clock Monitor failure --> Scenario 1: OSCCLK **recovers** prior to exiting Pseudo-Stop Mode. – MCU remains in Pseudo-Stop Mode, – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – Set SCMIF interrupt flag. *Some time later OSCCLK recovers.* – CM no longer indicates a failure, – 4096 OSCCLK cycles later Clock Quality Check indicates clock o.k., – SCM deactivated, – PLL disabled, – VREG disabled. – MCU remains in Pseudo-Stop Mode. S*ome time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Pseudo-Stop Mode using OSCCLK as system clock (SYSCLK), – Continue normal operation. *or an External Reset is applied.* – Exit Pseudo-Stop Mode using OSCCLK as system clock, – Start reset sequence. Scenario 2: OSCCLK **does not recover** prior to exiting Pseudo-Stop Mode. – MCU remains in Pseudo-Stop Mode, – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – Set SCMIF interrupt flag, – Keep performing Clock Quality Checks (could continue infinitely) while in Pseudo-Stop Mode. S*ome time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Pseudo-Stop Mode in SCM using PLL clock (f_{SCM}) as system clock – Continue to perform additional Clock Quality Checks until OSCCLK is o.k. again. *or an External RESET is applied.* – Exit Pseudo-Stop Mode in SCM using PLL clock (f_{SCM}) as system clock – Start reset sequence, – Continue to perform additional Clock Quality Checks until OSCCLK is o.k.again.

Table 4-3 Outcome of Clock Loss in Pseudo-Stop Mode

CME	SCME	SCMIE	CRG Actions
1	1	1	Clock failure --> – VREG enabled, – PLL enabled, – SCM activated, – Start Clock Quality Check, – SCMIF set. *SCMIF generates Self Clock Mode wakeup interrupt.* – Exit Pseudo-Stop Mode in SCM using PLL clock (f_{SCM}) as system clock, – Continue to perform a additional Clock Quality Checks until OSCCLK is o.k. again.

4.3.3.2 Wake-up from Full Stop (PSTP=0)

The MCU requires an external interrupt or an external reset in order to wake-up from Stop-Mode.

If the MCU gets an external reset during Full Stop Mode active, the CRG asynchronously restores all configuration bits in the register space to its default settings and will perform a maximum of 50 clock *check_windows*(see **4.1.4 Clock Quality Checker**). After completing the clock quality check the CRG starts the reset generator. After completing the reset sequence processing begins by fetching the normal reset vector. Full Stop-Mode is left and the MCU is in Run Mode again.

If the MCU is woken-up by an interrupt, the CRG will also perform a maximum of 50 clock *check_window*s (see **4.1.4 Clock Quality Checker**). If the clock quality check is successful, the CRG will release all system and core clocks and will continue with normal operation. If all clock checks within the Timeout-Window are failing, the CRG will switch to Self-Clock Mode or generate a clock monitor reset (CMRESET) depending on the setting of the SCME bit.

Because the PLL has been powered-down during Stop-Mode the PLLSEL bit is cleared and the MCU runs on OSCCLK after leaving Stop-Mode. The software must manually set the PLLSEL bit again, in order to switch system and core clocks to the PLLCLK.

NOTE: *In Full Stop Mode the clock monitor is disabled and any loss of clock will not be detected.*

Section 5 Resets

5.1 General

This section describes how to reset the CRG and how the CRG itself controls the reset of the MCU. It explains all special reset requirements. Since the reset generator for the MCU is part of the CRG this section also describes all automatic actions that occur during or as a result of individual reset conditions. The reset values of registers and signals are provided in **Section 3 Memory Map and Registers**. All reset sources are listed in **Table 5-1**. Refer to MCU specification for related vector addresses and priorities.

Table 5-1Reset Summary

Reset Source	Local Enable
Power on Reset	None
Low Voltage Reset	None
External Reset	None
Clock Monitor Reset	PLLCTL (CME=1, SCME=0)
COP Watchdog Reset	COPCTL (CR[2:0] nonzero)

5.2 Description of Reset Operation

The reset sequence is initiated by any of the following events:

- Low level is detected at the $\overline{\text{RESET}}$ pin (External Reset).
- Power on is detected.
- Low voltage is detected.
- COP watchdog times out.
- Clock monitor failure is detected and Self-Clock Mode was disabled (SCME=0).

Upon detection of any reset event, an internal circuit drives the $\overline{\text{RESET}}$ pin low for 128 SYSCLK cycles (see **Figure 5-1**). Since entry into reset is asynchronous it does not require a running SYSCLK. However, the internal reset circuit of the CRG cannot sequence out of current reset condition without a running SYSCLK. The number of 128 SYSCLK cycles might be increased by n=3 to 6 additional SYSCLK cycles depending on the internal synchronization latency. After 128+n SYSCLK cycles the $\overline{\text{RESET}}$ pin is released. The reset generator of the CRG waits for additional 64 SYSCLK cycles and then samples the RESET pin to determine the originating source. **Table 5-2** shows which vector will be fetched.

Table 5-2 Reset Vector Selection

sampled $\overline{\text{RESET}}$ pin (64 cycles after release)	Clock Monitor Reset pending	COP Reset pending	Vector fetch
1	0	0	POR / LVR / External Reset
1	1	X	Clock Monitor Reset
1	0	1	COP Reset

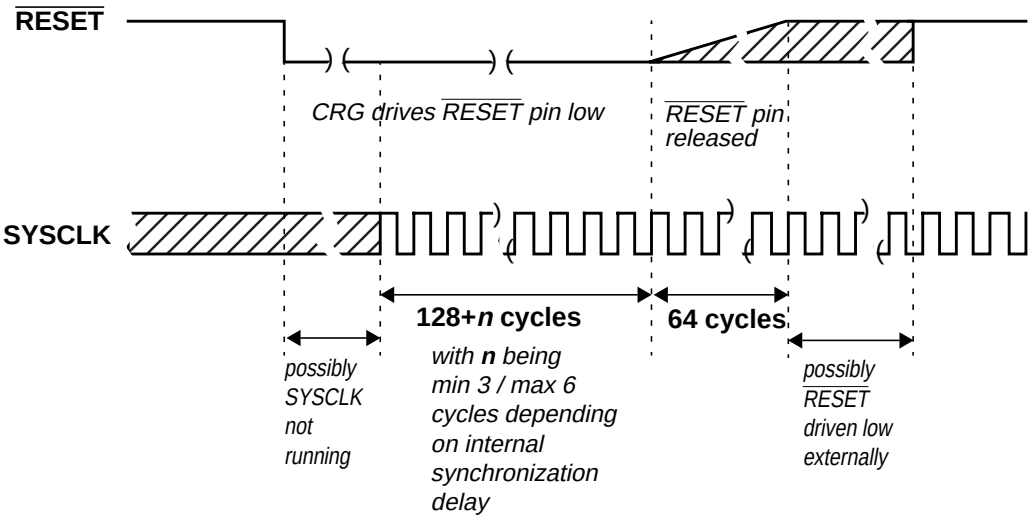
Table 5-2 Reset Vector Selection

sampled $\overline{\text{RESET}}$ pin (64 cycles after release)	Clock Monitor Reset pending	COP Reset pending	Vector fetch
0	X	X	POR / LVR / External Reset with rise of $\overline{\text{RESET}}$ pin

NOTE: *External circuitry connected to the $\overline{\text{RESET}}$ pin should not include a large capacitance that would interfere with the ability of this signal to rise to a valid logic one within 64 SYSCLK cycles after the low drive is released.*

The internal reset of the MCU remains asserted while the reset generator completes the 192 SYSCLK long reset sequence. The reset generator circuitry always makes sure the internal reset is deasserted synchronously after completion of the 192 SYSCLK cycles. In case the $\overline{\text{RESET}}$ pin is externally driven low for more than these 192 SYSCLK cycles (External Reset), the internal reset remains asserted too.

Figure 5-1 RESET Timing



5.2.1 Clock Monitor Reset

The CRG generates a Clock Monitor Reset in case all of the following conditions are true:

- Clock monitor is enabled (CME=1)
- Loss of clock is detected
- Self-Clock Mode is disabled (SCME=0).

The reset event asynchronously forces the configuration registers to their default settings (see **Section Section 3 Memory Map and Registers**). In detail the CME and the SCME are reset to logical ‘1’ (which doesn’t change the state of the CME bit, because it has already been set). As a consequence the CRG

immediately enters Self Clock Mode and starts its internal reset sequence. In parallel the clock quality check starts. As soon as clock quality check indicates a valid Oscillator Clock the CRG switches to OSCCLK and leaves Self Clock Mode. Since the clock quality checker is running in parallel to the reset generator, the CRG may leave Self Clock Mode while still completing the internal reset sequence. When the reset sequence is finished the CRG checks the internally latched state of the clock monitor fail circuit. If a clock monitor fail is indicated processing begins by fetching the Clock Monitor Reset vector.

5.2.2 Computer Operating Properly Watchdog (COP) Reset

When COP is enabled, the CRG expects sequential write of $55 and $AA (in this order) to the ARMCOP register during the selected time-out period. Once this is done, the COP time-out period restarts. If the program fails to do this the CRG will generate a reset. Also, if any value other than $55 or $AA is written, the CRG immediately generates a reset. In case windowed COP operation is enabled writes ($55 or $AA) to the ARMCOP register must occur in the last 25% of the selected time-out period. A premature write the CRG will immediately generate a reset.

As soon as the reset sequence is completed the reset generator checks the reset condition. If no clock monitor failure is indicated and the latched state of the COP timeout is true, processing begins by fetching the COP vector.

5.2.3 Power On Reset, Low Voltage Reset

The on-chip voltage regulator detects when VDD to the MCU has reached a certain level and asserts power on reset or low voltage reset or both. As soon as a power on reset or low voltage reset is triggered the CRG performs a quality check on the incoming clock signal. As soon as clock quality check indicates a valid Oscillator Clock signal the reset sequence starts using the Oscillator clock. If after 50 check windows the clock quality check indicated a non-valid Oscillator Clock the reset sequence starts using Self-Clock Mode.

Figure 5-2 and **Figure 5-3** show the power-up sequence for cases when the $\overline{\text{RESET}}$ pin is tied to VDD and when the $\overline{\text{RESET}}$ pin is held low.

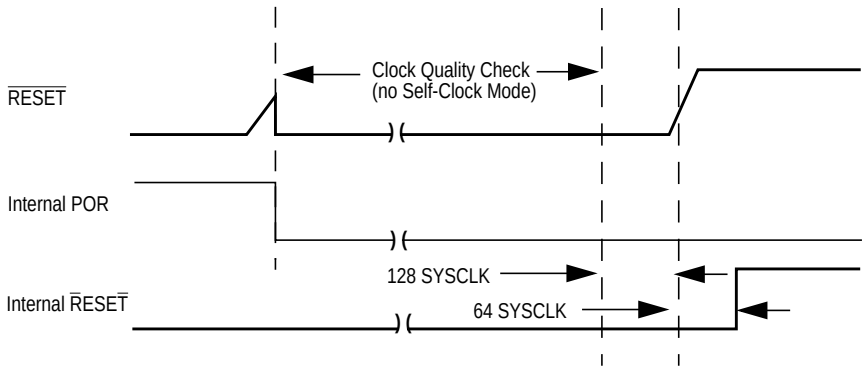


Figure 5-2 $\overline{\text{RESET}}$ pin tied to VDD (by a pull-up resistor)

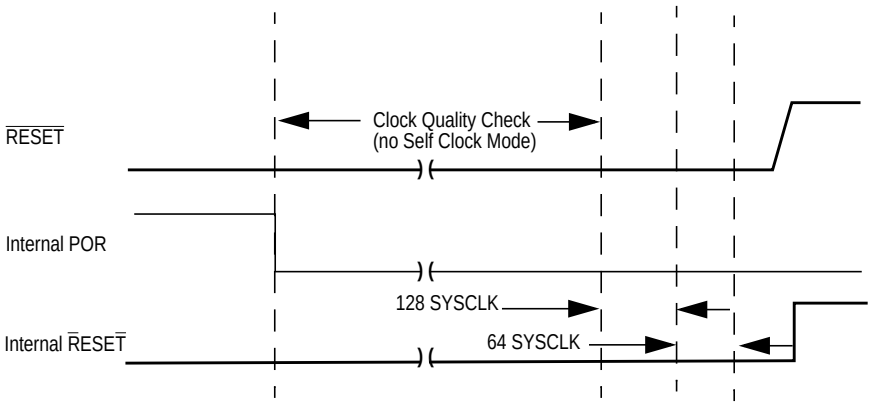


Figure 5-3 RESET pin held low externally

Section 6 Interrupts

6.1 General

The interrupts/reset vectors requested by the CRG are listed in **Table 6-1**. Refer to MCU specification for related vector addresses and priorities.

Table 6-1 CRG Interrupt Vectors

Interrupt Source	CCR Mask	Local Enable
Real time interrupt	I bit	CRGINT (RTIE)
LOCK interrupt	I bit	CRGINT (LOCKIE)
SCM interrupt	I bit	CRGINT (SCMIE)

6.2 Description of Interrupt Operation

6.2.1 Real Time Interrupt

The CRG generates a real time interrupt when the selected interrupt time period elapses. RTI interrupts are locally disabled by setting the RTIE bit to zero. The real time interrupt flag (RTIF) is set to1 when a timeout occurs, and is cleared to 0 by writing a 1 to the RTIF bit.

The RTI continues to run during Pseudo Stop Mode if the PRE bit is set to 1. This feature can be used for periodic wakeup from Pseudo Stop if the RTI interrupt is enabled.

6.2.2 PLL Lock Interrupt

The CRG generates a PLL Lock interrupt when the LOCK condition of the PLL has changed, either from a locked state to an unlocked state or vice versa. Lock interrupts are locally disabled by setting the LOCKIE bit to zero. The PLL Lock interrupt flag (LOCKIF) is set to1 when the LOCK condition has changed, and is cleared to 0 by writing a 1 to the LOCKIF bit.

6.2.3 Self Clock Mode Interrupt

The CRG generates a Self Clock Mode interrupt when the SCM condition of the system has changed, either entered or exited Self Clock Mode. SCM conditions can only change if the Self Clock Mode enable bit (SCME) is set to 1. SCM conditions are caused by a failing clock quality check after power on reset (POR) or low voltage reset (LVR) or recovery from Full Stop Mode (PSTP=0) or Clock Monitor failure. For details on the clock quality check refer to **4.1.4 Clock Quality Checker**. If the clock monitor is enabled (CME=1) a loss of external clock will also cause a SCM condition (SCME=1).

SCM interrupts are locally disabled by setting the SCMIE bit to zero. The SCM interrupt flag (SCMIF) is set to1 when the SCM condition has changed, and is cleared to 0 by writing a 1 to the SCMIF bit.

FINAL PAGE OF
54
PAGES

MC9S12DP256
Port Integration Module (PIM)
Block Guide
V03.05

Covers also
MC9S12DP512

Original Release Date: 19 NOV 2001
Revised: 24 MAR 2005

Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V02.00	19 FEB 2001			Initial version for 2nd Barracuda revision started from Integration Guide PIM_9DP256 V01.00. Summary of changes: - Added Port A, B, E, K, and BKGD pin. - Added MODRR register. - Moved priority information into Table 2-1 and removed Table 4-1 - Removed reference to IPBus from Block Diagram
V02.01	28 MAR 2001	28 MAR 2001		- Updated due to requirements in SRS supplement
V02.02	17 JUL 2001	17 JUL 2001		- 1st official version by Technical Publishing
V02.03	03 AUG 2001	03 AUG 2001		- Capitalized all pin names to match Barracuda DUG. - Added full register names in memory map table. - Corrected typo in PPSJ description.
V02.04	11 OCT 2001	11 OCT 2001		- Updated references w.r.t. new family name HCS12.
V02.05	31 OCT 2001	31 OCT 2001		- Minor cleanup.
V02.06	12 NOV 2001	12 NOV 2001		- Removed subsection on unbonded port pins on 80 pin package. Shall be stated in DUG.
V03.00	19 NOV 2001	11 FEB 2001		- Added CAN0 to PJ[7:6] routing feature. - Removed table notes from table 3-4 to 3-6. - Added port ABEK note in Overview. - Updated block diagram. - Removed ports ABEK from table 2-1. - Changed to SRSv3 document format. - General cosmetic changes. - Replaced key wakeup limits by variabeles defined in DUG.
V03.01	04 MAR 2002	04 MAR 2002		- Document format updates.
V03.02	23 APR 2002	23 APR 2002		- Corrected PTH I/O register bit swap of bit 7 & 6 (SS2, SCK2).
V03.03	04 NOV 2002	04 NOV 2002		- Made document generic for 9DP256 and 9DT256 derivatives by introducing conditional text.
V03.04	24 JUL 2003	24 JUL 2003		- Corrected column header in table Pulse Detection Criteria
V03.05	24 MAR 2005	24 JUL 2003		- Added conditional text note on front page of DP256 doc version to cover also DP512.

Table of Contents

Section 5 Initialization/Application Information

List of Figures

List of Tables

Preface

Terminology

Acronyms and Abbreviations	

Section 1 Introduction

1.1 Overview

The Port Integration Module establishes the interface between the peripheral modules and the I/O pins for all ports except AD0 and AD1.

NOTE: *Port A, B, E, and K are related to the core logic and multiplexed bus interface. Refer to the HCS12 Core User Guide for details.*

This section covers:

- Port T connected to the timer module
- The serial port S associated with 2 SCI and 1 SPI modules
- Port M associated with 4 CAN and 1 BDLC module
- Port P connected to the PWM and 2 SPI modules, which also can be used as an external interrupt source
- The standard I/O ports H and J associated with the first and fifth CAN module and the IIC interface. These ports can also be used as external interrupt sources.

Each I/O pin can be configured by several registers in order to select data direction and drive strength, to enable and select pull-up or pull-down resistors. On certain pins also interrupts can be enabled which result in status flags.

The I/O's of 2 CAN and all 3 SPI modules can be routed from their default location to determined pins.

The implementation of the Port Integration Module is device dependent.

1.2 Features

A standard port pin has the following minimum features:

- Input/output selection
- 5V output drive with two selectable drive strengths
- 5V digital and analog input
- Input with selectable pull-up or pull-down device

Optional features:

- Open drain for wired-or connections
- Interrupt inputs with glitch filtering

1.3 Block Diagram

The following figure is a block diagram of the PIM_9DP256.

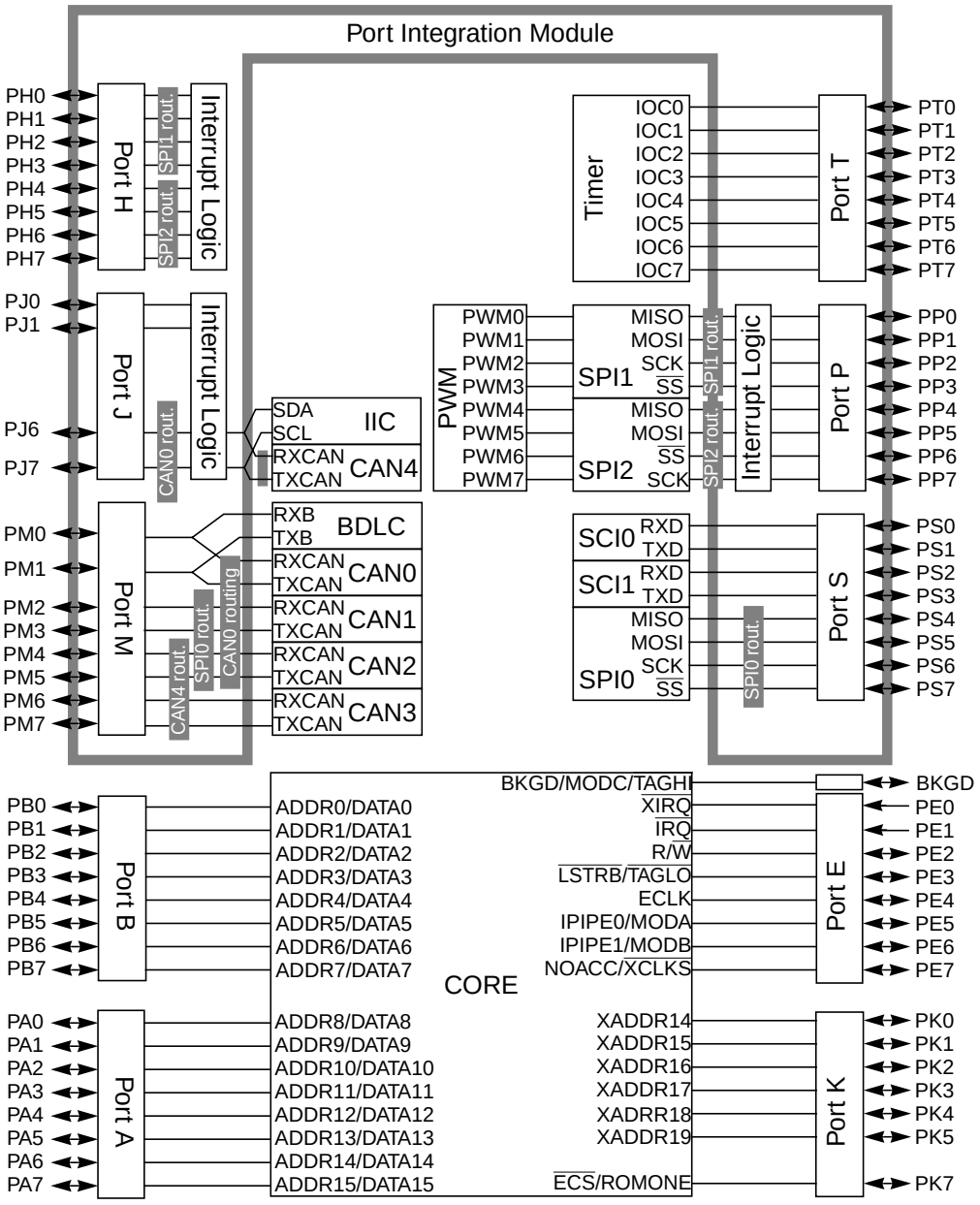


Figure 1-1 PIM_9DP256 Block Diagram

Section 2 External Signal Description

2.1 Overview

This section lists and describes the signals that do connect off-chip.

2.2 Signal properties

Table 2-1 shows all the pins and their functions that are controlled by the PIM_9DP256. If there is more than one function associated with a pin, the priority is indicated by the position in the table from top (highest priority) to down (lowest priority).

Table 2-1 Pin Functions and Priorities

Port	Pin Name	Pin Function & Priority	Description	Pin Function after Reset
Port T	PT[7:0]	IOC[7:0]	Enhanced Capture Timer Channels 7 to 0	GPIO
		GPIO	General-purpose I/O	
Port S	PS7	$\overline{SS0}$	Serial Peripheral Interface 0 slave select output in master mode, input in slave mode or master mode.	GPIO
		GPIO	General-purpose I/O	
	PS6	SCK0	Serial Peripheral Interface 0 serial clock pin	
		GPIO	General-purpose I/O	
	PS5	MOSI0	Serial Peripheral Interface 0 master out/slave in pin	
		GPIO	General-purpose I/O	
	PS4	MISO0	Serial Peripheral Interface 0 master in/slave out pin	
		GPIO	General-purpose I/O	
	PS3	TXD1	Serial Communication Interface 1 transmit pin	
		GPIO	General-purpose I/O	
	PS2	RXD1	Serial Communication Interface 1 receive pin	
		GPIO	General-purpose I/O	
	PS1	TXD0	Serial Communication Interface 0 transmit pin	
		GPIO	General-purpose I/O	
	PS0	RXD0	Serial Communication Interface 0 receive pin	
		GPIO	General-purpose I/O	

Port	Pin Name	Pin Function & Priority	Description	Pin Function after Reset
Port M	PM7	TXCAN3	MSCAN3 transmit pin	GPIO
		TXCAN4	MSCAN4 transmit pin	
		GPIO	General-purpose I/O	
	PM6	RXCAN3	MSCAN3 receive pin	
		RXCAN4	MSCAN4 receive pin	
		GPIO	General-purpose I/O	
	PM5	TXCAN2	MSCAN2 transmit pin	
		TXCAN0	MSCAN0 transmit pin	
		TXCAN4	MSCAN4 transmit pin	
		SCK0	Serial Peripheral Interface 0 serial clock pin	
		GPIO	General-purpose I/O	
	PM4	RXCAN2	MSCAN2 receive pin	
		RXCAN0	MSCAN0 receive pin	
		RXCAN4	MSCAN4 receive pin	
		MOSI0	Serial Peripheral Interface 0 master out/slave in pin	
		GPIO	General-purpose I/O	
	PM3	TXCAN1	MSCAN1 transmit pin	
		TXCAN0	MSCAN0 transmit pin	
		$\overline{SS0}$[1]	Serial Peripheral Interface 0 slave select output in master mode, input for slave mode or master mode.	
		GPIO	General-purpose I/O	
	PM2	RXCAN1	MSCAN1 receive pin	
		RXCAN0	MSCAN0 receive pin	
		MISO0[1]	Serial Peripheral Interface 0 master in/slave out pin	
		GPIO	General-purpose I/O	
	PM1	TXCAN0	MSCAN0 transmit pin	
		TXB	BDLC transmit pin	
		GPIO	General-purpose I/O	
	PM0	RXCAN0	MSCAN0 receive pin	
		RXB	BDLC receive pin	
		GPIO	General-purpose I/O	

Port	Pin Name	Pin Function & Priority	Description	Pin Function after Reset
Port P	PP7	PWM7	Pulse Width Modulator channel 7	GPIO
		SCK2	Serial Peripheral Interface 2 serial clock pin	
		GPIO/KWP7	General-purpose I/O with interrupt	
	PP6	PWM6	Pulse Width Modulator channel 6	
		$\overline{SS2}$	Serial Peripheral Interface 2 slave select output in master mode, input for slave mode or master mode.	
		GPIO/KWP6	General-purpose I/O with interrupt	
	PP5	PWM5	Pulse Width Modulator channel 5	
		MOSI2	Serial Peripheral Interface 2 master out/slave in pin	
		GPIO/KWP5	General-purpose I/O with interrupt	
	PP4	PWM4	Pulse Width Modulator channel 4	
		MISO2	Serial Peripheral Interface 2 master in/slave out pin	
		GPIO/KWP4	General-purpose I/O with interrupt	
	PP3	PWM3	Pulse Width Modulator channel 3	
		$\overline{SS1}$	Serial Peripheral Interface 1 slave select output in master mode, input for slave mode or master mode.	
		GPIO/KWP3	General-purpose I/O with interrupt	
	PP2	PWM2	Pulse Width Modulator channel 2	
		SCK1	Serial Peripheral Interface 1 serial clock pin	
		GPIO/KWP2	General-purpose I/O with interrupt	
	PP1	PWM1	Pulse Width Modulator channel 1	
		MOSI1	Serial Peripheral Interface 1 master out/slave in pin	
		GPIO/KWP1	General-purpose I/O with interrupt	
	PP0	PWM0	Pulse Width Modulator channel 0	
		MISO1	Serial Peripheral Interface 1 master in/slave out pin	
		GPIO/KWP0	General-purpose I/O with interrupt	

Port	Pin Name	Pin Function & Priority	Description	Pin Function after Reset
Port H	PH7	$\overline{SS2}$	Serial Peripheral Interface 2 slave select output in master mode, input for slave mode or master mode.	GPIO
		GPIO/KWH7	General-purpose I/O with interrupt	
	PH6	SCK2	Serial Peripheral Interface 2 serial clock pin	
		GPIO/KWH6	General-purpose I/O with interrupt	
	PH5	MOSI2	Serial Peripheral Interface 2 master out/slave in pin	
		GPIO/KWH5	General-purpose I/O with interrupt	
	PH4	MISO2	Serial Peripheral Interface 2 master in/slave out pin	
		GPIO/KWH4	General-purpose I/O with interrupt	
	PH3	$\overline{SS1}$	Serial Peripheral Interface 1 slave select output in master mode, input for slave mode or master mode.	
		GPIO/KWH3	General-purpose I/O with interrupt	
	PH2	SCK1	Serial Peripheral Interface 1 serial clock pin	
		GPIO/KWH2	General-purpose I/O with interrupt	
	PH1	MOSI1	Serial Peripheral Interface 1 master out/slave in pin	
		GPIO/KWH1	General-purpose I/O with interrupt	
	PH0	MISO1	Serial Peripheral Interface 1 master in/slave out pin	
		GPIO/KWH0	General-purpose I/O with interrupt	
Port J	PJ7	TXCAN4	MSCAN4 transmit pin	GPIO
		SCL	Inter Integrated Circuit serial clock line	
		TXCAN0	MSCAN0 transmit pin	
		GPIO/KWJ7	General-purpose I/O with interrupt	
	PJ6	RXCAN4	MSCAN4 receive pin	
		SDA	Inter Integrated Circuit serial data line	
		RXCAN0	MSCAN0 receive pin	
		GPIO/KWJ6	General-purpose I/O with interrupt	
	PJ[1:0]	GPIO/KWJ[1:0]	General-purpose I/O with interrupt	

NOTES:

1. If CAN0 is routed to PM[3:2] the SPI0 can still be used in bidirectional master mode. *Refer to SPI Block Guide for details.*

Section 3 Memory Map/Register Definition

This section provides a detailed description of all registers.

The following table shows the register map of the Port Integration Module.

Table 3-1 PIM_9DP256 Memory Map

Address offset	Use	Access
$00	Port T I/O Register (PTT)	RW
$01	Port T Input Register (PTIT)	R
$02	Port T Data Direction Register (DDRT)	RW
$03	Port T Reduced Drive Register (RDRT)	RW
$04	Port T Pull Device Enable Register (PERT)	RW
$05	Port T Polarity Select Register (PPST)	RW
$06	Reserved	-
$07	Reserved	-
$08	Port S I/O Register (PTS)	RW
$09	Port S Input Register (PTIS)	R
$0A	Port S Data Direction Register (DDRS)	RW
$0B	Port S Reduced Drive Register (RDRS)	RW
$0C	Port S Pull Device Enable Register (PERS)	RW
$0D	Port S Polarity Select Register (PPSS)	RW
$0E	Port S Wired-Or Mode Register (WOMS)	RW
$0F	Reserved	-
$10	Port M I/O Register (PTM)	RW
$11	Port M Input Register (PTIM)	R
$12	Port M Data Direction Register (DDRM)	RW
$13	Port M Reduced Drive Register (RDRM)	RW
$14	Port M Pull Device Enable Register (PERM)	RW
$15	Port M Polarity Select Register (PPSM)	RW
$16	Port M Wired-Or Mode Register (WOMM)	RW
$17	Module Routing Register (MODRR)	RW
$18	Port P I/O Register (PTP)	RW
$19	Port P Input Register (PTIP)	R
$1A	Port P Data Direction Register (DDRP)	RW
$1B	Port P Reduced Drive Register (RDRP)	RW
$1C	Port P Pull Device Enable Register (PERP)	RW
$1D	Port P Polarity Select Register (PPSP)	RW
$1E	Port P Interrupt Enable Register (PIEP)	RW
$1F	Port P Interrupt Flag Register (PIFP)	RW
$20	Port H I/O Register (PTH)	RW
$21	Port H Input Register (PTIH)	R
$22	Port H Data Direction Register (DDRH)	RW
$23	Port H Reduced Drive Register (RDRH)	RW
$24	Port H Pull Device Enable Register (PERH)	RW
$25	Port H Polarity Select Register (PPSH)	RW

$26	Port H Interrupt Enable Register (PIEH)	RW
$27	Port H Interrupt Flag Register (PIFH)	RW
$28	Port J I/O Register (PTJ)	RW[1]
$29	Port J Input Register (PTIJ)	R
$2A	Port J Data Direction Register (DDRJ)	RW[1]
$2B	Port J Reduced Drive Register (RDRJ)	RW[1]
$2C	Port J Pull Device Enable Register (PERJ)	RW[1]
$2D	Port J Polarity Select Register (PPSJ)	RW[1]
$2E	Port J Interrupt Enable Register (PIEJ)	RW[1]
$2F	Port J Interrupt Flag Register (PIFJ)	RW[1]
$30 - $3F	Reserved	-

NOTES:

1. Write access not applicable for one or more register bits. Please refer to detailed signal description.

NOTE: *Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.*

3.1 Register Descriptions

The following table summarizes the effect on the various configuration bits, data direction (DDR), output level (I/O), reduced drive (RDR), pull enable (PE), pull select (PS) and interrupt enable (IE) for the ports. The configuration bit PS is used for two purposes:

1. Configure the sensitive interrupt edge (rising or falling), if interrupt is enabled.
2. Select either a pull-up or pull-down device if PE is active.

Table 3-2 Pin Configuration Summary

DDR	IO	RDR	PE	PS	IE[1]	Function	Pull Device	Interrupt
0	X	X	0	X	0	Input	Disabled	Disabled
0	X	X	1	0	0	Input	Pull Up	Disabled
0	X	X	1	1	0	Input	Pull Down	Disabled
0	X	X	0	0	1	Input	Disabled	falling edge
0	X	X	0	1	1	Input	Disabled	rising edge
0	X	X	1	0	1	Input	Pull Up	falling edge
0	X	X	1	1	1	Input	Pull Down	rising edge
1	0	0	X	X	0	Output, full drive to 0	Disabled	Disabled
1	1	0	X	X	0	Output, full drive to 1	Disabled	Disabled
1	0	1	X	X	0	Output, reduced drive to 0	Disabled	Disabled
1	1	1	X	X	0	Output, reduced drive to 1	Disabled	Disabled
1	0	0	X	0	1	Output, full drive to 0	Disabled	falling edge
1	1	0	X	1	1	Output, full drive to 1	Disabled	rising edge
1	0	1	X	0	1	Output, reduced drive to 0	Disabled	falling edge
1	1	1	X	1	1	Output, reduced drive to 1	Disabled	rising edge

NOTES:
1. Applicable only on port P, H and J.

NOTE: *All bits of all registers in this module are completely synchronous to internal clocks during a register read.*

3.1.1 Port T Registers

Address Offset: $__00

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write	PTT7	PTT6	PTT5	PTT4	PTT3	PTT2	PTT1	PTT0
ECT:	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-1 Port T I/O Register (PTT)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.

Address Offset: $__01

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIT7	PTIT6	PTIT5	PTIT4	PTIT3	PTIT2	PTIT1	PTIT0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-2 Port T Input Register (PTIT)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This can also be used to detect overload or short circuit conditions on output pins.

Address Offset: $__02

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-3 Port T Data Direction Register (DDRT)

Read:Anytime.

Write:Anytime.

This register configures each port T pin as either input or output.
The ECT forces the I/O state to be an output for each timer port associated with an enabled output compare. In these cases the data direction bits will not change.
The DDRT bits revert to controlling the I/O direction of a pin when the associated timer output compare is disabled.
The timer input capture always monitors the state of the pin.

DDRT[7:0] — Data Direction Port T

1 = Associated pin is configured as output.
0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTT or PTIT registers, when changing the DDRT register.

Address Offset: $__03

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RDRT7	RDRT6	RDRT5	RDRT4	RDRT3	RDRT2	RDRT1	RDRT0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-4 Port T Reduced Drive Register (RDRT)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port T output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRT[7:0] — Reduced Drive Port T

1 = Associated pin drives at about 1/3 of the full drive strength.
0 = Full drive strength at output.

Address Offset: $__04

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PERT7	PERT6	PERT5	PERT4	PERT3	PERT2	PERT1	PERT0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-5 Port T Pull Device Enable Register (PERT)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input. This bit has no effect if the port is used as output. Out of reset no pull device is enabled.

PERT[7:0] — Pull Device Enable Port T

1 = Either a pull-up or pull-down device is enabled.
0 = Pull-up or pull-down device is disabled.

Address Offset: $__05

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PPST7	PPST6	PPST5	PPST4	PPST3	PPST2	PPST1	PPST0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-6 Port T Polarity Select Register (PPST)

Read:Anytime.

Write:Anytime.

This register selects whether a pull-down or a pull-up device is connected to the pin.

PPST[7:0] — Pull Select Port T

1 = A pull-down device is connected to the associated port T pin, if enabled by the associated bit in register PERT and if the port is used as input.

0 = A pull-up device is connected to the associated port T pin, if enabled by the associated bit in register PERT and if the port is used as input.

3.1.2 Port S Registers

Address Offset: $__08

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PTS7	PTS6	PTS5	PTS4	PTS3	PTS2	PTS1	PTS0
SPI/SCI	SS0	SCK0	MOSI0	MISO0	TXD1	RXD1	TXD0	RXD0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-7 Port S I/O Register (PTS)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.
The SPI pins (PS[7:4]) configuration is determined by several status bits in the SPI module. *Refer to SPI Block Guide for details*.
The SCI ports associated with transmit pins 3 and 1 are configured as outputs if the transmitter is enabled. The SCI ports associated with receive pins 2 and 0 are configured as inputs if the receiver is enabled. *Refer to SCI Block Guide for details.*

Address Offset: $__09

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIS7	PTIS6	PTIS5	PTIS4	PTIS3	PTIS2	PTIS1	PTIS0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-8 Port S Input Register (PTIS)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This also can be used to detect overload or short circuit conditions on output pins.

Address Offset:$__0A

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRS7	DDRS6	DDRS5	DDRS4	DDRS3	DDRS2	DDRS1	DDRS0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-9 Port S Data Direction Register (DDRS)

Read:Anytime.

Write:Anytime.

This register configures each port S pin as either input or output
If SPI is enabled, the SPI determines the pin direction. *Refer to SPI Block Guide for details.*
If the associated SCI transmit or receive channel is enabled this register has no effect on the pins. The pin is forced to be an output if a SCI transmit channel is enabled, it is forced to be an input if the SCI receive channel is enabled.
The DDRS bits revert to controlling the I/O direction of a pin when the associated channel is disabled.

DDRS[7:0] — Data Direction Port S
1 = Associated pin is configured as output.
0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTS or PTIS registers, when changing the DDRS register.

Address Offset: $__0B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RDRS7	RDRS6	RDRS5	RDRS4	RDRS3	RDRS2	RDRS1	RDRS0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-10 Port S Reduced Drive Register (RDRS)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port S output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRS[7:0] — Reduced Drive Port S

1 = Associated pin drives at about 1/3 of the full drive strength.

0 = Full drive strength at output.

Address Offset: $__0C

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PERS7	PERS6	PERS5	PERS4	PERS3	PERS2	PERS1	PERS0
Reset:	1	1	1	1	1	1	1	1

= Reserved or unimplemented

Figure 3-11 Port S Pull Device Enable Register (PERS)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input or as output in wired-or (open drain) mode. This bit has no effect if the port is used as push-pull output. Out of reset a pull-up device is enabled.

PERS[7:0] — Pull Device Enable Port S

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: $__0D

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PPSS7	PPSS6	PPSS5	PPSS4	PPSS3	PPSS2	PPSS1	PPSS0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-12 Port S Polarity Select Register (PPSS)

Read:Anytime.

Write:Anytime.

This register selects whether a pull-down or a pull-up device is connected to the pin.

PPSS[7:0] — Pull Select Port S

1 = A pull-down device is connected to the associated port S pin, if enabled by the associated bit in register PERS and if the port is used as input.

0 = A pull-up device is connected to the associated port S pin, if enabled by the associated bit in register PERS and if the port is used as input or as wired-or output.

Address Offset: $__0E

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	WOMS7	WOMS6	WOMS5	WOMS4	WOMS3	WOMS2	WOMS1	WOMS0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-13 Port S Wired-Or Mode Register (WOMS)

Read:Anytime.

Write:Anytime.

This register configures the output pins as wired-or. If enabled the output is driven active low only (open-drain). A logic level of “1” is not driven. It applies also to the SPI and SCI outputs and allows a multipoint connection of several serial modules. This bit has no influence on pins used as inputs.

WOMS[7:0] — Wired-Or Mode Port S

1 = Output buffers operate as open-drain outputs.

0 = Output buffers operate as push-pull outputs.

3.1.3 Port M Registers

Address Offset: $__10

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PTM7	PTM6	PTM5	PTM4	PTM3	PTM2	PTM1	PTM0
CAN:	TXCAN3	RXCAN3	TXCAN2	RXCAN2	TXCAN1	RXCAN1	TXCAN0	RXCAN0
BDLC:							TXB	RXB
CAN0:			TXCAN0	RXCAN0	TXCAN0	RXCAN0		
CAN4:	TXCAN4	RXCAN4	TXCAN4	RXCAN4				
SPI0:			SCK0	MOSI0	SS0	MISO0		
Reset	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-14 Port M I/O Register (PTM)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.

PM[7:6]

The CAN3 function (TXCAN3 and RXCAN3) takes precedence over the CAN4 and the general purpose I/O function if the CAN3 module is enabled.
The CAN4 function (TXCAN4 and RXCAN4) takes precedence over the general purpose I/O function if the CAN4 module is enabled. *Refer to MSCAN Block Guide for details.*

PM[5:4]

The CAN2 function (TXCAN2 and RXCAN2) takes precedence over the CAN0, CAN4, the SPI0 and the general purpose I/O function if the CAN2 module is enabled.
The CAN0 function (TXCAN0 and RXCAN0) takes precedence over the CAN4, the SPI0 and the general purpose I/O function if the CAN0 module is enabled.
The CAN4 function (TXCAN4 and RXCAN4) takes precedence over the SPI and general purpose I/O function if the CAN4 module is enabled. *Refer to MSCAN Block Guide for details.*
The SPI0 function (SCK0 and MOSI0) takes precedence of the general purpose I/O function if the SPI0 is enabled. *Refer to SPI Block Guide for details.*

PM[3:2]

The CAN1 function (TXCAN1 and RXCAN1) takes precedence over the CAN0, the SPI0 and the general purpose I/O function if the CAN1 module is enabled.
The CAN0 function (TXCAN0 and RXCAN0) takes precedence over the SPI0 and the general purpose I/O function if the CAN0 module is enabled. *Refer to MSCAN Block Guide for details.*
The SPI0 function ($\overline{SS0}$ and MISO0) takes precedence of the general purpose I/O function if the SPI0 is enabled and not in bidirectional mode. *Refer to SPI Block Guide for details.*

PM[1:0]

The CAN0 function (TXCAN0 and RXCAN0) takes precedence over the BDLC and the general purpose I/O function if the CAN0 module is enabled. *Refer to MSCAN Block Guide for details.*
The BDLC function takes precedence over the general purpose I/O function associated if enabled. *Refer to BDLC Block Guide for details.*

Address Offset: $__11

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIM7	PTIM6	PTIM5	PTIM4	PTIM3	PTIM2	PTIM1	PTIM0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-15 Port M Input Register (PTIM)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This can also be used to detect overload or short circuit conditions on output pins.

Address Offset: $__12

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRM7	DDRM6	DDRM5	DDRM4	DDRM3	DDRM2	DDRM1	DDRM0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-16 Port M Data Direction Register (DDRM)

Read:Anytime.

Write:Anytime.

This register configures each port M pin as either input or output.
The CAN/BDLC forces the I/O state to be an output for each port line associated with an enabled output (TXCAN[3:0], TXB). It also forces the I/O state to be an input for each port line associated with an enabled input (RXCAN[3:0], RXB). In those cases the data direction bits will not change.
The DDRM bits revert to controlling the I/O direction of a pin when the associated peripheral module is disabled.

DDRM[7:0] — Data Direction Port M
1 = Associated pin is configured as output.
0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTM or PTIM registers, when changing the DDRM register.

Address Offset: $__13

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RDRM7	RDRM6	RDRM5	RDRM4	RDRM3	RDRM2	RDRM1	RDRM0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-17 Port M Reduced Drive Register (RDRM)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port M output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRM[7:0] — Reduced Drive Port M

1 = Associated pin drives at about 1/3 of the full drive strength.

0 = Full drive strength at output.

Address Offset: $__14

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PERM7	PERM6	PERM5	PERM4	PERM3	PERM2	PERM1	PERM0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-18 Port M Pull Device Enable Register (PERM)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input or wired-or output. This bit has no effect if the port is used as push-pull output. Out of reset no pull device is enabled.

PERM[7:0] — Pull Device Enable Port M

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: $__15

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PPSM7	PPSM6	PPSM5	PPSM4	PPSM3	PPSM2	PPSM1	PPSM0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-19 Port M Polarity Select Register (PPSM)

Read:Anytime.

Write:Anytime.

This register selects whether a pull-down or a pull-up device is connected to the pin. If CAN is active a pull-up device can be activated on the RXCAN[3:0] inputs, but not a pull-down. If BDLC is active a pull-down device can be activated on the RXB pin but not a pull-up.

PPSM[7:0] — Pull Select Port M

1 = A pull-down device is connected to the associated port M pin, if enabled by the associated bit in register PERM and if the port is used as a general purpose or BDLC input but not as RXCAN.

0 = A pull-up device is connected to the associated port M pin, if enabled by the associated bit in register PERM and if the port is used as general purpose or RXCAN input but not as BDLC.

Address Offset: $__16

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	WOMM7	WOMM6	WOMM5	WOMM4	WOMM3	WOMM2	WOMM1	WOMM0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-20 Port M Wired-Or Mode Register (WOMM)

Read:Anytime.

Write:Anytime.

This register configures the output pins as wired-or. If enabled the output is driven active low only (open-drain). A logic level of “1” is not driven. It applies also to the CAN and BDLC outputs and allows a multipoint connection of several serial modules. This bit has no influence on pins used as inputs.

WOMM[7:0] — Wired-Or Mode Port M

1 = Output buffers operate as open-drain outputs.

0 = Output buffers operate as push-pull outputs.

Address Offset: $__17

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	MODRR6	MODRR5	MODRR4	MODRR3	MODRR2	MODRR1	MODRR0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-21 Module Routing Register (MODRR)

Read:Anytime.

Write:Anytime.

This register configures the re-routing of CAN0, CAN4, SPI0, SPI1, and SPI2 on defined port pins.

MODRR[1:0] — CAN0 Routing

Table 3-3 CAN0 Routing

MODRR[1]	MODRR[0]	RXCAN0	TXCAN0
0	0	PM0	PM1
0	1	PM2	PM3
1	0	PM4	PM5
1	1	PJ6	PJ7

MODRR[3:2] — CAN4 Routing

Table 3-4 CAN4 Routing

MODRR[3]	MODRR[2]	RXCAN4	TXCAN4
0	0	PJ6	PJ7
0	1	PM4	PM5
1	0	PM6	PM7
1	1	Reserved	

MODRR[4] — SPI0 Routing

Table 3-5 SPI0 Routing

MODRR[4]	MISO0	MOSI0	SCK0	$\overline{SS0}$
0	PS4	PS5	PS6	PS7
1	PM2	PM4	PM5	PM3

MODRR[5] — SPI1 Routing

Table 3-6 SPI1 Routing

MODRR[5]	MISO1	MOSI1	SCK1	$\overline{SS1}$
0	PP0	PP1	PP2	PP3
1	PH0	PH1	PH2	PH3

MODRR[6] — SPI2 Routing

Table 3-7 SPI2 Routing

MODRR[6]	MISO2	MOSI2	SCK2	$\overline{SS2}$
0	PP4	PP5	PP7	PP6
1	PH4	PH5	PH6	PH7

3.1.4 Port P Registers

Address Offset: $__18

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PTP7	PTP6	PTP5	PTP4	PTP3	PTP2	PTP1	PTP0
PWM:	PWM7	PWM6	PWM5	PWM4	PWM3	PWM2	PWM1	PWM0
SPI:	SCK2	$\overline{SS2}$	MOSI2	MISO2	$\overline{SS1}$	SCK1	MOSI1	MISO1
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-22 Port P I/O Register (PTP)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.
The PWM function takes precedence over the general purpose I/O function if the associated PWM channel is enabled. While channels 6-0 are output only if the respective channel is enabled, channel 7 can be PWM output or input if the shutdown feature is enabled. *Refer to PWM Block Guide for details.*
The SPI function takes precedence over the general purpose I/O function associated with if enabled. *Refer to SPI Block Guide for details.*
If both PWM and SPI are enabled the PWM functionality takes precedence.

Address Offset: $__19

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIP7	PTIP6	PTIP5	PTIP4	PTIP3	PTIP2	PTIP1	PTIP0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-23 Port P Input Register (PTIP)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This can be also used to detect overload or short circuit conditions on output pins.

Address Offset: $__1A

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRP7	DDRP6	DDRP5	DDRP4	DDRP3	DDRP2	DDRP1	DDRP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-24 Port P Data Direction Register (DDRP)

Read:Anytime.

Write:Anytime.

This register configures each port P pin as either input or output.
If the associated PWM channel or SPI module is enabled this register has no effect on the pins.
The PWM forces the I/O state to be an output for each port line associated with an enabled PWM7-0 channel. Channel 7 can force the pin to input if the shutdown feature is enabled.
If a SPI module is enabled, the SPI determines the pin direction. *Refer to SPI Block Guide for details.*
The DDRM bits revert to controlling the I/O direction of a pin when the associated PWM channel is disabled.

DDRP[7:0] — Data Direction Port P
1 = Associated pin is configured as output.
0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTP or PTIP registers, when changing the DDRP register.

Address Offset: $__1B

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RDRP7	RDRP6	RDRP5	RDRP4	RDRP3	RDRP2	RDRP1	RDRP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-25 Port P Reduced Drive Register (RDRP)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port P output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRP[7:0] — Reduced Drive Port P

1 = Associated pin drives at about 1/3 of the full drive strength.

0 = Full drive strength at output.

Address Offset: $__1C

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PERP7	PERP6	PERP5	PERP4	PERP3	PERP2	PERP1	PERP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-26 Port P Pull Device Enable Register (PERP)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input. This bit has no effect if the port is used as output. Out of reset no pull device is enabled.

PERP[7:0] — Pull Device Enable Port P

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: $__1D

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PPSP7	PPSP6	PPSP5	PPSP4	PPSP3	PPSP2	PPSP1	PPSP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-27 Port P Polarity Select Register (PPSP)

Read:Anytime.

Write:Anytime.

This register serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.

PPSP[7:0] — Polarity Select Port P

1 = Rising edge on the associated port P pin sets the associated flag bit in the PIFP register.A pull-down device is connected to the associated port P pin, if enabled by the associated bit in register PERP and if the port is used as input.

0 = Falling edge on the associated port P pin sets the associated flag bit in the PIFP register.A pull-up device is connected to the associated port P pin, if enabled by the associated bit in register PERP and if the port is used as input.

Address Offset: $__1E

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PIEP7	PIEP6	PIEP5	PIEP4	PIEP3	PIEP2	PIEP1	PIEP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-28 Port P Interrupt Enable Register (PIEP)

Read:Anytime.

Write:Anytime.

This register disables or enables on a per pin basis the edge sensitive external interrupt associated with port P.

PIEP[7:0] — Interrupt Enable Port P

1 = Interrupt is enabled.

0 = Interrupt is disabled (interrupt flag masked).

Address Offset: $__1F

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PIFP7	PIFP6	PIFP5	PIFP4	PIFP3	PIFP2	PIFP1	PIFP0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-29 Port P Interrupt Flag Register (PIFP)

Read:Anytime.

Write:Anytime.

Each flag is set by an active edge on the associated input pin. This could be a rising or a falling edge based on the state of the PPSP register. To clear this flag, write “1” to the corresponding bit in the PIFP register. Writing a “0” has no effect.

PIFP[7:0] — Interrupt Flags Port P

1 = Active edge on the associated bit has occurred (an interrupt will occur if the associated enable bit is set).
Writing a “1” clears the associated flag.

0 = No active edge pending.
Writing a “0” has no effect.

3.1.5 Port H Registers

Address Offset:$__20

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PTH7	PTH6	PTH5	PTH4	PTH3	PTH2	PTH1	PTH0
SPI:	SS2	SCK2	MOSI2	MISO2	SS1	SCK1	MOSI1	MISO1
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-30 Port H I/O Register (PTH)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.

The SPI function takes precedence over the general purpose I/O function associated with if enabled. *Refer to SPI Block Guide for details.*

Address Offset: $__21

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIH7	PTIH6	PTIH5	PTIH4	PTIH3	PTIH2	PTIH1	PTIH0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-31 Port H Input Register (PTIH)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This can also be used to detect overload or short circuit conditions on output pins.

Address Offset: $__22

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRH7	DDRH6	DDRH5	DDRH4	DDRH3	DDRH2	DDRH1	DDRH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-32 Port H Data Direction Register (DDRH)

Read:Anytime.

Write:Anytime.

This register configures each port H pin as either input or output.

DDRH[7:0] — Data Direction Port H

1 = Associated pin is configured as output.

0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTH or PTIH registers, when changing the DDRH register.

Address Offset: $__23

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	RDRH7	RDRH6	RDRH5	RDRH4	RDRH3	RDRH2	RDRH1	RDRH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-33 Port H Reduced Drive Register (RDRH)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port H output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRH[7:0] — Reduced Drive Port H

1 = Associated pin drives at about 1/3 of the full drive strength.

0 = Full drive strength at output.

Address Offset: $__24

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PERH7	PERH6	PERH5	PERH4	PERH3	PERH2	PERH1	PERH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-34 Port H Pull Device Enable Register (PERH)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input. This bit has no effect if the port is used as output. Out of reset no pull device is enabled.

PERH[7:0] — Pull Device Enable Port H

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: $__25

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PPSH7	PPSH6	PPSH5	PPSH4	PPSH3	PPSH2	PPSH1	PPSH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-35 Port H Polarity Select Register (PPSH)

Read:Anytime.

Write:Anytime.

This register serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.

PPSH[7:0] — Polarity Select Port H

1 = Rising edge on the associated port H pin sets the associated flag bit in the PIFH register. A pull-down device is connected to the associated port H pin, if enabled by the associated bit in register PERH and if the port is used as input.

0 = Falling edge on the associated port H pin sets the associated flag bit in the PIFH register. A pull-up device is connected to the associated port H pin, if enabled by the associated bit in register PERH and if the port is used as input.

Address Offset: $__26

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PIEH7	PIEH6	PIEH5	PIEH4	PIEH3	PIEH2	PIEH1	PIEH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-36 Port H Interrupt Enable Register (PIEH)

Read:Anytime.

Write:Anytime.

This register disables or enables on a per pin basis the edge sensitive external interrupt associated with port H.

PIEH[7:0] — Interrupt Enable Port H

1 = Interrupt is enabled.

0 = Interrupt is disabled (interrupt flag masked).

Address Offset: $__27

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PIFH7	PIFH6	PIFH5	PIFH4	PIFH3	PIFH2	PIFH1	PIFH0
Reset:	0	0	0	0	0	0	0	0

= Reserved or unimplemented

Figure 3-37 Port H Interrupt Flag Register (PIFH)

Read:Anytime.

Write:Anytime.

Each flag is set by an active edge on the associated input pin. This could be a rising or a falling edge based on the state of the PPSH register. To clear this flag, write “1” to the corresponding bit in the PIFH register. Writing a “0” has no effect.

PIFH[7:0] — Interrupt Flags Port H

1 = Active edge on the associated bit has occurred (an interrupt will occur if the associated enable bit is set).
Writing a “1” clears the associated flag.

0 = No active edge pending.
Writing a “0” has no effect.

3.1.6 Port J Registers

Address Offset: $__28

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTJ7	PTJ6	0	0	0	0	PTJ1	PTJ0
Write:								
CAN4:	TXCAN4	RXCAN4						
IIC:	SCL	SDA						
CAN0:	TXCAN0	RXCAN0						
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-38 Port J I/O Register (PTJ)

Read:Anytime.

Write:Anytime.

If the data direction bits of the associated I/O pins are set to 1, a read returns the value of the port register, otherwise the value at the pins is read.

PJ[7:6]

The CAN4 function (TXCAN4 and RXCAN4) takes precedence over the IIC, the CAN0 and the general purpose I/O function if the CAN4 module is enabled.

The IIC function (SCL and SDA) takes precedence over CAN0 and the general purpose I/O function if the IIC is enabled. If the IIC module takes precedence the SDA and SCL outputs are configured as open drain outputs. *Refer to IIC Block Guide for details.*

The CAN0 function (TXCAN0 and RXCAN0) takes precedence over the general purpose I/O function if the CAN0 module is enabled.*Refer to MSCAN Block Guide for details.*

Address Offset: $__29

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PTIJ7	PTIJ6	0	0	0	0	PTIJ1	PTIJ0
Write:								
Reset:	-	-	-	-	-	-	-	-

= Reserved or unimplemented

Figure 3-39 Port J Input Register (PTIJ)

Read:Anytime.

Write:Never, writes to this register have no effect.

This register always reads back the status of the associated pins. This can be used to detect overload or short circuit conditions on output pins.

Address Offset: $__2A

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	DDRJ7	DDRJ6	0	0	0	0	DDRJ1	DDRJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-40 Port J Data Direction Register (DDRJ)

Read:Anytime.

Write:Anytime.

This register configures each port J pin as either input or output.
The CAN forces the I/O state to be an output on PJ7 (TXCAN4) and an input on pin PJ6 (RXCAN4). The IIC takes control of the I/O if enabled. In these cases the data direction bits will not change. The DDRJ bits revert to controlling the I/O direction of a pin when the associated peripheral module is disabled.

DDRJ[7:6][1:0] — Data Direction Port J
1 = Associated pin is configured as output.
0 = Associated pin is configured as input.

Due to internal synchronization circuits, it can take up to 2 bus cycles until the correct value is read on PTJ or PTIJ registers, when changing the DDRJ register.

Address Offset: $__2B

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	RDRJ7	RDRJ6	0	0	0	0	RDRJ1	RDRJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-41 Port J Reduced Drive Register (RDRJ)

Read:Anytime.

Write:Anytime.

This register configures the drive strength of each port J output pin as either full or reduced. If the port is used as input this bit is ignored.

RDRJ[7:6][1:0] — Reduced Drive Port J
1 = Associated pin drives at about 1/3 of the full drive strength.
0 = Full drive strength at output.

Address Offset: $__2C

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PERJ7	PERJ6	0	0	0	0	PERJ1	PERJ0
Write:								
Reset:	1	1	-	-	-	-	1	1

= Reserved or unimplemented

Figure 3-42 Port J Pull Device Enable Register (PERJ)

Read:Anytime.

Write:Anytime.

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input or as wired-or output. This bit has no effect if the port is used as push-pull output. Out of reset a pull-up device is enabled.

PERJ[7:6][1:0] — Pull Device Enable Port J

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: $__2D

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PPSJ7	PPSJ6	0	0	0	0	PPSJ1	PPSJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-43 Port J Polarity Select Register (PPSJ)

Read:Anytime.

Write:Anytime.

This register serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.

PPSJ[7:6][1:0] — Polarity Select Port J

1 = Rising edge on the associated port J pin sets the associated flag bit in the PIFJ register. A pull-down device is connected to the associated port J pin, if enabled by the associated bit in register PERJ and if the port is used as input.

0 = Falling edge on the associated port J pin sets the associated flag bit in the PIFJ register. A pull-up device is connected to the associated port J pin, if enabled by the associated bit in register PERJ and if the port is used as general purpose input or as IIC port.

Address Offset: $__2E

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIEJ7	PIEJ6	0	0	0	0	PIEJ1	PIEJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-44 Port J Interrupt Enable Register (PIEJ)

Read:Anytime.

Write:Anytime.

This register disables or enables on a per pin basis the edge sensitive external interrupt associated with port J.

PIEJ[7:6][1:0] — Interrupt Enable Port J

1 = Interrupt is enabled.

0 = Interrupt is disabled (interrupt flag masked).

Address Offset: $__2F

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIFJ7	PIFJ6	0	0	0	0	PIFJ1	PIFJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

= Reserved or unimplemented

Figure 3-45 Port J Interrupt Flag Register (PIFJ)

Read:Anytime.

Write:Anytime.

Each flag is set by an active edge on the associated input pin. This could be a rising or a falling edge based on the state of the PPSJ register. To clear this flag, write “1” to the corresponding bit in the PIFJ register. Writing a “0” has no effect.

PIFJ[7:6][1:0] — Interrupt Flags Port J

1 = Active edge on the associated bit has occurred (an interrupt will occur if the associated enable bit is set).
Writing a “1” clears the associated flag.

0 = No active edge pending.
Writing a “0” has no effect.

Section 4 Functional Description

4.1 General

Each pin can act as general purpose I/O. In addition the pin can act as an output from a peripheral module or an input to a peripheral module.

A set of configuration registers is common to all ports. All registers can be written at any time, however a specific configuration might not become active.

Example:

Selecting a pull-up resistor. This resistor does not become active while the port is used as a push-pull output.

4.1.1 I/O register

This register holds the value driven out to the pin if the port is used as a general purpose I/O.

Writing to this register has only an effect on the pin if the port is used as general purpose output. When reading this address, the value of the pins is returned if the data direction register bits are set to 0.

If the data direction register bits are set to 1, the contents of the I/O register is returned. This is independent of any other configuration (**Figure 4-1**).

4.1.2 Input register

This is a read-only register and always returns the value of the pin (**Figure 4-1**).

4.1.3 Data direction register

This register defines whether the pin is used as an input or an output.

If a peripheral module controls the pin the contents of the data direction register is ignored (**Figure 4-1**).

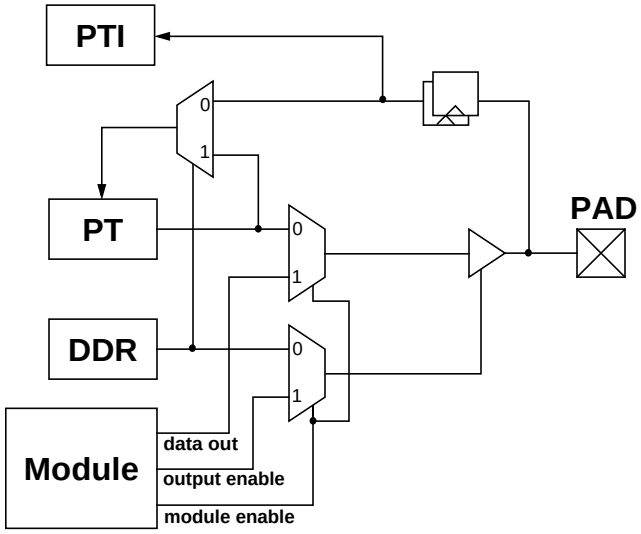


Figure 4-1 Illustration of I/O pin functionality

4.1.4 Reduced drive register

If the port is used as an output the register allows the configuration of the drive strength.

4.1.5 Pull device enable register

This register turns on a pull-up or pull-down device.

It becomes only active if the pin is used as an input or as a wired-or output.

4.1.6 Polarity select register

This register selects either a pull-up or pull-down device if enabled.

It becomes only active if the pin is used as an input. A pull-up device can be activated if the pin is used as a wired-or output.

4.2 Port T

This port is associated with the ECT module.

Port T pins PT[7:0] can be used for either general-purpose I/O, or with the channels of the Enhanced Capture Timer.

During reset, port T pins are configured as high-impedance inputs.

4.3 Port S

This port is associated with SCI0, SCI1 and SPI0.

Port S pins PS[7:0] can be used either for general-purpose I/O, or with the SCI and SPI subsystems.

During reset, port S pins are configured as inputs with pull-up.

The SPI0 pins can be re-routed. Refer to **Figure 3-21**.

4.4 Port M

This port is associated with the BDLC, CAN4-0 and SPI0.

Port M pins PM[7:0] can be used for either general purpose I/O, or with the CAN, J1850 and SPI subsystems.

During reset, port M pins are configured as high-impedance inputs.

The CAN0, CAN4 and SPI0 pins can be re-routed. Refer to **Figure 3-21**.

4.4.1 Module Routing Register

This register allows to re-route the CAN0, CAN4, SPI0, SPI1, and SPI2 pins to predefined pins.

NOTE: *The purpose of the Module Routing Register is to provide maximum flexibility for future derivatives of the MC9S12DP256 with a lower number of MSCAN and SPI modules.*

Table 4-1 Implemented modules on derivatives

Number of modules	MSCAN modules				SPI modules		
	CAN0	CAN1	CAN2	CAN4	SPI0	SPI1	SPI2
4	X	X	X	X	-	-	-
3	X	X	-	X	X	X	X
2	X	-	-	X	X	X	-
1	X	-	-	-	X	-	-

4.5 Port P

This port is associated with the PWM, SPI1 and SPI2.

Port P pins PP[7:0] can be used for either general purpose I/O, or with the PWM and SPI subsystems.

The pins are shared between the PWM channels and the SPI1 and SPI2 modules. If the PWM is enabled the pins become PWM output channels with the exception of pin 7 which can be PWM input or output. If

SPI1 or SPI2 are enabled and PWM is disabled, the respective pin configuration is determined by several status bits in the SPI modules.

During reset, port P pins are configured as high-impedance inputs.

The SPI1 and SPI2 pins can be re-routed. Refer to **Figure 3-21**.

Port P offers 8 I/O pins with edge triggered interrupt capability in wired-or fashion. The interrupt enable as well as the sensitivity to rising or falling edges can be individually configured on per pin basis. All 8 bits/pins share the same interrupt vector. Interrupts can be used with the pins configured as inputs or outputs.

An interrupt is generated when a bit in the port interrupt flag register and its corresponding port interrupt enable bit are both set. This external interrupt feature is capable to wake up the CPU when it is in STOP or WAIT mode.

A digital filter on each pin prevents pulses (**Figure 4-3**) shorter than a specified time from generating an interrupt. The minimum time varies over process conditions, temperature and voltage (**Figure 4-2** and **Table 4-2**).

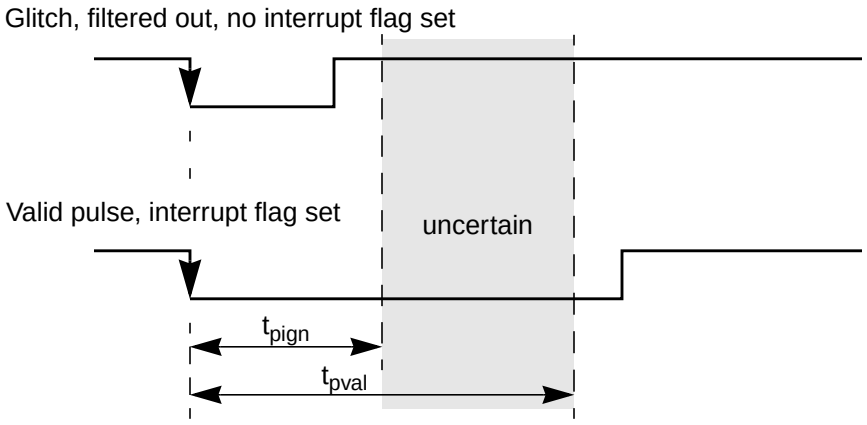


Figure 4-2 Interrupt Glitch Filter on Port P, H and J (PPS=0)

Table 4-2 Pulse Detection Criteria

Pulse	Mode		
	$\overline{STOP}$		STOP[1]
		Unit	
Ignored	$t_{pulse} \leq 3$	bus clocks	$t_{pulse} \leq t_{pign}$
Uncertain	$3 < t_{pulse} < 4$	bus clocks	$t_{pign} < t_{pulse} < t_{pval}$
Valid	$t_{pulse} \geq 4$	bus clocks	$t_{pulse} \geq t_{pval}$

NOTES:
1. These values include the spread of the oscillator frequency over temperature, voltage and process.

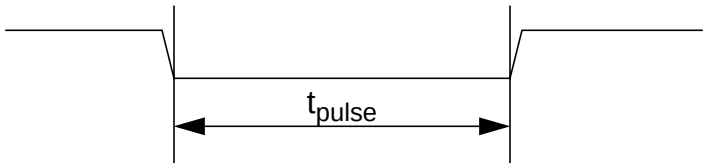


Figure 4-3 Pulse Illustration

A valid edge on an input is detected if 4 consecutive samples of a passive level are followed by 4 consecutive samples of an active level directly or indirectly.

The filters are continuously clocked by the bus clock in RUN and WAIT mode. In STOP mode the clock is generated by a single RC oscillator in the Port Integration Module. To maximize current saving the RC oscillator runs only if the following condition is true on any pin:

Sample count <= 4 and port interrupt enabled (PIE=1) and port interrupt flag not set (PIF=0).

4.6 Port H

This port is associated with the SPI1 and SPI2.

Port H pins PH[7:0] can be used for either general purpose I/O, or with the SPI subsystems.

During reset, port H pins are configured as high-impedance inputs.

Port H pins can be used with the routed SPI1 and SPI2 modules. Refer to **Figure 3-21**.

Port H offers 8 I/O ports with the same interrupt features as port P.

4.7 Port J

This port is associated with the CAN4, CAN0 and the IIC.

Port J pins PJ[7:6] and PJ[1:0] can be used for either general purpose I/O, or with the CAN and IIC subsystems.

During reset, port J pins are configured as inputs with pull-up.

If IIC takes precedence the pins become IIC open-drain output pins.

The CAN4 pins can be re-routed. Refer to **Figure 3-21**.

Port J pins can be used with the routed CAN0 modules. Refer to **Figure 3-21**.

Port J offers 4 I/O ports with the same interrupt features as port P.

4.8 Port A, B, E, K, and BKGD pin

All port and pin logic is located in the core module. *Refer to MEBI in HCS12 Core User Guide for details.*

4.9 External Pin Descriptions

All ports start up as general purpose inputs on reset.

4.10 Low Power Options

4.10.1 Run Mode

No low power options exist for this module in run mode.

4.10.2 Wait Mode

No low power options exist for this module in wait mode.

4.10.3 Stop Mode

All clocks are stopped. There are asynchronous paths to generate interrupts from STOP on port P, H and J.

Section 5 Initialization/Application Information

TBD

Index

–I–

Block Guide End Sheet

FINAL PAGE OF
54
PAGES

ECT_16B8C
Block User Guide
V01.06

Original Release Date: 2-Sep-1999
Revised: Jul 05, 2004

Motorola Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
0.1	2-Sep-99	2-Sep-99		Original draft. Distributed only within Motorola QS9000 Verified.
0.2	24-Sep-99	24-Sep-99		• Changed the specs as per MSRS format. • Modified ECT16b8c Block diagram. • Modified IP Bus signal names and their description. • Modified ECT output signal names. • Deleted bits 3-0 of TSCR1 register in Register Map(Sheet 1 of 2). • Modified register addresses in the description of TSFRZ,WAIT,NORMAL mode(Modes of Operation). • In Figure 1-6 changed text font to Halvetica. • Renamed TMSK1 and TMSK2 register as TIE and TSCR2 also renamed TSCR as TSCR1. • Modified TFLG2 bit setting sentence. • Added explanation about the abbreviation(M clock,PACLK) used. • Removed duplication of lines at the end of register description of PACN3/PACN2. • Corrected the reset value of MCCNT from $FF to $FFFF in the description of register MCCTL. • Corrected Table format for delay counter select and Modulus counter Prescalar Select. • Corrected all the cross-references used in section 3 of the document. • Deleted and added some module specific signals. • Changed all interrupts from active LOW to active HIGH. • Added description about successful output comapre and forced output compare taking place simultaneously and their effect on flag. • Added abbreviation section. • In Fig 1-3 changed host data bus to IPbus.
0.3	25-Oct-99	25-Oct-99		• Changed block name in Section 2. • Removed signals ipp_ect_ic_ibe and ipp_ect_ic_offval from port list.
0.4	11-Nov-99	11-Nov-99		• Removed signal ipb_read & ipb_write from portlist. ipb_rwb added

Version Number	Revision Date	Effective Date	Author	Description of Changes
0.5	1-Dec-99	1-Dec-99		• Incorporated feedback received from Joachim Kruecken on 30-Nov-99
01.00	10-May-01	11-May-01		• formal changes for SRS compliance (cover, master pages, paragraph formats, register templates) • updated cross-references • marked TIMTST register description with ’non_cust’ tag
01.01	19-July-01			• Document names have been added • Names and Variable definitions have been hidden
01.02	10-Jan-02	10-Jan-02		• Note added in Section 3.3.18 for PACN1/PACN0
01.03	18-Jul-02	18-Jul-02		• Changed the register bit information of MCCTL on bits • ICLAT and FLMC
01.04	11-Nov-02	11-Nov-02		• Eliminated all references of "n" to "x" .e.g. IOSn to IOSx
01.05	12-Mar-04	12-Mar-04		• Description of OC7M and TCTL1 registers modified, note on OC added in section 4.
01.06	05-Jul-04	05-Jul-04		• Included description about delay counter in section 4.2.1.3 • Modified the description about TCNT in section 3. 3.5

Table of Contents

Section 4 Functional Description

Section 5 Reset

Section 6 Interrupts

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

The HCS12 Enhanced Capture Timer module has the features of the HCS12 Standard Timer module enhanced by additional features in order to enlarge the field of applications, in particular for automotive ABS applications.

This design specification describes the standard timer as well as the additional features.

The basic timer consists of a 16-bit, software-programmable counter driven by a prescaler. This timer can be used for many purposes, including input waveform measurements while simultaneously generating an output waveform. Pulse widths can vary from microseconds to many seconds.

A full access for the counter registers or the input capture/output compare registers should take place in one clock cycle. Accessing high byte and low byte separately for all of these registers may not yield the same result as accessing them in one word.

1.2 Features

- 16-Bit Buffer Register for four Input Capture (IC) channels.
- Four 8-Bit Pulse Accumulators with 8-bit buffer registers associated with the four buffered IC channels. Configurable also as two 16-Bit Pulse Accumulators.
- 16-Bit Modulus Down-Counter with 4-bit Prescaler.
- Four user selectable Delay Counters for input noise immunity increase.

1.3 Modes of Operation

STOP: Timer and modulus counter are off since clocks are stopped.

FREEZE: Timer and modulus counter keep on running, unless TSFRZ in TSCR($06) is set to one.

WAIT: Counters keep on running, unless TSWAI in TSCR ($06) is set to one.

NORMAL: Timer and modulus counter keep on running, unless TEN in TSCR($06) respectively MCEN in MCCTL ($26) are cleared.

1.4 Block Diagram

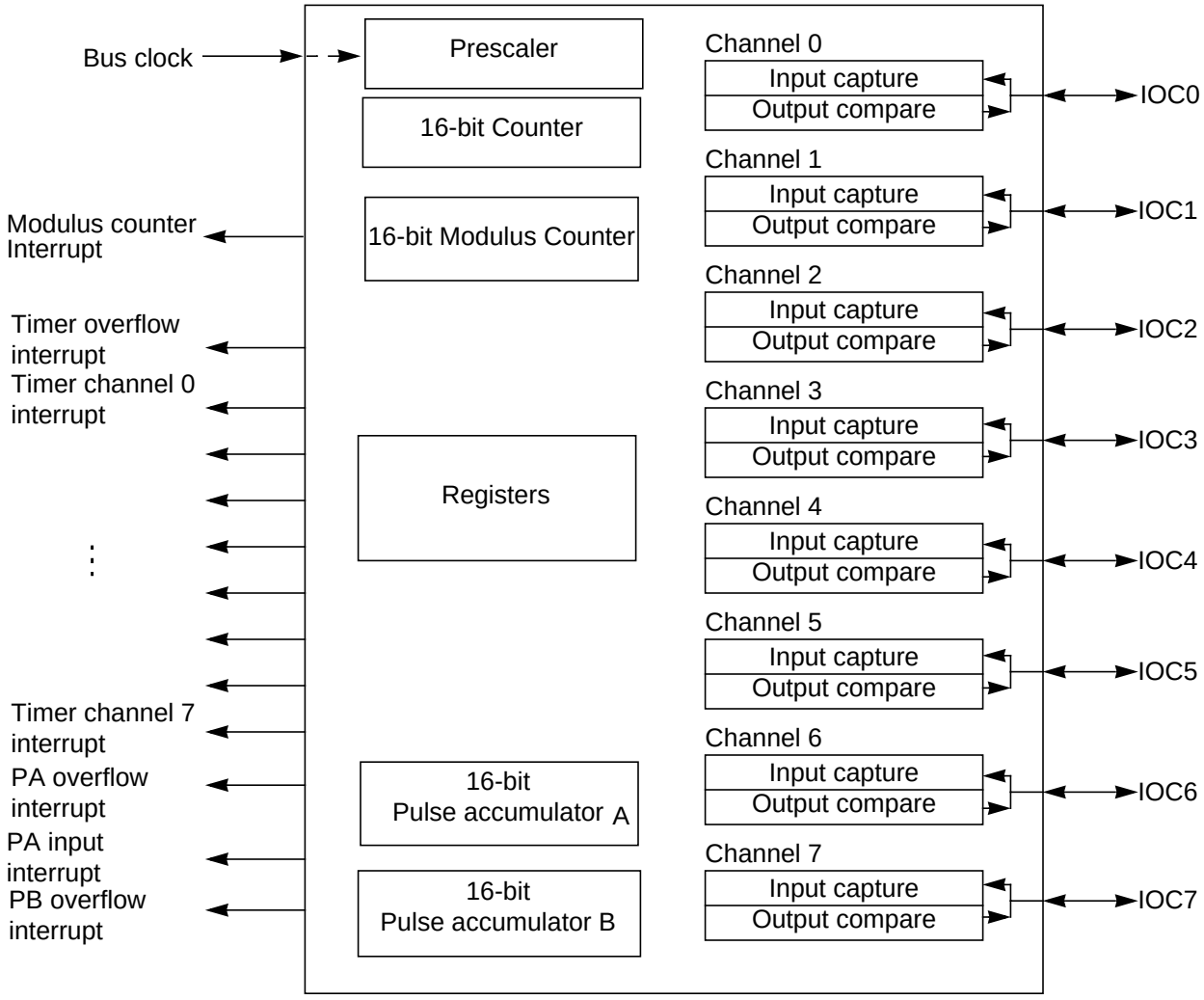


Figure 1-1 Timer Block Diagram

Section 2 Signal Description

2.1 Overview

The ECT_16B8C module has a total 8 external pins.

2.2 Detailed Signal Descriptions

2.2.1 IOC7 - Input capture and Output compare channel 7

This pin serves as input capture or output compare for channel 7.

2.2.2 IOC6 - Input capture and Output compare channel 6

This pin serves as input capture or output compare for channel 6.

2.2.3 IOC5 - Input capture and Output compare channel 5

This pin serves as input capture or output compare for channel 7.

2.2.4 IOC4 - Input capture and Output compare channel 4

This pin serves as input capture or output compare for channel 4.

2.2.5 IOC3 - Input capture and Output compare channel 3

This pin serves as input capture or output compare for channel 3.

2.2.6 IOC2 - Input capture and Output compare channel 2

This pin serves as input capture or output compare for channel 2.

2.2.7 IOC1 - Input capture and Output compare channel 1

This pin serves as input capture or output compare for channel 1.

2.2.8 IOC0 - Input capture and Output compare channel 0

This pin serves as input capture or output compare for channel 0.

NOTE: *For the description of interrupts see Section 6 Interrupts.*

Section 3 Memory Map and Registers

3.1 Overview

This section provides a detailed description of all memory and registers.

3.2 Module Memory Map

The memory map for the ECT module is given below in **Table 3-1**. The Address listed for each register is the address offset. The total address for each register is the sum of the base address for the ECT module and the address offset for each register.

Table 3-1 Module Memory Map

Offset	Use	Access
$_00	Timer Input Capture/Output Compare Select (TIOS)	Read/Write
$_01	Timer Compare Force Register (CFORC)	Read/Write[1]
$_02	Output Compare 7 Mask Register (OC7M)	Read/Write
$_03	Output Compare 7 Data Register (OC7D)	Read/Write
$_04	Timer Count Register High (TCNT)	Read/Write[2]
$_05	Timer Count Register Low (TCNT)	Read/Write[2]
$_06	Timer System Control Register1 (TSCR1)	Read/Write
$_07	Timer Toggle Overflow Register (TTOV)	Read/Write
$_08	Timer Control Register1 (TCTL1)	Read/Write
$_09	Timer Control Register2 (TCTL2)	Read/Write
$_0A	Timer Control Register3 (TCTL3)	Read/Write
$_0B	Timer Control Register4 (TCTL4)	Read/Write
$_0C	Timer Interrupt Enable Register (TIE)	Read/Write
$_0D	Timer System Control Register2 (TSCR2)	Read/Write
$_0E	Main Timer Interrupt Flag1 (TFLG1)	Read/Write
$_0F	Main Timer Interrupt Flag2 (TFLG2)	Read/Write
$_10	Timer Input Capture/Output Compare Register0 High (TC0)	Read/Write[3]
$_11	Timer Input Capture/Output Compare Register0 Low (TC0)	Read/Write[3]
$_12	Timer Input Capture/Output Compare Register1 High (TC1)	Read/Write[3]
$_13	Timer Input Capture/Output Compare Register1 Low (TC1)	Read/Write[3]
$_14	Timer Input Capture/Output Compare Register2 High (TC2)	Read/Write[3]
$_15	Timer Input Capture/Output Compare Register2 Low (TC2)	Read/Write[3]
$_16	Timer Input Capture/Output Compare Register3 High (TC3)	Read/Write[3]

Table 3-1 Module Memory Map

$_17	Timer Input Capture/Output Compare Register3 Low (TC3)	Read/Write[3]
$_18	Timer Input Capture/Output Compare Register4 High (TC4)	Read/Write[3]
$_19	Timer Input Capture/Output Compare Register4 Low (TC4)	Read/Write[3]
$_1A	Timer Input Capture/Output Compare Register5 High (TC5)	Read/Write[3]
$_1B	Timer Input Capture/Output Compare Register5 Low (TC5)	Read/Write[3]
$_1C	Timer Input Capture/Output Compare Register6 High (TC6)	Read/Write[3]
$_1D	Timer Input Capture/Output Compare Register6 Low (TC6)	Read/Write[3]
$_1E	Timer Input Capture/Output Compare Register7 High (TC7)	Read/Write[3]
$_1F	Timer Input Capture/Output Compare Register7 Low (TC7)	Read/Write[3]
$_20	16-Bit Pulse Accumulator A Control Register (PACTL)	Read/Write
$_21	Pulse Accumulator A Flag Register (PAFLG)	Read/Write
$_22	Pulse Accumulator Count Register3 (PACN3)	Read/Write
$_23	Pulse Accumulator Count Register2 (PACN2)	Read/Write
$_24	Pulse Accumulator Count Register1 (PACN1)	Read/Write
$_25	Pulse Accumulator Count Register0 (PACN0)	Read/Write
$_26	16-Bit Modulus Down Counter Register (MCCTL)	Read/Write
$_27	16-Bit Modulus Down Counter Flag Register (MCFLG)	Read/Write
$_28	Input Control Pulse Accumulator Register (ICPAR)	Read/Write
$_29	Delay Counter Control Register (DLYCT)	Read/Write
$_2A	Input Control Overwrite Register (ICOVW)	Read/Write
$_2B	Input Control System Control Register (ICSYS)	Read/Write[4]
$_2C	Reserved	--
$_2D	Timer Test Register (TIMTST)	Read/Write[2]
$_2E	Reserved	--
$_2F	Reserved	--
$_30	16-Bit Pulse Accumulator B Control Register (PBCTL)	Read/Write
$_31	16-Bit Pulse Accumulator B Flag Register (PBFLG)	Read/Write
$_32	8-Bit Pulse Accumulator Holding Register3 (PA3H)	Read/Write[5]
$_33	8-Bit Pulse Accumulator Holding Register2 (PA2H)	Read/Write[5]
$_34	8-Bit Pulse Accumulator Holding Register1 (PA1H)	Read/Write[5]
$_35	8-Bit Pulse Accumulator Holding Register0 (PA0H)	Read/Write[5]
$_36	Modulus Down-Counter Count Register High (MCCNT)	Read/Write

Table 3-1 Module Memory Map

$_37	Modulus Down-Counter Count Register Low (MCCNT)	Read/Write
$_38	Timer Input Capture Holding Register0 High (TC0H)	Read/Write[5]
$_39	Timer Input Capture Holding Register0 Low (TC0H)	Read/Write[5]
$_3A	Timer Input Capture Holding Register1 High(TC1H)	Read/Write[5]
$_3B	Timer Input Capture Holding Register1 Low (TC1H)	Read/Write[5]
$_3C	Timer Input Capture Holding Register2 High (TC2H)	Read/Write[5]
$_3D	Timer Input Capture Holding Register2 Low (TC2H)	Read/Write[5]
$_3E	Timer Input Capture Holding Register3 High (TC3H)	Read/Write[5]
$_3F	Timer Input Capture Holding Register3 Low (TC3H)	Read/Write[5]

1. Always read $00.
2. Only writable in special modes (test_mode = 1).
3. Write to these registers have no meaning or effect during input capture.
4. May be written once (test_mode = 0) but writes are always permitted when test_mode = 0
5. Write has no effect.

3.3 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

3.3.1 TIOS — Timer Input Capture/Output Compare Select Register

Register offset: $_00

	BIT7	6	5	4	3	2	1	BIT0
R W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Figure 3-1 Timer Input Capture/Output Compare Register (TIOS)

Read or write anytime.

IOS[7:0] — Input Capture or Output Compare Channel Configuration

0 = The corresponding channel acts as an input capture

1 = The corresponding channel acts as an output compare.

3.3.2 CFORC — Timer Compare Force Register

Register offset: $_01

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	0	0
W	FOC7	FOC6	FOC5	FOC4	FOC3	FOC2	FOC1	FOC0
RESET:	0	0	0	0	0	0	0	0

Figure 3-2 Timer Compare Force Register (CFORC)

Read anytime but will always return $00 (1 state is transient). Write anytime.

FOC[7:0] — Force Output Compare Action for Channel 7-0

A write to this register with the corresponding data bit(s) set causes the action which is programmed for output compare “n” to occur immediately. The action taken is the same as if a successful comparison had just taken place with the TCn register except the interrupt flag does not get set.

NOTE: *A successful channel 7 output compare overrides any channel 6:0 compares.If forced output compare on any channel occurs at the same time as the successful output compare then forced output compare action will take precedence and interrupt flag won’t get set.*

3.3.3 OC7M — Output Compare 7 Mask Register

Register offset: $_02

	BIT7	6	5	4	3	2	1	BIT0
R W	OC7M7	OC7M6	OC7M5	OC7M4	OC7M3	OC7M2	OC7M1	OC7M0
RESET:	0	0	0	0	0	0	0	0

Figure 3-3 Output Compare 7 Mask Register (OC7M)

Read or write anytime.

Setting the OC7Mn (n ranges from 0 to 6) bit of OC7M register configures the corresponding port to be an output port when the IOS7 bit and the corresponding IOSn (n ranges from 0 to 6) bit of TIOS register are set to be an output compare. Refer to the note on Section **4.2.4** for more insight.

NOTE: *A successful channel 7 output compare overrides any channel 6:0 compares. For each OC7M bit that is set, the output compare action reflects the corresponding OC7D bit.*

3.3.4 OC7D — Output Compare 7 Data Register

Register offset: $_03

	BIT7	6	5	4	3	2	1	BIT0
R W	OC7D7	OC7D6	OC7D5	OC7D4	OC7D3	OC7D2	OC7D1	OC7D0
RESET:	0	0	0	0	0	0	0	0

Figure 3-4 Output Compare 7 Data Register (OC7D)

Read or write anytime.

A channel 7 output compare can cause bits in the output compare 7 data register to transfer to the timer port data register depending on the output compare 7 mask register.

3.3.5 TCNT — Timer Count Register

Register offset: $_04-$_05

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R W	tcnt 15	tcnt 14	tcnt 13	tcnt 12	tcnt 11	tcnt 10	tcnt 9	tcnt 8	tcnt 7	tcnt 6	tcnt 5	tcnt 4	tcnt 3	tcnt 2	tcnt 1	tcnt 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-5 Timer Count Register (TCNT)

The 16-bit main timer is an up counter.

A full access for the counter register should take place in one clock cycle. A separate read (any mode)/ write (test mode) for high byte and low byte will give a different result than accessing them as a word.

Read anytime.

Write has no meaning or effect in the normal mode; only writable in special modes (test_mode = 1).

The period of the first count after a write to the TCNT registers may be a different size because the write is not synchronized with the prescaler clock.

3.3.6 TSCR1 — Timer System Control Register 1

Register offset: $_06

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 Timer System Control Register 1 (TSCR1)

Read or write anytime.

TEN — Timer Enable

0 = Disables the main timer, including the counter. Can be used for reducing power consumption.

1 = Allows the timer to function normally.

If for any reason the timer is not active, there is no ÷64 clock for the pulse accumulator since the ÷64 is generated by the timer prescaler.

TSWAI — Timer Module Stops While in Wait

0 = Allows the timer module to continue running during wait.

1 = Disables the timer module when the MCU is in the wait mode. Timer interrupts cannot be used to get the MCU out of wait.

TSWAI also affects pulse accumulators and modulus down counters.

TSFRZ — Timer and Modulus Counter Stop While in Freeze Mode

0 = Allows the timer and modulus counter to continue running while in freeze mode.

1 = Disables the timer and modulus counter whenever the MCU is in freeze mode. This is useful for emulation.

TSFRZ does not stop the pulse accumulator.

TFFCA — Timer Fast Flag Clear All

0 = Allows the timer flag clearing to function normally.

1 = For TFLG1($0E), a read from an input capture or a write to the output compare channel ($10–$1F) causes the corresponding channel flag, CnF, to be cleared. For TFLG2 ($0F), any access to the TCNT register ($04, $05) clears the TOF flag. Any access to the PACN3 and PACN2 registers ($22, $23) clears the PAOVF and PAIF flags in the PAFLG register ($21). Any access to the PACN1 and PACN0 registers ($24, $25) clears the PBOVF flag in the PBFLG register ($31). This has the advantage of eliminating software overhead in a separate clear sequence. Extra care is required to avoid accidental flag clearing due to unintended accesses.

3.3.7 TTOV — Timer Toggle On Overflow Register 1

Register offset: $_07

	BIT7	6	5	4	3	2	1	BIT0
R W	TOV7	TOV6	TOV5	TOV4	TOV3	TOV2	TOV1	TOV0
RESET:	0	0	0	0	0	0	0	0

Figure 3-7 Timer Toggle On Overflow Register 1 (TTOV)

Read or write anytime.

TOVn — Toggle On Overflow Bits

TOVn toggles output compare pin on overflow. This feature only takes effect when in output compare mode. When set, it takes precedence over forced output compare but not channel 7 override events.

0 = Toggle output compare pin on overflow feature disabled
1 = Toggle output compare pin on overflow feature enabled

3.3.8 TCTL1/TCTL2 — Timer Control Register 1/Timer Control Register 2

Register offset:$_08

	BIT7	6	5	4	3	2	1	BIT0
R W	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
RESET	0	0	0	0	0	0	0	0

Register offset:$_09

	BIT7	6	5	4	3	2	1	BIT0
R W	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
RESET	0	0	0	0	0	0	0	0

Figure 3-8 Timer Control Register 1/Timer Control Register 2 (TCTL1/TCTL2)

Read or write anytime.

OMn — Output Mode

OLn — Output Level

These eight pairs of control bits are encoded to specify the output action to be taken as a result of a successful OCn(n varies from 0 to 7) compare. When either OMn or OLn is one, the port associated with OCn becomes an output tied to OCn when the corresponding IOSn bit of TIOS register is set and TEN bit of TSCR1 register is set. Refer to the note on Section **4.2.4** for more insight.

NOTE: *To enable output action by OMn and OLn bits on timer port, the corresponding bit in OC7M should be cleared.*

Table 3-2 Compare Result Output Action

OMn	OLn	Action
0	0	Timer disconnected from output pin logic
0	1	Toggle OCn output line
1	0	Clear OCn output line to zero
1	1	Set OCn output line to one

To operate the 16-bit pulse accumulators A and B (PACA and PACB) independently of input capture or output compare 7 and 0 respectively the user must set the corresponding bits IOSn = 1, OMn = 0 and OLn = 0. OC7M7 or OC7M0 in the OC7M register must also be cleared.

3.3.9 TCTL3/TCTL4 — Timer Control Register 3/Timer Control Register 4

Register offset: $_0A

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
RESET:	0	0	0	0	0	0	0	0

Register offset: $_0B

	BIT7	6	5	4	3	2	1	BIT0
R W	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
RESET:	0	0	0	0	0	0	0	0

Figure 3-9 Timer Control Register 3/Timer Control Register 4 (TCTL3/TCTL4)

Read or write anytime.

EDGnB, EDGnA — Input Capture Edge Control

These eight pairs of control bits configure the input capture edge detector circuits.

The four pairs of control bits of TCTL4 also configure the 8 bit pulse accumulators PAC0 - 3.

For 16 - bit pulse accumulator PACB, EDGE0B & EDGE0A, control bits of TCTL4 will decide the active edge.

Table 3-3 Edge Detector Circuit Configuration

EDGnB	EDGnA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

3.3.10 TIE — Timer Interrupt Enable Register

Register offset: $_0C

	BIT7	6	5	4	3	2	1	BIT0
R W	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
RESET:	0	0	0	0	0	0	0	0

Figure 3-10 Timer Interrupt Enable Register (TIE)

Read or write anytime.

The bits in TIE correspond bit-for-bit with the bits in the TFLG1 status register. If cleared, the corresponding flag is disabled from causing a hardware interrupt. If set, the corresponding flag is enabled to cause a interrupt.

C7I–C0I — Input Capture/Output Compare “n” Interrupt Enable

3.3.11 TSCR2 — Timer System Control Register 2

Register offset: $_0D

	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-11 Timer System Control Register 2 (TSCR2)

Read or write anytime.

TOI — Timer Overflow Interrupt Enable
- 0 = Interrupt inhibited
- 1 = Hardware interrupt requested when TOF flag set

TCRE — Timer Counter Reset Enable

This bit allows the timer counter to be reset by a successful output compare 7 event. This mode of operation is similar to an up-counting modulus counter.
- 0 = Counter reset inhibited and counter free runs
- 1 = Counter reset by a successful output compare 7

If TC7 = $0000 and TCRE = 1, TCNT will stay at $0000 continuously. If TC7 = $FFFF and TCRE = 1, TOF will never be set when TCNT is reset from $FFFF to $0000.

PR2, PR1, PR0 — Timer Prescaler Select

These three bits specify the number of ÷2 stages that are to be inserted between the bus clock and the main timer counter.

Table 3-4 Prescaler Selection

PR2	PR1	PR0	Prescale Factor
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

The newly selected prescale factor will not take effect until the next synchronized edge where all prescale counter stages equal zero.

3.3.12 TFLG1 — Main Timer Interrupt Flag 1

Register offset: $_0E

	BIT7	6	5	4	3	2	1	BIT0
R W	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
RESET:	0	0	0	0	0	0	0	0

Figure 3-12 Main Timer Interrupt Flag 1 (TFLG1)

TFLG1 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write a one to the bit.

Use of the TFMOD bit in the ICSYS register ($2B) in conjunction with the use of the ICOVW register ($2A) allows a timer interrupt to be generated after capturing two values in the capture and holding registers instead of generating an interrupt for every capture.

Read anytime. Write used in the clearing mechanism (set bits cause corresponding bits to be cleared). Writing a zero will not affect current status of the bit.

When TFFCA bit in TSCR register is set, a read from an input capture or a write into an output compare channel ($10–$1F) will cause the corresponding channel flag CnF to be cleared.

C7F–C0F — Input Capture/Output Compare Channel “n” Flag.

C0F can also be set by 16 - bit Pulse Accumulator B (PACB). C3F - C0F can also be set by 8 - bit pulse accumulators PAC3 - PAC0.

3.3.13 TFLG2 — Main Timer Interrupt Flag 2

Register offset: $_0F

	BIT7	6	5	4	3	2	1	BIT0
R	TOF	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-13 Main Timer Interrupt Flag 2 (TFLG2)

TFLG2 indicates when interrupt conditions have occurred. To clear a bit in the flag register, write the bit to one.

Read anytime. Write used in clearing mechanism (set bits cause corresponding bits to be cleared).

Any access to TCNT will clear TFLG2 register if the TFFCA bit in TSCR register is set.

TOF — Timer Overflow Flag

Set when 16-bit free-running timer overflows from $FFFF to $0000. This bit is cleared automatically by a write to the TFLG2 register with bit 7 set. (See also TCRE control bit explanation.)

3.3.14 Timer Input Capture/Output Compare Registers 0-7

TC0 — Timer Input Capture/Output Compare Register 0 **Register offset: $_10–$_11**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0	tc0
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC1 — Timer Input Capture/Output Compare Register 1 **Register offset: $_12–$_13**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1	tc1
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC2 — Timer Input Capture/Output Compare Register 2 **Register offset: $_14–$_15**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2	tc2
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC3 — Timer Input Capture/Output Compare Register 3 **Register offset: $_16–$_17**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3	tc3
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC4 — Timer Input Capture/Output Compare Register 4 **Register offset: $_18–$_19**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4	tc4
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC5 — Timer Input Capture/Output Compare Register 5 **Register offset: $_1A–$_1B**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5	tc5
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC6 — Timer Input Capture/Output Compare Register 6 **Register offset: $_1C–$_1D**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6	tc6
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TC7 — Timer Input Capture/Output Compare Register 7 **Register offset: $_1E–$_1F**

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7	tc7
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-14 Timer Input Capture/Output Compare Registers 0-7

Depending on the TIOS bit for the corresponding channel, these registers are used to latch the value of the free-running counter when a defined transition is sensed by the corresponding input capture edge detector or to trigger an output action for output compare.

Read anytime. Write anytime for output compare function. Writes to these registers have no meaning or effect during input capture. All timer input capture/output compare registers are reset to $0000.

3.3.15 PACTL — 16-Bit Pulse Accumulator A Control Register

Register offset: $_20

	BIT7	6	5	4	3	2	1	BIT0
R	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-15 16-Bit Pulse Accumulator Control Register (PACTL)

16-Bit Pulse Accumulator A (PACA) is formed by cascading the 8-bit pulse accumulators PAC3 and PAC2.

When PAEN is set, the PACA is enabled. The PACA shares the input pin with IC7.

Read: any time
Write: any time

PAEN — Pulse Accumulator A System Enable

0 = 16-Bit Pulse Accumulator A system disabled. 8-bit PAC3 and PAC2 can be enabled when their related enable bits in ICPAR ($28) are set.
Pulse Accumulator Input Edge Flag (PAIF) function is disabled.

1 = 16-Bit Pulse Accumulator A system enabled. The two 8-bit pulse accumulators PAC3 and PAC2 are cascaded to form the PACA 16-bit pulse accumulator. When PACA in enabled, the PACN3 and PACN2 registers contents are respectively the high and low byte of the PACA. PA3EN and PA2EN control bits in ICPAR ($28) have no effect.
Pulse Accumulator Input Edge Flag (PAIF) function is enabled.

PAEN is independent from TEN. With timer disabled, the pulse accumulator can still function unless pulse accumulator is disabled.

PAMOD — Pulse Accumulator Mode

This bit is active only when the Pulse Accumulator A is enabled (PAEN = 1).

0 = event counter mode

1 = gated time accumulation mode

PEDGE — Pulse Accumulator Edge Control

This bit is active only when the Pulse Accumulator A is enabled (PAEN = 1).

For PAMOD bit = 0 (event counter mode).

0 = falling edges on PT7 pin cause the count to be incremented

1 = rising edges on PT7 pin cause the count to be incremented

For PAMOD bit = 1 (gated time accumulation mode).

0 = PT7 input pin high enables bus clock divided by 64 to Pulse Accumulator and the trailing falling edge on PT7 sets the PAIF flag.

1 = PT7 input pin low enables bus clock divided by 64 to Pulse Accumulator and the trailing rising edge on PT7 sets the PAIF flag

Table 3-5 Pin Action

PAMOD	PEDGE	Pin Action
0	0	Falling edge
0	1	Rising edge
1	0	Div. by 64 clock enabled with pin high level
1	1	Div. by 64 clock enabled with pin low level

If the timer is not active (TEN = 0 in TSCR), there is no divide-by-64 since the ÷64 clock is generated by the timer prescaler.

CLK1, CLK0 — Clock Select Bits

Table 3-6 Clock Selection

CLK1	CLK0	Clock Source
0	0	Use timer prescaler clock as timer counter clock
0	1	Use PACLK as input to timer counter clock
1	0	Use PACLK/256 as timer counter clock frequency
1	1	Use PACLK/65536 as timer counter clock frequency

For the description of PACLK please refer **Figure 4-4**.

If the pulse accumulator is disabled (PAEN = 0), the prescaler clock from the timer is always used as an input clock to the timer counter. The change from one selected clock to the other happens immediately after these bits are written.

PAOVI — Pulse Accumulator A Overflow Interrupt enable

0 = interrupt inhibited

1 = interrupt requested if PAOVF is set

PAI — Pulse Accumulator Input Interrupt enable

0 = interrupt inhibited

1 = interrupt requested if PAIF is set

3.3.16 PAFLG — Pulse Accumulator A Flag Register

Register offset: $_21

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	PAOVF	PAIF
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-16 Pulse Accumulator A Flag Register (PAFLG)

Read or write anytime. When the TFFCA bit in the TSCR register is set, any access to the PACNT register will clear all the flags in the PAFLG register.

PAOVF — Pulse Accumulator A Overflow Flag

Set when the 16-bit pulse accumulator A overflows from $FFFF to $0000,or when 8-bit pulse accumulator 3 (PAC3) overflows from $FF to $00.

When PACMX = 1, PAOVF bit can also be set if 8 - bit pulse accumulator 3 (PAC3) reaches $FF followed by an active edge on PT3.

This bit is cleared automatically by a write to the PAFLG register with bit 1 set.

PAIF — Pulse Accumulator Input edge Flag

Set when the selected edge is detected at the PT7 input pin. In event mode the event edge triggers PAIF and in gated time accumulation mode the trailing edge of the gate signal at the PT7 input pin triggers PAIF.

This bit is cleared by a write to the PAFLG register with bit 0 set.
Any access to the PACN3, PACN2 registers will clear all the flags in this register when TFFCA bit in register TSCR($06) is set.

3.3.17 PACN3, PACN2 — Pulse Accumulators Count Registers

Register offset: $_22

	BIT7	6	5	4	3	2	1	BIT0
R W	pacnt7(15)	pacnt6(14)	pacnt5(13)	pacnt4(12)	pacnt3(11)	pacnt2(10)	pacnt1(9)	pacnt0(8)
RESET:	0	0	0	0	0	0	0	0

Figure 3-17 Pulse Accumulators Count Register 3 (PACN3)

Register offset: $_23

	BIT7	6	5	4	3	2	1	BIT0
R W	pacnt7	pacnt6	pacnt5	pacnt4	pacnt3	pacnt2	pacnt1	pacnt0
RESET:	0	0	0	0	0	0	0	0

Figure 3-18 Pulse Accumulators Count Register 2 (PACN2)

Read or write any time.

The two 8-bit pulse accumulators PAC3 and PAC2 are cascaded to form the PACA 16-bit pulse accumulator. When PACA in enabled (PAEN=1 in PACTL, $20) the PACN3 and PACN2 registers contents are respectively the high and low byte of the PACA.

When PACN3 overflows from $FF to $00, the Interrupt flag PAOVF in PAFLG ($21) is set.

Full count register access should take place in one clock cycle. A separate read/write for high byte and low byte will give a different result than accessing them as a word.

NOTE : *When clocking pulse and write to the registers occurs simultaneously , write takes priority and the register is not incremented.*

3.3.18 PACN1, PACN0 — Pulse Accumulators Count Registers

Register offset: $_24

	BIT7	6	5	4	3	2	1	BIT0
R W	pacnt7(15)	pacnt6(14)	pacnt5(13)	pacnt4(12)	pacnt3(11)	pacnt2(10)	pacnt1(9)	pacnt0(8)
RESET:	0	0	0	0	0	0	0	0

Figure 3-19 Pulse Accumulators Count Register 1 (PACN1)

Register offset: $_25

	BIT7	6	5	4	3	2	1	BIT0
R W	pacnt7	pacnt6	pacnt5	pacnt4	pacnt3	pacnt2	pacnt1	pacnt0
RESET:	0	0	0	0	0	0	0	0

Figure 3-20 Pulse Accumulators Count Register 0 (PACN0)

Read or write any time.

The two 8-bit pulse accumulators PAC1 and PAC0 are cascaded to form the PACB 16-bit pulse accumulator. When PACB in enabled, (PBEN=1 in PBCTL, $30) the PACN1 and PACN0 registers contents are respectively the high and low byte of the PACB.

When PACN1 overflows from $FF to $00, the Interrupt flag PBOVF in PBFLG ($31) is set.

Full count register access should take place in one clock cycle. A separate read/write for high byte and low byte will give a different result than accessing them as a word

NOTE : *When clocking pulse and write to the registers occurs simultaneously , write takes priority and the register is not incremented.*

3.3.19 MCCTL — 16-Bit Modulus Down-Counter Control Register

Register offset: $_26

	BIT7	6	5	4	3	2	1	BIT0
R	MCZI	MODMC	RDMCL	0	0	MCEN	MCPR1	MCPR0
W				ICLAT	FLMC			
RESET:	0	0	0	0	0	0	0	0

Figure 3-21 16-Bit Modulus Down-Counter Control Register (MCCTL)

Read or write any time.

MCZI — Modulus Counter Underflow Interrupt Enable
0 = Modulus counter interrupt is disabled.

1 = Modulus counter interrupt is enabled.

MODMC — Modulus Mode Enable

0 = The counter counts once from the value written to it and will stop at $0000.

1 = Modulus mode is enabled. When the counter reaches $0000, the counter is loaded with the latest value written to the modulus count register.

NOTE: *For proper operation, the MCEN bit should be cleared before modifying the MODMC bit in order to reset the modulus counter to $FFFF.*

RDMCL — Read Modulus Down-Counter Load

0 = Reads of the modulus count register will return the present value of the count register.

1 = Reads of the modulus count register will return the contents of the load register.

ICLAT — Input Capture Force Latch Action

When input capture latch mode is enabled (LATQ and BUFEN bit in ICSYS ($2B) are set, a write one to this bit immediately forces the contents of the input capture registers TC0 to TC3 and their corresponding 8-bit pulse accumulators to be latched into the associated holding registers. The pulse accumulators will be automatically cleared when the latch action occurs.

Writing zero to this bit has no effect. Read of this bit will return always zero.

FLMC — Force Load Register into the Modulus Counter Count Register

This bit is active only when the modulus down-counter is enabled (MCEN=1).

A write one into this bit loads the load register into the modulus counter count register. This also resets the modulus counter prescaler.

Write zero to this bit has no effect.

When MODMC=0, counter starts counting and stops at $0000.

Read of this bit will return always zero.

MCEN — Modulus Down-Counter Enable

0 = Modulus counter disabled.

1 = Modulus counter is enabled.

When MCEN=0, the counter is preset to $FFFF. This will prevent an early interrupt flag when the modulus down-counter is enabled.

MCPR1, MCPR0 — Modulus Counter Prescaler select

These two bits specify the division rate of the modulus counter prescaler.

The newly selected prescaler division rate will not be effective until a load of the load register into the modulus counter count register occurs.

Table 3-7 Modulus Counter Prescaler Select

MCPR1	MCPR0	Prescaler division rate
0	0	1

Table 3-7 Modulus Counter Prescaler Select

MCPR1	MCPR0	Prescaler division rate
0	1	4
1	0	8
1	1	16

3.3.20 MCFLG — 16-Bit Modulus Down-Counter FLAG Register

Register offset: $_27

	BIT7	6	5	4	3	2	1	BIT0
R	MCZF	0	0	0	POLF3	POLF2	POLF1	POLF0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-22 16-Bit Modulus Down-Counter FLAG Register (MCFLG)

Read: any time
Write: Only for clearing bit 7

MCZF — Modulus Counter Underflow Flag

The flag is set when the modulus down-counter reaches $0000.

A write one to this bit clears the flag. Write zero has no effect.

Any access to the MCCNT register will clear the MCZF flag in this register when TFFCA bit in register TSCR($06) is set.

POLF3 – POLF0 — First Input Capture Polarity Status

This are read only bits. Write to these bits has no effect.

Each status bit gives the polarity of the first edge which has caused an input capture to occur after capture latch has been read.

Each POLFn corresponds to a timer PORTn input.

0 = The first input capture has been caused by a falling edge.
1 = The first input capture has been caused by a rising edge.

3.3.21 ICPAR — Input Control Pulse Accumulators Register

Register offset: $_28

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	PA3EN	PA2EN	PA1EN	PA0EN
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-23 Input Control Pulse Accumulators Register (ICPAR)

The 8-bit pulse accumulators PAC3 and PAC2 can be enabled only if PAEN in PATCL ($20) is cleared. If PAEN is set, PA3EN and PA2EN have no effect.

The 8-bit pulse accumulators PAC1 and PAC0 can be enabled only if PBEN in PBTCL ($30) is cleared. If PBEN is set, PA1EN and PA0EN have no effect.

Read or write any time.

PAnEN — 8-Bit Pulse Accumulator Enable
0 = 8-Bit Pulse Accumulator is disabled.
1 = 8-Bit Pulse Accumulator is enabled.

3.3.22 DLYCT — Delay Counter Control Register

Register offset: $_29

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	DLY1	DLY0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-24 Delay Counter Control Register (DLYCT)

Read or write any time.

If enabled, after detection of a valid edge on input capture pin, the delay counter counts the pre-selected number of bus clock cycles, then it will generate a pulse on its output. The pulse is generated only if the level of input signal, after the preset delay, is the opposite of the level before the transition.This will avoid reaction to narrow input pulses.

After counting, the counter will be cleared automatically.

Delay between two active edges of the input signal period should be longer than the selected counter delay.

DLYn — Delay Counter Select

Table 3-8 Delay Counter Select

DLY1	DLY0	Delay
0	0	Disabled (bypassed)
0	1	256 bus clock cycles
1	0	512 bus clock cycles
1	1	1024 bus clock cycles

3.3.23 ICOVW — Input Control Overwrite Register

Register offset: $_2A

	BIT7	6	5	4	3	2	1	BIT0
R W	NOVW7	NOVW6	NOVW5	NOVW4	NOVW3	NOVW2	NOVW1	NOVW0
RESET:	0	0	0	0	0	0	0	0

Figure 3-25 Input Control Overwrite Register (ICOVW)

Read or write any time.

An IC register is empty when it has been read or latched into the holding register.

A holding register is empty when it has been read.

NOVWn — No Input Capture Overwrite

0 = The contents of the related capture register or holding register can be overwritten when a new input capture or latch occurs.

1 = The related capture register or holding register cannot be written by an event unless they are empty (see **4.2.1 IC Channels**). This will prevent the captured value to be overwritten until it is read or latched in the holding register.

3.3.24 ICSYS — Input Control System Control Register

Register offset: $_2B

	BIT7	6	5	4	3	2	1	BIT0
R W	SH37	SH26	SH15	SH04	TFMOD	PACMX	BUFEN	LATQ
RESET:	0	0	0	0	0	0	0	0

Figure 3-26 Input Control System Register (ICSYS)

Read: any time
Write: Can be written once (test_mode =0). Writes are always permitted when test_mode =1.

SHxy — Share Input action of Input Capture Channels x and y

0 = Normal operation
1 = The channel input ‘x’ causes the same action on the channel ‘y’. The port pin ‘x’ and the corresponding edge detector is used to be active on the channel ‘y’.

TFMOD — Timer Flag-setting Mode

Use of the TFMOD bit in the ICSYS register ($2B) in conjunction with the use of the ICOVW register ($2A) allows a timer interrupt to be generated after capturing two values in the capture and holding registers instead of generating an interrupt for every capture.

By setting TFMOD in queue mode, when NOVW bit is set and the corresponding capture and holding registers are emptied, an input capture event will first update the related input capture register with the main timer contents. At the next event the TCn data is transferred to the TCnH register, The TCn is updated and the CnF interrupt flag is set.

In all other input capture cases the interrupt flag is set by a valid external event on PTn.

0 = The timer flags C3F–C0F in TFLG1 ($0E) are set when a valid input capture transition on the corresponding port pin occurs.
1 = If in queue mode (BUFEN=1 and LATQ=0), the timer flags C3F–C0F in TFLG1 ($0E) are set only when a latch on the corresponding holding register occurs.
If the queue mode is not engaged, the timer flags C3F–C0F are set the same way as for TFMOD=0.

PACMX — 8-Bit Pulse Accumulators Maximum Count

0 = Normal operation. When the 8-bit pulse accumulator has reached the value $FF, with the next active edge, it will be incremented to $00.
1 = When the 8-bit pulse accumulator has reached the value $FF, it will not be incremented further. The value $FF indicates a count of 255 or more.

BUFEN — IC Buffer Enable

0 = Input Capture and pulse accumulator holding registers are disabled.
1 = Input Capture and pulse accumulator holding registers are enabled. The latching mode is defined by LATQ control bit.
Write one into ICLAT bit in MCCTL ($26), when LATQ is set will produce latching of input capture and pulse accumulators registers into their holding registers.

LATQ — Input Control Latch or Queue Mode Enable

The BUFEN control bit should be set in order to enable the IC and pulse accumulators holding registers. Otherwise LATQ latching modes are disabled.

Write one into ICLAT bit in MCCTL ($26), when LATQ and BUFEN are set will produce latching of input capture and pulse accumulators registers into their holding registers.

0 = Queue Mode of Input Capture is enabled.
The main timer value is memorized in the IC register by a valid input pin transition.
With a new occurrence of a capture, the value of the IC register will be transferred to its holding register and the IC register memorizes the new timer value.
1 = Latch Mode is enabled. Latching function occurs when modulus down-counter reaches zero or a zero is written into the count register MCCNT (see **4.2.1.2 Buffered IC Channels**).
With a latching event the contents of IC registers and 8-bit pulse accumulators are transferred to their holding registers. 8-bit pulse accumulators are cleared.

3.3.25 TIMTST — Timer Test Register

Register offset: $_2D

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	TCBYP	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-27 Timer Test Register (TIMTST)

Read: any time
Write: only in special mode (test_mode = 1).

TCBYP — Main Timer Divider Chain Bypass
0 = Normal operation
1 = For testing only. The 16-bit free-running timer counter is divided into two 8-bit halves and the prescaler is bypassed. The clock drives both halves directly.
When the high byte of timer counter TCNT ($04) overflows from $FF to $00, the TOF flag in TFLG2 ($0F) will be set.

3.3.26 PBCTL — 16-Bit Pulse Accumulator B Control Register

Register offset: $_30

	BIT7	6	5	4	3	2	1	BIT0
R	0	PBEN	0	0	0	0	PBOVI	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-28 16-Bit Pulse Accumulator B Control Register (PBCTL)

Read or write any time.

16-Bit Pulse Accumulator B (PACB) is formed by cascading the 8-bit pulse accumulators PAC1 and PAC0.

When PBEN is set, the PACB is enabled. The PACB shares the input pin with IC0.

PBEN — Pulse Accumulator B System Enable
0 = 16-bit Pulse Accumulator system disabled. 8-bit PAC1 and PAC0 can be enabled when their related enable bits in ICPAR ($28) are set.

1 = Pulse Accumulator B system enabled. The two 8-bit pulse accumulators PAC1 and PAC0 are cascaded to form the PACB 16-bit pulse accumulator. When PACB in enabled, the PACN1 and PACN0 registers contents are respectively the high and low byte of the PACB.
PA1EN and PA0EN control bits in ICPAR ($28) have no effect.

PBEN is independent from TEN. With timer disabled, the pulse accumulator can still function unless pulse accumulator is disabled.

PBOVI — Pulse Accumulator B Overflow Interrupt enable
0 = interrupt inhibited
1 = interrupt requested if PBOVF is set

3.3.27 PBFLG — Pulse Accumulator B Flag Register

Register offset: $_31

	BIT7	6	5	4	3	2	1	BIT0
R	0	0	0	0	0	0	PBOVF	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-29 Pulse Accumulator B Flag Register (PBFLG)

Read or write any time.

PBOVF — Pulse Accumulator B Overflow Flag

This bit is set when the 16-bit pulse accumulator B overflows from $FFFF to $0000, or when 8-bit pulse accumulator 1 (PAC1) overflows from $FF to $00.

This bit is cleared by a write to the PBFLG register with bit 1 set.

Any access to the PACN1 and PACN0 registers will clear the PBOVF flag in this register when TFFCA bit in register TSCR($06) is set.

When PACMX = 1, PBOVF bit can also be set if 8 - bit pulse accumulator 1 (PAC1) reaches $FF and followed an active edge comes on PT1.

3.3.28 PA3H–PA0H — 8-Bit Pulse Accumulators Holding Registers

Register offset: $_32

	BIT7	6	5	4	3	2	1	BIT0
R	PA3H7	PA3H6	PA3H5	PA3H4	PA3H3	PA3H2	PA3H1	PA3H0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-30 8-Bit Pulse Accumulators Holding Register 3 (PA3H)

Register offset: $_33

	BIT7	6	5	4	3	2	1	BIT0
R	PA2H7	PA2H6	PA2H5	PA2H4	PA2H3	PA2H2	PA2H1	PA2H0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-31 8-Bit Pulse Accumulators Holding Register 2 (PA2H)

Register offset: $_34

	BIT7	6	5	4	3	2	1	BIT0
R	PA1H7	PA1H6	PA1H5	PA1H4	PA1H3	PA1H2	PA1H1	PA1H0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-32 8-Bit Pulse Accumulators Holding Register 1 (PA1H)

Register offset: $_35

	BIT7	6	5	4	3	2	1	BIT0
R	PA0H7	PA0H6	PA0H5	PA0H4	PA0H3	PA0H2	PA0H1	PA0H0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-33 8-Bit Pulse Accumulators Holding Register 0 (PA0H)

Read: any time
Write: has no effect.

These registers are used to latch the value of the corresponding pulse accumulator when the related bits in register ICPAR ($28) are enabled (see **4.2.2 Pulse Accumulators**).

3.3.29 MCCNT — Modulus Down-Counter Count Register

Register address:$_36-$_37

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R W	mccnt 15	mccnt 14	mccnt 13	mccnt 12	mccnt 11	mccnt 10	mccnt 9	mccnt 8	mccnt 7	mccnt 6	mccnt 5	mccnt 4	mccnt 3	mccnt 2	mccnt 1	mccnt 0
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3-34 Modulus Down-Counter Count Register (MCCNT)

Read or write any time.

A full access for the counter register should take place in one clock cycle. A separate read/write for high byte and low byte will give different result than accessing them as a word.

If the RDMCL bit in MCCTL register is cleared, reads of the MCCNT register will return the present value of the count register. If the RDMCL bit is set, reads of the MCCNT will return the contents of the load register.

If a $0000 is written into MCCNT and modulus counter while LATQ and BUFEN in ICSYS ($2B) register are set, the input capture and pulse accumulator registers will be latched.

With a $0000 write to the MCCNT, the modulus counter will stay at zero and does not set the MCZF flag in MCFLG register.

If modulus mode is enabled (MODMC=1), a write to this address will update the load register with the value written to it. The count register will not be updated with the new value until the next counter underflow.

The FLMC bit in MCCTL ($26) can be used to immediately update the count register with the new value if an immediate load is desired.

If modulus mode is not enabled (MODMC=0), a write to this address will clear the prescaler and will immediately update the counter register with the value written to it and down-counts once to $0000.

3.3.30 Timer Input Capture Holding Registers 0-3

Register offset: $_38-$_39

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-35 Timer Input Capture Holding Register 0 (TC0H)

Register offset: $_3A-$_3B

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-36 Timer Input Capture Holding Register 1 (TC1H)

Register offset: $_3C-$_3D

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-37 Timer Input Capture Holding Register 2 (TC2H)

Register offset: $_3E-$_3F

	BIT15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT0
R	TC15	TC14	TC13	TC12	TC11	TC10	TC9	TC8	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-38 Timer Input Capture Holding Register 3 (TC3H)

Read: any time
Write: has no effect.

These registers are used to latch the value of the input capture registers TC0 – TC3. The corresponding IOSn bits in TIOS ($00) should be cleared (see **4.2.1 IC Channels**).

Section 4 Functional Description

4.1 General

This section provides a complete functional description of the ECT block, detailing the operation of the design from the end user perspective in a number of subsections.

Refer to the Timer Block Diagrams from **Figure 4-1** to **Figure 4-5** as necessary.

Figure 4-1 Detailed Timer Block Diagram in Latch mode

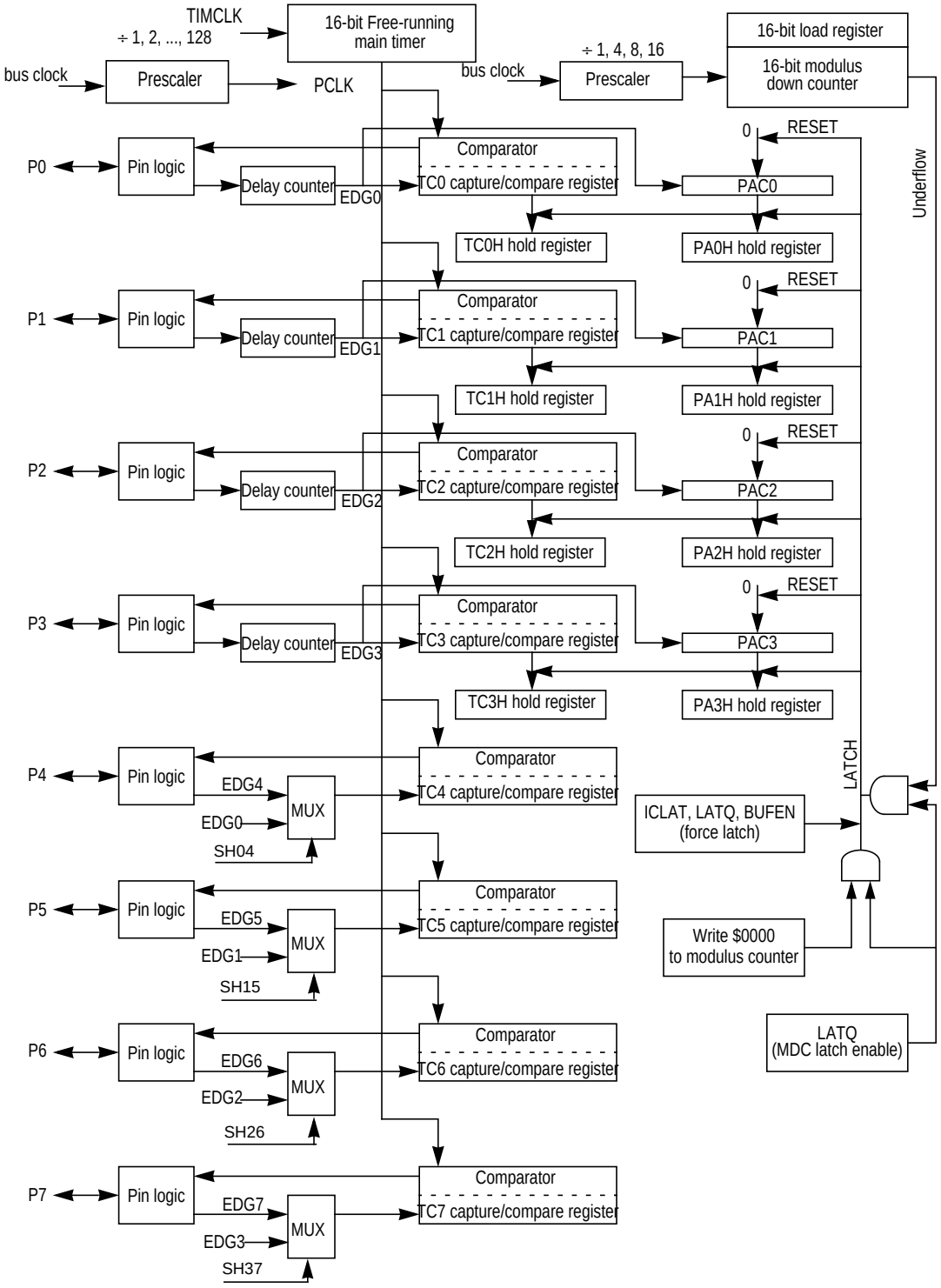


Figure 4-2 Detailed Timer Block Diagram in Queue mode

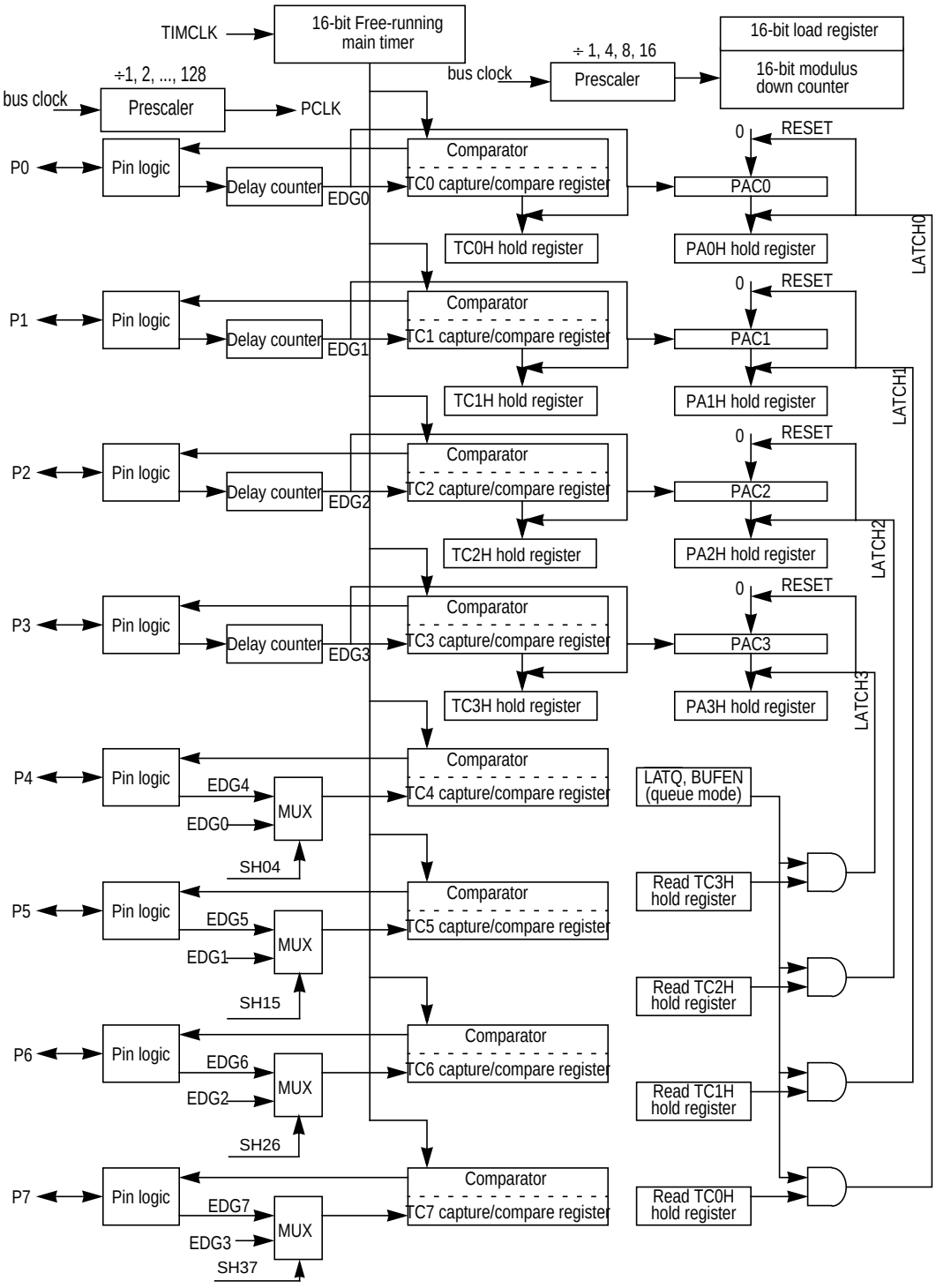


Figure 4-3 8-Bit Pulse Accumulators Block Diagram

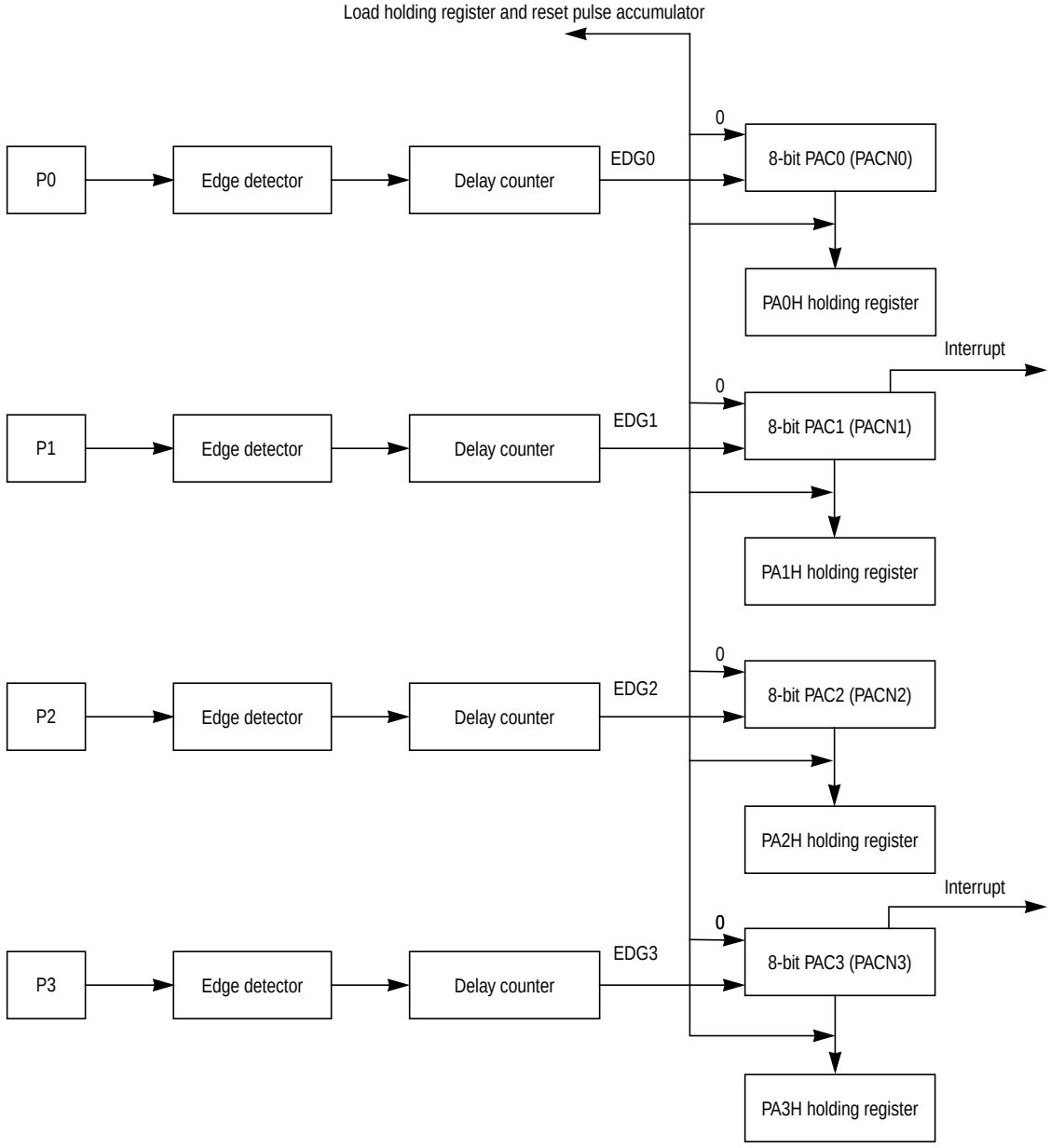


Figure 4-4 16-Bit Pulse Accumulators Block Diagram

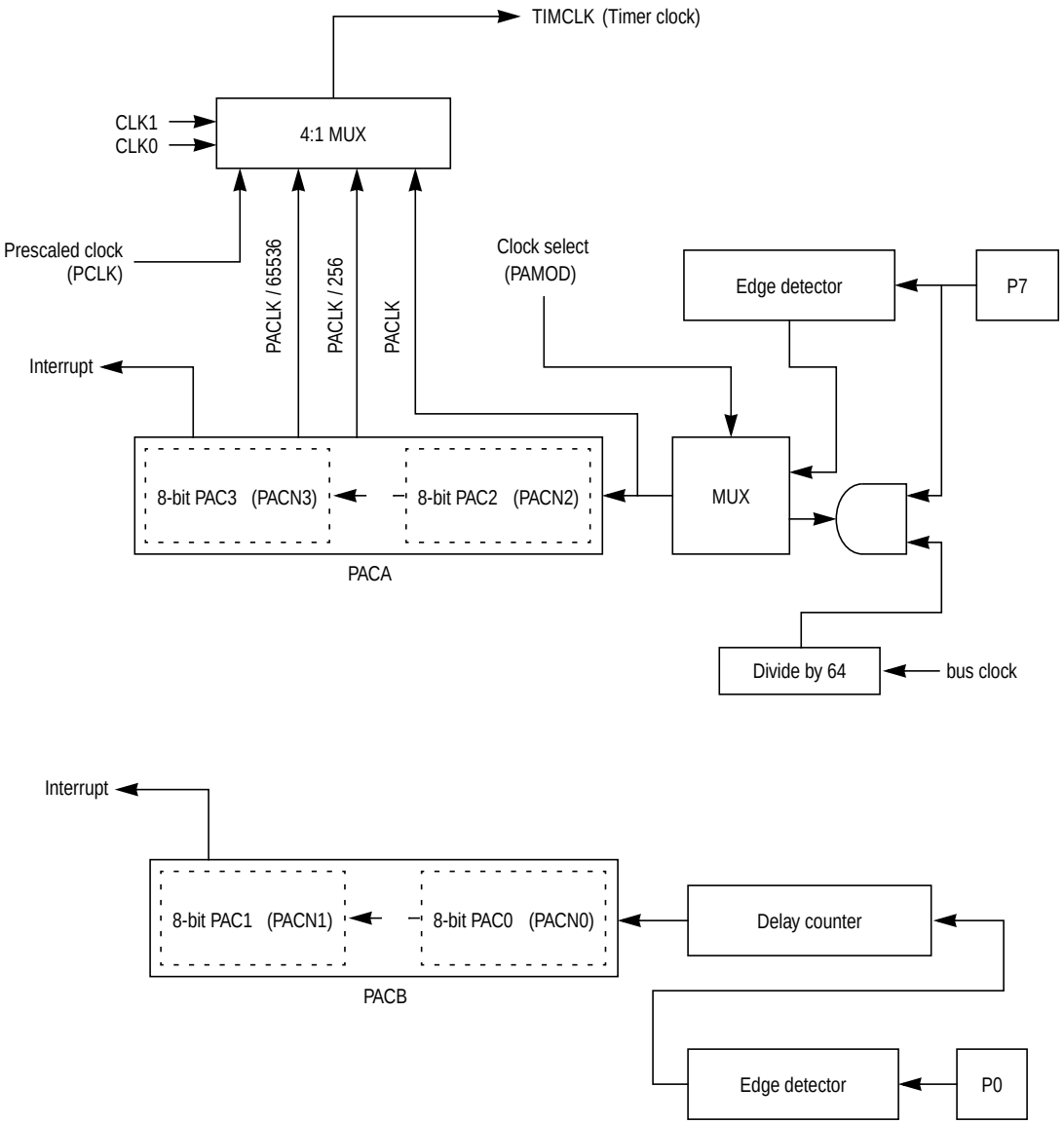
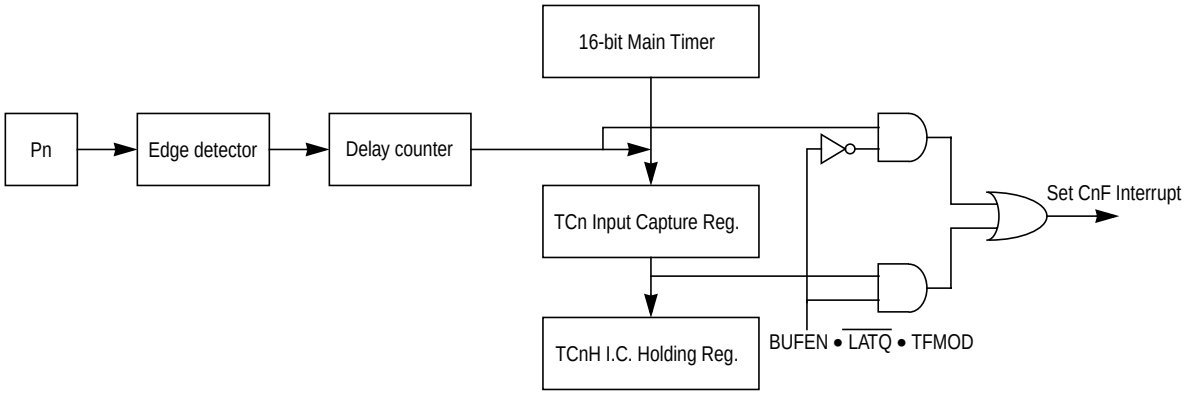


Figure 4-5 Block Diagram for Port7 with Output compare/Pulse Accumulator A



4.2 Enhanced Capture Timer Modes of Operation

The Enhanced Capture Timer has 8 Input Capture, Output Compare (IC/OC) channels same as on the HC12 standard timer (timer channels TC0 to TC7). When channels are selected as input capture by selecting the IOSn bit in TIOS register, they are called Input Capture (IC) channels.

Four IC channels are the same as on the standard timer with one capture register each which memorizes the timer value captured by an action on the associated input pin.

Four other IC channels, in addition to the capture register, have also one buffer each called holding register. This permits to memorize two different timer values without generation of any interrupt.

Four 8-bit pulse accumulators are associated with the four buffered IC channels. Each pulse accumulator has a holding register to memorize their value by an action on its external input. Each pair of pulse accumulators can be used as a 16-bit pulse accumulator.

The 16-bit modulus down-counter can control the transfer of the IC registers contents and the pulse accumulators to the respective holding registers for a given period, every time the count reaches zero.

The modulus down-counter can also be used as a stand-alone time base with periodic interrupt capability.

4.2.1 IC Channels

The IC channels are composed of four standard IC registers and four buffered IC channels.

An **IC register** is **empty** when it has been read or latched into the holding register.

A **holding register** is **empty** when it has been read.

4.2.1.1 Non-Buffered IC Channels

The main timer value is memorized in the IC register by a valid input pin transition. If the corresponding NOVWx bit of the ICOVW register is cleared, with a new occurrence of a capture, the contents of IC register are overwritten by the new value.

If the corresponding NOVWx bit of the ICOVW register is set, the capture register cannot be written unless it is empty.

This will prevent the captured value to be overwritten until it is read.

4.2.1.2 Buffered IC Channels

There are two modes of operations for the buffered IC channels.

- IC Latch Mode:

When enabled (LATQ=1), the main timer value is memorized in the IC register by a valid input pin transition. See **Figure 4-1**

The value of the buffered IC register is latched to its holding register by the Modulus counter for a given period when the count reaches zero, by a write $0000 to the modulus counter or by a write to ICLAT in the MCCTL register.

If the corresponding NOVWn bit of the ICOVW register is cleared, with a new occurrence of a capture, the contents of IC register are overwritten by the new value. In case of latching, the contents of its holding register are overwritten.

If the corresponding NOVWn bit of the ICOVW register is set, the capture register or its holding register cannot be written by an event unless they are empty (see **4.2.1**). This will prevent the captured value to be overwritten until it is read or latched in the holding register.

- IC queue mode:

When enabled (LATQ=0), the main timer value is memorized in the IC register by a valid input pin transition. See **Figure 4-2**

If the corresponding NOVWn bit of the ICOVW register is cleared, with a new occurrence of a capture, the value of the IC register will be transferred to its holding register and the IC register memorizes the new timer value.

If the corresponding NOVWn bit of the ICOVW register is set, the capture register or its holding register cannot be written by an event unless they are empty (see **4.2.1**).

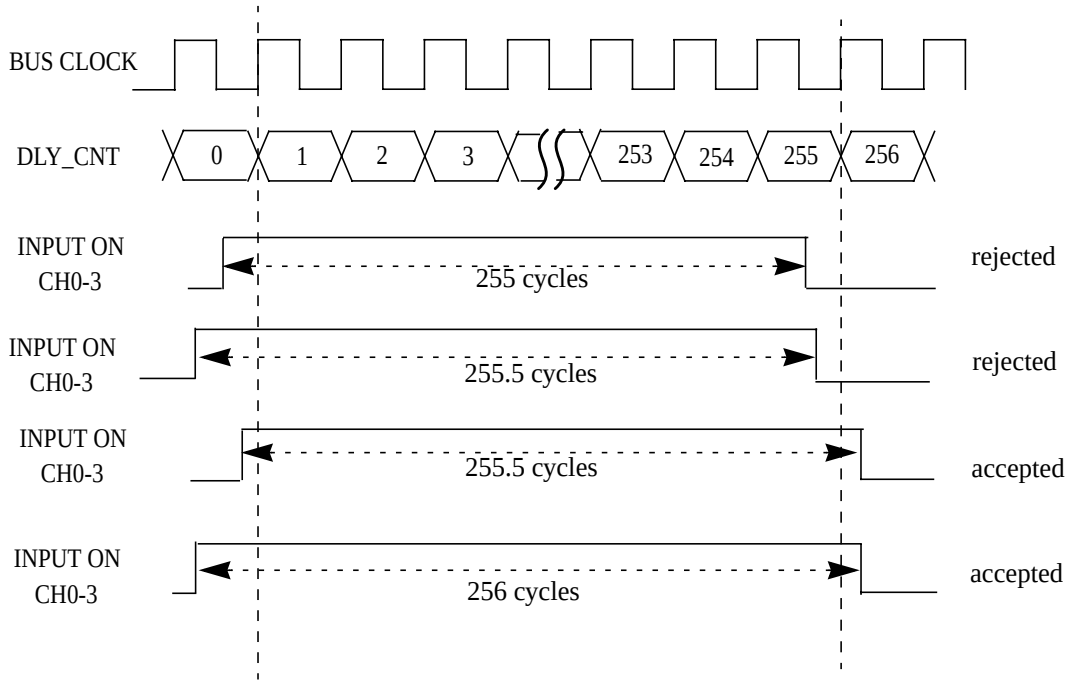
In queue mode, reads of holding register will latch the corresponding pulse accumulator value to its holding register.

4.2.1.3 Delayed IC channels

There are four delay counters in this module associated with IC channels 0 - 3. The use of this feature is

explained in the diagram and notes below.

Figure 4-6 Channel Input validity with delay counter feature



In the diagram above a delay counter value of 256 bus cycles is considered.

1. Input pulses with a duration of (DLY_CNT - 1) cycles or shorter are rejected.
2. Input pulses with a duration between (DLY_CNT - 1) and DLY_CNT cycles may be rejected or accepted, depending on their relative alignment with the sample points.
3. Input pulses with a duration between (DLY_CNT - 1) and DLY_CNT cycles may be rejected or accepted, depending on their relative alignment with the sample points.
4. Input pulses with a duration of DLY_CNT or longer are accepted.

4.2.2 Pulse Accumulators

There are four 8-bit pulse accumulators with four 8-bit holding registers associated with the four IC buffered channels. A pulse accumulator counts the number of active edges at the input of its channel.

The user can prevent 8-bit pulse accumulators counting further than $FF by PACMX control bit in ICSYS ($2B). In this case a value of $FF means that 255 counts or more have occurred.

Each pair of pulse accumulators can be used as a 16-bit pulse accumulator. See **Figure 4-4**

There are two modes of operation for the pulse accumulators.

4.2.2.1 Pulse Accumulator latch mode

The value of the pulse accumulator is transferred to its holding register when the modulus down-counter reaches zero, a write $0000 to the modulus counter or when the force latch control bit ICLAT is written.

At the same time the pulse accumulator is cleared.

4.2.2.2 Pulse Accumulator queue mode

When queue mode is enabled, reads of an input capture holding register will transfer the contents of the associated pulse accumulator to its holding register.

At the same time the pulse accumulator is cleared.

4.2.3 Modulus Down-Counter

The modulus down-counter can be used as a time base to generate a periodic interrupt. It can also be used to latch the values of the IC registers and the pulse accumulators to their holding registers.

The action of latching can be programmed to be periodic or only once.

4.2.4 Channel Configurations

Timer Channels can be configured as input capture channels or output compare channels. Following are the ways a port can be configured as an output for OC.

The pin associated with channel 7 becomes output-tied to OC7 when

- TEN = 1, IOS7 = 1, and either or both of OM7 and OL7 are set. or
- OC7M7 =1 and IOS7 = 1.

When masking, the timer does not have to be enabled so that the pin associated with OCn becomes an output tied to OCn.

The pins associated with channels 0-6 become output-tied to OCn (n=0..6) when

- TEN = 1, IOSn = 1, and either or both of OMn and OLn are set or
- OC7Mn =1, IOS7 = 1 and IOSn = 1

Once the pin is configured as OC, its initial state is zero and its status is changed (if needed) on consecutive clock cycles following the write which enabled the ECT to drive the pin. In other words after a pin starts to be driven by ECT OC logic, it is forced low for at least one clock cycle.

Section 5 Reset

5.1 General

The reset state of each individual bit is listed within the Register Description section (**Section 3 Memory Map and Registers**) which details the registers and their bit-fields.

Section 6 Interrupts

6.1 General

This section describes interrupts originated by the ECT_16B8C block.The MCU must service the interrupt requests. **Table 6-1** lists the interrupts generated by the ECT to communicate with the MCU.

Table 6-1 ECT Interrupts

Interrupt Source	Description
Timer Channel 7-0	Active high timer channel interrupts 7-0
Modulus counter underflow	Active high modulus counter interrupt
Pulse Accumulator B Overflow	Active high pulse accumulator B interrupt
Pulse Accumulator A Input	Active high pulse accumulator A input interrupt
Pulse Accumulator A Overflow	Pulse accumulator overflow interrupt
Timer Overflow	Timer Overflow interrupt

6.2 Description of Interrupt Operation

The ECT_16B8C only originates interrupt requests. The following is a description of how the module makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt number are chip dependent.

6.2.1 Channel [7:0] Interrupt

This active high output will be asserted by the module to request a timer channel 7 - 0 interrupt to be serviced by the system controller.

6.2.2 Modulus Counter Interrupt

This active high output will be asserted by the module to request a modulus counter underflow interrupt to be serviced by the system controller.

6.2.3 Pulse Accumulator B Overflow Interrupt)

This active high output will be asserted by the module to request a timer pulse accumulator B overflow interrupt to be serviced by the system controller.

6.2.4 Pulse Accumulator A Input Interrupt

This active high output will be asserted by the module to request a timer pulse accumulator A input interrupt to be serviced by the system controller.

6.2.5 Pulse Accumulator A Overflow Interrupt

This active high output will be asserted by the module to request a timer pulse accumulator A overflow interrupt to be serviced by the system controller.

6.2.6 Timer Overflow Interrupt

This active high output will be asserted by the module to request a timer overflow interrupt to be serviced by the system controller.

User Guide End Sheet

FINAL PAGE OF
60
PAGES

EETS4K
Block User Guide
V02.07

Original Release Date: 22 FEB 2001
Revised: 8 APR 2003

Motorola, Inc

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.00	22FEB01	15NOV00		Initial Version.
V02.00	22MAY01	27MAR01		Do not set PVIOL for erase verify command if address written to is in a protected area. Allow data writes of $00 and $40 to ECLKDIV.
V02.01	25MAY01			Make formats SRS V2 compliant. Reorder and restructure document. Add overview block diagram.
V02.02	19JUL01			Add document names. Hide names and variable definitions.
V02.03	29OCT01			Minor cleanup.
V02.04	11MAR02			Modify document number. Fix cross references.
V02.05	09JUL02			Modify document number. Rearrange Section **3.2 Module Memory Map**.
V02.06	23JAN03			Add description of EADDR and EDATA registers.
V02.07	08APR03			Modify description of **3.3.5 EPROT — EEPROM Protection Register** to clarify mass erase restrictions. Modify description of CBEIF and CCIF flags in **3.3.6 ESTAT — EEPROM Status Register**.

Table of Contents

Section 5 Resets

Section 6 Interrupts

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

This document describes the EETS4K module which is a 4K byte EEPROM (Non-Volatile) memory. The EETS4K block uses a small sector Flash memory to emulate EEPROM functionality. It is an array of electrically erasable and programmable, non-volatile memory. The EEPROM memory is organized as 2048 rows of 2 bytes (1 word). The EEPROM memory's erase sector size is 2 rows or 2 words (4 bytes).

The EEPROM memory may be read as either bytes, aligned words or misaligned words. Read access time is one bus cycle for byte and aligned word, and two bus cycles for misaligned words.

Program and erase functions are controlled by a command driven interface. Both sector erase and mass erase of the entire EEPROM memory are supported. An erased bit reads '1' and a programmed bit reads '0'. The high voltage required to program and erase is generated internally by on-chip charge pumps.

It is not possible to read from the EEPROM memory while it is being erased or programmed.

The EEPROM memory is ideal for data storage for single-supply applications allowing for field reprogramming without requiring external programming voltage sources.

WARNING

A word must be erased before being programmed. Cumulative programming of bits within a word is not allowed.

1.1.1 Glossary

Command Sequence

A three-step MCU instruction sequence to program, erase or erase-verify the EEPROM.

1.2 Features

- 4K bytes of EEPROM memory.
- Minimum erase sector of 4 bytes.
- Automated program and erase algorithms.
- Interrupts on EEPROM command completion and command buffer empty.
- Fast sector erase and word program operation.
- 2-stage command pipeline.
- Flexible protection scheme for protection against accidental program or erase.
- Single power supply program and erase.

1.3 Modes of Operation

- Program and erase operation (please refer to **4.1** for details).

1.4 Block Diagram

Figure 1-1 shows a block diagram of the EETS4K module.

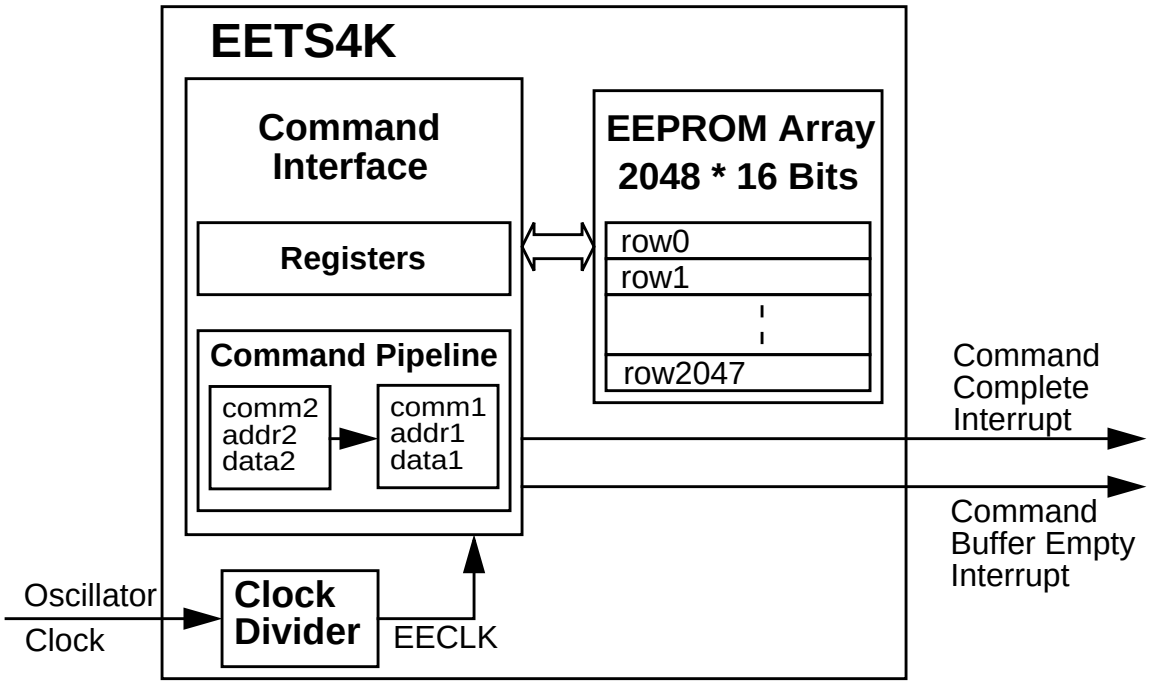


Figure 1-1 Module Block Diagram

Section 2 External Signal Description

2.1 Overview

The EETS4K module contains no signals that connect off chip.

Section 3 Memory Map and Registers

3.1 Overview

This section describes the EETS4K memory map and registers.

3.2 Module Memory Map

Figure 3-1 shows the EETS4K memory map. Location of the EEPROM array in the MCU memory map is defined in the specific MCU Device User Guide and is reflected in the INITEE register contents defined in the HCS12 Core User Guide. Shown within the EEPROM array are a protection/reserved field and user-defined EEPROM protected sectors. The 16 byte protection/reserved field is located in the EEPROM array from address $_FF0 to $_FFF. A description of this protection/reserved field is given in **Table 3-1**.

Table 3-1 EEPROM Protection/Reserved Field

Array Address	Size (bytes)	Description
$_FF0 - $FFC	13	Reserved
$_FFD	1	EEPROM Protection byte
$_FFE - $_FFF	2	Reserved

The EEPROM module has hardware interlocks which protect data from accidental corruption. A protected sector is located at the higher address end of the EEPROM array, just below address $_FFF. The protected sector in the EEPROM array can be sized from 64 bytes to 512 bytes. In addition, the EPOPEN bit in the EPROT register (see section **3.3.5**) can globally protect the entire EEPROM array.

NOTE

Chip security is defined at the MCU level.

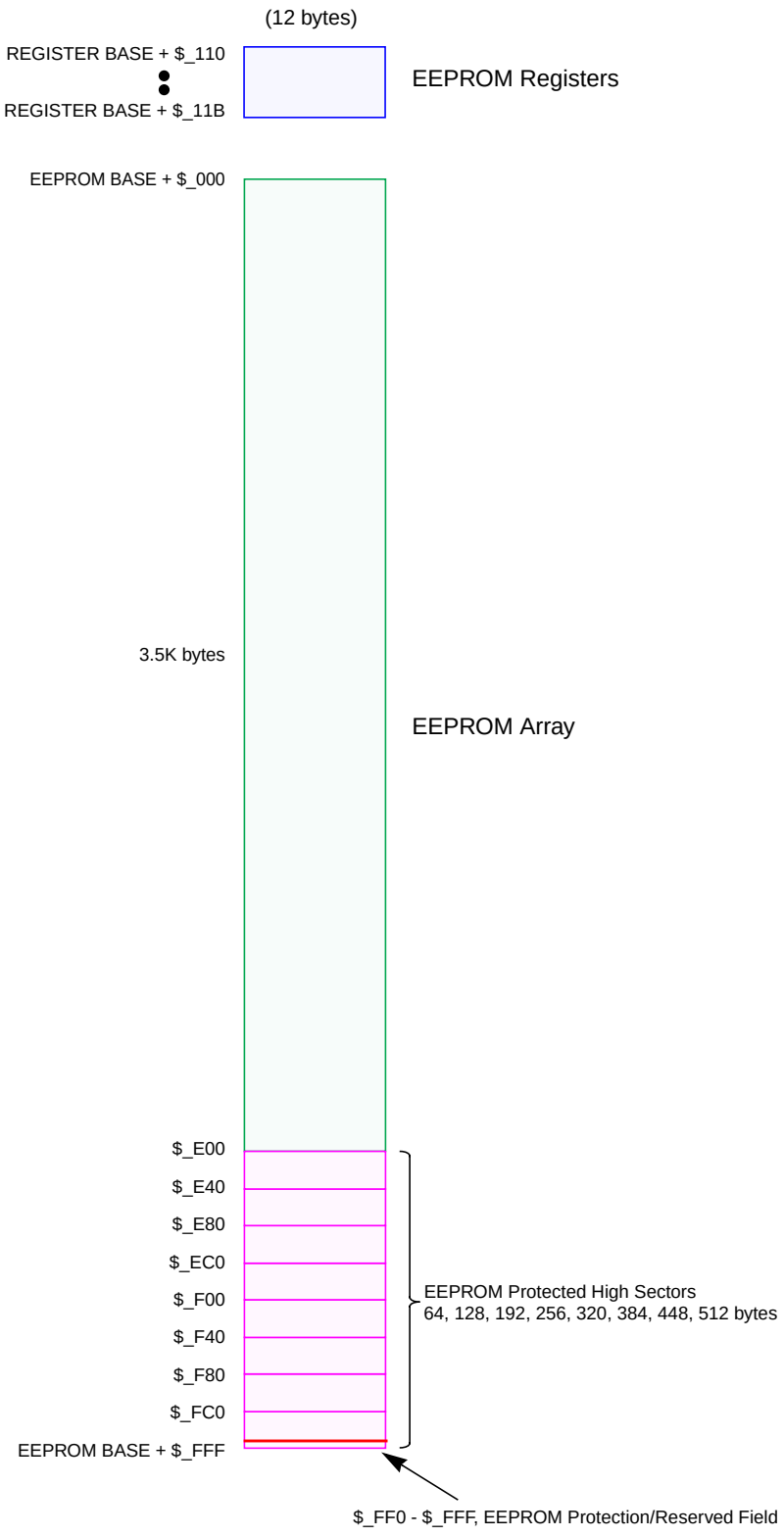


Figure 3-1 EEPROM Memory Map

The EEPROM module also contains a set of 12 control and status registers located in address space BASE + $110 to BASE + $11B.

Table 3-2 gives an overview on all EETS4K registers.

Table 3-2 EEPROM Register Memory Map

Address Offset	Use	Access
$_00	EEPROM Clock Divider Register (ECLKDIV)	R/W
$_01	RESERVED1[1]	R
$_02	RESERVED2[1]	R
$_03	EEPROM Configuration Register (ECNFG)	R/W
$_04	EEPROM Protection Register (EPROT)	R/W
$_05	EEPROM Status Register (ESTAT)	R/W
$_06	EEPROM Command Register (ECMD)	R/W
$_07	RESERVED3[1]	R
$_08	EEPROM High Address Register (EADDRHI)	R/W
$_09	EEPROM Low Address Register (EADDRLO)	R/W
$_0A	EEPROM High Data Register (EDATAHI)	R/W
$_0B	EEPROM Low Data Register (EDATALO)	R/W

NOTES:
1. Intended for factory test purposes only.

NOTE: *Register Address = Register Base Address + $110 + Address Offset, where the Register Base Address is defined by the HCS12 Core INITRG register and the Address Offset is defined by the EEPROM module.*

3.3 Register Descriptions

3.3.1 ECLKDIV — EEPROM Clock Divider Register

The ECLKDIV register is used to control timed events in program and erase algorithms.

Register address BASE + $110

	7	6	5	4	3	2	1	0
R	EDIVLD	PRDIV8	EDIV5	EDIV4	EDIV3	EDIV2	EDIV1	EDIV0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 EEPROM Clock Divider Register (ECLKDIV)

All bits in the ECLKDIV register are readable, bits 6-0 are write once and bit 7 is not writable.

EDIVLD — Clock Divider Loaded.

1 = Register has been written to since the last reset.

0 = Register has not been written.

PRDIV8 — Enable Prescaler by 8.

1 = Enables a prescaler by 8, to divide the EEPROM module input oscillator clock before feeding into the CLKDIV divider.

0 = The input oscillator clock is directly fed into the ECLKDIV divider.

EDIV[5:0] — Clock Divider Bits.

The combination of PRDIV8 and EDIV[5:0] effectively divides the EEPROM module input oscillator clock down to a frequency of 150kHz - 200kHz. The maximum divide ratio is 512. Please refer to section **4.1.1** for more information.

3.3.2 RESERVED1

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $111

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 RESERVED1

All bits read zero and are not writable.

3.3.3 RESERVED2

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $112

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-4 RESERVED2

All bits read zero and are not writable.

3.3.4 ECNFG — EEPROM Configuration Register

The ECNFG register enables the EEPROM interrupts.

Register address BASE + $113

	7	6	5	4	3	2	1	0
R	CBEIE	CCIE	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-5 EEPROM Configuration Register (ECNFG)

CBEIE and CCIE are readable and writable. Bits 5-0 read zero and are not writable.

CBEIE — Command Buffer Empty Interrupt Enable.

The CBEIE bit enables the interrupts in case of an empty command buffer in the EEPROM.
1 = An interrupt will be requested whenever the CBEIF flag, **Figure 3-7**, is set.
0 = Command Buffer Empty interrupts disabled.

CCIE — Command Complete Interrupt Enable.

The CCIE bit enables the interrupts in case of all commands being completed in the EEPROM.
1 = An interrupt will be requested whenever the CCIF, **Figure 3-7**, flag is set.
0 = Command Complete interrupts disabled.

3.3.5 EPROT — EEPROM Protection Register

The EPROT register defines which EEPROM sectors are protected against program or erase.

Register address BASE + $114

	7	6	5	4	3	2	1	0
R	EPOPEN	NV6	NV5	NV4	EPDIS	EP2	EP1	EP0
W								
Reset:	F	F	F	F	F	F	F	F

= Unimplemented or Reserved

Figure 3-6 EEPROM Protection Register (EPROT)

The EPROT register is loaded from EEPROM array address $_FFD during reset, as indicated by the “F” in **Figure 3-6**.

All bits in the EPROT register are readable. Bits NV[6:4] are not writable. The EPOPEN and EPDIS bits in the EPROT register can only be written to the protected state (i.e. 0). The EP[2:0] bits can be written anytime until bit EPDIS is cleared. If the EPOPEN bit is cleared, then the state of the EPDIS and EP[2:0] bits is irrelevant.

To change the EEPROM protection that will be loaded on reset, the upper sector of EEPROM must first be unprotected, then the EEPROM Protect byte located at address $_FFD must be written to.

A protected EEPROM sector is disabled by the EPDIS bit while the size of the protected sector is defined by the EP bits in the EPROT register.

Trying to alter any of the protected areas will result in a protect violation error and PVIOL flag will be set in the ESTAT register. A mass erase of a whole EEPROM block is only possible when protection is fully disabled by setting the EPOPEN and EPDIS bits. An attempt to mass erase an EEPROM block while protection is enabled will set the PVIOL flag in the ESTAT register.

EPOPEN — Opens the EEPROM for program or erase.

1 = The EEPROM sectors not protected are enabled for program or erase.

0 = The whole EEPROM array is protected. In this case the EPDIS and EP bits within the protection register are ignored.

EPDIS — EEPROM Protection address range Disable.

The EPDIS bit determines whether there is a protected area in the space of the EEPROM address map.

1 = Protection disabled.

0 = Protection enabled.

EP[2:0] — EEPROM Protection Address Size.

The EP[2:0] bits determine the size of the protected sector. Refer to **Table 3-3**.

Table 3-3 EEPROM Address Range Protection

EP[2:0]	Protected Address Range	Protected Size
000	$_FC0-$_FFF	64 bytes
001	$_F80-$_FFF	128 bytes
010	$_F40-$_FFF	192 bytes
011	$_F00-$_FFF	256 bytes
100	$_EC0-$_FFF	320 bytes
101	$_E80-$_FFF	384 bytes
110	$_E40-$_FFF	448 bytes
111	$_E00-$_FFF	512 bytes

NV[6:4] — Non-Volatile Flag Bits.

These three bits are available to the user as non-volatile flags.

3.3.6 ESTAT — EEPROM Status Register

The ESTAT register defines the EEPROM state machine command status and EEPROM array access, protection and erase verify status.

Register address BASE + $115

	7	6	5	4	3	2	1	0
R	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
W								
Reset:	1	1	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-7 EEPROM Status Register (ESTAT)

Register bits CBEIF, PVIOL and ACCERR are readable and writable, bits CCIF and BLANK are readable and not writable, bits 3, 1 and 0 read zero and are not writable.

CBEIF — Command Buffer Empty Interrupt Flag.

The CBEIF flag indicates that the address, data and command buffers are empty so that a new command sequence can be started. The CBEIF flag is cleared by writing a “1” to CBEIF. Writing a “0” to the CBEIF flag has no effect on CBEIF. Writing a “0” to CBEIF after writing an aligned word to the EEPROM address space but before CBEIF is cleared will abort a command sequence and cause

the ACCERR flag in the ESTAT register to be set. Writing a “0” to CBEIF outside of a command sequence will not set the ACCERR flag. The CBEIF flag is used together with the CBEIE bit in the ECNFG register to generate an interrupt request.

1 = Buffers are ready to accept a new command.

0 = Buffers are full.

CCIF — Command Complete Interrupt Flag.

The CCIF flag indicates that there are no more commands pending. The CCIF flag is cleared when CBEIF is cleared and sets automatically upon completion of all active and pending commands. The CCIF flag does not set when an active command completes and a pending command is fetched from the command buffer. Writing to the CCIF flag has no effect. The CCIF flag is used together with the CCIE bit in the ECNFG register to generate an interrupt request.

1 = All commands are completed.

0 = Command in progress.

PVIOL — Protection Violation.

The PVIOL flag indicates an attempt was made to program or erase an address in a protected EEPROM memory area (see **4.1.4 Illegal EEPROM Operations**) . The PVIOL flag is cleared by writing a “1” to PVIOL. Writing a “0” to the PVIOL flag has no effect on PVIOL. While PVIOL is set, it is not possible to launch another command in the EEPROM.

1 = A protection violation has occurred.

0 = No failure.

ACCERR — EEPROM Access Error.

The ACCERR flag indicates an illegal access to the selected EEPROM array (see **4.1.4 Illegal EEPROM Operations**) . This can be either a violation of the command sequence, issuing an illegal command (illegal combination of the CMDBx bits in the ECMD register) or the execution of a CPU STOP instruction while a command is executing (CCIF=0). The ACCERR flag is cleared by writing a “1” to ACCERR. Writing a “0” to the ACCERR flag has no effect on ACCERR. While ACCERR is set, it is not possible to launch another command in the EEPROM.

1 = Access error has occurred.

0 = No failure.

BLANK — Array has been verified as erased.

The BLANK flag indicates that an erase verify command has checked the EEPROM array and found it to be erased. The BLANK flag is cleared by hardware when CBEIF is cleared as part of a new valid command sequence. Writing to the BLANK flag has no effect on BLANK.

1 = EEPROM array verifies as erased.

0 = If an erase verify command has been requested, and the CCIF flag is set, then a zero in BLANK indicates array is not erased.

3.3.7 ECMD — EEPROM Command Register

The ECMD register defines the EEPROM commands.

Register address BASE + $116

	7	6	5	4	3	2	1	0
R	0	CMDB6	CMDB5	0	0	CMDB2	0	CMDB0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-8 EEPROM Command Register (ECMD)

Bits 7, 4, 3 and 1 read zero and are not writable. Bits CMDB6, CMDB5, CMDB2 and CMDB0 are readable and writable during a command sequence.

CMDB — Valid normal mode commands are shown in **Table 3-4**. Any other command than those mentioned in **Table 3-4** sets the ACCERR bit in the ESTAT register (**3.3.6**).

Table 3-4 EEPROM Normal Mode Commands

Command	Meaning
$05	Erase Verify
$20	Word Program
$40	Sector Erase
$41	Mass Erase
$60	Sector Modify

3.3.8 RESERVED3

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $117

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-9 RESERVED3

All bits read zero and are not writable.

3.3.9 EADDR — EEPROM Address Register

EADDRHI and EADDRLO are the EEPROM address registers.

Register address Base + $118

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	EABHI		
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-10 EEPROM Address High Register (EADDRHI)

Register address Base + $119

	7	6	5	4	3	2	1	0
R	EABLO							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-11 EEPROM Address Low Register (EADDRLO)

In normal modes, all EADDRHI and EADDRLO bits read zero and are not writable.

In special modes, all EADDRHI and EADDRLO bits are readable and writable except EADDRHI[7:3] which are not writable and always read zero.

For sector erase, the MCU address bits AB[1:0] are ignored.

For mass erase, any address within the block is valid to start the command.

3.3.10 EDATA — EEPROM Data Register

EDATAHI and EDATALO are the EEPROM data registers.

Register address BASE + $11A

	7	6	5	4	3	2	1	0
R	EDHI							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-12 EEPROM Data High Register (EDATAHI)

Register address BASE + $11B

	7	6	5	4	3	2	1	0
R W	EDLO							
Reset:	0	0	0	0	0	0	0	0

Figure 3-13 EEPROM Data Low Register (EDATALO)

In normal modes, all EDATAHI and EDATALO bits read zero and are not writable.

In special modes, all EDATAHI and EDATALO bits are readable and writable.

Section 4 Functional Description

4.1 Program and Erase Operation

Write and read operations are both used for the program and erase algorithms described in this section. These algorithms are controlled by a state machine whose timebase EECLK is derived from the oscillator clock via a programmable divider. The command register as well as the associated address and data registers operate as a buffer and a register (2-stage FIFO) so that a new command along with the necessary data and address can be stored to the buffer while the previous command is still in progress. The pipelined operation allows a simplification of command launching. Buffer empty as well as command completion are signalled by flags in the EEPROM status register. Interrupts for the EEPROM will be generated if enabled.

The next four subsections describe:

- How to write the ECLKDIV register.
- The write sequences used to program and erase the EEPROM, but also to perform more sophisticated commands like sector modify and erase verify.
- Valid EEPROM commands.
- Errors resulting from illegal EEPROM operations.

4.1.1 Writing the ECLKDIV Register

Prior to issuing any program or erase command, it is first necessary to write the ECLKDIV register to divide the oscillator down to within 150kHz to 200kHz range. The program and erase timings are also a function of the bus clock, such that the ECLKDIV determination must take this information into account. If we define:

- EECLK as the clock of the EEPROM timing control block
- Tbus as the period of the bus clock
- INT(x) as taking the integer part of x (e.g. INT(4.323)=4),

then ECLKDIV register bits PRDIV8 and EDIV[5:0] are to be set as described in **Figure 4-1**.

For example, if the oscillator clock is 950kHz and the bus clock is 10MHz, ECLKDIV bits EDIV[5:0] should be set to 4 (binary 000100) and bit PRDIV8 set to 0. The resulting EECLK is then 190kHz. As a result, the EEPROM algorithm timings are increased over optimum target by:

$$(200 - 190)/200 \times 100 = 5\%$$

NOTE

Command execution time will increase proportionally with the period of EECLK.

WARNING

Because of the impact of clock synchronization on the accuracy of the functional timings, programming or erasing the EEPROM cannot be performed if the bus clock runs at less than 1 MHz. Programming the EEPROM with an oscillator clock < 150kHz should be avoided. Setting ECLKDIV to a value such that EECLK < 150kHz can reduce the lifetime of the EEPROM due to overstress. Setting ECLKDIV to a value such that (1/EECLK+Tbus) < 5μs can result in incomplete programming or erasure of the memory array cells.

If the ECLKDIV register is written, the bit EDIVLD is set automatically. If this bit is zero, the register has not been written since the last reset. Program and erase commands will not be executed if this register has not been written to.

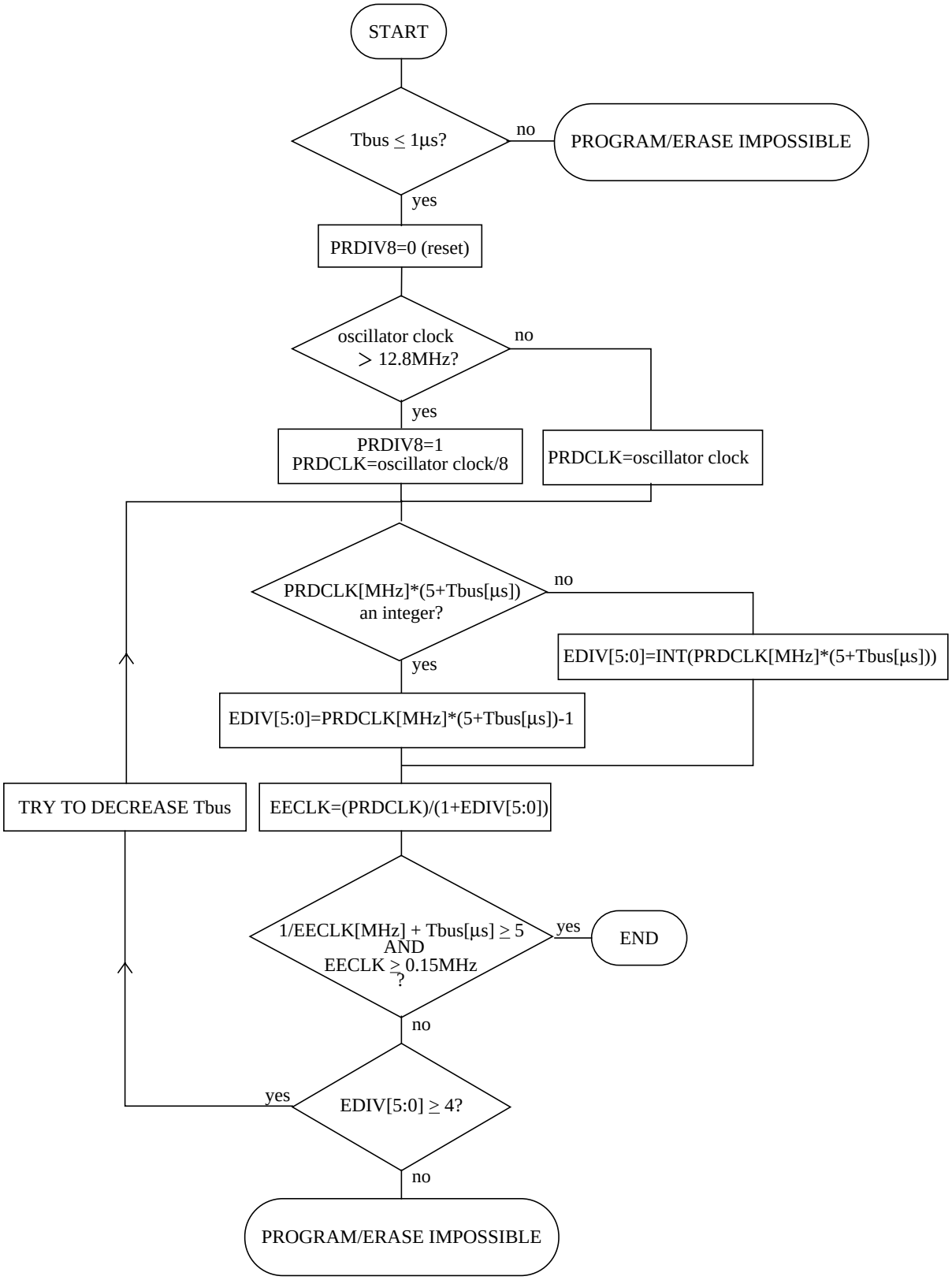


Figure 4-1 PRDIV8 and EDIV bits Determination Procedure

4.1.2 Program and Erase

A Command State Machine is used to supervise the write sequencing for program and erase. More specialized commands like sector modify or erase verify follow the same flow. Before starting a command sequence, it is necessary to check that there is no pending access error or protection violation (the ACCERR and PVIOL flags should be cleared in the ESTAT register).

After this initial step, the CBEIF flag should be tested to ensure that the address, data and command buffers are empty. If so, the command sequence can be started. The following 3-step command write sequence must be strictly adhered to and no intermediate access to the EEPROM array is permitted between the 3 steps. It is possible to read any EEPROM register during a command sequence. The command sequence is as follows:

1. Write the aligned data word to be programmed to the valid EEPROM address space. The address and data will be stored in internal buffers. For program, all address bits are valid. For erase, the value of the data bytes is don't care. For mass erase, the address can be anywhere in the available address space of the array. For sector erase, the address bits[1:0] are ignored.
2. Write the program or erase command to the command buffer. These commands are listed in **Table 4-1**.
3. Clear the CBEIF flag by writing a "1" to it to launch the command. When the CBEIF flag is cleared, the CCIF flag is cleared by hardware indicating that the command was successfully launched. The CBEIF flag will be set again indicating the address, data and command buffers are ready for a new command sequence to begin.

The completion of the command is indicated by the CCIF flag setting. The CCIF flag only sets when all active and pending commands have been completed.

NOTE

The Command State Machine will flag errors in program or erase write sequences by means of the ACCERR (access error) and PVIOL (protection violation) flags in the ESTAT register. An erroneous command write sequence will abort and set the appropriate flag. If set, the user must clear the ACCERR or PVIOL flags before commencing another command write sequence. By writing a 0 to the CBEIF flag the command sequence can be aborted after the word write to the EEPROM address space or after writing a command to the ECMD register and before the command is launched. Writing a "0" to the CBEIF flag in this way will set the ACCERR flag.

A summary of the program algorithm is shown in **Figure 4-2**. For the erase algorithm, the user writes either a mass erase or sector erase command to the ECMD register.

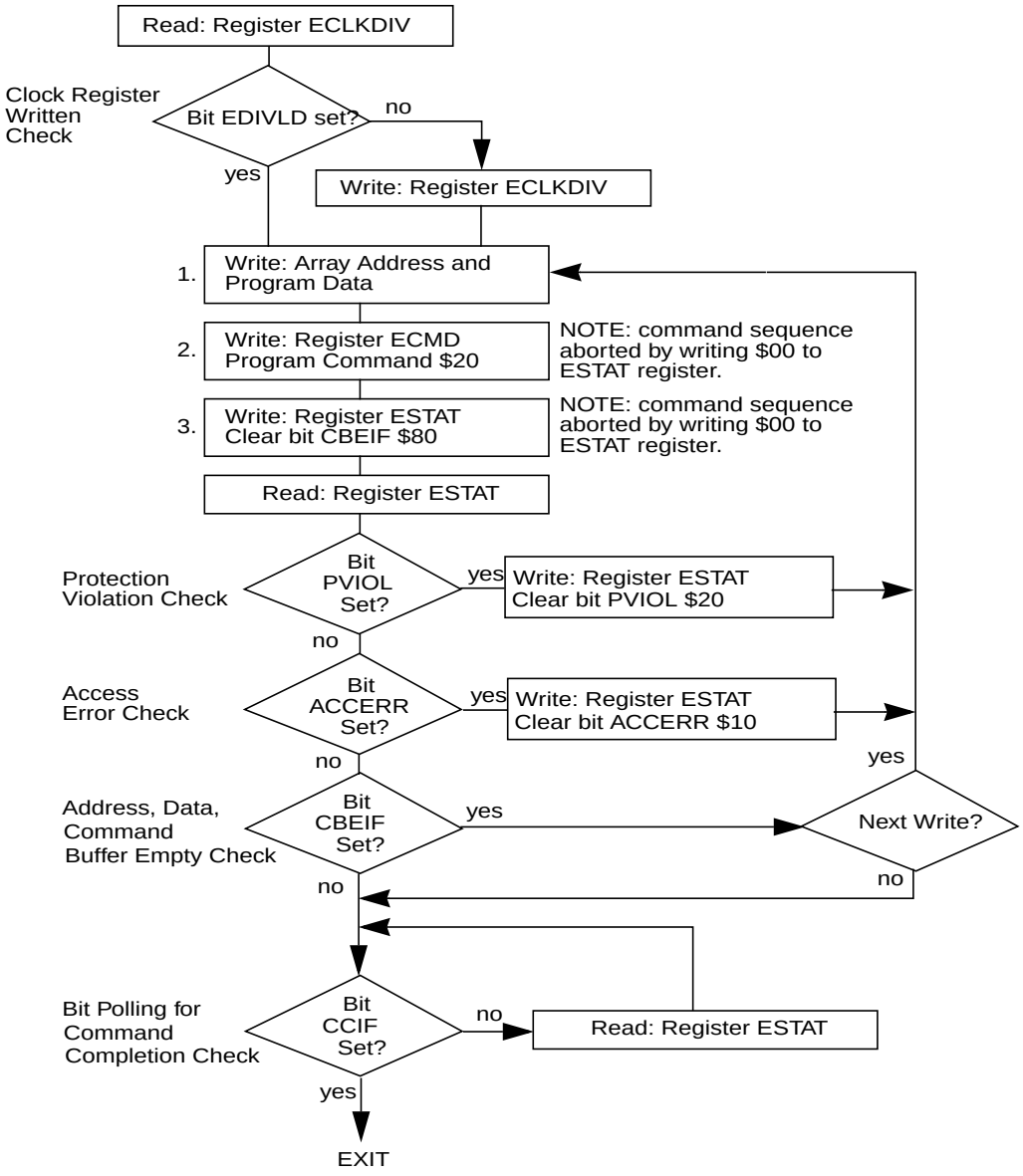


Figure 4-2 Example Program Algorithm

4.1.3 Valid EEPROM Commands

Table 4-1 summarizes the valid EEPROM commands. Also shown are the effects of the commands on the EEPROM array.

Table 4-1 Valid EEPROM Commands

ECMD	Meaning	Function on EEPROM Array
$05	Erase Verify	Verify all memory bytes of the EEPROM array are erased. If the array is erased, the BLANK bit will set in the ESTAT register upon command completion.
$20	Program	Program a word (two bytes).
$40	Sector Erase	Erase two words (four bytes) of EEPROM array.
$41	Mass Erase	Erase all of the EEPROM array. A mass erase of the full array is only possible when EPDIS and EPOPEN are set.
$60	Sector Modify	Erase two words of EEPROM, re-program one word.

WARNING

It is not permitted to program an EEPROM word without first erasing the sector in which that word resides.

The sector modify command executes a two-step algorithm which first erases a sector (2 words) of EEPROM array and then re-programs one of the words in that sector. The EEPROM sector which is erased by the sector modify command is the sector containing the address of the aligned array write which starts the valid command sequence. That same address is re-programmed with the data that was written. By launching a sector modify command and then pipelining a program command, it is possible to completely replace the contents of an EEPROM sector.

4.1.4 Illegal EEPROM Operations

The ACCERR flag will be set during the command write sequence if any of the following illegal operations are performed causing the command write sequence to immediately abort:

1. Writing to the EEPROM address space before initializing ECLKDIV.
2. Writing a misaligned word or a byte to the valid EEPROM address space.
3. Writing to the EEPROM address space while CBEIF is not set.
4. Writing a second word to the EEPROM address space before executing a program or erase command on the previously written word.
5. Writing to any EEPROM register other than ECMD after writing a word to the EEPROM address space.
6. Writing a second command to the ECMD register before executing the previously written

command.

7. Writing an invalid command to the ECMD register in normal mode.
8. Writing to any EEPROM register other than ESTAT (to clear CBEIF) after writing to the command register (ECMD).
9. The part enters STOP mode and a program or erase command is in progress. The command is aborted and any pending command is killed.
10. A “0” is written to the CBEIF bit in the ESTAT register.

The ACCERR flag will not be set if any EEPROM register is read during the command sequence.

If the EEPROM array is read during execution of an algorithm (i.e. CCIF bit in the ESTAT register is low), the read will return non-valid data and the ACCERR flag will not be set.

When an ACCERR flag is set in the ESTAT register, the Command State Machine is locked. It is not possible to launch another command until the ACCERR flag is cleared.

The PVIOL flag will be set during the command write sequence after the word write to the EEPROM address space and the command sequence will be aborted if any of the following illegal operations are performed.

1. Writing a EEPROM address to program in a protected area of the EEPROM.
2. Writing a EEPROM address to erase in a protected area of the EEPROM.
3. Writing the mass erase command to ECMD while any protection is enabled.

When the PVIOL flag is set in the ESTAT register the Command State Machine is locked. It is not possible to launch another command until the PVIOL flag is cleared.

4.2 Wait Mode

When the MCU enters WAIT mode and if any command is active (CCIF=0), that command and any pending command will be completed.

The EETS4K module can recover the MCU from WAIT if the interrupts are enabled (see **Section 6**).

4.3 Stop Mode

If a command is active (CCIF = 0) when the MCU enters the STOP mode, the command will be aborted, and the data being programmed or erased is lost. The high voltage circuitry to the EEPROM array will be switched off when entering STOP mode. CCIF and ACCERR flags will be set. Upon exit from STOP, the CBEIF flag is set and any pending command will not be executed. The ACCERR flag must be cleared before returning to normal operation.

WARNING

As active commands are immediately aborted when the MCU enters STOP mode, it is strongly recommended that the user does not use the STOP command during program and erase execution.

4.4 Background Debug Mode

In Background Debug Mode (BDM), the EPROT register is writable. If the chip is unsecured then all EEPROM commands listed in **Table 4-1** can be executed. In special single chip mode if the chip is secured then the only possible command to execute is mass erase.

Section 5 Resets

5.1 General

If a reset occurs while any command is in progress that command will be immediately aborted. The state of the word being programmed or the sector / block being erased is not guaranteed.

Section 6 Interrupts

6.1 General

The EETS4K block can generate an interrupt when all commands are completed or the address, data and command buffers are empty.

Table 6-1 EEPROM Interrupt Sources

Interrupt Source	Interrupt Flag	Local Enable	Global (CCR) Mask
EEPROM Address, Data and Command Buffers empty	CBEIF (ESTAT register)	CBEIE	I Bit
All Commands are completed on EEPROM	CCIF (ESTAT register)	CCIE	I Bit

NOTE

Vector addresses and their relative interrupt priority are determined at the MCU level.

6.2 Description of Interrupt Operation

For a detailed description of the register bits, refer to the EEPROM Configuration register and EEPROM Status register sections (respectively **3.3.4** and **3.3.6**).

Block Guide End Sheet

FINAL PAGE OF
38
PAGES

FTS512K4
Block User Guide
V01.06

Original Release Date: 08 FEB 2001
Revised: 01 APR 2003

Motorola, Inc

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V01.00	30MAY01	30MAY01		Generated from generic HCS12 block guide(V02.00) Made formats SRS V2 compliant. Remove non-customer information. Reorder and restructure document. Add overview block diagram.
V01.01	19JUL01	19JUL01		Add document names. Hide names and variable definitions.
V01.02	30JAN02	30JAN02		Add description of WRALL bit. Add description of the Address and Data registers. Modify for use of 64Kx16 arrays.
v01.03	23MAR02	23MAR02		Modify FSEC register to include KEYEN[1:0]. Update security restrictions found in **4.5 Flash Security**: (i) $0000 and $FFFF keys are illegal. (ii) No back-to-back writes of keys allowed. (iii) Writing more than 4 keys in a sequence will not unsecure. (iv) Incorrect key sequence results in lock-up with exit by reset only.
V01.04	02AUG02			Modify document number. Fix **Table 3-2** entry for MCU Address Range $C000-$FFFF.
V01.05	02DEC02			Fix bit 15 entry in **Figure 3-10**.
V01.06	01APR03			Fix sector size in **Table 4-1**. Modify description of CBEIF and CCIF flags in **3.3.6 FSTAT — Flash Status Register**. Modify description of **3.3.5 FPROT — Flash Protection Register** to clarify mass erase restrictions.

Table of Contents

Section 5 Resets

Section 6 Interrupts

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

This document describes the FTS512K4 module which is a 512K byte Flash (Non-Volatile) memory. The Flash memory contains 4 blocks of 128K bytes with each block organized as 1024 rows of 128 bytes. The Flash block's erase sector size is 8 rows (1024 bytes).

The Flash memory may be read as either bytes, aligned words or misaligned words. Read access time is one bus cycle for byte and aligned word, and two bus cycles for misaligned words.

Program and erase functions are controlled by a command driven interface. Both sector erase and mass erase of an entire 128K byte Flash block are supported. An erased bit reads '1' and a programmed bit reads '0'. The high voltage required to program and erase is generated internally by on-chip charge pumps.

All Flash blocks can be programmed or erased at the same time. However, it is not possible to read from a Flash block while it is being erased or programmed.

The Flash memory is ideal for program and data storage for single-supply applications allowing for field reprogramming without requiring external programming voltage sources.

WARNING

A word must be erased before being programmed. Cumulative programming of bits within a word is not allowed.

1.1.1 Glossary

Banked Register

A register operating on one Flash block which shares the same register address as the equivalent registers for the other Flash blocks. The active register bank is selected by two bank-select bits in the unbanked register space.

Common Register

A register which operates on all Flash blocks.

Command Sequence

A three-step MCU instruction sequence to program, erase or erase-verify a Flash block.

1.2 Features

- 512K bytes of flash memory comprising four 128k byte blocks.
- Each block in the Flash module can be read, programmed or erased concurrently.

- Automated program and erase algorithm.
- Interrupts on Flash command completion and command buffer empty.
- Fast sector erase and word program operation.
- 2-stage command pipeline.
- Flexible protection scheme for protection against accidental program or erase.
- Single power supply program and erase.
- Security feature.

1.3 Modes of Operation

- Program and erase operation (please refer to **4.1** for details).

1.4 Block Diagram

Figure 1-1 shows a block diagram of the FTS512K4 module.

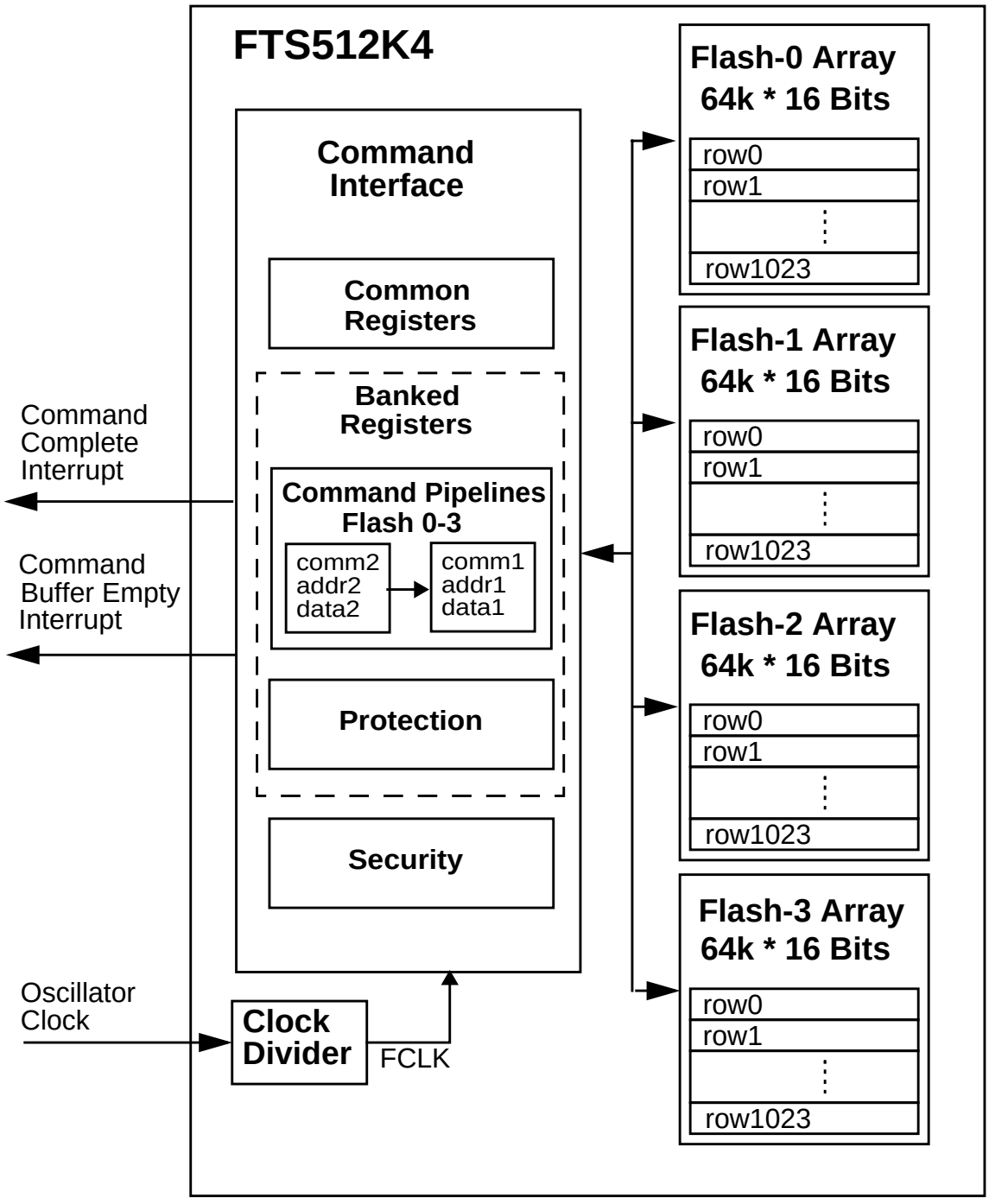


Figure 1-1 Module Block Diagram

Section 2 External Signal Description

2.1 Overview

The FTS512K4 module contains no signals that connect off-chip.

Section 3 Memory Map and Registers

3.1 Overview

This section describes the FTS512K4 memory map and registers

3.2 Modules Memory Map

Figure 3-1shows the FTS512K4 memory map The HCS12 architecture places the Flash array addresses between $4000 and $FFFF, which corresponds to three 16K byte pages. The content of the HCS12 Core PPAGE register is used to map the logical middle page ranging from address $8000 to $BFFF to any physical 16K byte page in the physical memory.[1] Shown within the pages are a protection/options field, described in **Table 3-1**,and user defined Flash protected sectors, described in **Table 3-2**

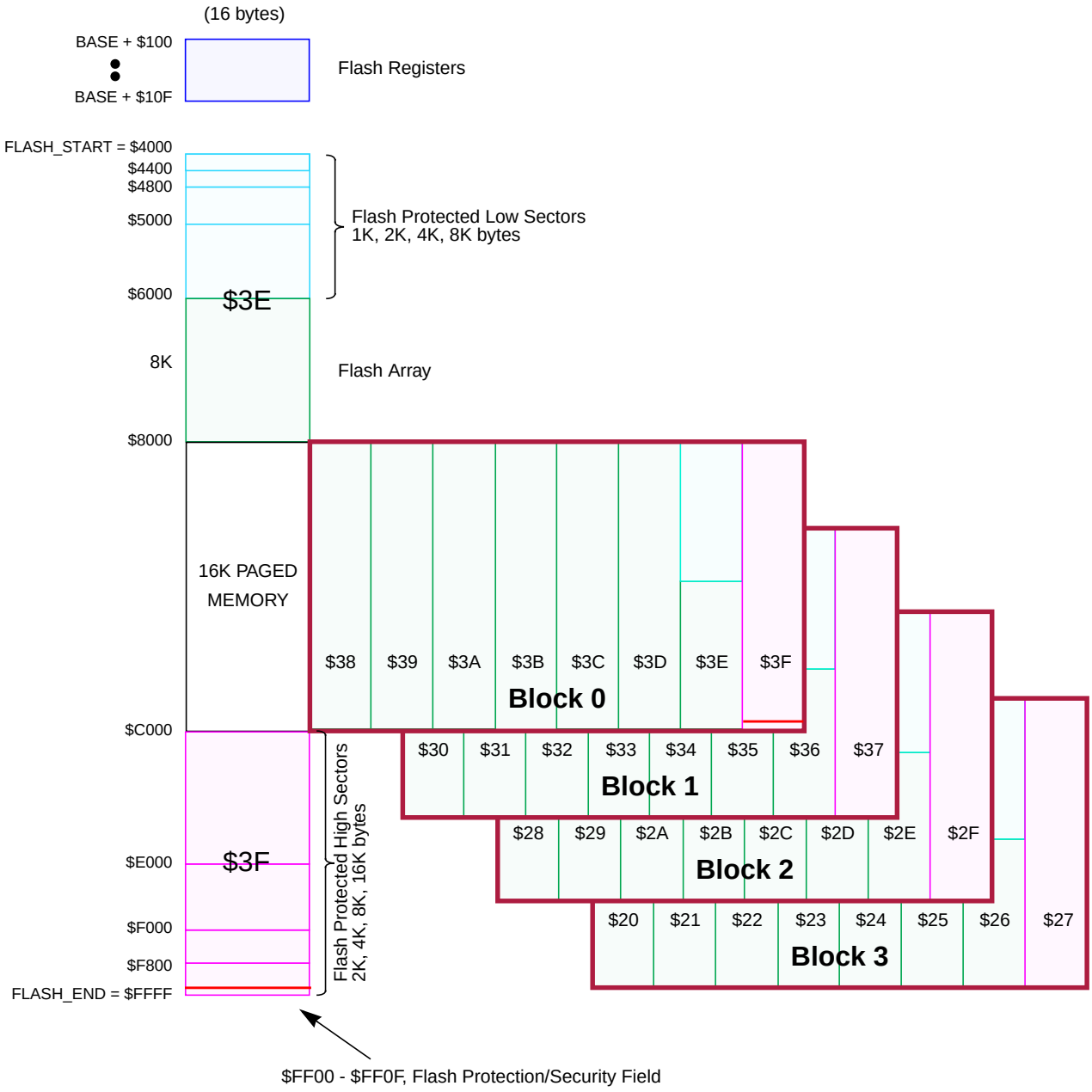
The FPOPEN bit in the FPROT register (see **3.3.5**) can globally protect the entirety of the corresponding Flash block However, for all Flash blocks, two protected areas, one starting from the Flash block starting address (called lower) towards higher addresses and the other one growing downward from the Flash block end address (called higher) can be activated For Flash block 0, the higher page is mainly targeted to hold the boot loader code since it covers the vector space.

Security information that allows the MCU to prevent intrusive access to the Flash module is stored in the Flash Protection/Options field of Flash block 0 as described in **Table 3-1**.

Table 3-1 Flash Protection/Options Field

Array Address	Size (bytes)	Description
$FF00 - $FF07	8	Backdoor Comparison Keys
$FF08 - $FF09	2	Reserved
$FF0A	1	Block 3 Flash Protection byte Refer to Section **3.3.5**
$FF0B	1	Block 2 Flash Protection byte Refer to Section **3.3.5**
$FF0C	1	Block 1 Flash Protection byte Refer to Section **3.3.5**
$FF0D	1	Block 0 Flash Protection byte Refer to Section **3.3.5**
$FF0E	1	Reserved
$FF0F	1	Flash Options/Security byte Refer to Section **3.3.2**

NOTES:
1. By placing $3F or $3E in the PPAGE register, the bottom respectively top “fixed” 16Kbytes pages can be seen twice in the MCU memory map.



Note: $20-$3F correspond to the PPAGE register content

Figure 3-1 Flash Memory Map

Table 3-2 Flash Memory Map Summary

MCU Address Range	PPAGE	Protectable Low Range	Protectable High Range	Flash Block	Block Relative Address[1]
$4000-$7FFF	Unpaged ($3E)	$4000-$43FF $4000-$47FF $4000-$4FFF $4000-$5FFF	N.A.	0	$18000-$1BFFF
$8000-$BFFF	$20	N.A.	N.A.	3	$00000-$03FFF
	$21	N.A.	N.A.		$04000-$07FFF
	$22	N.A.	N.A.		$08000-$0BFFF
	$23	N.A.	N.A.		$0C000-$0FFFF
	$24	N.A.	N.A.		$10000-$13FFF
	$25	N.A.	N.A.		$14000-$17FFF
	$26	$8000-$83FF $8000-$87FF $8000-$8FFF $8000-$9FFF	N.A.		$18000-$1BFFF
	$27	N.A.	$B800-$BFFF $B000-$BFFF $A000-$BFFF $8000-$BFFF		$1C000-$1FFFF
$8000-$BFFF	$28	N.A.	N.A.	2	$00000-$03FFF
	$29	N.A.	N.A.		$04000-$07FFF
	$2A	N.A.	N.A.		$08000-$0BFFF
	$2B	N.A.	N.A.		$0C000-$0FFFF
	$2C	N.A.	N.A.		$10000-$13FFF
	$2D	N.A.	N.A.		$14000-$17FFF
	$2E	$8000-$83FF $8000-$87FF $8000-$8FFF $8000-$9FFF	N.A.		$18000-$1BFFF
	$2F	N.A.	$B800-$BFFF $B000-$BFFF $A000-$BFFF $8000-$BFFF		$1C000-$1FFFF

Table 3-2 Flash Memory Map Summary

MCU Address Range	PPAGE	Protectable Low Range	Protectable High Range	Flash Block	Block Relative Address[1]
$8000-$BFFF	$30	N.A.	N.A.	1	$00000-$03FFF
	$31	N.A.	N.A.		$04000-$07FFF
	$32	N.A.	N.A.		$08000-$0BFFF
	$33	N.A.	N.A.		$0C000-$0FFFF
	$34	N.A.	N.A.		$10000-$13FFF
	$35	N.A.	N.A.		$14000-$17FFF
	$36	$8000-$83FF $8000-$87FF $8000-$8FFF $8000-$9FFF	N.A.		$18000-$1BFFF
	$37	N.A.	$B800-$BFFF $B000-$BFFF $A000-$BFFF $8000-$BFFF		$1C000-$1FFFF
$8000-$BFFF	$38	N.A.	N.A.	0	$00000-$03FFF
	$39	N.A.	N.A.		$04000-$07FFF
	$3A	N.A.	N.A.		$08000-$0BFFF
	$3B	N.A.	N.A.		$0C000-$0FFFF
	$3C	N.A.	N.A.		$10000-$13FFF
	$3D	N.A.	N.A.		$14000-$17FFF
	$3E	$8000-$83FF $8000-$87FF $8000-$8FFF $8000-$9FFF	N.A.		$18000-$1BFFF
	$3F	N.A.	$B800-$BFFF $B000-$BFFF $A000-$BFFF $8000-$BFFF		$1C000-$1FFFF
$C000-$FFFF	Unpaged ($3F)	N.A.	$F800-$FFFF $F000-$FFFF $E000-$FFFF $C000-$FFFF	0	$1C000-$1FFFF

NOTES:
1. Inside each Flash block of size 128 Kbyte.

The Flash module also contains a set of 16 control and status registers located in address space BASE + $100 to BASE + $10F In order to accommodate four Flash blocks with a minimum register address space, a set of registers (BASE+$104 to BASE+$10B) is duplicated in four banks The active bank is selected by the BKSEL bits in the unbanked Flash Configuration Register (FCNFG) A summary of these registers is given in **Table 3-3**.

Table 3-3 Flash Register Memory Map

Address Offset	Use	Access
$_00	Flash Clock Divider Register (FCLKDIV)	R/W
$_01	Flash Security Register (FSEC)	R
$_02	Flash Test Mode Register (FTSTMOD)[1]	R
$_03	Flash Configuration Register (FCNFG)	R/W
$_04	Flash Protection Register (FPROT)	R/W
$_05	Flash Status Register (FSTAT)	R/W
$_06	Flash Command Register (FCMD)	R/W
$_07	RESERVED1[1]	R
$_08	Flash High Address Register (FADDRHI)[1]	R
$_09	Flash Low Address Register (FADDRLO)[1]	R
$_0A	Flash High Data Register (FDATAHI)[1]	R
$_0B	Flash Low Data Register (FDATALO)[1]	R
$_0C	RESERVED2[1]	R
$_0D	RESERVED3[1]	R
$_0E	RESERVED4[1]	R
$_0F	RESERVED5[1]	R

NOTES:
1. Intended for factory test purposes only.

NOTE: *Register Address = Register Base Address + $100 + Address Offset, where the Register Base Address is defined by the HCS12 Core INITRG register and the Address Offset is defined by the Flash module.*

3.3 Register Descriptions

3.3.1 FCLKDIV — Flash Clock Divider Register

The unbanked FCLKDIV register is used to control timed events in program and erase algorithms.

Register address BASE + $100

	7	6	5	4	3	2	1	0
R	FDIVLD	PRDIV8	FDIV5	FDIV4	FDIV3	FDIV2	FDIV1	FDIV0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 Flash Clock Divider Register (FCLKDIV)

All bits in the FCLKDIV register are readable, bits 6-0 are write once and bit 7 is not writable

FDIVLD — Clock Divider Loaded

1 = Register has been written to since the last reset

0 = Register has not been written

PRDIV8 — Enable Prescaler by 8

1 = Enables a prescaler by 8, to divide the Flash module input oscillator clock before feeding into the CLKDIV divider

0 = The input oscillator clock is directly fed into the FCLKDIV divider

FDIV[5:0] — Clock Divider Bits

The combination of PRDIV8 and FDIV[5:0] effectively divides the Flash module input oscillator clock down to a frequency of 150kHz - 200kHz The maximum divide ratio is 512 Please refer to section **4.1.1** for more information

3.3.2 FSEC — Flash Security Register

This unbanked FSEC register holds all bits associated with the security of the MCU.

Register address BASE + $101

	7	6	5	4	3	2	1	0
R	KEYEN1	KEYEN0	NV5	NV4	NV3	NV2	SEC1	SEC0
W								
Reset:	F	F	F	F	F	F	F	F

= Unimplemented or Reserved

Figure 3-3 Flash Security Register (FSEC)

All bits in the FSEC register are readable but not writable.

The FSEC register is loaded from the Flash Protection/Options field byte at $FF0F during the reset sequence, indicated by “F” in **Figure 3-3**

KEYEN[1:0]— Backdoor Key Security Enable Bits.

The KEYEN[1:0] bits define the enabling of the Backdoor Key Access to the Flash module as shown in **Table 3-4**

Table 3-4 Flash KEYEN States

KEYEN[1:0]	Description
00	Backdoor Key Access to Flash module DISABLED
01	Backdoor Key Access to Flash module DISABLED
10	Backdoor Key Access to Flash module **ENABLED**
11	Backdoor Key Access to Flash module DISABLED

NV[5:2] — Non-Volatile Flag Bits

These 4 bits are available to the user as non-volatile flags

SEC[1:0] — Flash Security Bits

The SEC[1:0] bits define the security state of the MCU as shown in **Table 3-5** If the Flash module is unsecured using the Backdoor Key Access, the SEC bits are forced to “10”.

Table 3-5 Flash Security States

SEC[1:0]	Description
00	secured
01	secured
10	unsecured
11	secured

The security function in the Flash module is described in section **4.5**.

3.3.3 FTSTMOD — Flash Test Mode Register

The unbanked FTSTMOD register is used primarily to control the Flash Special modes.

Register address BASE + $102

	7	6	5	4	3	2	1	0
R W	N/A	N/A	N/A	WRALL	0	0	0	N/A
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-4 Flash Test Mode Register (FTSTMOD)

In normal modes, all bits in the FTSTMOD register read zero and are not writable. The WRALL bit is writable only in special modes. The purpose of this bit is to launch a command on all blocks in parallel. This can be useful for mass erase and erase verify operations. All other bits in this register must be written to zero at all times.

WRALL —Write to all register banks.

If this bit is set, all banked registers sharing the same address will be written simultaneously.
1 = Write to all register banks.
0 = Write only to the bank selected via BKSEL.

3.3.4 FCNFG — Flash Configuration Register

The unbanked FCNFG register enables the Flash interrupts, gates the security backdoor writes and selects the register bank to be operated on.

Register address BASE + $103

	7	6	5	4	3	2	1	0
R W	CBEIE	CCIE	KEYACC	0	0	0	BKSEL1	BKSEL0
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-5 Flash Configuration Register (FCNFG)

CBEIE, CCIE, KEYACC, BKSEL1 and BKSEL0 are readable and writable. Bits 4-2 read zero and are not writable.

CBEIE — Command Buffer Empty Interrupt Enable.

The CBEIE bit enables the interrupts in case of an empty command buffer in the Flash module.
1 = An interrupt will be requested whenever the CBEIF flag, **Figure 3-7**, is set.
0 = Command Buffer Empty interrupts disabled.

CCIE — Command Complete Interrupt Enable.

The CCIE bit enables the interrupts in case of all commands being completed in the Flash module.

1 = An interrupt will be requested whenever the CCIF, **Figure 3-7**, flag is set.

0 = Command Complete interrupts disabled.

KEYACC — Enable Security Key Writing.

1 = Writes to Flash array are interpreted as keys to open the backdoor. Reads of the Flash array return invalid data.

0 = Flash module writes are interpreted as the start of a program or erase sequence.

BKSEL[1:0] — Register Bank Select.

These bits are used to select one of the four register banks. The register bank associated with Flash block 0 is the default out of reset. The bank selection is according to **Table 3-6**.

Table 3-6 Flash Register Bank Selects

BKSEL[1:0]	Selected Register Bank
00	Flash 0
01	Flash 1
10	Flash 2
11	Flash 3

3.3.5 FPROT — Flash Protection Register

The banked FPROT register defines which Flash sectors are protected against program or erase.

Register address BASE + $104

	7	6	5	4	3	2	1	0
R W	FPOPEN	NV6	FPHDIS	FPHS1	FPHS0	FPLDIS	FPLS1	FPLS0
Reset:	F	F	F	F	F	F	F	F

= Unimplemented or Reserved

Figure 3-6 Flash Protection Register (FPROT)

The FPROT register is readable in normal and special modes. Bit NV6 is not writable. FPOPEN, FPHDIS and FPLDIS bits in the FPROT register can only be written to the protected state (i.e. 0). FPLS[1:0] can be written anytime until bit FPLDIS is cleared. FPHS[1:0] bits can be written anytime until bit FPHDIS is

cleared. If the FPOPEN bit is cleared, then the state of the FPHDIS, FPHS[1:0], FPLDIS and FPLS[1:0] bits is irrelevant. The FPROT register is loaded from Flash block 0 during reset as shown in **Table 3-7**.

Table 3-7 Loading of the Protection Register from Flash

Flash Address	Protection byte for
$FF0D	Flash 0
$FF0C	Flash 1
$FF0B	Flash 2
$FF0A	Flash 3

To change the Flash protection that will be loaded on reset, the upper sector of Flash block 0 must be unprotected, then the Flash Protect/Security byte located as described in **Table 3-1** must be written.

A protected Flash sector is disabled by the bits FPHDIS and FPLDIS while the size of the protected sector is defined by FPHS[1:0] and FPLS[1:0] in the FPROT register.

Trying to alter any of the protected areas will result in a protect violation error and bit PVIOL will be set in the Flash Status Register (FSTAT). A mass erase of a whole Flash block is only possible when protection is fully disabled by setting the FPOPEN, FPLDIS, and FPHDIS bits. An attempt to mass erase a Flash block while protection is enabled in that block will set the PVIOL bit in the FSTAT register.

FPOPEN — Opens the Flash array for program or erase.

1 = The Flash sectors not protected are enabled for program or erase.

0 = The whole Flash array is protected. In this case the FPHDIS, FPHS[1:0], FPLDIS and FPLS[1:0] bits within the protection register are ignored.

FPHDIS — Flash Protection Higher address range Disable.

The FPHDIS bit determines whether there is a protected area in the higher space of the Flash block.

1 = Protection disabled.

0 = Protection enabled.

FPHS[1:0] — Flash Protection Higher Address Size.

The FPHS[1:0] bits determine the size of the protected sector. Refer to **Table 3-8**.

Table 3-8 Flash Higher Address Range Protection

FPHS[1:0]	Protected Address Range	Protected Size
00	see **Table 3-2**	2K bytes
01		4K
10		8K
11		16K

FPLDIS — Flash Protection Lower address range Disable.
The FPLDIS bit determines whether there is a protected sector in the lower space of the Flash block.

1 = Protection disabled.
0 = Protection enabled.

FPLS[1:0] — Flash Protection Lower Address Size.
The FPLS[1:0] bits determine the size of the protected sector. Refer to **Table 3-9**.

Table 3-9 Flash Lower Address Range Protection

FPLS[1:0]	Protected Address Range	Protected Size
00	see **Table 3-2**	1K Bytes
01		2K
10		4K
11		8K

NV6 — Non-Volatile Flag Bit.

The NV6 bit should remain in the erased state “1” for future enhancements.

3.3.6 FSTAT — Flash Status Register

The banked FSTAT register defines the Flash state machine command status and Flash array access, protection and erase verify status.

Register address BASE + $105

	7	6	5	4	3	2	1	0
R	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
W								
Reset:	1	1	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-7 Flash Status Register (FSTAT)

Register bits CBEIF, PVIOL and ACCERR are readable and writable, bits CCIF and BLANK are readable and not writable, bits 3, 1 and 0 read zero and are not writable.

CBEIF — Command Buffer Empty Interrupt Flag.

The CBEIF flag indicates that the address, data and command buffers are empty so that a new command sequence can be started. The CBEIF flag is cleared by writing a “1” to CBEIF. Writing a "0" to the CBEIF flag has no effect on CBEIF. Writing a "0" to CBEIF after writing an aligned word to the Flash address space but before CBEIF is cleared will abort a command sequence and cause the

ACCERR flag in the FSTAT register to be set. Writing a "0" to CBEIF outside of a command sequence will not set the ACCERR flag. The CBEIF flag is used together with the CBEIE bit in the FCNFG register to generate an interrupt request (see also **Figure 6-1**).

1 = Buffers are ready to accept a new command.
0 = Buffers are full.

CCIF — Command Complete Interrupt Flag.

The CCIF flag indicates that there are no more commands pending. The CCIF flag is cleared when CBEIF is clear and sets automatically upon completion of all active and pending commands. The CCIF flag does not set when an active commands completes and a pending command is fetched from the command buffer. Writing to the CCIF flag has no effect. The CCIF flag is used together with the CCIE bit in the FCNFG register to generate an interrupt request (see also **Figure 6-1**).

1 = All commands are completed.
0 = Command in progress.

PVIOL — Protection Violation.

The PVIOL flag indicates an attempt was made to program or erase an address in a protected Flash memory area. The PVIOL flag is cleared by writing a “1” to PVIOL. Writing a “0” to the PVIOL flag has no effect on PVIOL. While PVIOL is set in any of the FSTAT registers, it is not possible to launch another command in any of the Flash blocks.

1 = A protection violation has occurred.
0 = No failure.

ACCERR — Flash Access Error.

The ACCERR flag indicates an illegal access to the selected Flash block caused by either a violation of the command sequence, issuing an illegal command (illegal combination of the CMDBx bits in the FCMD register) or the execution of a CPU STOP instruction while a command is executing (CCIF=0). The ACCERR flag is cleared by writing a “1” to ACCERR. Writing a “0” to the ACCERR flag has no effect on ACCERR. While ACCERR is set in any of the FSTAT registers, it is not possible to launch another command in any of the Flash blocks.

1 = Access error has occurred.
0 = No failure.

BLANK — Array has been verified as erased.

The BLANK flag indicates that an erase verify command has checked the Flash block and found it to be erased. The BLANK flag is cleared by hardware when CBEIF is cleared as part of a new valid command sequence. Writing to the BLANK flag has no effect on BLANK.

1 = Flash block verifies as erased.
0 = If an erase verify command has been requested, and the CCIF flag is set, then a zero in BLANK indicates the block is not erased.

3.3.7 FCMD — Flash Command Register

The banked FCMD register defines the Flash commands.

Register address BASE + $106

	7	6	5	4	3	2	1	0
R	0	CMDB6	CMDB5	0	0	CMDB2	0	CMDB0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-8 Flash Command Buffer and Register (FCMD)

Bits 7, 4, 3 and 1 read zero and are not writable. Bits CMDB6, CMDB5, CMDB2 and CMDB0 are readable and writable during a command sequence.

CMDB — Valid normal mode commands are shown in **Table 3-10**. Any commands other than those mentioned in **Table 3-10** sets the ACCERR bit in the FSTAT register (**3.3.6**).

Table 3-10 Flash Normal Mode Commands

Command	Meaning
$05	Erase Verify
$20	Word Program
$40	Sector Erase
$41	Mass Erase

3.3.8 RESERVED1

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $107

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-9 RESERVED1

All bits read zero and are not writable.

3.3.9 FADDR — Flash Address Register

FADDRHI and FADDRLO are the banked Flash address registers.

Register address Base + $108

	15	14	13	12	11	10	9	8
R W	FABHI							
Reset:	0	0	0	0	0	0	0	0

Figure 3-10 Flash Address High Register (FADDRHI)

Register address Base + $109

	15	14	13	12	11	10	9	8
R W	FABLO							
Reset:	0	0	0	0	0	0	0	0

Figure 3-11 Flash Address Low Register (FADDRLO)

In normal modes, the FADDR (FADDRHI, FADDRLO) register reads zeros and is not writable.

The FADDRHI and FADDRLO registers can be written in special modes by writing to address BASE + $108 and BASE + $109 in the register space.

For sector erase, the MCU address bits AB[9:0] are ignored.

For mass erase, any address within the block is valid to start the command.

3.3.10 FDATA — Flash Data Register

FDATAHI and FDATALO are the banked Flash data registers.

Register address BASE + $10A

	7	6	5	4	3	2	1	0
R W	FDHI							
Reset:	0	0	0	0	0	0	0	0

Figure 3-12 Flash Data High Register (FDATAHI)

Register address BASE + $10B

	7	6	5	4	3	2	1	0
R W	FDLO							
Reset:	0	0	0	0	0	0	0	0

Figure 3-13 Flash Data Low Register (FDATALO)

In normal modes, all FDATA bits read zero and are not writable.

In special modes, all FDATA bits are readable and writable when writing to an address within the Flash address range.

3.3.11 RESERVED2

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $10C

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-14 RESERVED2

All bits read zero and are not writable.

3.3.12 RESERVED3

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $10D

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-15 RESERVED3

All bits read zero and are not writable.

3.3.13 RESERVED4

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $10E

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-16 RESERVED4

All bits read zero and are not writable.

3.3.14 RESERVED5

This register is reserved for factory testing and is not accessible to the user.

Register address BASE + $10F

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-17 RESERVED5

All bits read zero and are not writable.

Section 4 Functional Description

4.1 Program and Erase Operation

Write and read operations are both used for the program and erase algorithms described in this section. These algorithms are controlled by a state machine whose timebase FCLK is derived from the oscillator clock via a programmable divider. The command register as well as the associated address and data registers operate as a buffer and a register (2-stage FIFO) so that a new command along with the necessary data and address can be stored to the buffer while the previous command is still in progress. This pipelined operation allows a time optimization when programming more than one word on a specific row, as the high voltage generation can be kept ON in between two programming commands. The pipelined operation also allows a simplification of command launching. Buffer empty as well as command completion are signalled by flags in the Flash status register. Interrupts for the Flash will be generated if enabled.

The next four subsections describe:

- How to write the FCLKDIV register.
- The write sequences used to program, erase and erase-verify the Flash.
- Valid Flash commands.
- Errors resulting from illegal Flash operations.

4.1.1 Writing the FCLKDIV Register

Prior to issuing any program or erase command, it is first necessary to write the FCLKDIV register to divide the oscillator down to within the 150kHz to 200kHz range. The program and erase timings are also a function of the bus clock, such that the FCLKDIV determination must take this information into account. If we define:

- FCLK as the clock of the Flash timing control block
- Tbus as the period of the bus clock
- INT(x) as taking the integer part of x (e.g. INT(4.323)=4),

then FCLKDIV register bits PRDIV8 and FDIV[5:0] are to be set as described in **Figure 4-1**.

For example, if the oscillator clock frequency is 4Mz and the bus clock is 25MHz, FCLKDIV bits FDIV[5:0] should be set to 20 (010100) and bit PRDIV8 set to 0. The resulting FCLK is then 190kHz. As a result, the Flash algorithm timings are increased over optimum target by:

$$(200 - 190)/200 \times 100 = 5\%$$

NOTE

Command execution time will increase proportionally with the period of FCLK.

WARNING

Because of the impact of clock synchronization on the accuracy of the functional timings, programming or erasing the Flash cannot be performed if the bus clock runs at less than 1 MHz. Programming or erasing the Flash with an input clock < 150kHz should be avoided. Setting FCLKDIV to a value such that FCLK < 150kHz can destroy the Flash due to overstress. Setting FCLKDIV to a value such that (1/FCLK+Tbus) < 5μs can result in incomplete programming or erasure of the memory array cells.

If the FCLKDIV register is written, the bit FDIVLD is set automatically. If this bit is zero, the register has not been written since the last reset. Program and erase commands will not be executed if this register has not been written to.

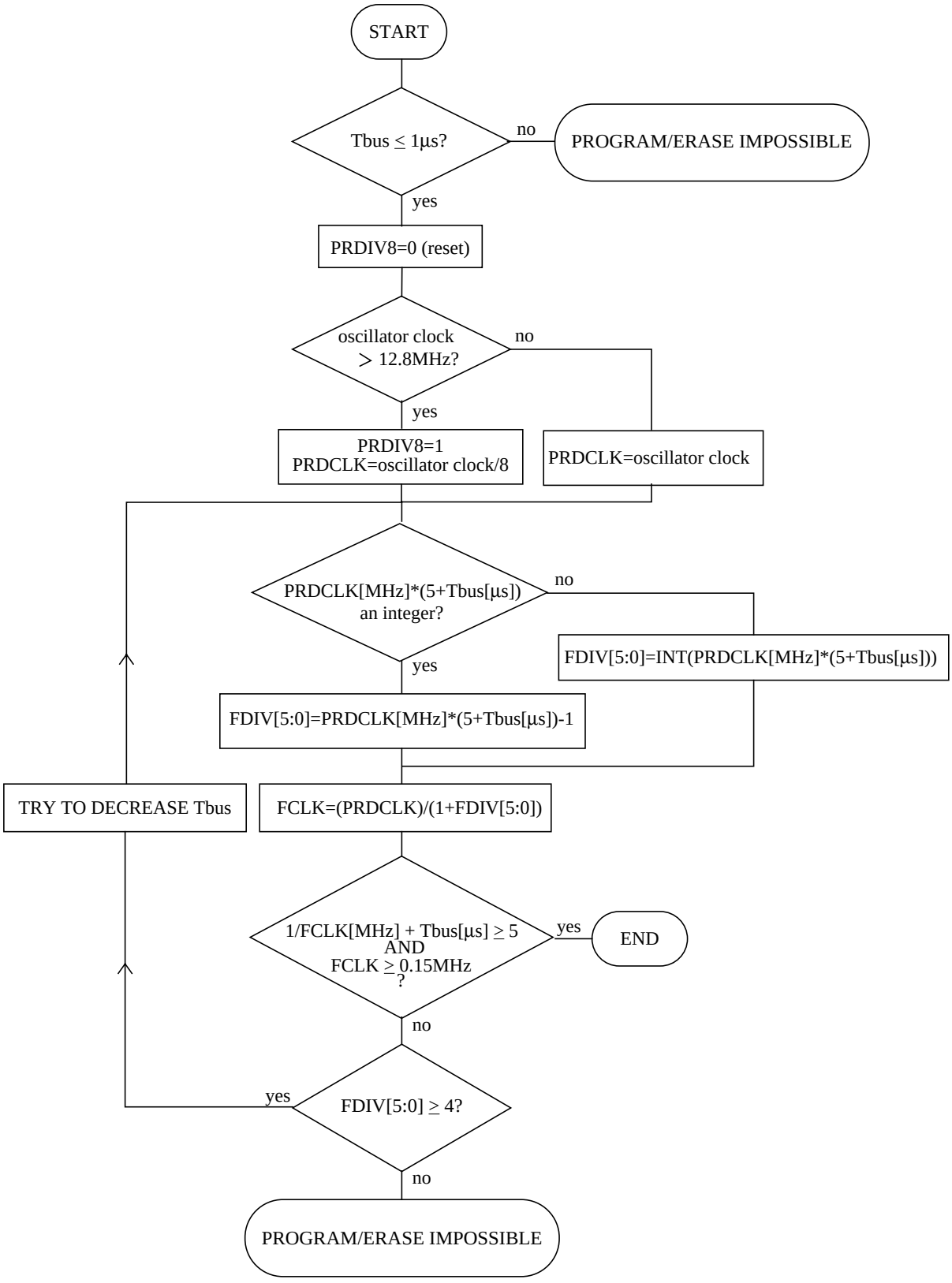


Figure 4-1 PRDIV8 and FDIV bits Determination Procedure

4.1.2 Program and Erase Sequences in Normal Mode

A Command State Machine is used to supervise the write sequencing for program and erase. The erase verify command follows the same flow. Before starting a command sequence, it is required that there are no pending access error or protection violations in any of the Flash blocks (the ACCERR and PVIOL flags should be cleared in the FSTAT registers). It is also required that the Flash FCNFG register and the HCS12 Core PPAGE register are set to select the Flash array address space to operate on. This initialization procedure is as follows:

1. Verify that the ACCERR and PVIOL flags in the FSTAT register are cleared in all banks. This requires a check of the FSTAT content for all conditions of the BKSEL bits in the FCNFG register.
2. Write to the BKSEL bit in the FCNFG register to select the bank of registers corresponding to the Flash block to be programmed or erased (see **Table 3-6**).
3. Write to the HCS12 Core PPAGE register ($x030) to select one of the 16K byte pages to be programmed, if programming in the $8000-$BFFF address range. There is no need to set the PPAGE register when programming in the $4000-$7FFF or $C000-$FFFF address ranges.

After this optional initialization step, the CBEIF flag should be tested to ensure that the address, data and command buffers are empty. If the CBEIF flag is set, the program/erase command write sequence can be started. The following 3-step command write sequence must be strictly adhered to and no intermediate writes to the Flash module are permitted between the steps. However, the user is allowed to read any Flash register during a command write sequence. The command write sequence is as follows:

1. Write the aligned data word to be programmed to the valid Flash address space. The address and data will be stored in internal buffers. For program, all address bits are valid. For erase, the value of the data bytes is ignored. For mass erase, the address can be anywhere in the available address space of the block to be erased. For sector erase, the address bits[9:0] are ignored.
2. Write the program or erase command to the command buffer. These commands are listed in **Table 4-1**.
3. Clear the CBEIF flag by writing a “1” to it to launch the command. When the CBEIF flag is cleared, the CCIF flag is cleared by hardware indicating that the command was successfully launched. The CBEIF flag will be set again indicating the address, data and command buffers are ready for a new command sequence to begin.

The completion of the command is indicated by the setting of the CCIF flag. The CCIF flag only sets when all active and pending commands have been completed.

NOTE

The Command State Machine will flag errors in program or erase write sequences by means of the ACCERR (access error) and PVIOL (protection violation) flags in the FSTAT register. An erroneous command write sequence will abort and set the appropriate flag. If set, the user must clear the ACCERR or PVIOL flags before commencing another command write sequence. By writing a “0” to the CBEIF flag, the command sequence can be aborted after the word write to the Flash address space or after writing a

command to the FCMD register and before the command is launched. Writing a “0” to the CBEIF flag in this way will set the ACCERR flag.

A summary of the program algorithm is shown in **Figure 4-2**. For the erase algorithm, the user writes either a mass or sector erase command to the FCMD register.

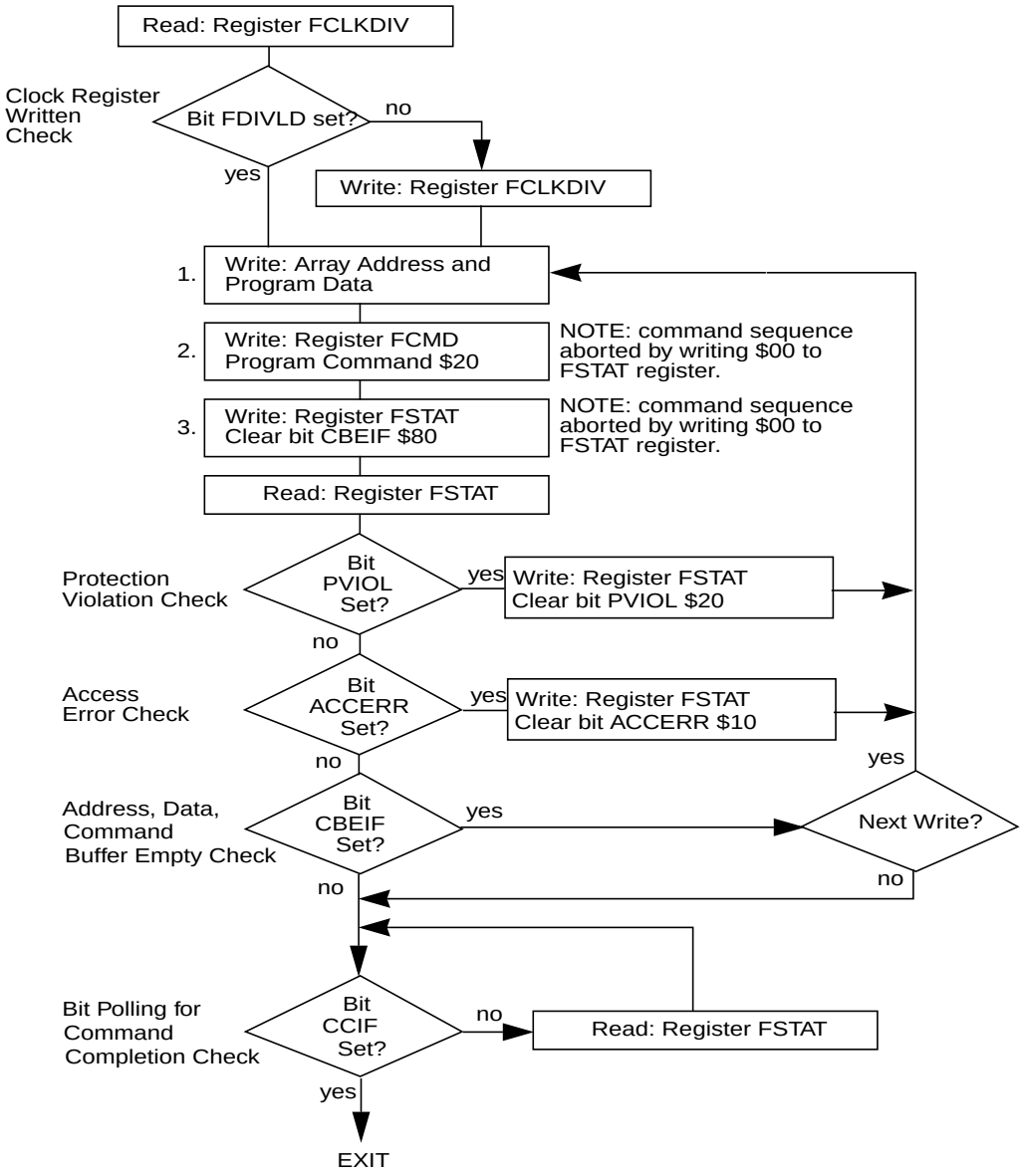


Figure 4-2 Example Program Algorithm

4.1.3 Valid Flash Commands

Table 4-1 summarizes the valid Flash commands. Also shown are the effects of the commands on the Flash array.

Table 4-1 Valid Flash Commands

FCMD	Meaning	Function on Flash Array
$05	Erase Verify	Verify all memory bytes of the Flash block are erased. If the block is erased, the BLANK bit will set in the FSTAT register upon command completion.
$20	Program	Program a word (two bytes).
$40	Sector Erase	Erase 512 words of Flash.
$41	Mass Erase	Erase all of the Flash block. A mass erase of the full block is only possible when FPLDIS, FPHDIS and FPOPEN are set.

WARNING

It is not permitted to program a Flash word without first erasing the sector in which that word resides.

4.1.4 Illegal Flash Operations

The ACCERR flag will be set during the command write sequence if any of the following illegal operations are performed causing the command write sequence to immediately abort:

1. Writing to the Flash address space before initializing FCLKDIV.
2. Writing to the Flash address space in the range $8000-$BFFF when the HCS12 Core PPAGE register does not select a 16K byte page in the Flash block selected by the BKSEL bit in the FCNFG register.
3. Writing to the Flash address space $4000-$7FFF or $C000-$FFFF with the BKSEL bits in the FCNFG register not selecting Flash block 0.
4. Writing a misaligned word or a byte to the valid Flash address space.
5. Writing to the Flash address space while CBEIF is not set.
6. Writing a second word to the Flash address space before executing a program or erase command on the previously written word.
7. Writing to any Flash register other than FCMD after writing a word to the Flash address space.
8. Writing a second command to the FCMD register before executing the previously written command.
9. Writing an invalid command to the FCMD register.
10. Writing to any Flash register other than FSTAT (to clear CBEIF) after writing to the command

register (FCMD).

11. The part enters STOP mode and a program or erase command is in progress. The command is aborted and any pending command is killed.
12. When security is enabled, a command other than mass erase originating from a non-secure memory or from the Background Debug Mode is written to FCMD.
13. A “0” is written to the CBEIF bit in the FSTAT register.

The ACCERR flag will not be set if any Flash register is read during the command sequence.

If the Flash array is read during execution of an algorithm (i.e. CCIF bit in the FSTAT register is low), the read will return non-valid data and the ACCERR flag will not be set.

If an ACCERR flag is set in either of the FSTAT registers, the Command State Machine is locked. It is not possible to launch another command on any block until the ACCERR flag is cleared.

The PVIOL flag will be set during the command write sequence after the word write to the Flash address space if any of the following illegal operations are performed, causing the command sequence to immediately abort:

1. Writing a Flash address to program in a protected area of the Flash block.
2. Writing a Flash address to erase in a protected area of the Flash block.
3. Writing the mass erase command to FCMD while any protection is enabled. See Protection register description in **3.3.5**.

If a PVIOL flag is set in any of the FSTAT registers, the Command State Machine is locked. It is not possible to launch another command on any block until the PVIOL flag is cleared.

4.2 Wait Mode

When the MCU enters WAIT mode and if any command is active (CCIF=0), that command and any pending command will be completed.

The FTS512K4 module can recover the part from WAIT if the interrupts are enabled (see **Section 6**).

4.3 Stop Mode

If a command is active (CCIF = 0) when the MCU enters the STOP mode, the command will be aborted and the data being programmed or erased is lost. The high voltage circuitry to the Flash block will be switched off when entering STOP mode. CCIF and ACCERR flags will be set. If commands are active in multiple blocks when STOP occurs, then all the corresponding CCIF and ACCERR flags will be set. Upon exit from STOP, the CBEIF flag is set and any pending command will not be executed. All ACCERR flags must be cleared before returning to normal operation.

WARNING

As active commands are immediately aborted when the MCU enters STOP mode, it is strongly recommended that the user does not use the STOP command during program and erase execution.

4.4 Background Debug Mode

In Background Debug Mode (BDM), the FPROT registers are writable. If the MCU is unsecured, then all Flash commands listed in **Table 4-1** can be executed. If the MCU is secured and is in Special Single Chip mode, the only possible command to execute is mass erase.

4.5 Flash Security

The Flash module provides the necessary security information to the MCU. After each reset, the Flash module determines the security state of the MCU as defined in section **3.3.2**.

The contents of the Flash Protection/Options byte at $FF0F in the Flash Protection/Options Field must be changed directly by programming $FF0F when the device is unsecured and the higher address sector is unprotected. If the Flash Protection/Options byte is left in the secure state, any reset will cause the MCU to return to the secure operating mode.

4.5.1 Unsecuring via the Backdoor Key Access

The MCU may only be unsecured by using the Backdoor Key Access feature which requires knowledge of the contents of the Backdoor Keys (four 16-bit words programmed at addresses $FF00 - $FF07). If KEYEN[1:0]=10 and the KEYACC bit is set, a write to a Backdoor Key address in the Flash array triggers a comparison between the written data and the Backdoor Key data stored in the Flash array. If all four words of data are written to the correct addresses in the correct order and the data matches the Backdoor Keys stored in the Flash array, the MCU will be unsecured. The data must be written to the Backdoor Keys sequentially staring with $FF00-1 and ending with $FF06-7. $0000 and $FFFF keys are not permitted. When the KEYACC bit is set, reads of the Flash array will return invalid data.

The user code stored in the Flash array must have a method of receiving the Backdoor Key from an external stimulus. This external stimulus would typically be through one of the on-chip serial ports.

If KEYEN[1:0]=10 in the FSEC register, the MCU can be unsecured by the Backdoor Access Sequence described below:

1. Set the KEYACC bit in the Flash Configuration Register (FCNFG).
2. Write the correct four 16-bit words to Flash addresses $FF00 - $FF07 sequentially starting with $FF00.
3. Clear the KEYACC bit.
4. If all four 16-bit words match the Backdoor Keys stored in Flash addresses $FF00 - $FF07, the MCU is unsecured and bits SEC[1:0] in the FSEC register are forced to the unsecure state of “10”.

The Backdoor Access Sequence is monitored by the internal Security State Machine. An illegal operation during the Backdoor Access Sequence will cause the Security State Machine to lock, leaving the MCU in the secured state. A reset of the MCU will cause the Security State Machine to exit the lock state and allow a new Backdoor Access Sequence to be attempted. The following illegal operations will lock the Security State Machine:

1. If any of the four 16-bit words does not match the backdoor keys programmed in the Flash array.
2. If the four 16-bit words are written in the wrong sequence.
3. If more than four 16-bit words are written.
4. If any of the four 16-bit words written are $0000 or $FFFF.
5. If the KEYACC bit does not remain set while the four 16-bit words are written.

After the Backdoor Access Sequence has been correctly matched, the MCU will be unsecured. The Flash security byte can be programmed to the unsecure state, if desired.

In the unsecure state, the user has full control of the contents of the four word Backdoor Key by programming it in bytes $FF00 - $FF07 of the Flash Protection/Options Field.

The security as defined in the Flash Security/Options byte ($FF0F) is not changed by using the Backdoor Access Sequence to unsecure. The Backdoor Keys stored in addresses $FF00 - $FF07 are unaffected by the Backdoor Access Sequence. After the next reset sequence, the security state of the Flash module is determined by the Flash Security/Options byte ($FF0F). The Backdoor Access Sequence has no effect on the program and erase protections defined in the Flash Protection Register (FPROT).

It is not possible to unsecure the MCU in Special Single Chip mode by the Backdoor Access Sequence via the Background Debug Mode.

Section 5 Resets

5.1 General

If a reset occurs while any command is in progress that command will be immediately aborted. The state of the word being programmed or the sector / block being erased is not guaranteed.

Section 6 Interrupts

6.1 General

The FTS512K4 module can generate an interrupt when all Flash commands are completed or the address, data and command buffers are empty.

Table 6-1 Flash Interrupt Sources

Interrupt Source	Interrupt Flag	Local Enable	Global (CCR) Mask
Flash Address, Data and Command Buffers empty	CBEIF (FSTAT from any Flash block)	CBEIE	I Bit
All Commands are completed on Flash	CCIF (FSTAT from any Flash block)	CCIE	I Bit

NOTE

Vector addresses and their relative interrupt priority are determined at the MCU level

6.2 Description of Interrupt Operation

Figure 6-1 shows the logic used for generating interrupt via the relevant block.

This system uses the CBEIF and CCIF flags in combination with the enable bits CBIE and CCIE in addition to the BKSEL bits) to discriminate for the interrupt generation. By taking account of the possible selected bank, the system is prevented from generating false interrupts when the command buffer is empty in an unselected bank.

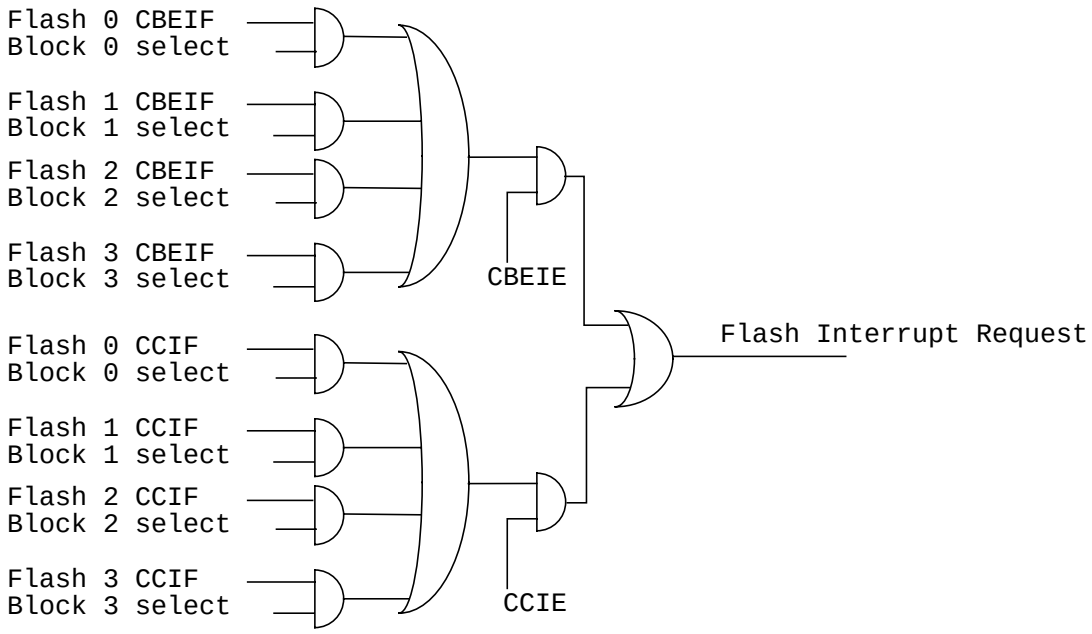


Figure 6-1 Flash Interrupt Implementation

For a detailed description of the register bits, refer to the Flash Configuration register and Flash Status register sections (respectively **3.3.4** and **3.3.6**).

Block Guide End Sheet

FINAL PAGE OF
48
PAGES

HCS12 Inter-Integrated Circuit(IIC) Block Guide V02.08

Original Release Date: 08 SEP 1999
Revised: Jun 3, 2004

8/16 Bit Division,TSPG
Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
0.1	8-Sep-99		Vipin Agrawal, Puneet Goel	Original draft. Distributed only within Motorola
0.2	30-Sep-99		Puneet Goel	Minor corrections as suggested by Joachim Kruecken.
2.0	12-Feb-01		Gautam Kar, Gurdarshan Kalra	Reformatted for SRS v2.0
2.1	2-Mar-2001		Gurdarshan Kalra	Minor corrections as suggested by Jens Winkler
2.2	6-Mar-2001		Gurdarshan Kalra	Minor corrections as suggested by Jens Winkler
2.03	26-Mar-2001		Gurdarshan Kalra Jens Winkler	Minor updates in format
2.04	19-July-2001		Dirk Rowald	Document names have been added, Names and variable definitions have been hidden
2.05	7-Mar-2002		Stephen Zhou	Minor updates in format
2.06	18-Aug-2002		Stephen Zhou	Reformated for SRS3.0,and add examples for programing general use and some diagrams to make it more user friendly as suggested by Joachim
2.07	11-Apr-2003		Stephen Zhou	Clearly claim support 400kps; Add notes for TCF bit in Section 5.1.3 Correct Section 7 for IBIF is cleared by writing ‘1’
2.08	3-Jun-2004		Vicers Cai	Correct the wrong divider values for SDA Hold from IBC=$60 to IBC=$7F

Table of Contents

Section 5 Initialization/Application Information

Section 6 Resets

Section 7 Interrupts

List of Figures

List of Tables

Preface

N/A.

Section 1 Introduction

1.1 Overview

The Inter-IC Bus (IIC or I2C) is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices Being a two-wire device, the IIC Bus minimizes the need for large numbers of connections between devices, and eliminates the need for an address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface will operate at baud rates of up to 100kbps with maximum capacitive bus loading.With reduced bus slew rate, the device is capable of operating at higher baud rates, up to a maximum of [MCUbus]clock/20. The module can operate up to a baud rate of 400kbps provided the IIC bus slew rate is less than 100ns. The maximum communication interconnect length and the number of devices that can be connected to the bus are limited by a maximum bus capacitance of 400pF in all instances.

1.2 Features

The IIC module has the following key features:

- Compatible with I2C Bus standard
- Multi-master operation
- Software programmable for one of 256 different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection

1.3 Modes of Operation

The IIC functions the same in normal, special, and emulation modes. It has two low power modes, wait and stop modes.

- Run Mode

  This is the basic mode of operation.

- Wait Mode

  IIC operation in wait mode can be configured. Depending on the state of internal bits, the IIC can operate normally when the CPU is in wait mode or the IIC clock generation can be turned off and the IIC module enters a power conservation state during wait mode. In the latter case, any transmission or reception in progress stops at wait mode entry.

- Stop Mode

  The IIC is inactive in stop mode for reduced power consumption. The STOP instruction does not affect IIC register states.

1.4 Block Diagram

The block diagram of the IIC module is shown in **Figure 1-1**

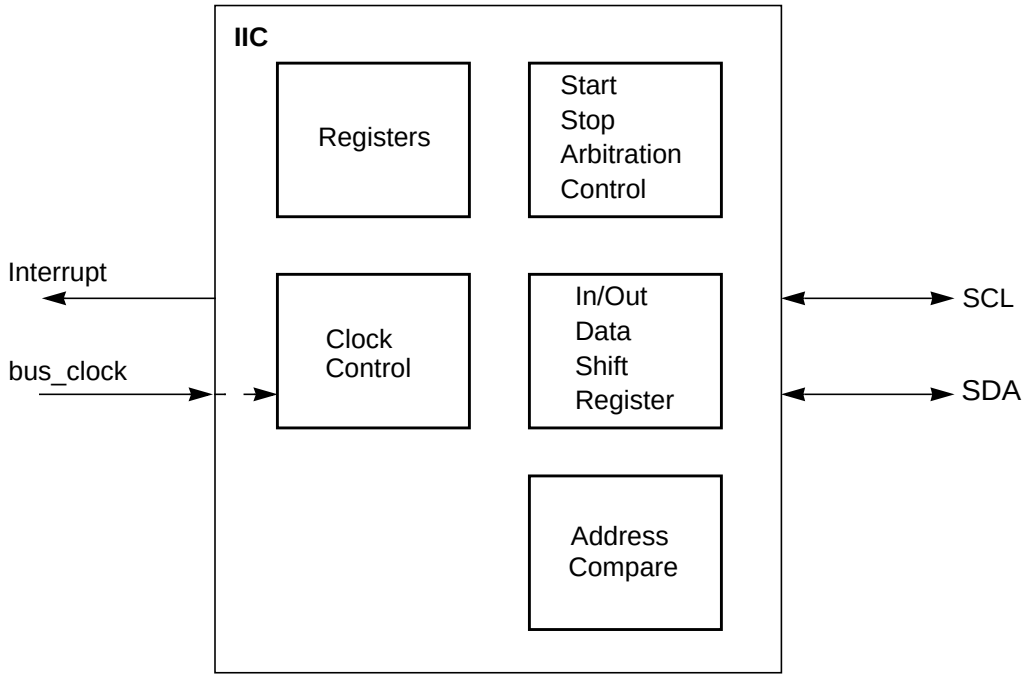
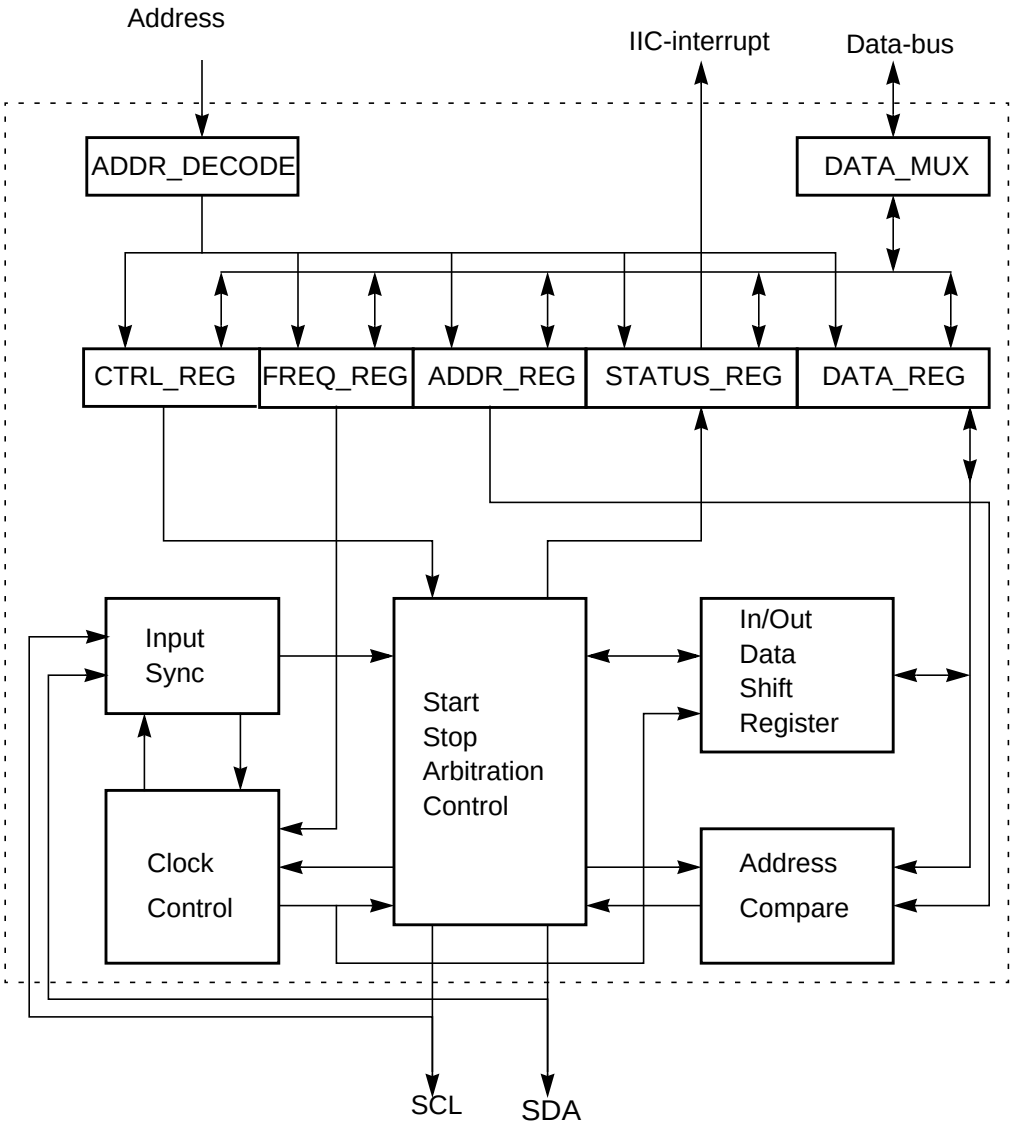


Figure 1-1 IIC Block Diagram


Address
IIC-interrupt
Data-bus
ADDR_DECODE
DATA_MUX
CTRL_REG
FREQ_REG
ADDR_REG
STATUS_REG
DATA_REG
Input
Sync
Start
Stop
Arbitration
Control
In/Out
Data
Shift
Register
Clock
Control
Address
Compare
SCL
SDA

Section 2 External Signal Description

2.1 Overview

The IIC module has a total of 2 external pins.

2.2 Detailed Signal Descriptions

2.2.1 SCL

This is the bidirectional Serial Clock Line (SCL) of the module, compatible to the IIC-Bus specification.

2.2.2 SDA

This is the bidirectional Serial Data line (SDA) of the module, compatible to the IIC-Bus specification.

Section 3 Memory Map/Register Definition

3.1 Overview

This section provides a detailed description of all memory and registers for the IIC module.

3.2 Module Memory Map

The memory map for the IIC module is given below in **Table 3-1**. The Address listed for each register is the address offset.The total address for each register is the sum of the base address for the IIC module and the address offset for each register.

Table 3-1 Module Memory Map

Address	Use	Access
Base Address + $_0	IIC-Bus Address Register (IBAD)	Read/Write
Base Address + $_1	IIC-Bus Frequency Divider Register (IBFD)	Read/Write
Base Address + $_2	IIC-Bus Control Register (IBCR)	Read/Write
Base Address + $_3	IIC-Bus Status Register (IBSR)	Read/Write
Base Address + $_4	IIC-Bus Data I/O Register (IBDR)	Read/Write

3.3 Register Descriptions

This section consists of register descriptions in address order.Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

3.3.1 IIC Address Register

Register address: Base Address + $0000)

	7	6	5	4	3	2	1	0
R	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-1 IIC Bus Address Register (IBAD)

Read and write anytime{iic_regs}

This register contains the address the IIC Bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.{iic_slave}

ADR7–ADR1 — Slave Address

Bit 1 to bit 7 contain the specific slave address to be used by the IIC Bus module.{iic_slave}

The default mode of IIC Bus is slave mode for an address match on the bus.

RESERVED

Bit 0 of the IBAD is reserved for future compatibility. This bit will always read 0.{iic_regs}

3.3.2 IIC Frequency Divider Register

Register address: Base address + $0001

	7	6	5	4	3	2	1	0
R W	IBC7	IBC6	IBC5	IBC4	IBC3	IBC2	IBC1	IBC0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 IIC Bus Frequency Divider Register (IBFD)

Read and write anytime{iic_regs}

IBC7–IBC0 — I-Bus Clock Rate 7–0

This field is used to prescale the clock for bit rate selection. {iic_div} The bit clock generator is implemented as a prescale divider - IBC7-6, prescaled shift register - IBC5-3 select the prescaler divider and IBC2-0 select the shift register tap point. {iic_div}The IBC bits are decoded to give the Tap and Prescale values as shown in **Table 3-2** {iic_div}

Table 3-2 I-Bus Tap and Prescale Values

IBC2-0 (bin)	SCL Tap (clocks)	SDA Tap (clocks)
000	5	1
001	6	1
010	7	2
011	8	2
100	9	3
101	10	3
110	12	4
111	15	4

IBC5-3 (bin)	scl2start (clocks)	scl2stop (clocks)	scl2tap (clocks)	tap2tap (clocks)
000	2	7	4	1
001	2	7	4	2
010	2	9	6	4
011	6	9	6	8
100	14	17	14	16
101	30	33	30	32
110	62	65	62	64
111	126	129	126	128

Table 3-3 Multiplier Factor

IBC7-6	MUL
00	01
01	02
10	04
11	RESERVED

The number of clocks from the falling edge of SCL to the first tap (Tap[1]) is defined by the values shown in the scl2tap column of **Table 3-2**, all subsequent tap points are separated by 2^{IBC5-3} as shown in the tap2tap column in **Table 3-2**.{iic_div} The SCL Tap is used to generated the SCL period and the SDA Tap is used to determine the delay from the falling edge of SCL to SDA changing, the SDA hold time.{iic_div}

IBC7-6 defines the multiplier factor MUL. {iic_div, iic_ack_addon}The values of MUL are shown in the **Table 3-3**{iic_div, iic_ack_addon}

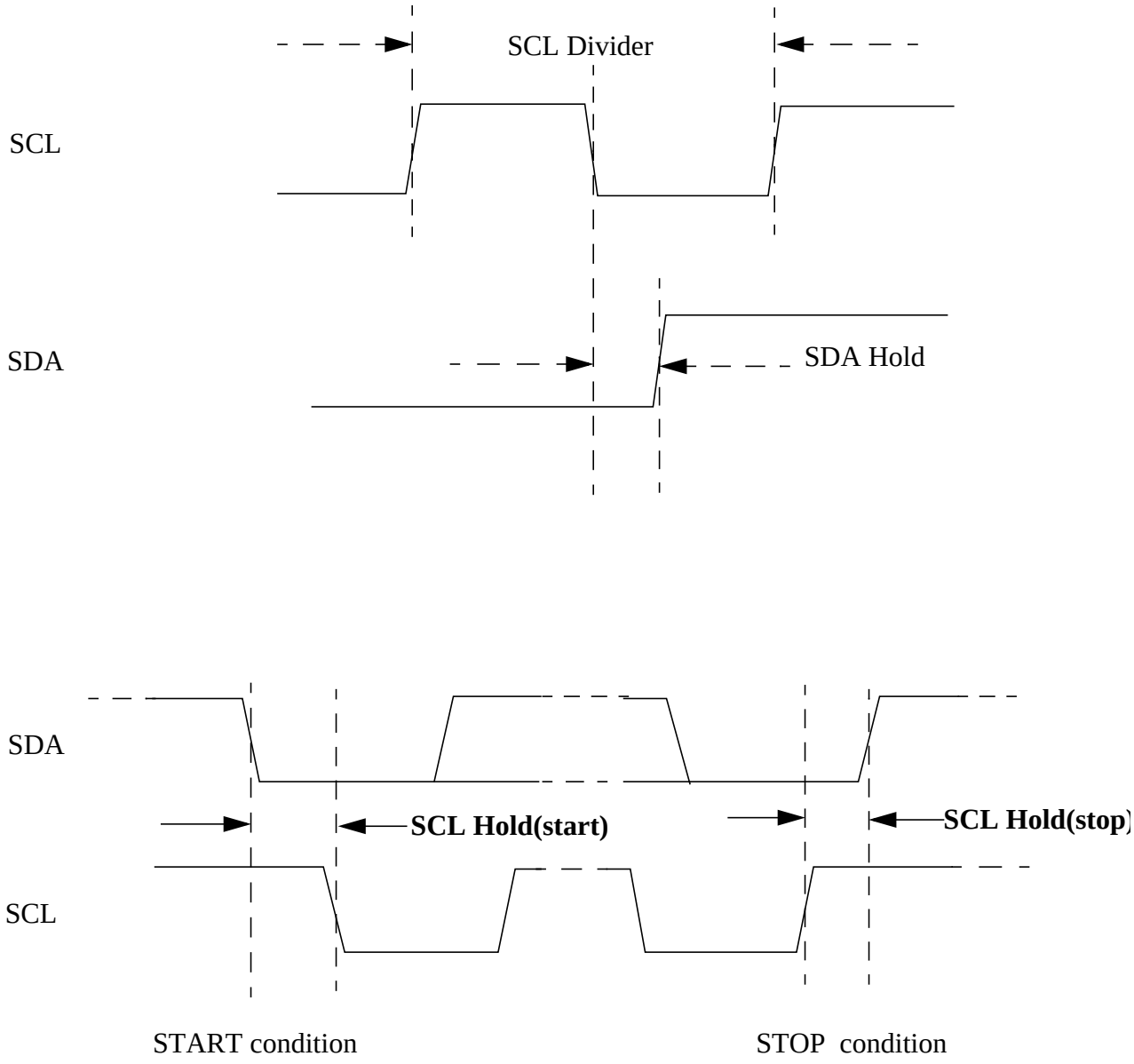


Figure 3-3 SCL divider and SDA hold

The equation used to generate the divider values from the IBFD bits is:

SCL Divider = MUL x {2 x (scl2tap + [(SCL_Tap -1) x tap2tap] + 2)}{iic_div}

The SDA hold delay is equal to the CPU clock period multiplied by the SDA Hold value shown in **Table 3-4**. {iic_div}The equation used to generate the SDA Hold value from the IBFD bits is:

SDA Hold = MUL x {scl2tap + [(SDA_Tap - 1) x tap2tap] + 3} {iic_div}

The equation for SCL Hold values to generate the start and stop conditions from the IBFD bits is:

SCL Hold(start) = MUL x [scl2start + (SCL_Tap - 1) x tap2tap] {iic_div}

SCL Hold(stop) = MUL x [scl2stop + (SCL_Tap - 1) x tap2tap] {iic_div}

Table 3-4 IIC Divider and Hold Values

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
MUL=1				
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	53
17	128	21	58	65

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	1022	1025
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3E	3072	513	1534	1537
3F	3840	513	1918	1921
MUL=2				
40	40	14	12	22
41	44	14	14	24
42	48	16	16	26
43	52	16	18	28
44	56	18	20	30
45	60	18	22	32
46	68	20	26	36
47	80	20	32	42
48	56	14	20	30
49	64	14	24	34
4A	72	18	28	38
4B	80	18	32	42
4C	88	22	36	46
4D	96	22	40	50
4E	112	26	48	58
4F	136	26	60	70
50	96	18	36	50
51	112	18	44	58
52	128	26	52	66
53	144	26	60	74
54	160	34	68	82
55	176	34	76	90
56	208	42	92	106

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
57	256	42	116	130
58	160	18	76	82
59	192	18	92	98
5A	224	34	108	114
5B	256	34	124	130
5C	288	50	140	146
5D	320	50	156	162
5E	384	66	188	194
5F	480	66	236	242
60	320	34	156	162
61	384	34	188	194
62	448	66	220	226
63	512	66	252	258
64	576	98	284	290
65	640	98	316	322
66	768	130	380	386
67	960	130	476	482
68	640	66	316	322
69	768	66	380	386
6A	896	130	444	450
6B	1024	130	508	514
6C	1152	194	572	578
6D	1280	194	636	642
6E	1536	258	764	770
6F	1920	258	956	962
70	1280	130	636	642
71	1536	130	764	770
72	1792	258	892	898
73	2048	258	1020	1026
74	2304	386	1148	1154
75	2560	386	1276	1282
76	3072	514	1532	1538

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
77	3840	514	1916	1922
78	2560	258	1276	1282
79	3072	258	1532	1538
7A	3584	514	1788	1794
7B	4096	514	2044	2050
7C	4608	770	2300	2306
7D	5120	770	2556	2562
7E	6144	1026	3068	3074
7F	7680	1026	3836	3842
MUL=4				
80	80	28	24	44
81	88	28	28	48
82	96	32	32	52
83	104	32	36	56
84	112	36	40	60
85	120	36	44	64
86	136	40	52	72
87	160	40	64	84
88	112	28	40	60
89	128	28	48	68
8A	144	36	56	76
8B	160	36	64	84
8C	176	44	72	92
8D	192	44	80	100
8E	224	52	96	116
8F	272	52	120	140
90	192	36	72	100
91	224	36	88	116
92	256	52	104	132
93	288	52	120	148
94	320	68	136	164
95	352	68	152	180

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
96	416	84	184	212
97	512	84	232	260
98	320	36	152	164
99	384	36	184	196
9A	448	68	216	228
9B	512	68	248	260
9C	576	100	280	292
9D	640	100	312	324
9E	768	132	376	388
9F	960	132	472	484
A0	640	68	312	324
A1	768	68	376	388
A2	896	132	440	452
A3	1024	132	504	516
A4	1152	196	568	580
A5	1280	196	632	644
A6	1536	260	760	772
A7	1920	260	952	964
A8	1280	132	632	644
A9	1536	132	760	772
AA	1792	260	888	900
AB	2048	260	1016	1028
AC	2304	388	1144	1156
AD	2560	388	1272	1284
AE	3072	516	1528	1540
AF	3840	516	1912	1924
B0	2560	260	1272	1284
B1	3072	260	1528	1540
B2	3584	516	1784	1796
B3	4096	516	2040	2052
B4	4608	772	2296	2308
B5	5120	772	2552	2564

IBC[7:0] (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
B6	6144	1028	3064	3076
B7	7680	1028	3832	3844
B8	5120	516	2552	2564
B9	6144	516	3064	3076
BA	7168	1028	3576	3588
BB	8192	1028	4088	4100
BC	9216	1540	4600	4612
BD	10240	1540	5112	5124
BE	12288	2052	6136	6148
BF	15360	2052	7672	7684

3.3.3 IIC Control Register

Register address: Base address + $0002

	7	6	5	4	3	2	1	0
R	IBEN	IBIE	MS/$\overline{SL}$	Tx/$\overline{Rx}$	TXAK	0	0	IBSWAI
W						RSTA		
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-4 IIC-Bus Control Register (IBCR)

Read and write anytime{iic_regs}

IBEN — I-Bus Enable

This bit controls the software reset of the entire IIC Bus module.

0 = The module is reset and disabled.{iic_disable} This is the power-on reset situation. When low the interface is held in reset but registers can still be accessed{iic_disable}

1 = The IIC Bus module is enabled. {iic_div, iic_ack, iic_receive, iic_transmit}This bit must be set before any other IBCR bits have any effect{iic_disable}

If the IIC Bus module is enabled in the middle of a byte transfer the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected.Master mode will not be aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the IIC Bus module losing arbitration, after which bus operation would return to normal.

IBIE — I-Bus Interrupt Enable

0 = Interrupts from the IIC Bus module are disabled. {iic_int} Note that this does not clear any currently pending interrupt condition.{iic_int}

1 = Interrupts from the IIC Bus module are enabled. {iic_int}An IIC Bus interrupt occurs provided the IBIF bit in the status register is also set.{iic_int}

MS/$\overline{SL}$ — Master/Slave mode select bit

Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus, and the master mode is selected.{iic_receive, iic_transmit} When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave.{iic_receive, iic_transmit} A STOP signal should only be generated if the IBIF flag is set. MS/$\overline{SL}$ is cleared without generating a STOP signal when the master loses arbitration.

0 = Slave Mode
1 = Master Mode

Tx/$\overline{Rx}$ — Transmit/Receive mode select bit

This bit selects the direction of master and slave transfers. {iic_receive, iic_transmit} When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high.

0 = Receive
1 = Transmit

TXAK — Transmit Acknowledge enable

This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers.{iic_receive, iic_transmit} The IIC module will always acknowledge address matches, provided it is enabled, regardless of the value of TXAK. {iic_ack}Note that values written to this bit are only used when the IIC Bus is a receiver, not a transmitter.

0 = An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte data{iic_receive, iic_transmit}
1 = No acknowledge signal response is sent (i.e., acknowledge bit = 1){iic_receive, iic_transmit}

RSTA — Repeat Start

Writing a 1 to this bit will generate a repeated START condition on the bus, provided it is the current bus master. {iic_receive, iic_transmit}This bit will always be read as a low.{iic_regs, iic_receive, iic_transmit} Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration.

1 = Generate repeat start cycle

RESERVED

Bit 1 of the IBCR is reserved for future compatibility. This bit will always read 0.

{iic_regs}

IBSWAI — I-Bus Interface Stop in WAIT mode

0 = IIC Bus module clock operates normally{iic_wait}
1 = Halt IIC Bus module clock generation in WAIT mode{iic_wait}

Wait mode is entered via execution of a CPU WAI instruction. In the event that the IBSWAI bit is set, all clocks internal to the IIC will be stopped and any transmission currently in progress will halt.{iic_wait} If

the CPU were woken up by a source other than the IIC module, then clocks would restart and the IIC would continue where it left off in the previous transmission.{iic_wait} It is not possible for the IIC to wake up the CPU when its internal clocks are stopped.

If it were the case that the IBSWAI bit was cleared when the WAI instruction was executed, the IIC internal clocks and interface would remain alive, continuing the operation which was currently underway. It is also possible to configure the IIC such that it will wake up the CPU via an interrupt at the conclusion of the current operation. See the discussion on the IBIF and IBIE bits in the IBSR and IBCR, respectively.

3.3.4 IIC Status Register

Register address: Base address + $0003

	7	6	5	4	3	2	1	0
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W								
RESET:	1	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-5 IIC Bus Status Register (IBSR)

This status register is read-only with exception of bit 1 (IBIF) and bit 4 (IBAL), which are software clearable{iic_regs}

TCF — Data transferring bit

While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. Note that this bit is only valid during or immediately following a transfer to the IIC module or from the IIC module.{iic_int}

0 = Transfer in progress
1 = Transfer complete

IAAS — Addressed as a slave bit

When its own specific address (I-Bus Address Register) is matched with the calling address, this bit is set.{iic_slave}The CPU is interrupted provided the IBIE is set.{iic_int}Then the CPU needs to check the SRW bit and set its Tx/$\overline{\text{Rx}}$ mode accordingly.Writing to the I-Bus Control Register clears this bit.{iic_int}

0 = Not addressed
1 = Addressed as a slave

IBB — Bus busy bit

This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and {iic_receive, iic_transmit}

0 = the bus enters idle state.
1 = Bus is busy

IBAL — Arbitration Lost

The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances:

1. SDA sampled low when the master drives a high during an address or data transmit cycle.{iic_arb}
2. SDA sampled low when the master drives a high during the acknowledge bit of a data receive cycle.{iic_arb}
3. A start cycle is attempted when the bus is busy.{iic_arb}
4. A repeated start cycle is requested in slave mode.{iic_arb}
5. A stop condition is detected when the master did not request it.{iic_arb}

This bit must be cleared by software, by writing a one to it. A write of zero has no effect on this bit.

RESERVED

Bit 3 of IBSR is reserved for future use. A read operation on this bit will return 0.{iic_regs}

SRW — Slave Read/Write

When IAAS is set this bit indicates the value of the R/W command bit of the calling address sent from the master. {iic_receive, iic_transmit}

This bit is only valid when the I-Bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated.

Checking this bit, the CPU can select slave transmit/receive mode according to the command of the master.

0 = Slave receive, master writing to slave
1 = Slave transmit, master reading from slave

IBIF — I-Bus Interrupt

The IBIF bit is set when one of the following conditions occurs:

– arbitration lost (IBAL bit set)
– byte transfer complete (TCF bit set)
– addressed as slave (IAAS bit set)

It will cause a processor interrupt request if the IBIE bit is set. This bit must be cleared by software, writing a one to it. A write of zero has no effect on this bit.

RXAK — Received Acknowledge

The value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus.{iic_ack} If RXAK is high, it means no acknowledge signal is detected at the 9th clock.{iic_ack}

0 = Acknowledge received
1 = No acknowledge received

3.3.5 IIC Data I/O Register

Register address

	7	6	5	4	3	2	1	0
R W	D7	D6	D5	D4	D3	D2	D1	D0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 IIC Bus Data I/O Register (IBDR)

In master transmit mode, when data is written to the IBDR a data transfer is initiated. {iic_transmit}The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving.{iic_receive} In slave mode, the same functions are available after an address match has occurred.{iic_receive, iic_transmit}Note that the Tx/Rx bit in the IBCR must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. {iic_receive, iic_transmit} For instance, if the IIC is configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.

Reading the IBDR will return the last byte received while the IIC is configured in either master receive or slave receive modes. {iic_receive}The IBDR does not reflect every byte that is transmitted on the IIC bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of MS/$\overline{SL}$ is used for the address transfer and should com.prise of the calling address (in position D7-D1) concatenated with the required R/$\overline{W}$ bit (in position D0).

Section 4 Functional Description

4.1 General

This section provides a complete functional description of the IIC.

4.2 I-Bus Protocol

The IIC Bus system uses a Serial Data line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. Logic AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal. They are described briefly in the following sections and illustrated in **Figure 4-1**.

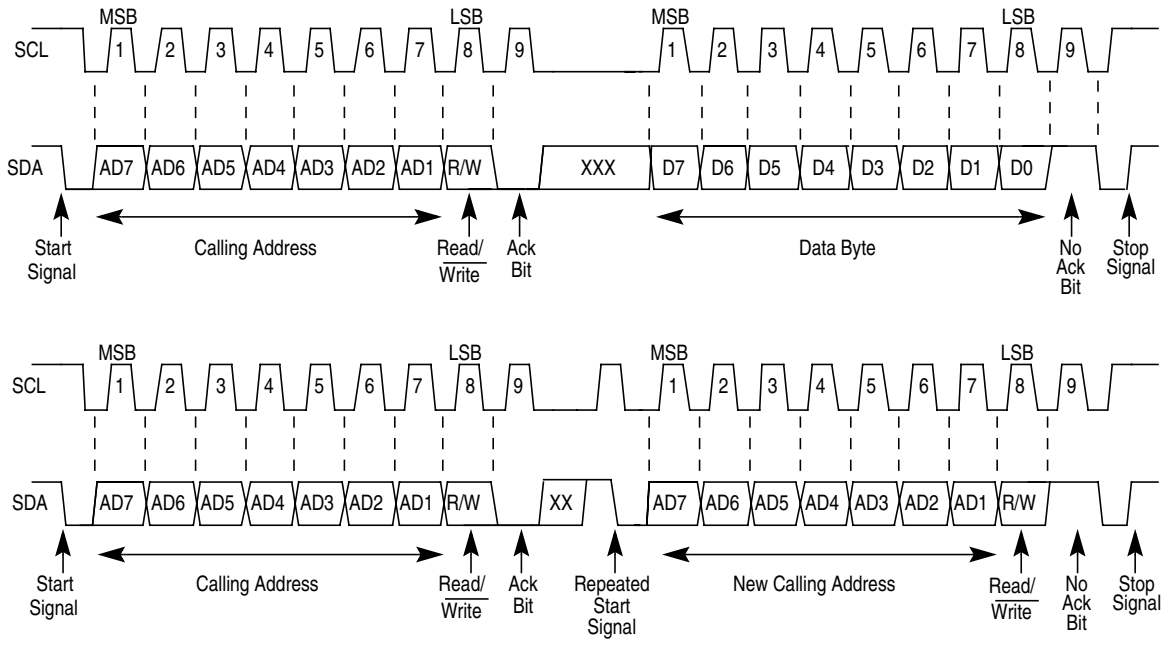


Figure 4-1 IIC-Bus Transmission Signals

4.2.1 START Signal

When the bus is free, i.e. no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal.As shown in Figure 4-1, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

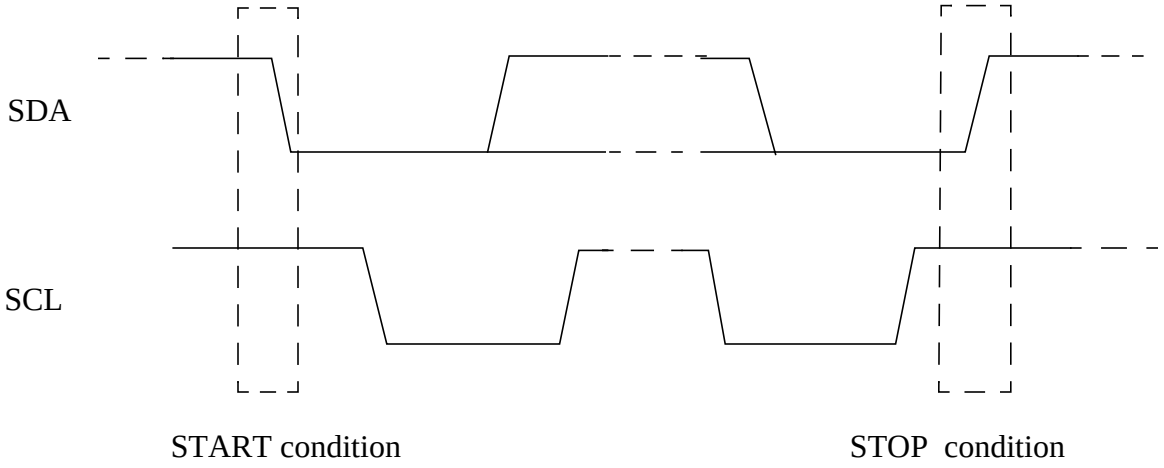


Figure 4-2 Start and Stop conditions

4.2.2 Slave Address Transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer, the slave transmits data to the master.
0 = Write transfer, the master transmits data to the slave.

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see Figure 4-1).

No two slaves in the system may have the same address. If the IIC Bus is master, it must not transmit an address that is equal to its own slave address. The IIC Bus cannot be master and slave at the same time.However, if arbitration is lost during an address cycle the IIC Bus will revert to slave mode and operate correctly even if it is being addressed by another master.

4.2.3 Data Transfer

Once successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in Figure 4-1. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte has to be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. So one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so the slave releases the SDA line for the master to generate STOP or START signal.

4.2.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL at logical “1” (see **Figure 4-1**).

The master can generate a STOP even if the slave has generated an acknowledge at which point the slave must release the bus.

4.2.5 Repeated START Signal

As shown in **Figure 4-1**, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

4.2.6 Arbitration Procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure, a bus master loses arbitration if it transmits logic “1” while another master transmits logic “0”. The losing masters immediately switch over to slave receive mode and stop driving SDA output. In this case the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

4.2.7 Clock Synchronization

Since wire-AND logic is performed on SCL line, a high-to-low transition on SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached.However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see **Figure 4-2**). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods.The first device to complete its high period pulls the SCL line low again.

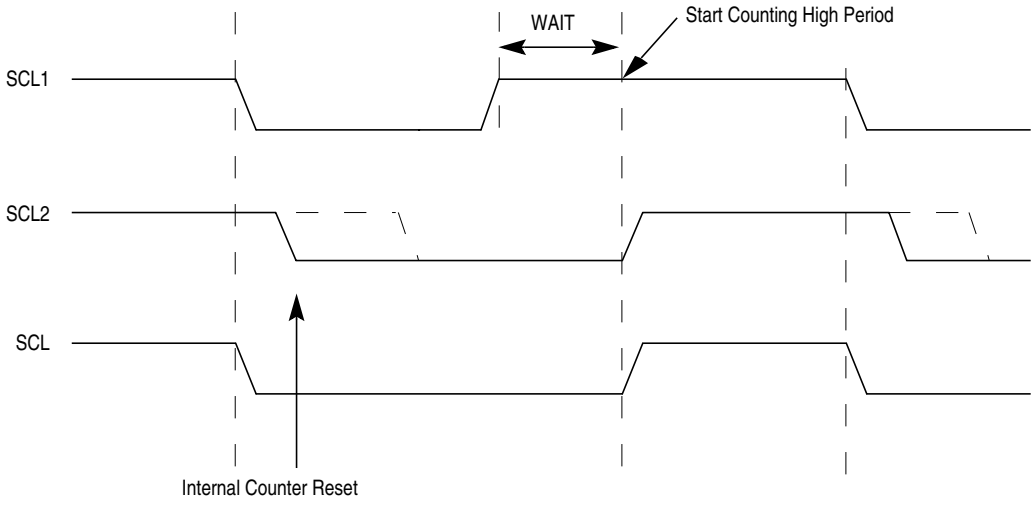


Figure 4-3 IIC-Bus Clock Synchronization

4.2.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

4.2.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low the slave can drive SCL low for the required period and then release it.If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

4.3 Modes of Operation

The IIC functions the same in normal, special, and emulation modes. It has two low power modes, wait and stop modes

4.3.1 Run Mode

This is the basic mode of operation.

4.3.2 Wait Mode

IIC operation in wait mode can be configured. Depending on the state of internal bits, the IIC can operate normally when the CPU is in wait mode or the IIC clock generation can be turned off and the IIC module enters a power conservation state during wait mode. In the later case, any transmission or reception in progress stops at wait mode entry.

4.3.3 Stop Mode

The IIC is inactive in stop mode for reduced power consumption. The STOP instruction does not affect IIC register states.

Section 5 Initialization/Application Information

5.1 IIC Programming Examples

5.1.1 Initialization Sequence

Reset will put the IIC Bus Control Register to its default status. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the Frequency Divider Register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the IIC Bus Address Register (IBAD) to define its slave address.
3. Set the IBEN bit of the IIC Bus Control Register (IBCR) to enable the IIC interface system.
4. Modify the bits of the IIC Bus Control Register (IBCR) to select Master/Slave mode, Transmit/Receive mode and interrupt enable or not.

5.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the IIC Bus Busy bit (IBB) must be tested to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period it may be necessary to wait until the IIC is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of a program which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
CHFLAG    BRSET   IBSR,#$20,*     ;WAIT FOR IBB FLAG TO CLEAR
TXSTART   BSET    IBCR,#$30       ;SET TRANSMIT AND MASTER MODE;i.e. GENERATE START CONDITION
          MOVB    CALLING,IBDR    ;TRANSMIT THE CALLING ADDRESS, D0=R/W
IBFREE    BRCLR   IBSR,#$20,*     ;WAIT FOR IBB FLAG TO SET
```

5.1.3 Post-Transfer Software Response

Successful transmission or reception of a byte will set the TCF (data transferring) bit and the IBIF (interrupt flag) bit in the IBSR status register. An interrupt service routine can be called by this action if

the IBIE (interrupt enable) bit in the IBCR control register is set. The IBIF (interrupt flag) bit can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress by reading the IBDR data register in receive mode or writing the IBDR in transmit mode. The TCF bit should not be used as a data transfer complete flag as the flag timing is dependent on a number of factors including the IIC bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. It is recommended that transfer complete situations are detected using the IBIF flag

Software may service the IIC I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Note that polling should monitor the IBIF bit rather than the TCF bit since their operation is different when arbitration is lost.

Note that when an interrupt occurs at the end of the address cycle the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit in IBDR, then the Tx/Rx bit should be toggled at this stage.

During slave mode address cycles (IAAS=1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the Tx/Rx bit is programmed accordingly. For slave mode data cycles (IAAS=0) the SRW bit is not valid, the Tx/Rx bit in the control register should be read to determine the direction of the current transfer.

The following is an example of a software response by a 'master transmitter' in the interrupt routine .

```
ISR         BCLR    IBSR,#$02           ;CLEAR THE IBIF FLAG
            BRCLR   IBCR,#$20,SLAVE     ;BRANCH IF IN SLAVE MODE
            BRCLR   IBCR,#$10,RECEIVE   ;BRANCH IF IN RECEIVE MODE
            BRSET   IBSR,#$01,END       ;IF NO ACK, END OF TRANSMISSION
TRANSMIT    MOVB    DATABUF,IBDR        ;TRANSMIT NEXT BYTE OF DATA
```

5.1.4 Generation of STOP

A data transfer ends with a STOP signal generated by the 'master' device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```
MASTX       TST     TXCNT               ;GET VALUE FROM THE TRANSMITING COUNTER
            BEQ     END                 ;END IF NO MORE DATA
            BRSET   IBSR,#$01,END       ;END IF NO ACK
            MOVB    DATABUF,IBDR        ;TRANSMIT NEXT BYTE OF DATA
            DEC     TXCNT               ;DECREASE THE TXCNT
            BRA     EMASTX              ;EXIT
END         BCLR    IBCR,#$20           ;GENERATE A STOP CONDITION
EMASTX      RTI                         ;RETURN FROM INTERRUPT
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data which can be done by setting the transmit acknowledge bit (TXAK)

before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must be generated first. The following is an example showing how a STOP signal is generated by a master receiver.

```
MASR    DEC     RXCNT       ;DECREASE THE RXCNT
        BEQ     ENMASR      ;LAST BYTE TO BE READ
        MOVB    RXCNT,D1    ;CHECK SECOND LAST BYTE
        DEC     D1          ;TO BE READ
        BNE     NXMAR       ;NOT LAST OR SECOND LAST
LAMAR   BSET    IBCR,#$08   ;SECOND LAST, DISABLE ACK
                            ;TRANSMITTING
        BRA     NXMAR
ENMASR  BCLR    IBCR,#$20   ;LAST ONE, GENERATE 'STOP' SIGNAL
NXMAR   MOVB    IBDR,RXBUF  ;READ DATA AND STORE
        RTI
```

5.1.5 Generation of Repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
RESTART BSET    IBCR,#$04   ;ANOTHER START (RESTART)
        MOVB    CALLING,IBDR;TRANSMIT THE CALLING ADDRESS;D0=R/W
```

5.1.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received . If IAAS is set, software should set the transmit/receive mode select bit (Tx/Rx bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears the IAAS automatically. Note that the only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred, interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may now be initiated by writing information to IBDR, for slave transmits, or dummy reading from IBDR, in slave receive mode. The slave will drive SCL low in-between byte transfers, SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an 'end of data' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so that the master can generate a STOP signal.

5.1.7 Arbitration Lost

If several masters try to engage the bus simultaneously, only one master wins and the others lose arbitration. The devices which lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS/SL=0. If one master attempts to start transmission while the bus is being engaged by another master, the hardware will inhibit the transmission; switch the MS/SL bit from 1 to 0 without generating STOP condition; generate an interrupt to CPU and set the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

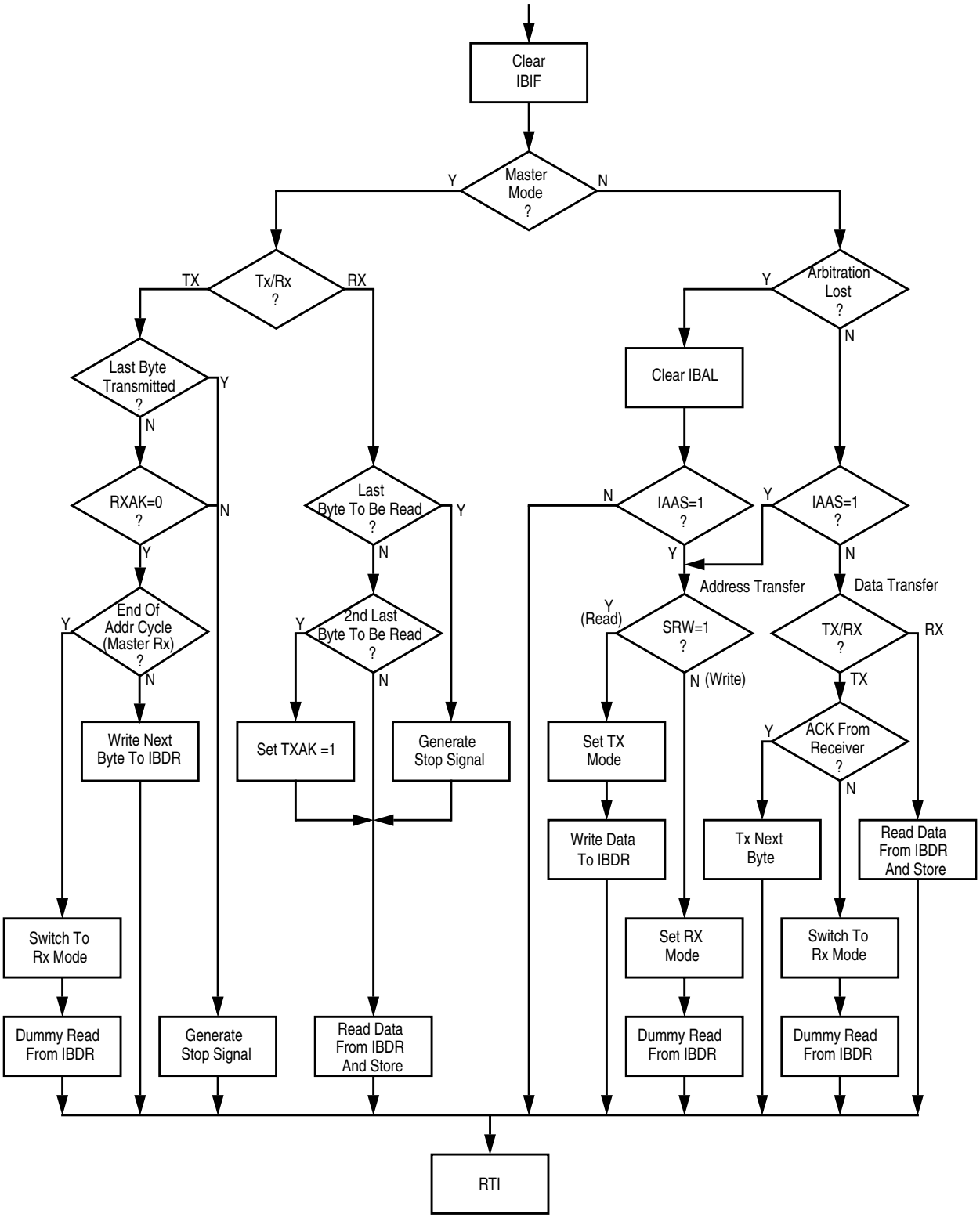


Figure 5-1 Flow-Chart of Typical IIC Interrupt Routine

Section 6 Resets

6.1 General

The reset state of each individual bit is listed within the Register Description section (see **Section 3 Memory Map/Register Definition**) which details the registers and their bit-fields.

Section 7 Interrupts

7.1 General

IIC uses only one interrupt vector.

Table 7-1 Interrupt Summary

Interrupt	Offset	Vector	Priority	Source	Description
IIC Interrupt	-	-	-	IBAL, TCF, IAAS bits in IBSR register	When either of IBAL, TCF or IAAS bits is set may cause an interrupt based on Arbitration lost, Transfer Complete or Address Detect conditions.

7.2 Interrupt Description

Internally there are three types of interrupts in IIC. The interrupt service routine can determine the interrupt type by reading the Status Register.

IIC Interrupt can be generated on

1. Arbitration Lost condition (IBAL bit set)
2. Byte Transfer condition (TCF bit set)
3. Address Detect condition (IAAS bit set)

The IIC interrupt is enabled by the IBIE bit in the IIC Control Register. It must be cleared by writing ‘1’ to the IBIF bit in the interrupt service routine.

Block Guide End Sheet

FINAL PAGE OF
46
PAGES

MOTOROLA
intelligence everywhere™

digital dna™

Interrupt (INT) Module V1

HCS12 Microcontrollers

S12INTV1/D
Rev. 1.00
5/2003

MOTOROLA.COM/SEMICONDUCTORS

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
1.02	5/1/2003	5/1/2003	John Langan	Creation of block user guide from core user guide version 1.5 (Oct. 12, 2001). Changes include: updating format and making end-customer friendly. Original release.

Table of Contents

List of Figures

List of Tables

Section 1 Introduction to Interrupt (INT)

Section 2 External Signal Description

Section 3 Memory Map/Register Definition

Section 4 Functional Description

Section 1 Introduction to Interrupt (INT)

This section describes the functionality of the Interrupt (INT) sub-block of the S12 Core Platform.

A block diagram of the Interrupt sub-block is shown in **Figure 1-1**.

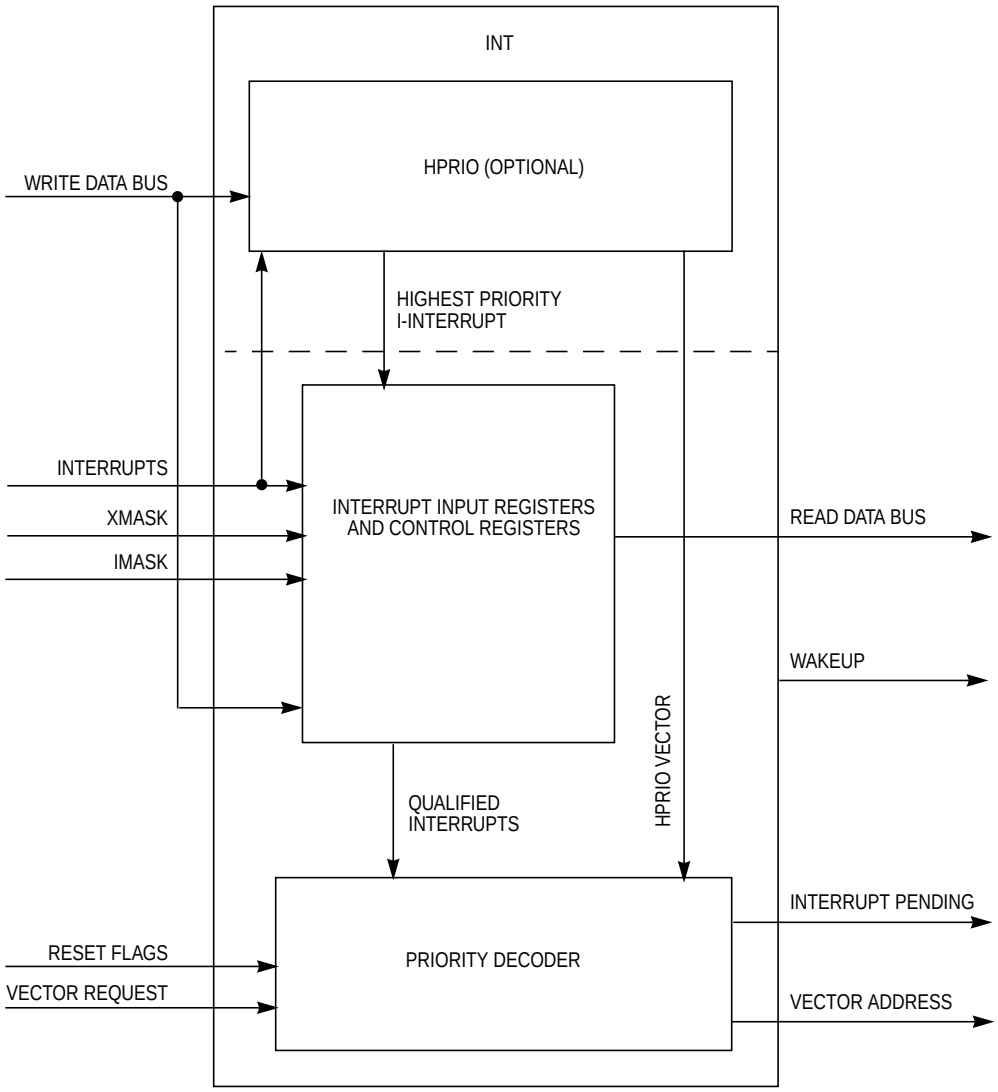


Figure 1-1 Interrupt Block Diagram

1.1 Overview

The Interrupt sub-block decodes the priority of all system exception requests and provides the applicable vector for processing the exception. The INT supports I-bit maskable and X-bit maskable interrupts, a nonmaskable Unimplemented Opcode Trap, a nonmaskable software interrupt (SWI) or Background Debug Mode request, and three system reset vector requests. All interrupt related exception requests are handled by the Interrupt sub-block (INT).

1.2 Features

The INT includes these features:

- Provides two to 122 I-bit maskable interrupt vectors ($FF00–$FFF2)
- Provides one X-bit maskable interrupt vector ($FFF4)
- Provides a nonmaskable software interrupt (SWI) or Background Debug Mode request vector ($FFF6)
- Provides a nonmaskable Unimplemented Opcode Trap (TRAP) vector ($FFF8)
- Provides three system reset vectors ($FFFA–$FFFE) (Reset, CMR, and COP)
- Determines the appropriate vector and drives it onto the address bus at the appropriate time
- Signals the CPU that interrupts are pending
- Provides control registers which allow testing of interrupts
- Provides additional input signals which prevents requests for servicing I and X interrupts
- Wakes the system from stop or wait mode when an appropriate interrupt occurs or whenever $\overline{\text{XIRQ}}$ is active, even if $\overline{\text{XIRQ}}$ is masked
- Provides asynchronous path for all I and X interrupts, ($FF00–$FFF4)
- (Optional) Selects and stores the highest priority I interrupt based on the value written into the HPRIO register

1.3 Modes of Operation

The functionality of the INT sub-block in various modes of operation is discussed in the subsections that follow.

1.3.1 Normal Operation

The INT operates the same in all normal modes of operation.

1.3.2 Special Operation

Interrupts may be tested in special modes through the use of the interrupt test registers.

1.3.3 Emulation Modes

The INT operates the same in emulation modes as in normal modes.

1.4 Low-Power Options

The INT does not contain any user-controlled options for reducing power consumption. The operation of the INT in low-power modes is discussed in the following subsections.

1.4.1 Run Mode

The INT does not contain any options for reducing power in run mode.

1.4.2 Wait Mode

Clocks to the INT can be shut off during system wait mode and the asynchronous interrupt path will be used to generate the wake-up signal upon recognition of a valid interrupt or any $\overline{\text{XIRQ}}$ request.

1.4.3 Stop Mode

Clocks to the INT can be shut off during system stop mode and the asynchronous interrupt path will be used to generate the wake-up signal upon recognition of a valid interrupt or any $\overline{\text{XIRQ}}$ request.

Section 2 External Signal Description

Most interfacing with the Interrupt sub-block is done within the Core. However, the Interrupt does receive direct input from the Multiplexed External Bus Interface (MEBI) sub-block of the Core for the $\overline{\text{IRQ}}$ and $\overline{\text{XIRQ}}$ pin data.

Section 3 Memory Map/Register Definition

A summary of the registers associated with the Interrupt sub-block is shown in **Figure 3-1**. Detailed descriptions of the registers and associated bits are given in the subsections that follow.

Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
$0015	ITCR	Read	0	0	0	WRTINT	ADR3	ADR2	ADR1	ADR0
		Write								
$0016	ITEST	Read	INTE	INTC	INTA	INT8	INT6	INT4	INT2	INT0
		Write								
$001F	HPRIO	Read	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0
		Write								

= Unimplemented

Figure 3-1 Interrupt Register Summary

3.1 Interrupt Test Control Register

Register address: Base + $0015

	7	6	5	4	3	2	1	0
R	0	0	0	WRTINT	ADR3	ADR2	ADR1	ADR0
W								
Reset:	0	0	0	0	1	1	1	1

= Unimplemented or Reserved

Figure 3-2 Interrupt Test Control Register (ITCR)

Read: see individual bit descriptions

Write: see individual bit descriptions

WRTINT — Write to the Interrupt Test Registers

Read: anytime

Write: only in special modes and with I-bit mask and X-bit mask set.

1 = Disconnect the interrupt inputs from the priority decoder and use the values written into the ITEST registers instead.

0 = Disables writes to the test registers; reads of the test registers will return the state of the interrupt inputs.

NOTE: *Any interrupts which are pending at the time that WRTINT is set will remain until they are overwritten.*

ADR3–ADR0 — Test Register Select Bits

Read: anytime

Write: anytime

These bits determine which test register is selected on a read or write. The hexadecimal value written here will be the same as the upper nibble of the lower byte of the vector selects. That is, an “F” written into ADR3–ADR0 will select vectors $FFFE–$FFF0 while a “7” written to ADR3–ADR0 will select vectors $FF7E–$FF70.

3.2 Interrupt Test Registers

Register address: Base + $0016

	7	6	5	4	3	2	1	0
R W	INTE	INTC	INTA	INT8	INT6	INT4	INT2	INT0
Reset:	0	0	0	0	0	0	0	0

Figure 3-3 Interrupt TEST Registers (ITEST)

Read: Only in special modes. Reads will return either the state of the interrupt inputs of the Interrupt sub-block (WRTINT = 0) or the values written into the TEST registers (WRTINT = 1). Reads will always return zeroes in normal modes.

Write: Only in special modes and with WRTINT = 1 and CCR I mask = 1.

INTE–INT0 — Interrupt TEST Bits

These registers are used in special modes for testing the interrupt logic and priority independent of the system configuration. Each bit is used to force a specific interrupt vector by writing it to a logic one state. Bits are named INTE through INT0 to indicate vectors $FFxE through $FFx0. These bits can be written only in special modes and only with the WRTINT bit set (logic one) in the Interrupt Test Control Register (ITCR). In addition, I interrupts must be masked using the I bit in the CCR. In this

state, the interrupt input lines to the Interrupt sub-block will be disconnected and interrupt requests will be generated only by this register. These bits can also be read in special modes to view that an interrupt requested by a system block (such as a peripheral block) has reached the INT module.

There is a test register implemented for every 8 interrupts in the overall system. All of the test registers share the same address and are individually selected using the value stored in the ADR3–ADR0 bits of the Interrupt Test Control Register (ITCR).

NOTE: *When ADR3–ADR0 have the value of $F, only bits 2–0 in the ITEST register will be accessible. That is, vectors higher than $FFF4 cannot be tested using the test registers and bits 7–3 will always read as a logic zero. If ADR3–ADR0 point to an unimplemented test register, writes will have no effect and reads will always return a logic zero value.*

3.3 Highest Priority I Interrupt (Optional)

Register address: Base + $001F

	7	6	5	4	3	2	1	0
R	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0
W								
Reseet:	1	1	1	1	0	0	1	0

= Unimplemented or Reserved

Figure 3-4 Highest Priority I Interrupt Register (HPRIO)

Read: anytime

Write: only if I mask in CCR = 1

PSEL7–PSEL1 — Highest Priority I Interrupt Select Bits

The state of these bits determines which I-bit maskable interrupt will be promoted to highest priority (of the I-bit maskable interrupts). To promote an interrupt, the user writes the least significant byte of the associated interrupt vector address to this register. If an unimplemented vector address or a non I-bit masked vector address (value higher than $F2) is written, IRQ ($FFF2) will be the default highest priority interrupt.

Section 4 Functional Description

The Interrupt sub-block processes all exception requests made by the CPU. These exceptions include interrupt vector requests and reset vector requests. Each of these exception types and their overall priority level is discussed in the subsections below.

4.1 Interrupt Exception Requests

As shown in the block diagram in **Figure 1-1**, the INT contains a register block to provide interrupt status and control, an optional Highest Priority I Interrupt (HPRIO) block, and a priority decoder to evaluate whether pending interrupts are valid and assess their priority.

4.1.1 Interrupt Registers

The INT registers are accessible only in special modes of operation and function as described in **3.1 Interrupt Test Control Register** and **3.2 Interrupt Test Registers** previously.

4.1.2 Highest Priority I-Bit Maskable Interrupt

When the optional HPRIO block is implemented, the user is allowed to promote a single I-bit maskable interrupt to be the highest priority I interrupt. The HPRIO evaluates all interrupt exception requests and passes the HPRIO vector to the priority decoder if the highest priority I interrupt is active. RTI replaces the promoted interrupt source.

4.1.3 Interrupt Priority Decoder

The priority decoder evaluates all interrupts pending and determines their validity and priority. When the CPU requests an interrupt vector, the decoder will provide the vector for the highest priority interrupt request. Because the vector is not supplied until the CPU requests it, it is possible that a higher priority interrupt request could override the original exception that caused the CPU to request the vector. In this case, the CPU will receive the highest priority vector and the system will process this exception instead of the original request.

NOTE: *Care must be taken to ensure that all exception requests remain active until the system begins execution of the applicable service routine; otherwise, the exception request may not get processed.*

If for any reason the interrupt source is unknown (e.g., an interrupt request becomes inactive after the interrupt has been recognized but prior to the vector request), the vector address will default to that of the last valid interrupt that existed during the particular interrupt sequence. If the CPU requests an interrupt vector when there has never been a pending interrupt request, the INT will provide the Software Interrupt (SWI) vector address.

4.2 Reset Exception Requests

The INT supports three system reset exception request types: normal system reset or power-on-reset request, Crystal Monitor reset request, and COP Watchdog reset request. The type of reset exception request must be decoded by the system and the proper request made to the Core. The INT will then provide the service routine address for the type of reset requested.

4.3 Exception Priority

The priority (from highest to lowest) and address of all exception vectors issued by the INT upon request by the CPU is shown in **Table 4-1**.

Table 4-1 Exception Vector Map and Priority

Vector Address	Source
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Crystal monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI) or BDM vector request
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ signal
$FFF0–$FF00	Device-specific I-bit maskable interrupt sources (priority in descending order)

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

JAPAN:
Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

MOTOROLA

S12INTV1/D
Rev. 1.00
5/2003

MOTOROLA
intelligence everywhere™

digital dna™

Multiplexed External Bus Interface (MEBI) Module V3

Block User Guide

HCS12
Microcontrollers

S12MEBIV3/D
Rev. 3.00
2/2003

MOTOROLA.COM/SEMICONDUCTORS

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
3.00	2/5/2003	2/5/2003	John Langan	Original release

Table of Contents

List of Figures

List of Tables

Section 1 Introduction

This section describes the functionality of the Multiplexed External Bus Interface (MEBI) sub-block of the S12 Core Platform. The functionality of the module is closely coupled with the S12 CPU and the Memory Map Controller (MMC) sub-blocks.

Figure 1-1 is a block diagram of the MEBI. In **Figure 1-1**, the signals on the right hand side represent pins that are accessible externally. On some chips, these may not all be bonded out.

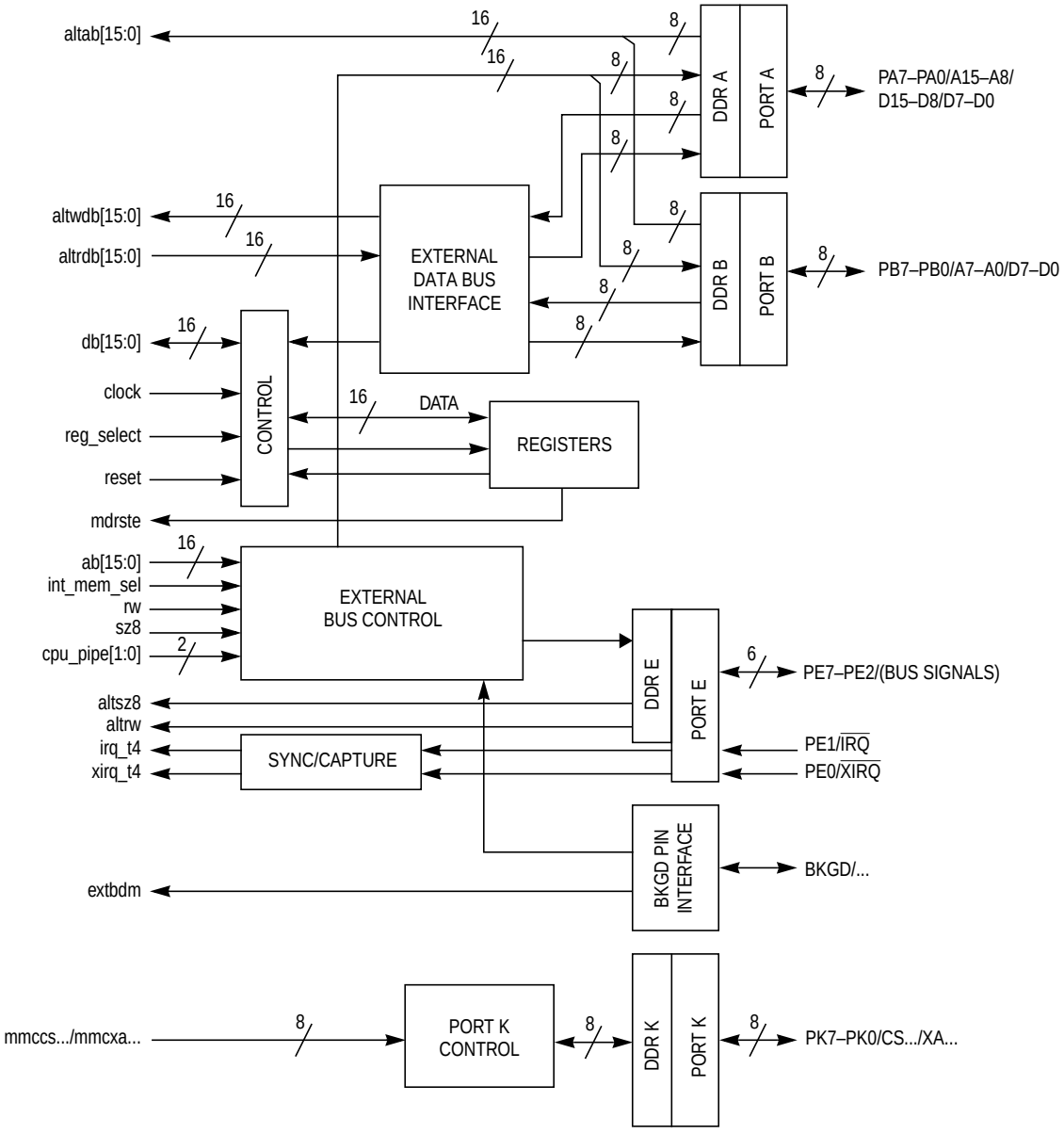


Figure 1-1 MEBI Block Diagram

1.1 Overview

The MEBI sub-block of the Core serves to provide access and/or visibility to internal Core data manipulation operations including timing reference information at the external boundary of the Core and/or system. Depending upon the system operating mode and the state of bits within the control registers of the MEBI, the internal 16-bit read and write data operations will be represented in 8-bit or 16-bit accesses externally. Using control information from other blocks within the system, the MEBI will determine the appropriate type of data access to be generated.

1.2 Features

The block name includes these distinctive features:

- External bus controller with four 8-bit Ports A,B, E, and K
- Data and data direction registers for Ports A, B, E, and K when used as general-purpose I/O
- Control register to enable/disable alternate functions on Ports E and K
- Mode control register
- Control register to enable/disable pull-ups on Ports A, B, E, and K
- Control register to enable/disable reduced output drive on Ports A, B, E, and K
- Control register to configure external clock behavior
- Control register to configure $\overline{\text{IRQ}}$ pin operation
- Logic to capture and synchronize external interrupt pin inputs

1.3 Modes of Operation

- Normal Expanded Wide Mode

  Ports A and B are configured as a 16-bit multiplexed address and data bus and Port E provides bus control and status signals. This mode allows 16-bit external memory and peripheral devices to be interfaced to the system.

- Normal Expanded Narrow Mode

  Ports A and B are configured as a 16-bit address bus and Port A is multiplexed with 8-bit data. Port E provides bus control and status signals. This mode allows 8-bit external memory and peripheral devices to be interfaced to the system.

- Normal Single-Chip Mode

  There is no external expansion bus in this mode. The processor program is executed from internal memory. Ports A, B, K, and most of E are available as general-purpose I/O.

- Special Single-Chip Mode

  This mode is generally used for debugging single-chip operation, boot-strapping, or security related operations. The active background mode is in control of CPU execution and BDM firmware is waiting for additional serial commands through the BKGD pin. There is no external expansion bus after reset in this mode.

- Emulation Expanded Wide Mode

  Developers use this mode for emulation systems in which the users target application is Normal Expanded Wide Mode.

- Emulation Expanded Narrow Mode

  Developers use this mode for emulation systems in which the users target application is Normal Expanded Narrow Mode.

- Special Test Mode

  Ports A and B are configured as a 16-bit multiplexed address and data bus and Port E provides bus control and status signals. In special test mode, the write protection of many control bits is lifted so that they can be thoroughly tested without needing to go through reset.

- Special Peripheral Mode

  This mode is intended for Motorola factory testing of the system. The CPU is inactive and an external (tester) bus master drives address, data, and bus control signals.

Section 2 External Signal Description

2.1 Overview

In typical implementations, the MEBI sub-block of the Core interfaces directly with external system pins. Some pins may not be bonded out in all implementations.

Table 2-1 outlines the pin names and functions and gives a brief description of their operation Reset state of these pins and associated pull-ups or pull-downs is dependent on the mode of operation and on the integration of this block at the chip level (chip dependent).

Table 2-1 External System Pins Associated With MEBI (Sheet 1 of 3)

Pin Name	Pin Functions	Description
BKGD/MODC/ $\overline{\text{TAGHI}}$	MODC	At the rising edge on $\overline{\text{RESET}}$, the state of this pin is registered into the MODC bit to set the mode. (This pin always has an internal pullup.)
	BKGD	Pseudo open-drain communication pin for the single-wire background debug mode. There is an internal pull-up resistor on this pin.
	$\overline{\text{TAGHI}}$	When instruction tagging is on, a zero at the falling edge of E tags the high half of the instruction word being read into the instruction queue.
PA7/A15/D15/D7 thru PA0/A8/D8/D0	PA7–PA0	General-purpose I/O pins, see PORTA and DDRA registers.
	A15–A8	High-order address lines multiplexed during ECLK low. Outputs except in special peripheral mode where they are inputs from an external tester system.
	D15–D8	High-order bidirectional data lines multiplexed during ECLK high in expanded wide modes, special peripheral mode, and visible internal accesses (IVIS = 1) in emulation expanded narrow mode. Direction of data transfer is generally indicated by R/$\overline{\text{W}}$.
	D15/D7 thru D8/D0	Alternate high-order and low-order bytes of the bidirectional data lines multiplexed during ECLK high in expanded narrow modes and narrow accesses in wide modes. Direction of data transfer is generally indicated by R/$\overline{\text{W}}$.
PB7/A7/D7 thru PB0/A0/D0	PB7–PB0	General-purpose I/O pins, see PORTB and DDRB registers.
	A7–A0	Low-order address lines multiplexed during ECLK low. Outputs except in special peripheral mode where they are inputs from an external tester system.
	D7–D0	Low-order bidirectional data lines multiplexed during ECLK high in expanded wide modes, special peripheral mode, and visible internal accesses (with IVIS = 1) in emulation expanded narrow mode. Direction of data transfer is generally indicated by R/$\overline{\text{W}}$.
PE7/NOACC	PE7	General-purpose I/O pin, see PORTE and DDRE registers.
	NOACC	CPU No Access output. Indicates whether the current cycle is a free cycle. Only available in expanded modes.

Table 2-1 External System Pins Associated With MEBI (Sheet 2 of 3)

Pin Name	Pin Functions	Description
PE6/IPIPE1/ MODB/CLKTO	MODB	At the rising edge of $\overline{\text{RESET}}$, the state of this pin is registered into the MODB bit to set the mode.
	PE6	General-purpose I/O pin, see PORTE and DDRE registers.
	IPIPE1	Instruction pipe status bit 1, enabled by PIPOE bit in PEAR.
	CLKTO	System Clock Test Output. Only available in special modes. PIPOE = 1 overrides this function. The enable for this function is in the clock module.
PE5/IPIPE0/MO DA	MODA	At the rising edge on $\overline{\text{RESET}}$, the state of this pin is registered into the MODA bit to set the mode.
	PE5	General-purpose I/O pin, see PORTE and DDRE registers.
	IPIPE0	Instruction pipe status bit 0, enabled by PIPOE bit in PEAR.
PE4/ECLK	PE4	General-purpose I/O pin, see PORTE and DDRE registers.
	ECLK	Bus timing reference clock, can operate as a free-running clock at the system clock rate or to produce one low-high clock per visible access, with the high period stretched for slow accesses. ECLK is controlled by the NECLK bit in PEAR, the IVIS bit in MODE, and the ESTR bit in EBICTL.
PE3/$\overline{\text{LSTRB}}$/ $\overline{\text{TAGLO}}$	PE3	General-purpose I/O pin, see PORTE and DDRE registers.
	$\overline{\text{LSTRB}}$	Low strobe bar, zero indicates valid data on D7–D0.
	SZ8	In special peripheral mode, this pin is an input indicating the size of the data transfer (0 = 16-bit; 1 = 8-bit).
	$\overline{\text{TAGLO}}$	In expanded wide mode or emulation narrow modes, when instruction tagging is on and low strobe is enabled, a zero at the falling edge of E tags the low half of the instruction word being read into the instruction queue.
PE2/R/$\overline{\text{W}}$	PE2	General-purpose I/O pin, see PORTE and DDRE registers.
	R/$\overline{\text{W}}$	Read/write, indicates the direction of internal data transfers. This is an output except in special peripheral mode where it is an input.
PE1/$\overline{\text{IRQ}}$	PE1	General-purpose input-only pin, can be read even if $\overline{\text{IRQ}}$ enabled.
	$\overline{\text{IRQ}}$	Maskable interrupt request, can be level sensitive or edge sensitive.
PE0/$\overline{\text{XIRQ}}$	PE0	General-purpose input-only pin.
	$\overline{\text{XIRQ}}$	Non-maskable interrupt input.
PK7/$\overline{\text{ECS}}$	PK7	General-purpose I/O pin, see PORTK and DDRK registers.
	$\overline{\text{ECS}}$	Emulation chip select
PK6/$\overline{\text{XCS}}$	PK6	General-purpose I/O pin, see PORTK and DDRK registers.
	$\overline{\text{XCS}}$	External data chip select

Table 2-1 External System Pins Associated With MEBI (Sheet 3 of 3)

Pin Name	Pin Functions	Description
PK5/X19 thru PK0/X14	PK5–PK0	General-purpose I/O pins, see PORTK and DDRK registers.
	X19–X14	Memory expansion addresses

2.2 Detailed Signal Descriptions

Detailed descriptions of these pins can be found in the device specification for the particular chip being used.

Section 3 Memory Map/Register Definition

A summary of the registers associated with the MEBI sub-block is shown in **Figure 3-1**. Detailed descriptions of the registers and bits are given in the subsections that follow.On most chips the registers are mappable. Therefore, the upper bits may not be all zeros as shown in the table and descriptions.

Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
$0000	PORTA	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0001	PORTB	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0002	DDRA	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0003	DDRB	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0004	Reserved	Read	0	0	0	0	0	0	0	0
		Write								
$0005	Reserved	Read	0	0	0	0	0	0	0	0
		Write								
$0006	Reserved	Read	0	0	0	0	0	0	0	0
		Write								
$0007	Reserved	Read	0	0	0	0	0	0	0	0
		Write								
$0008	PORTE	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0009	DDRE	Read	Bit 7	6	5	4	3	2	0	0
		Write								
$000A	PEAR	Read	NOACCE	0	PIPOE	NECLK	LSTRE	RDWE	0	0
		Write								
$000B	MODE	Read	MODC	MODB	MODA	0	IVIS	0	EMK	EME
		Write								
$000C	PUCR	Read	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE
		Write								
$000D	RDRIV	Read	RDPK	0	0	RDPE	0	0	RDPB	RDPA
		Write								
$000E	EBICTL	Read	0	0	0	0	0	0	0	ESTR
		Write								
$000F	Reserved	Read	0	0	0	0	0	0	0	0
		Write								
$001E	IRQCR	Read	IRQE	IRQEN	0	0	0	0	0	0
		Write								
$0032	PORTK	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								
$0033	DDRK	Read	Bit 7	6	5	4	3	2	1	Bit 0
		Write								

= Unimplemented

Figure 3-1 MEBI Register Map Summary

3.1 Register Descriptions

3.1.1 Port A Data Register (PORTA)

Address: Base + $__00

	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	—	—	—	—	—	—	—	—
Single Chip:	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Expanded Wide, Emulation Narrow with IVIS, and Peripheral:	AB/DB15	AB/DB14	AB/DB13	AB/DB12	AB/DB11	AB/DB10	AB/DB9	AB/DB8
Expanded Narrow:	AB15 and DB15/DB7	AB14 and DB14/DB6	AB13 and DB13/DB5	AB12 and DB12/DB4	AB11 and DB11/DB3	AB10 and DB10/DB2	AB9 and DB9/DB1	AB8 and DB8/DB0

Figure 3-2 Port A Data Register (PORTA)

Read: anytime when register is in the map

Write: anytime when register is in the map

Port A bits 7 through 0 are associated with address lines A15 through A8 respectively and data lines D15/D7 through D8/D0 respectively. When this port is not used for external addresses such as in single-chip mode, these pins can be used as general-purpose I/O. Data Direction Register A (DDRA) determines the primary direction of each pin. DDRA also determines the source of data for a read of PORTA.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

NOTE: *To ensure that you read the value present on the PORTA pins, always wait at least one cycle after writing to the DDRA register before reading from the PORTA register.*

3.1.2 Port B Data Register (PORTB)

Address:	Base + $__01							
	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	—	—	—	—	—	—	—	—
Single Chip:	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Expanded Wide, Emulation Narrow with IVIS, and Peripheral:	AB/DB7	AB/DB6	AB/DB5	AB/DB4	AB/DB3	AB/DB2	AB/DB1	AB/DB0
Expanded Narrow:	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0

Figure 3-3 Port B Data Register (PORTB)

Read: anytime when register is in the map

Write: anytime when register is in the map

Port B bits 7 through 0 are associated with address lines A7 through A0 respectively and data lines D7 through D0 respectively. When this port is not used for external addresses, such as in single-chip mode, these pins can be used as general-purpose I/O. Data Direction Register B (DDRB) determines the primary direction of each pin. DDRB also determines the source of data for a read of PORTB.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

NOTE: *To ensure that you read the value present on the PORTB pins, always wait at least one cycle after writing to the DDRB register before reading from the PORTB register.*

3.1.3 Data Direction Register A (DDRA)

Address:	Base + $__02							
	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-4 Data Direction Register A (DDRA)

Read: anytime when register is in the map

Write: anytime when register is in the map

This register controls the data direction for Port A. When Port A is operating as a general-purpose I/O port, DDRA determines the primary direction for each Port A pin. A “1” causes the associated port pin to be an output and a “0” causes the associated pin to be a high-impedance input. The value in a DDR bit also affects the source of data for reads of the corresponding PORTA register. If the DDR bit is zero (input) the buffered pin input state is read. If the DDR bit is one (output) the associated port data register bit state is read.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally. It is reset to $00 so the DDR does not override the three-state control signals.

DDRA7–DDRA0 — Data Direction Port A

1 = Configure the corresponding I/O pin as an output

0 = Configure the corresponding I/O pin as an input

3.1.4 Data Direction Register B (DDRB)

Address: Base + $__03

	BIT 7	6	5	4	3	2	1	BIT 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-5 Data Direction Register B (DDRB)

Read: anytime when register is in the map

Write: anytime when register is in the map

This register controls the data direction for Port B. When Port B is operating as a general-purpose I/O port, DDRB determines the primary direction for each Port B pin. A “1” causes the associated port pin to be an output and a “0” causes the associated pin to be a high-impedance input. The value in a DDR bit also affects the source of data for reads of the corresponding PORTB register. If the DDR bit is zero (input) the buffered pin input state is read. If the DDR bit is one (output) the associated port data register bit state is read.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally. It is reset to $00 so the DDR does not override the three-state control signals.

DDRB7–DDRB0 — Data Direction Port B

1 = Configure the corresponding I/O pin as an output

0 = Configure the corresponding I/O pin as an input

3.1.5 Reserved Registers

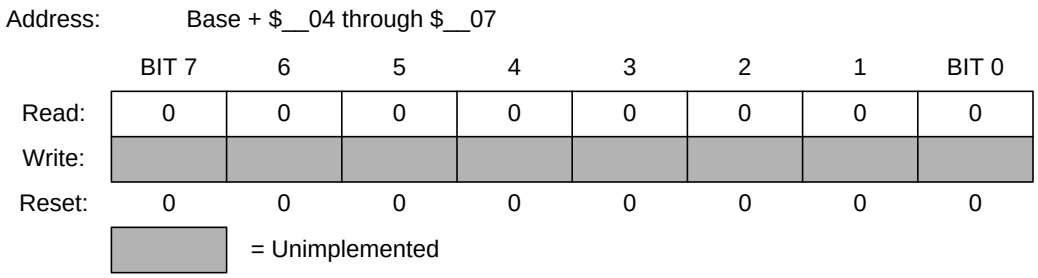


Figure 3-6 Reserved Registers

These register locations are not used (reserved). All unused registers and bits in this block return logic zeros when read. Writes to these registers have no effect.

These registers are not in the on-chip map in special peripheral mode.

3.1.6 Port E Data Register (PORTE)

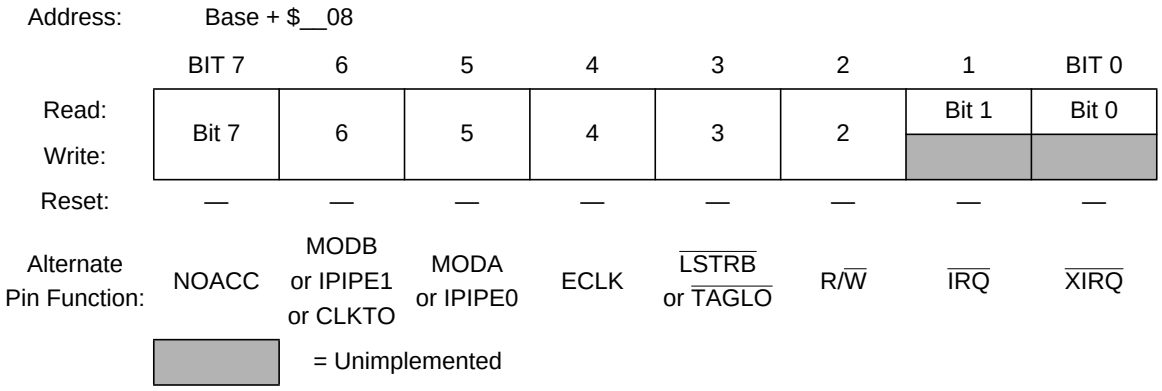


Figure 3-7 Port E Data Register (PORTE)

Read: anytime when register is in the map

Write: anytime when register is in the map

Port E is associated with external bus control signals and interrupt inputs. These include mode select (MODB/IPIPE1, MODA/IPIPE0), E clock, size ($\overline{\text{LSTRB}}$/$\overline{\text{TAGLO}}$), read/write (R/$\overline{\text{W}}$), $\overline{\text{IRQ}}$, and $\overline{\text{XIRQ}}$. When not used for one of these specific functions, Port E pins 7–2 can be used as general-purpose I/O and pins 1–0 can be used as general-purpose input. The Port E Assignment Register (PEAR) selects the function of each pin and DDRE determines whether each pin is an input or output when it is configured to be general-purpose I/O. DDRE also determines the source of data for a read of PORTE.

Some of these pins have software selectable pull-ups (PE7, ECLK, $\overline{\text{LSTRB}}$, R/$\overline{\text{W}}$, $\overline{\text{IRQ}}$, and $\overline{\text{XIRQ}}$). A single control bit enables the pull-ups for all of these pins when they are configured as inputs

This register is not in the on-chip map in special peripheral mode or in expanded modes when the EME bit is set. Therefore, these accesses will be echoed externally.

NOTE: *It is unwise to write PORTE and DDRE as a word access. If you are changing Port E pins from being inputs to outputs, the data may have extra transitions during the write. It is best to initialize PORTE before enabling as outputs.*

NOTE: *To ensure that you read the value present on the PORTE pins, always wait at least one cycle after writing to the DDRE register before reading from the PORTE register.*

3.1.7 Data Direction Register E (DDRE)

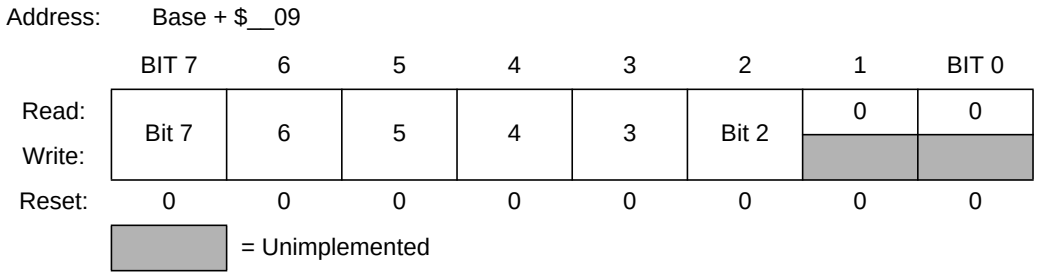


Figure 3-8 Data Direction Register E (DDRE)

Read: anytime when register is in the map

Write: anytime when register is in the map

Data Direction Register E is associated with Port E. For bits in Port E that are configured as general-purpose I/O lines, DDRE determines the primary direction of each of these pins. A “1” causes the associated bit to be an output and a “0” causes the associated bit to be an input. Port E bit 1 (associated with $\overline{\text{IRQ}}$) and bit 0 (associated with $\overline{\text{XIRQ}}$) cannot be configured as outputs. Port E, bits 1 and 0, can be read regardless of whether the alternate interrupt function is enabled. The value in a DDR bit also affects the source of data for reads of the corresponding PORTE register. If the DDR bit is zero (input) the buffered pin input state is read. If the DDR bit is one (output) the associated port data register bit state is read.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally. Also, it is not in the map in expanded modes while the EME control bit is set.

DDRE7–DDRE2 — Data Direction Port E

1 = Configure the corresponding I/O pin as an output

0 = Configure the corresponding I/O pin as an input

NOTE: *It is unwise to write PORTE and DDRE as a word access. If you are changing Port E pins from inputs to outputs, the data may have extra transitions during the write. It is best to initialize PORTE before enabling as outputs.*

3.1.8 Port E Assignment Register (PEAR)

Address: Base + $__0A

	BIT 7	6	5	4	3	2	1	BIT 0	
Read:	NOACCE	0	PIPOE	NECLK	LSTRE	RDWE	0	0	
Write:									
Reset:	0	0	0	0	0	0	0	0	Special Single Chip
Reset:	0	0	1	0	1	1	0	0	Special Test
Reset:	0	0	0	0	0	0	0	0	Peripheral
Reset:	1	0	1	0	1	1	0	0	Emulation Expanded Narrow
Reset:	1	0	1	0	1	1	0	0	Emulation Expanded Wide
Reset:	0	0	0	1	0	0	0	0	Normal Single Chip
Reset:	0	0	0	0	0	0	0	0	Normal Expanded Narrow
Reset:	0	0	0	0	0	0	0	0	Normal Expanded Wide

= Unimplemented

Figure 3-9 Port E Assignment Register (PEAR)

Read: anytime (provided this register is in the map).

Write: each bit has specific write conditions. Please refer to the descriptions of each bit on the following pages.

Port E serves as general-purpose I/O or as system and bus control signals. The PEAR register is used to choose between the general-purpose I/O function and the alternate control functions. When an alternate control function is selected, the associated DDRE bits are overridden.

The reset condition of this register depends on the mode of operation because bus control signals are needed immediately after reset in some modes. In normal single-chip mode, no external bus control signals are needed so all of Port E is configured for general-purpose I/O. In normal expanded modes, only the E clock is configured for its alternate bus control function and the other bits of Port E are configured for general-purpose I/O. As the reset vector is located in external memory, the E clock is required for this access. R/$\overline{W}$ is only needed by the system when there are external writable resources. If the normal expanded system needs any other bus control signals, PEAR would need to be written before any access that needed the additional signals. In special test and emulation modes, IPIPE1, IPIPE0, E, $\overline{LSTRB}$, and R/$\overline{W}$ are configured out of reset as bus control signals.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

NOACCE — CPU No Access Output Enable

Normal: write once

Emulation: write never

Special: write anytime

1 = The associated pin (Port E, bit 7) is output and indicates whether the cycle is a CPU free cycle.
0 = The associated pin (Port E, bit 7) is general-purpose I/O.

This bit has no effect in single-chip or special peripheral modes.

PIPOE — Pipe Status Signal Output Enable

Normal: write once

Emulation: write never

Special: write anytime.

1 = The associated pins (Port E, bits 6:5) are outputs and indicate the state of the instruction queue
0 = The associated pins (Port E, bits 6:5) are general-purpose I/O.

This bit has no effect in single-chip or special peripheral modes.

NECLK — No External E Clock

Normal and Special: write anytime

Emulation: write never

1 = The associated pin (Port E, bit 4) is a general-purpose I/O pin.
0 = The associated pin (Port E, bit 4) is the external E clock pin. External E clock is free-running if ESTR = 0

External E clock is available as an output in all modes.

LSTRE — Low Strobe ($\overline{\text{LSTRB}}$) Enable

Normal: write once

Emulation: write never

Special: write anytime.

1 = The associated pin (Port E, bit 3) is configured as the $\overline{\text{LSTRB}}$ bus control output. If BDM tagging is enabled, $\overline{\text{TAGLO}}$ is multiplexed in on the rising edge of ECLK and $\overline{\text{LSTRB}}$ is driven out on the falling edge of ECLK.
0 = The associated pin (Port E, bit 3) is a general-purpose I/O pin.

This bit has no effect in single-chip, peripheral, or normal expanded narrow modes.

NOTE: *$\overline{\text{LSTRB}}$ is used during external writes. After reset in normal expanded mode, $\overline{\text{LSTRB}}$ is disabled to provide an extra I/O pin. If $\overline{\text{LSTRB}}$ is needed, it should be enabled before any external writes. External reads do not normally need $\overline{\text{LSTRB}}$ because all 16 data bits can be driven even if the system only needs 8 bits of data.*

RDWE — Read/Write Enable

Normal: write once

Emulation: write never

Special: write anytime

1 = The associated pin (Port E, bit 2) is configured as the R/$\overline{W}$ pin

0 = The associated pin (Port E, bit 2) is a general-purpose I/O pin.

This bit has no effect in single-chip or special peripheral modes.

NOTE: *R/$\overline{W}$ is used for external writes. After reset in normal expanded mode, R/$\overline{W}$ is disabled to provide an extra I/O pin. If R/$\overline{W}$ is needed it should be enabled before any external writes.*

3.1.9 MODE Register (MODE)

Address: Base + $__0B

	BIT 7	6	5	4	3	2	1	BIT 0	
Read:	MODC	MODB	MODA	0	IVIS	0	EMK	EME	
Write:									
Reset:	0	0	0	0	0	0	0	0	Special Single Chip
Reset:	0	0	1	0	1	0	1	1	Emulation Expanded Narrow
Reset:	0	1	0	0	1	0	0	0	Special Test
Reset:	0	1	1	0	1	0	1	1	Emulation Expanded Wide
Reset:	1	0	0	0	0	0	0	0	Normal Single Chip
Reset:	1	0	1	0	0	0	0	0	Normal Expanded Narrow
Reset:	1	1	0	0	0	0	0	0	Peripheral
Reset:	1	1	1	0	0	0	0	0	Normal Expanded Wide

= Unimplemented

Figure 3-10 MODE Register (MODE)

Read: anytime (provided this register is in the map).

Write: each bit has specific write conditions. Please refer to the descriptions of each bit on the following pages.

The MODE register is used to establish the operating mode and other miscellaneous functions (i.e., internal visibility and emulation of Port E and K).

In special peripheral mode, this register is not accessible but it is reset as shown to system configuration features. Changes to bits in the MODE register are delayed one cycle after the write.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

MODC, MODB, and MODA — Mode Select Bits

These bits indicate the current operating mode.

If MODA = 1, then MODC, MODB, and MODA are write never.

If MODC = MODA = 0, then MODC, MODB, and MODA are writable with the exception that you cannot change to or from special peripheral mode

If MODC = 1, MODB = 0, and MODA = 0, then MODC is write never. MODB and MODA are write once, except that you cannot change to special peripheral mode. From normal single-chip, only normal expanded narrow and normal expanded wide modes are available.

Table 3-1 MODC, MODB, and MODA Write Capability(1)

MODC	MODB	MODA	Mode	MODx Write Capability
0	0	0	Special Single Chip	MODC, MODB, and MODA write anytime but not to 110(2)
0	0	1	Emulation Narrow	No write
0	1	0	Special Test	MODC, MODB, and MODA write anytime but not to 110(2)
0	1	1	Emulation Wide	No write
1	0	0	Normal Single Chip	MODC write never, MODB and MODA write once but not to 110
1	0	1	Normal Expanded Narrow	No write
1	1	0	Special Peripheral	No write
1	1	1	Normal Expanded Wide	No write

NOTES:

1. No writes to the MOD bits are allowed while operating in a SECURE mode. For more details, refer to the device user guide.
2. If you are in a special single-chip or special test mode and you write to this register, changing to normal single-chip mode, then one allowed write to this register remains. If you write to normal expanded or emulation mode, then no writes remain.

Table 3-2 Mode Select and State of Mode Bits

Input BKGD and Bit MODC	Input and Bit MODB	Input and Bit MODA	Mode Description
0	0	0	Special Single Chip, BDM allowed and ACTIVE. BDM is “allowed” in all other modes but a serial command is required to make BDM “active”.
0	0	1	Emulation Expanded Narrow, BDM allowed
0	1	0	Special Test (Expanded Wide), BDM allowed
0	1	1	Emulation Expanded Wide, BDM allowed
1	0	0	Normal Single Chip, BDM allowed
1	0	1	Normal Expanded Narrow, BDM allowed
1	1	0	Peripheral, BDM allowed but bus operations would cause bus conflicts (must not be used)
1	1	1	Normal Expanded Wide, BDM allowed

IVIS — Internal Visibility (for both read and write accesses)

This bit determines whether internal accesses generate a bus cycle that is visible on the external bus.

Normal: write once

Emulation: write never

Special: write anytime

1 = Internal bus operations are visible on external bus.

0 = No visibility of internal bus operations on external bus.

EMK — Emulate Port K

Normal: write once

Emulation: write never

Special: write anytime

1 = If in any expanded mode, PORTK and DDRK are removed from the memory map.

0 = PORTK and DDRK are in the memory map so Port K can be used for general-purpose I/O.

In single-chip modes, PORTK and DDRK are always in the map regardless of the state of this bit.

In special peripheral mode, PORTK and DDRK are never in the map regardless of the state of this bit.

EME — Emulate Port E

Normal and Emulation: write never

Special: write anytime

1 = If in any expanded mode or special peripheral mode, PORTE and DDRE are removed from the memory map. Removing the registers from the map allows the user to emulate the function of these registers externally.

0 = PORTE and DDRE are in the memory map so Port E can be used for general-purpose I/O.

In single-chip modes, PORTE and DDRE are always in the map regardless of the state of this bit.

3.1.10 Pull-Up Control Register (PUCR)

Address: Base + $__0C

	BIT 7	6	5	4	3	2	1	BIT 0
Read:	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE
Write:								
Reset:(1)	1	0	0	1	0	0	0	0

= Unimplemented

NOTES:
1. The default value of this parameter is shown. Please refer to the specific device User's Guide to determine the actual reset state of this register.

Figure 3-11 Pullup Control Register (PUCR)

Read: anytime (provided this register is in the map).

Write: anytime (provided this register is in the map).

This register is used to select pull resistors for the pins associated with the core ports. Pull resistors are assigned on a per-port basis and apply to any pin in the corresponding port that is currently configured as an input. The polarity of these pull resistors is determined by chip integration. Please refer to the specific device User's Guide to determine the polarity of these resistors.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

NOTE: *These bits have no effect when the associated pin(s) are outputs. (The pull resistors are inactive.)*

PUPKE — Pull-Up Port K Enable
1 = Enable pull resistors for Port K input pins.
0 = Port K pull resistors are disabled.

PUPEE — Pull-Up Port E Enable
1 = Enable pull resistors for Port E input pins bits 7, 4–0.
0 = Port E pull resistors on bits 7, 4–0 are disabled.

NOTE: *Bits 5 and 6 of Port E have pull resistors which are only enabled during reset. This bit has no effect on these pins.*

PUPBE — Pull-Up Port B Enable
1 = Enable pull resistors for all Port B input pins.
0 = Port B pull resistors are disabled.

PUPAE — Pull-Up Port A Enable
1 = Enable pull resistors for all Port A input pins.

0 = Port A pull resistors are disabled.

3.1.11 Reduced Drive Register (RDRIV)

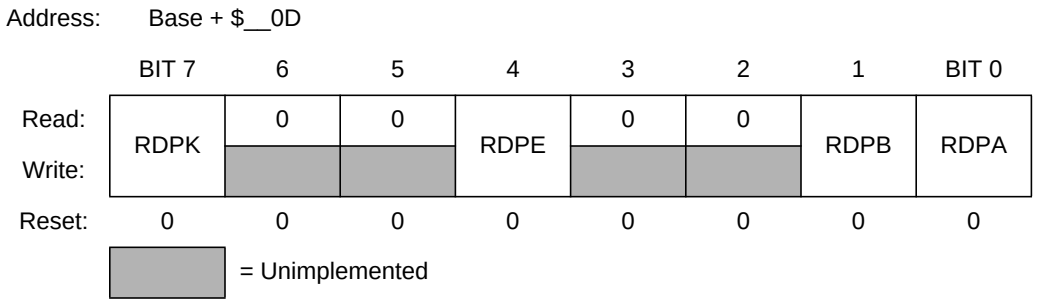


Figure 3-12 Reduced Drive Register (RDRIV)

Read: anytime (provided this register is in the map)

Write: anytime (provided this register is in the map)

This register is used to select reduced drive for the pins associated with the core ports. This gives reduced power consumption and reduced RFI with a slight increase in transition time (depending on loading). This feature would be used on ports which have a light loading. The reduced drive function is independent of which function is being used on a particular port.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

RDPK — Reduced Drive of Port K
1 = All Port K output pins have reduced drive enabled.
0 = All Port K output pins have full drive enabled.

RDPE — Reduced Drive of Port E
1 = All Port E output pins have reduced drive enabled.
0 = All Port E output pins have full drive enabled.

RDPB — Reduced Drive of Port B
1 = All Port B output pins have reduced drive enabled.
0 = All Port B output pins have full drive enabled.

RDPA — Reduced Drive of Ports A
1 = All Port A output pins have reduced drive enabled.
0 = All Port A output pins have full drive enabled.

3.1.12 External Bus Interface Control Register (EBICTL)

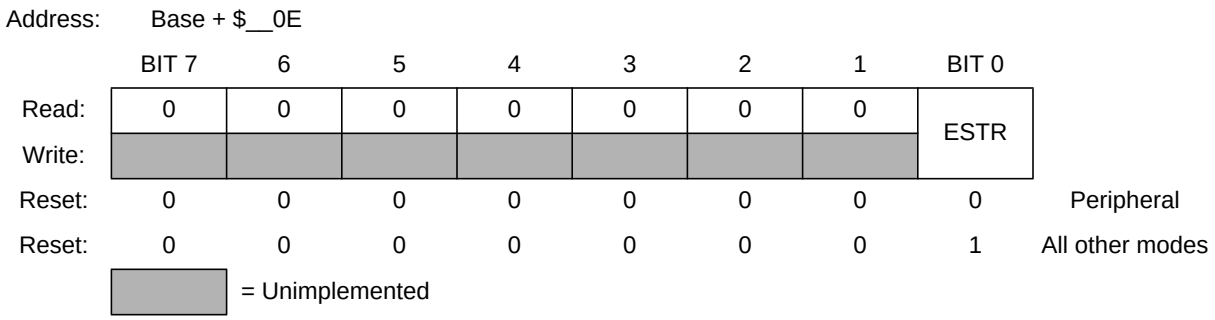


Figure 3-13 External Bus Interface Control Register (EBICTL)

Read: anytime (provided this register is in the map)

Write: refer to individual bit descriptions below

The EBICTL register is used to control miscellaneous functions (i.e., stretching of external E clock).

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

ESTR — E Clock Stretches

This control bit determines whether the E clock behaves as a simple free-running clock or as a bus control signal that is active only for external bus cycles.

Normal and Emulation: write once

Special: write anytime

1 = E stretches high during stretched external accesses and remains low during non-visible internal accesses.

0 = E never stretches (always free running).

This bit has no effect in single-chip modes.

3.1.13 Reserved Register

Address:	Base + $__0F							
Read:	0	0	0	0	0	0	0	0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-14 Reserved Register

This register location is not used (reserved). All bits in this register return logic zeros when read. Writes to this register have no effect.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

3.1.14 IRQ Control Register (IRQCR)

Address	Base + $__1E							
	Bit 7	6	5	4	3	2	1	Bit 0
Read:	IRQE	IRQEN	0	0	0	0	0	0
Write:								
Reset:	0	1	0	0	0	0	0	0

= Unimplemented

Figure 3-15 IRQ Control Register (IRQCR)

Read: see individual bit descriptions below

Write: see individual bit descriptions below

IRQE — IRQ Select Edge Sensitive Only

Special modes: read or write anytime

Normal and Emulation modes: read anytime, write once

1 = IRQ configured to respond only to falling edges. Falling edges on the IRQ pin will be detected anytime IRQE = 1 and will be cleared only upon a reset or the servicing of the IRQ interrupt.

0 = IRQ configured for low level recognition.

IRQEN — External IRQ Enable

Normal, Emulation, and Special modes: read or write anytime

1 = External IRQ pin is connected to interrupt logic.

0 = External IRQ pin is disconnected from interrupt logic.

NOTE: *When IRQEN = 0, the edge detect latch is disabled.*

3.1.15 Port K Data Register (PORTK).

Address:	Base + $32							
	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Alternate Pin Function	ECS	XCS	XAB19	XAB18	XAB17	XAB16	XAB15	XAB14
Reset:	—	—	—	—	—	—	—	—

Figure 3-16 Port K Data Register (PORTK)

Read: anytime

Write: anytime

This port is associated with the internal memory expansion emulation pins. When the port is not enabled to emulate the internal memory expansion, the port pins are used as general-purpose I/O. When Port K is operating as a general-purpose I/O port, DDRK determines the primary direction for each Port K pin. A “1” causes the associated port pin to be an output and a “0” causes the associated pin to be a high-impedance input. The value in a DDR bit also affects the source of data for reads of the corresponding PORTK register. If the DDR bit is zero (input) the buffered pin input is read. If the DDR bit is one (output) the output of the port data register is read.

This register is not in the map in peripheral or expanded modes while the EMK control bit in MODE register is set. Therefore, these accesses will be echoed externally.

When inputs, these pins can be selected to be high impedance or pulled up, based upon the state of the PUPKE bit in the PUCR register.

Bit 7 — Port K, Bit 7

This bit is used as an emulation chip select signal for the emulation of the internal memory expansion, or as general-purpose I/O, depending upon the state of the EMK bit in the MODE register. While this bit is used as a chip select, the external bit will return to its de-asserted state (V_{DD}) for approximately 1/4 cycle just after the negative edge of ECLK, unless the external access is stretched and ECLK is free-running (ESTR bit in EBICTL = 0). See the HCS12V1.5 MMC specification for additional details on when this signal will be active.

Bit 6 — Port K, Bit 6

This bit is used as an external chip select signal for most external accesses that are not selected by $\overline{\text{ECS}}$ (see the MMC specification for more details), depending upon the state the of the EMK bit in the MODE register. While this bit is used as a chip select, the external pin will return to its de-asserted state (V_{DD}) for approximately 1/4 cycle just after the negative edge of ECLK, unless the external access is stretched and ECLK is free-running (ESTR bit in EBICTL = 0).

Bits 5–0 — Port K, Bits 5–0

These six bits are used to determine which FLASH/ROM or external memory array page is being accessed. They can be viewed as expanded addresses XAB19–XAB14 of the 20-bit address used to access up to1M byte internal FLASH/ROM or external memory array. Alternatively, these bits can be used for general-purpose I/O depending upon the state of the EMK bit in the MODE register.

3.1.16 Port K Data Direction Register (DDRK)

Address: Base + $33

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 7	Bit 6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-17 Port K Data Direction Register (DDRK)

Read: anytime.

Write: anytime.

This register determines the primary direction for each Port K pin configured as general-purpose I/O. This register is not in the map in peripheral or expanded modes while the EMK control bit in MODE register is set. Therefore, these accesses will be echoed externally.

DDRK7–DDRK0 — Data Direction Port K Bits

1 = Associated pin is an output

0 = Associated pin is a high-impedance input

NOTE: *It is unwise to write PORTK and DDRK as a word access. If you are changing Port K pins from inputs to outputs, the data may have extra transitions during the write. It is best to initialize PORTK before enabling as outputs.*

NOTE: *To ensure that you read the correct value from the PORTK pins, always wait at least one cycle after writing to the DDRK register before reading from the PORTK register.*

Section 4 Functional Description

There are four main sub-blocks within the MEBI:

- External bus control
- External data bus interface
- Control
- Registers

4.1 External Bus Control

The external bus control generates the miscellaneous control functions (pipe signals, ECLK, $\overline{\text{LSTRB}}$, and R/$\overline{\text{W}}$) that will be sent external on Port E, bits 6–2. It also generates the external addresses.

4.1.1 Detecting Access Type from External Signals

The external signals $\overline{\text{LSTRB}}$, R/$\overline{\text{W}}$, and AB0 indicate the type of bus access that is taking place. Accesses to the internal RAM module are the only type of access that would produce $\overline{\text{LSTRB}}$ = AB0 = 1, because the internal RAM is specifically designed to allow misaligned 16-bit accesses in a single cycle. In these cases the data for the address that was accessed is on the low half of the data bus and the data for address + 1 is on the high half of the data bus. This is summarized in **Table 4-1**.

Table 4-1 Access Type vs. Bus Control Pins

$\overline{\text{LSTRB}}$	AB0	R/$\overline{\text{W}}$	Type of Access
1	0	1	8-bit read of an even address
0	1	1	8-bit read of an odd address
1	0	0	8-bit write of an even address
0	1	0	8-bit write of an odd address
0	0	1	16-bit read of an even address
1	1	1	16-bit read of an odd address (low/high data swapped)
0	0	0	16-bit write to an even address
1	1	0	16-bit write to an odd address (low/high data swapped)

4.1.2 Stretched Bus Cycles

In order to allow fast internal bus cycles to coexist in a system with slower external memory resources, the HCS12 supports the concept of stretched bus cycles (module timing reference clocks for timers and baud rate generators are not affected by this stretching). Control bits in the MISC register in the MMC sub-block of the Core specify the amount of stretch (0, 1, 2, or 3 periods of the internal bus-rate clock). While stretching, the CPU state machines are all held in their current state. At this point in the CPU bus cycle, write data would already be driven onto the data bus so the length of time write data is valid is extended in the case of a stretched bus cycle Read data would not be captured by the system until the E clock falling edge. In the case of a stretched bus cycle, read data is not required until the specified setup time before the falling edge of the stretched E clock. The chip selects, and R/$\overline{W}$ signals remain valid during the period of stretching (throughout the stretched E high time).

NOTE: *The address portion of the bus cycle is not stretched!*

4.2 External Data Bus Interface

The external data bus interface block manages data transfers from/to the external pins to/from the internal read and write data buses. This block selectively couples 8-bit or 16-bit data to the internal data bus to implement a variety of data transfers including 8-bit, 16-bit, 16-bit swapped, and 8-bit external to 16-bit internal accesses. Modes, addresses, chip selects, etc. affect the type of accesses performed during each bus cycle.

4.2.1 Internal Visibility

Internal visibility is available when the system is operating in expanded wide modes, special test mode, or emulation narrow mode. It is not available in single-chip, peripheral, or normal expanded narrow modes. Internal visibility is enabled by setting the IVIS bit in the MODE register.

If an internal access is made while E, R/$\overline{W}$, and $\overline{LSTRB}$ are configured as bus control outputs and internal visibility is off (IVIS = 0), E will remain low for the cycle, R/$\overline{W}$ will remain high, and the $\overline{LSTRB}$ pins will remain at their previous state. The address bus is not affected by the IVIS function, as address information is always driven.

When internal visibility is enabled (IVIS = 1), certain internal cycles will be blocked from going external to prevent possible corruption of external devices. Specifically, during cycles when the BDM is selected, R/$\overline{W}$ will remain high, data will maintain its previous state, and address and $\overline{LSTRB}$ pins will be updated with the internal value. During CPU no access cycles when the BDM is not driving, R/$\overline{W}$ will remain high, and address, data, and the $\overline{LSTRB}$ pins will remain at their previous state.

4.2.2 Secure Mode

When the system is operating in a secure mode, internal visibility is not available (i.e., IVIS = 1 has no effect). Also, the IPIPE signals will not be visible, regardless of operating mode. IPIPE1–IPIPE0 will display zeroes if they are enabled. In addition, the MOD bits in the MODE control register cannot be written.

4.3 Control

The control block generates the register read/write control signals and miscellaneous port control signals.

4.3.1 Low-Power Options

The MEBI does not contain any user-controlled options for reducing power consumption. The operation of the MEBI in low-power modes is discussed in the following subsections.

4.3.1.1 Run Mode

The MEBI does not contain any options for reducing power in run mode; however, the external addresses are conditioned to reduce power in single-chip modes. Expanded bus modes will increase power consumption.

4.3.1.2 Wait Mode

The MEBI does not contain any options for reducing power in wait mode.

4.3.1.3 Stop Mode

The MEBI will cease to function after execution of a CPU STOP instruction.

4.4 Registers

The register block includes the fourteen 8-bit registers and five reserved register locations associated with the MEBI sub-block.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

S12MEBIV3/D
Rev. 3.00
2/2003

MOTOROLA
intelligence everywhere™

digital dna™

Module Mapping Control (MMC) V4

HCS12 Microcontrollers

S12MMCV4/D
Rev. 4.00
2/2003

MOTOROLA.COM/SEMICONDUCTORS

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
4.00	2/4/2003	2/4/2003	John Langan	Creation of block user guide from core user guide version 1.5 (July 2, 2002). Changes include: updating format and making end customer friendly. Original release.

Table of Contents

List of Figures

List of Tables

Section 1 Introduction to Module Mapping Control (MMC)

Section 2 External Signal Description

Section 3 Memory Map/Register Definition

Section 4 Functional Description

Section 1 Introduction to Module Mapping Control (MMC)

This section describes the functionality of the Module Mapping Control (MMC) sub-block of the S12 Core Platform.

The block diagram of the MMC is shown in **Figure 1-1**.

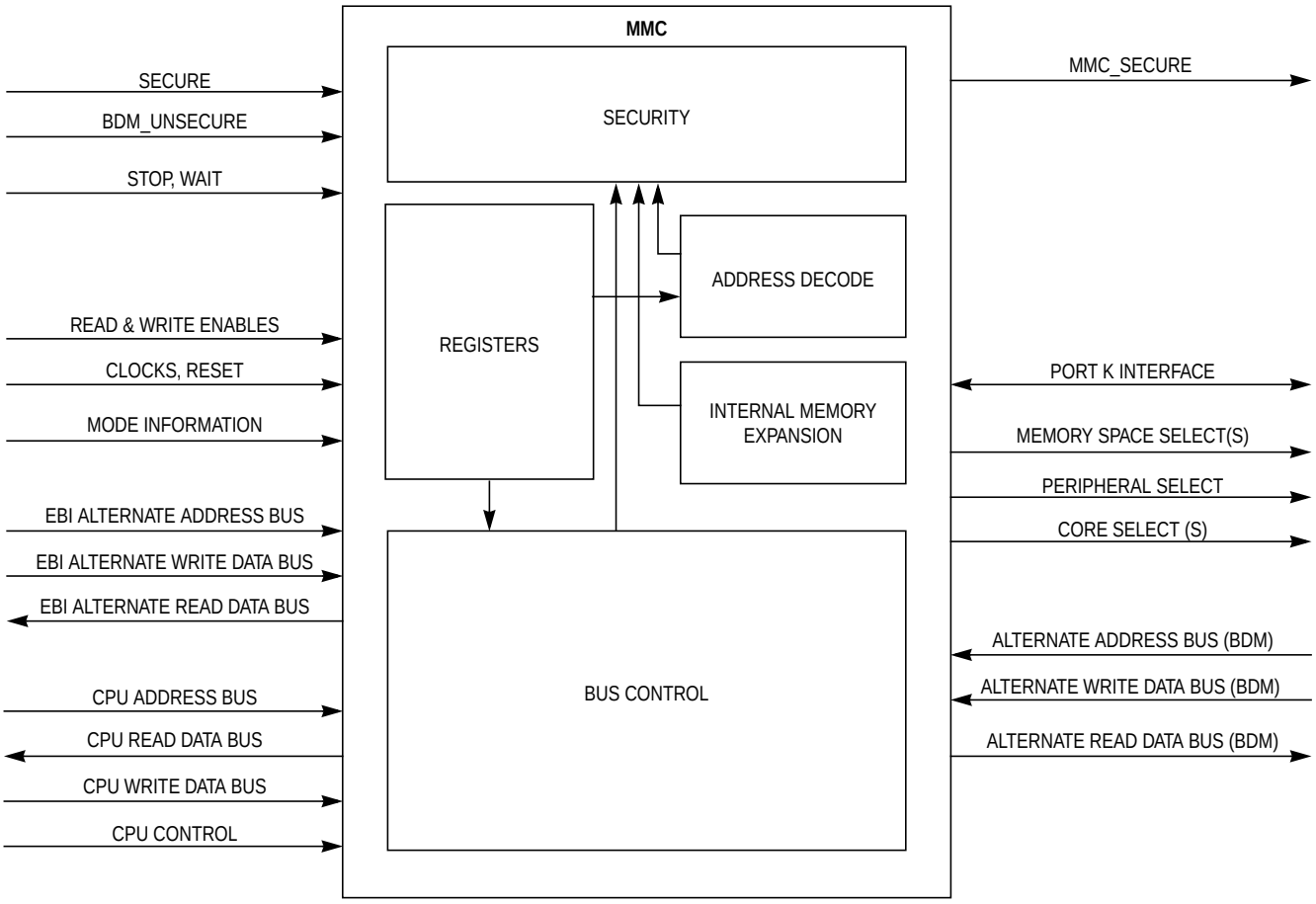


Figure 1-1 Module Mapping Control Block Diagram

1.1 Overview

The MMC is the submodule which controls memory map assignment and selection of internal resources and external space. Internal buses between the core and memories and between the core and peripherals is controlled in this module. The memory expansion is generated in this module.

1.2 Features

- Registers for mapping of address space for on-chip RAM, EEPROM, and FLASH (or ROM) memory blocks and associated registers
- Memory mapping control and selection based upon address decode and system operating mode
- Core address bus control
- Core data bus control and multiplexing
- Core security state decoding
- Emulation chip select signal generation ($\overline{\text{ECS}}$)
- External chip select signal generation ($\overline{\text{XCS}}$)
- Internal memory expansion
- External stretch and ROM mapping control functions via the MISC register
- Reserved registers for test purposes
- Configurable system memory options defined at integration of Core into the System-on-a-Chip (SOC).

1.3 Modes of Operation

Some of the registers operate differently depending on the mode of operation (i.e., normal expanded wide, special single chip, etc.). This is best understood from the register descriptions.

Section 2 External Signal Description

All interfacing with the MMC sub-block is done within the Core, it has no external signals.

Section 3 Memory Map/Register Definition

A summary of the registers associated with the MMC sub-block is shown in **Figure 3-1**. Detailed descriptions of the registers and bits are given in the subsections that follow.

Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
$0010	INITRM	Read	RAM15	RAM14	RAM13	RAM12	RAM11	0	0	RAMHAL
		Write								
$0011	INITRG	Read	0	REG14	REG13	REG12	REG11	0	0	0
		Write								
$0012	INITEE	Read	EE15	EE14	EE13	EE12	EE11	0	0	EEON
		Write								
$0013	MISC	Read	0	0	0	0	EXSTR1	EXSTR0	ROMHM	ROMON
		Write								
$0014	MTSTO	Read	BIT 7	6	5	4	3	2	1	BIT 0
		Write								
$0017	MTST1	Read	BIT 7	6	5	4	3	2	1	BIT 0
		Write								
$001C	MEMSIZ0	Read	REG_SW0	0	EEP_SW1	EEP_SW0	0	RAM_SW2	RAM_SW1	RAM_SW0
		Write								
$001D	MEMSIZ1	Read	ROM_SW1	ROM_SW0	0	0	0	0	PAG_SW1	PAG_SW0
		Write								
$0030	PPAGE	Read	0	0	PIX5	PIX4	PIX3	PIX2	PIX1	PIX0
		Write								
$0031	Reserved	Read	0	0	0	0	0	0	0	0
		Write								

= Unimplemented

Figure 3-1 Module Mapping Control Register Summary

3.1 Register Descriptions

3.1.1 Initialization of Internal RAM Position Register (INITRM)

Register address Base + $10

	7	6	5	4	3	2	1	0
R	RAM15	RAM14	RAM13	RAM12	RAM11	0	0	RAMHAL
W								
Reset	0	0	0	0	1	0	0	1

= Unimplemented or Reserved

Figure 3-2 Initialization of Internal RAM Position Register (INITRM)

Read: Anytime

Write: Once in Normal and Emulation Modes, anytime in Special Modes

NOTE: *Writes to this register take one cycle to go into effect.*

This register initializes the position of the internal RAM within the on-chip system memory map.

RAM15–RAM11 — Internal RAM Map Position

These bits determine the upper five bits of the base address for the system's internal RAM array.

RAMHAL — RAM High-Align

RAMHAL specifies the alignment of the internal RAM array.
0 = Aligns the RAM to the lowest address ($0000) of the mappable space
1 = Aligns the RAM to the higher address ($FFFF) of the mappable space

3.1.2 Initialization of Internal Registers Position Register (INITRG)

Register address Base + $11

	7	6	5	4	3	2	1	0
R	0	REG14	REG13	REG12	REG11	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 Initialization of Internal Registers Position Register (INITRG)

Read: Anytime

Write: Once in Normal and Emulation modes and anytime in Special modes

This register initializes the position of the internal registers within the on-chip system memory map. The registers occupy either a 1K byte or 2K byte space and can be mapped to any 2K byte space within the first 32K bytes of the system's address space.

REG14–REG11 — Internal Register Map Position

These four bits in combination with the leading zero supplied by bit 7 of INITRG determine the upper five bits of the base address for the system's internal registers (i.e., the minimum base address is $0000 and the maximum is $7FFF).

3.1.3 Initialization of Internal EEPROM Position Register (INITEE)

Register address Base + $12

	7	6	5	4	3	2	1	0
R	EE15	EE14	EE13	EE12	EE11	0	0	EEON
W								
Reset(1)	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

NOTES:
1. The reset state of this register is controlled at chip integration. Please refer to the specific device User's Guide to determine the actual reset state of this register.

Figure 3-4 Initialization of Internal EEPROM Position Register (INITEE)

Read: Anytime

Write: The EEON bit can be written to any time on all devices. Bits E11–E15 are "Write anytime in all modes" on most devices. On some devices, bits E11–E15 are "Write once in Normal and Emulation modes and write anytime in Special modes". See device User's Guide to determine the actual write access rights.

NOTE: *Writes to this register take one cycle to go into effect.*

This register initializes the position of the internal EEPROM within the on-chip system memory map.

EE15–EE11 — Internal EEPROM Map Position

These bits determine the upper five bits of the base address for the system's internal EEPROM array.

EEON — Enable EEPROM

This bit is used to enable the EEPROM memory in the memory map.
1 = Enables the EEPROM in the memory map at the address selected by EE15–EE11.
0 = Disables the EEPROM from the memory map.

3.1.4 Miscellaneous System Control Register (MISC)

Register address Base + $13

	7	6	5	4	3	2	1	0
R	0	0	0	0	EXSTR1	EXSTR0	ROMHM	ROMON
W								
Reset: Expanded or Emulation	0	0	0	0	1	1	0	—(1)
Reset: Peripheral or Single Chip	0	0	0	0	1	1	0	1
Reset: Special Test	0	0	0	0	1	1	0	0

= Unimplemented or Reserved

NOTES:
1. The reset state of this bit is determined at the chip integration level.

Figure 3-5 Miscellaneous System Control Register (MISC)

Read: Anytime

Write: As stated in each bit description

NOTE: *Writes to this register take one cycle to go into effect*

This register initializes miscellaneous control functions.

EXSTR1 and EXSTR0 — External Access Stretch Bits 1 and 0

Write: Once in Normal and Emulation modes and anytime in Special modes

This two bit field determines the amount of clock stretch on accesses to the external address space as shown in **Table 3-1**. In single chip and peripheral modes these bits have no meaning or effect.

Table 3-1 External Stretch Bit Definition

Stretch Bit EXSTR1	Stretch Bit EXSTR0	Number of E Clocks Stretched
0	0	0
0	1	1
1	0	2
1	1	3

ROMHM — FLASH EEPROM or ROM Only in Second Half of Memory Map

Write: Once in Normal and Emulation modes and anytime in Special modes

1 = Disables direct access to the FLASH EEPROM or ROM in the lower half of the memory map. These physical locations of the FLASH EEPROM or ROM can still be accessed through the Program Page window.

0 = The fixed page(s) of FLASH EEPROM or ROM in the lower half of the memory map can be accessed.

ROMON — Enable FLASH EEPROM or ROM

Write: Once in Normal and Emulation modes and anytime in Special modes

This bit is used to enable the FLASH EEPROM or ROM memory in the memory map.

1 = Enables the FLASH EEPROM or ROM in the memory map.

0 = Disables the FLASH EEPROM or ROM from the memory map.

3.1.5 Reserved Test Register Zero (MTST0)

Register address Base + $14

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 Reserved Test Register Zero (MTST0)

Read: Anytime

Write: No effect — this register location is used for internal test purposes.

3.1.6 Reserved Test Register One (MTST1)

Register address Base + $17

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	1	0	0	0	0

= Unimplemented or Reserved

Figure 3-7 Reserved Test Register One (MTST1)

Read: Anytime

Write: No effect — this register location is used for internal test purposes.

3.1.7 Memory Size Register Zero (MEMSIZ0)

Register address Base + $1C

	7	6	5	4	3	2	1	0
R	REG_SW0	0	EEP_SW1	EEP_SW0	0	RAM_SW2	RAM_SW1	RAM_SW0
W								
Reset	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

Figure 3-8 Memory Size Register Zero

Read: Anytime

Write: Writes have no effect

Reset: Defined at chip integration, see chip level documentation.

The MEMSIZ0 register reflects the state of the register, EEPROM and RAM memory space configuration switches at the Core boundary which are configured at system integration. This register allows read visibility to the state of these switches.

REG_SW0 — Allocated System Register Space

1 = Allocated system register space size is 2K byte
0 = Allocated system register space size is 1K byte

EEP_SW1:EEP_SW0 — Allocated System EEPROM Memory Space

The allocated system EEPROM memory space size is as given in **Table 3-2**.

Table 3-2 Allocated EEPROM Memory Space

EEP_SW1:EEP_SW0	Allocated EEPROM Space
00	0K byte
01	2K bytes
10	4K bytes
11	8K bytes

RAM_SW2:RAM_SW0 — Allocated System RAM Memory Space

The allocated system RAM memory space size is as given in **Table 3-3**.

Table 3-3 Allocated RAM Memory Space

RAM_SW2:RAM_SW0	Allocated RAM Space	RAM Mappable Region	INITRM Bits Used	RAM Reset Base Address(1)
000	2K bytes	2K bytes	RAM15–RAM11	$0800
001	4K bytes	4K bytes	RAM15–RAM12	$0000
010	6K bytes	8K bytes(2)	RAM15–RAM13	$0800
011	8K bytes	8K bytes	RAM15–RAM13	$0000
100	10K bytes	16K bytes2	RAM15–RAM14	$1800
101	12K bytes	16K bytes2	RAM15–RAM14	$1000
110	14K bytes	16K bytes2	RAM15–RAM14	$0800
111	16K bytes	16K bytes	RAM15–RAM14	$0000

NOTES:
1. The RAM Reset BASE Address is based on the reset value of the INITRM register, $09.
2. Alignment of the Allocated RAM space within the RAM mappable region is dependent on the value of RAMHAL.

NOTE: *As stated, the bits in this register provide read visibility to the system physical memory space allocations defined at system integration. The actual array size for any given type of memory block may differ from the allocated size. Please refer to the chip-level documentation for actual sizes.*

3.1.8 Memory Size Register One (MEMSIZ1)

Register address Base + $1D

	7	6	5	4	3	2	1	0
R	ROM_SW1	ROM_SW0	0	0	0	0	PAG_SW1	PAG_SW0
W								
Reset	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

Figure 3-9 Memory Size Register One

Read: Anytime

Write: Writes have no effect

Reset: Defined at chip integration, see chip level documentation.

The MEMSIZ1 register reflects the state of the FLASH or ROM physical memory space and paging switches at the Core boundary which are configured at system integration. This register allows read visibility to the state of these switches.

ROM_SW1:ROM_SW0 — Allocated System FLASH or ROM Physical Memory Space

The allocated system FLASH or ROM physical memory space is as given in **Table 3-4**.

Table 3-4 Allocated FLASH/ROM Physical Memory Space

ROM_SW1:ROM_SW0	Allocated FLASH or ROM Space
00	0K byte
01	16K bytes
10	48K bytes(1)
11	64K bytes(1)

NOTES:
1. The ROMHM software bit in the MISC register determines the accessibility of the FLASH/ROM memory space. Please refer to **3.1.8 Memory Size Register One (MEMSIZ1)** for a detailed functional description of the ROMHM bit.

PAG_SW1:PAG_SW0 — Allocated Off-Chip FLASH or ROM Memory Space

The allocated off-chip FLASH or ROM memory space size is as given in **Table 3-5**.

Table 3-5 Allocated Off-Chip Memory Options

PAG_SW1:PAG_SW0	Off-Chip Space	On-Chip Space
00	876K bytes	128K bytes
01	768K bytes	256K bytes
10	512K bytes	512K bytes
11	0K byte	1M byte

NOTE: *As stated, the bits in this register provide read visibility to the system memory space and on-chip/off-chip partitioning allocations defined at system integration. The actual array size for any given type of memory block may differ from the allocated size. Please refer to the chip-level documentation for actual sizes.*

3.1.9 Program Page Index Register (PPAGE)

Register address Base + $30

	7	6	5	4	3	2	1	0
R	0	0	PIX5	PIX4	PIX3	PIX2	PIX1	PIX0
W								
Reset(1)	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

NOTES:
1. The reset state of this register is controlled at chip integration. Please refer to the specific device User's Guide to determine the actual reset state of this register.

Figure 3-10 Program Page Index Register (PPAGE)

Read: Anytime
Write: Determined at chip integration. Generally it's: "Write anytime in all modes"; on some devices it will be: "Write only in Special modes". Check specific device documentation to determine which applies.
Reset: Defined at chip integration as either $00 (paired with write in any mode) or $3C (paired with write only in special modes), see chip level documentation.

The HCS12 Core architecture limits the physical address space available to 64K bytes. The Program Page Index Register allows for integrating up to 1M byte of FLASH or ROM into the system by using the six page index bits to page 16K byte blocks into the Program Page Window located from $8000 to $BFFF as defined in **Table 3-6**. CALL and RTC instructions have special access to read and write this register without using the address bus.

NOTE: *Normal writes to this register take one cycle to go into effect. Writes to this register using the special access of the CALL and RTC instructions will be complete before the end of the associated instruction.*

PIX5–PIX0 — Program Page Index Bits 5–0

These page index bits are used to select which of the 64 FLASH or ROM array pages is to be accessed in the Program Page Window as shown in **Table 3-6**.

Table 3-6 Program Page Index Register Bits

PIX5	PIX4	PIX3	PIX2	PIX1	PIX0	Program Space Selected
0	0	0	0	0	0	16K page 0
0	0	0	0	0	1	16K page 1
0	0	0	0	1	0	16K page 2
0	0	0	0	1	1	16K page 3
. . . . .	. . . . .	. . . .	. . . . .	. . . . .	. . . . .	. . . . .
1	1	1	1	0	0	16K page 60
1	1	1	1	0	1	16K page 61
1	1	1	1	1	0	16K page 62
1	1	1	1	1	1	16K page 63

Section 4 Functional Description

The MMC sub-block performs four basic functions of the Core operation: bus control, address decoding and select signal generation, memory expansion, and security decoding for the system. Each aspect is described in the following subsections.

4.1 Bus Control

The MMC controls the address bus and data buses that interface the Core with the rest of the system. This includes the multiplexing of the input data buses to the Core onto the main CPU read data bus and control of data flow from the CPU to the output address and data buses of the Core. In addition, the MMC handles all CPU read data bus swapping operations.

4.2 Address Decoding

As data flows on the Core address bus, the MMC decodes the address information, determines whether the internal Core register or firmware space, the peripheral space or a memory register or array space is being addressed and generates the correct select signal. This decoding operation also interprets the mode of operation of the system and the state of the mapping control registers in order to generate the proper select. The MMC also generates two external chip select signals, Emulation Chip Select ($\overline{ECS}$) and External Chip Select ($\overline{XCS}$).

4.2.1 Select Priority and Mode Considerations

Although internal resources such as control registers and on-chip memory have default addresses, each can be relocated by changing the default values in control registers. Normally, I/O addresses, control registers, vector spaces, expansion windows, and on-chip memory are mapped so that their address ranges do not overlap. The MMC will make only one select signal active at any given time. This activation is based upon the priority outlined in **Table 4-1**. If two or more blocks share the same address space, only the select signal for the block with the highest priority will become active. An example of this is if the registers and the RAM are mapped to the same space, the registers will have priority over the RAM and the portion of RAM mapped in this shared space will not be accessible. The expansion windows have the lowest priority. This means that registers, vectors, and on-chip memory are always visible to a program regardless of the values in the page select registers.

Table 4-1 Select Signal Priority

Priority	Address Space
Highest	BDM (internal to Core) firmware or register space
...	Internal register space
...	RAM memory block
...	EEPROM memory block
...	On-chip FLASH or ROM
Lowest	Remaining external space

In expanded modes, all address space not used by internal resources is by default external memory space. The data registers and data direction registers for Ports A and B are removed from the on-chip memory map and become external accesses. If the EME bit in the MODE register (see MEBI Block Guide) is set, the data and data direction registers for Port E are also removed from the on-chip memory map and become external accesses.

In Special Peripheral mode, the first 16 registers associated with bus expansion are removed from the on-chip memory map (PORTA, PORTB, DDRA, DDRB, PORTE, DDRE, PEAR, MODE, PUCR, RDRIV, and the EBI reserved registers).

In emulation modes, if the EMK bit in the MODE register (see MEBI Block Guide) is set, the data and data direction registers for Port K are removed from the on-chip memory map and become external accesses.

4.2.2 Emulation Chip Select Signal

When the EMK bit in the MODE register (see MEBI Block Guide) is set, Port K bit 7 is used as an active-low emulation chip select signal, $\overline{\text{ECS}}$. This signal is active when the system is in Emulation mode, the EMK bit is set and the FLASH or ROM space is being addressed subject to the conditions outlined in **4.3.2 Extended Address (XAB19:14) and ECS Signal Functionality**. When the EMK bit is clear, this pin is used for general purpose I/O.

4.2.3 External Chip Select Signal

When the EMK bit in the MODE register (see MEBI Block Guide) is set, Port K bit 6 is used as an active-low external chip select signal, $\overline{\text{XCS}}$. This signal is active only when the $\overline{\text{ECS}}$ signal described above is not active and when the system is addressing the external address space. Accesses to unimplemented locations within the register space or to locations that are removed from the map (i.e., Ports A and B in Expanded modes) will not cause this signal to become active. When the EMK bit is clear, this pin is used for general purpose I/O.

4.3 Memory Expansion

The HCS12 Core architecture limits the physical address space available to 64K bytes. The Program Page Index Register allows for integrating up to 1M byte of FLASH or ROM into the system by using the six page index bits to page 16K byte blocks into the Program Page Window located from $8000 to $BFFF in the physical memory space. The paged memory space can consist of solely on-chip memory or a combination of on-chip and off-chip memory. This partitioning is configured at system integration through the use of the paging configuration switches (*pag_sw1:pag_sw0*) at the Core boundary. The options available to the integrator are as given in **Table 4-2** (this table matches **Table 3-5** but is repeated here for easy reference).

Table 4-2 Allocated Off-Chip Memory Options

pag_sw1:pag_sw0	Off-Chip Space	On-Chip Space
00	876K byte2	128K byte2
01	768K byte2	256K byte2
10	512K byte2	512K byte2
11	0K byte	1M byte

Based upon the system configuration, the Program Page Window will consider its access to be either internal or external as defined in **Table 4-3**.

Table 4-3 External/Internal Page Window Access

pag_sw1:pag_sw0	Partitioning	PIX5:0 Value	Page Window Access
00	876K off-Chip, 128K on-Chip	$00–$37	External
		$38–$3F	Internal
01	768K off-chip, 256K on-chip	$00–$2F	External
		$30–$3F	Internal
10	512K off-chip, 512K on-chip	$00–$1F	External
		$20–$3F	Internal
11	0K off-chip, 1M on-chip	N/A	External
		$00–$3F	Internal

NOTE: *The partitioning as defined in* **Table 4-3** *applies only to the allocated memory space and the actual on-chip memory sizes implemented in the system may differ. Please refer to the chip-level documentation for actual sizes.*

The PPAGE register holds the page select value for the Program Page Window. The value of the PPAGE register can be manipulated by normal read and write (some devices don't allow writes in some modes) instructions as well as the CALL and RTC instructions.

Control registers, vector spaces, and a portion of on-chip memory are located in unpaged portions of the 64K byte physical address space. The stack and I/O addresses should also be in unpaged memory to make them accessible from any page.

The starting address of a service routine must be located in unpaged memory because the 16-bit exception vectors cannot point to addresses in paged memory. However, a service routine can call other routines that are in paged memory. The upper 16K byte block of memory space ($C000–$FFFF) is unpaged. It is recommended that all reset and interrupt vectors point to locations in this area.

4.3.1 CALL and Return from Call Instructions

CALL and RTC are uninterruptable instructions that automate page switching in the program expansion window. CALL is similar to a JSR instruction, but the subroutine that is called can be located anywhere in the normal 64K byte address space or on any page of program expansion memory. CALL calculates and stacks a return address, stacks the current PPAGE value, and writes a new instruction-supplied value to PPAGE. The PPAGE value controls which of the 64 possible pages is visible through the 16K byte expansion window in the 64K byte memory map. Execution then begins at the address of the called subroutine.

During the execution of a CALL instruction, the CPU:

- Writes the old PPAGE value into an internal temporary register and writes the new instruction-supplied PPAGE value into the PPAGE register.
- Calculates the address of the next instruction after the CALL instruction (the return address), and pushes this 16-bit value onto the stack.
- Pushes the old PPAGE value onto the stack.
- Calculates the effective address of the subroutine, refills the queue, and begins execution at the new address on the selected page of the expansion window.

This sequence is uninterruptable; there is no need to inhibit interrupts during CALL execution. A CALL can be performed from any address in memory to any other address.

The PPAGE value supplied by the instruction is part of the effective address. For all addressing mode variations except indexed-indirect modes, the new page value is provided by an immediate operand in the instruction. In indexed-indirect variations of CALL, a pointer specifies memory locations where the new page value and the address of the called subroutine are stored. Using indirect addressing for both the new page value and the address within the page allows values calculated at run time rather than immediate values that must be known at the time of assembly.

The RTC instruction terminates subroutines invoked by a CALL instruction. RTC unstacks the PPAGE value and the return address and refills the queue. Execution resumes with the next instruction after the CALL.

During the execution of an RTC instruction, the CPU:

- Pulls the old PPAGE value from the stack
- Pulls the 16-bit return address from the stack and loads it into the PC
- Writes the old PPAGE value into the PPAGE register
- Refills the queue and resumes execution at the return address

This sequence is uninterruptable; an RTC can be executed from anywhere in memory, even from a different page of extended memory in the expansion window.

The CALL and RTC instructions behave like JSR and RTS, except they use more execution cycles. Therefore, routinely substituting CALL/RTC for JSR/RTS is not recommended. JSR and RTS can be used to access subroutines that are on the same page in expanded memory. However, a subroutine in expanded memory that can be called from other pages must be terminated with an RTC. And the RTC unstacks a

PPAGE value. So any access to the subroutine, even from the same page, must use a CALL instruction so that the correct PPAGE value is in the stack.

4.3.2 Extended Address (XAB19:14) and $\overline{\text{ECS}}$ Signal Functionality

If the EMK bit in the MODE register is set (see MEBI Block Guide) the PIX5:0 values will be output on XAB19:14 respectively (Port K bits 5:0) when the system is addressing within the physical Program Page Window address space ($8000–$BFFF) and is in an expanded mode. When addressing anywhere else within the physical address space (outside of the paging space), the XAB19:14 signals will be assigned a constant value based upon the physical address space selected. In addition, the active-low emulation chip select signal, $\overline{\text{ECS}}$, will likewise function based upon the assigned memory allocation. In the cases of 48K byte and 64K byte allocated physical FLASH/ROM space, the operation of the $\overline{\text{ECS}}$ signal will additionally depend upon the state of the ROMHM bit (see **3.1.4 Miscellaneous System Control Register (MISC)**) in the MISC register. **Table 4-4**, **Table 4-5**, **Table 4-6**, and **Table 4-7** summarize the functionality of these signals based upon the allocated memory configuration. Again, this signal information is only available externally when the EMK bit is set and the system is in an expanded mode.

Table 4-4 0K Byte Physical FLASH/ROM Allocated

Address Space	Page Window Access	ROMHM	$\overline{\text{ECS}}$	XAB19:14
$0000–$3FFF	N/A	N/A	1	$3D
$4000–$7FFF	N/A	N/A	1	$3E
$8000–$BFFF	N/A	N/A	0	PIX5:0
$C000–$FFFF	N/A	N/A	0	$3F

Table 4-5 16K Byte Physical FLASH/ROM Allocated

Address Space	Page Window Access	ROMHM	$\overline{\text{ECS}}$	XAB19:14
$0000–$3FFF	N/A	N/A	1	$3D
$4000–$7FFF	N/A	N/A	1	$3E
$8000–$BFFF	N/A	N/A	1	PIX5:0
$C000–$FFFF	N/A	N/A	0	$3F

Table 4-6 48K Byte Physical FLASH/ROM Allocated

Address Space	Page Window Access	ROMHM	$\overline{\text{ECS}}$	XAB19:14
$0000–$3FFF	N/A	N/A	1	$3D
$4000–$7FFF	N/A	0	0	$3E
	N/A	1	1	
$8000–$BFFF	External	N/A	1	PIX5:0
	Internal	N/A	0	
$C000–$FFFF	N/A	N/A	0	$3F

Table 4-7 64K Byte Physical FLASH/ROM Allocated

Address Space	Page Window Access	ROMHM	$\overline{ECS}$	XAB19:14
$0000–$3FFF	N/A	0	0	$3D
	N/A	1	1	
$4000–$7FFF	N/A	0	0	$3E
	N/A	1	1	
$8000–$BFFF	External	N/A	1	PIX5:0
	Internal	N/A	0	
$C000–$FFFF	N/A	N/A	0	$3F

A graphical example of a memory paging for a system configured as 1M byte on-chip FLASH/ROM with 64K allocated physical space is given in **Figure 4-1**.

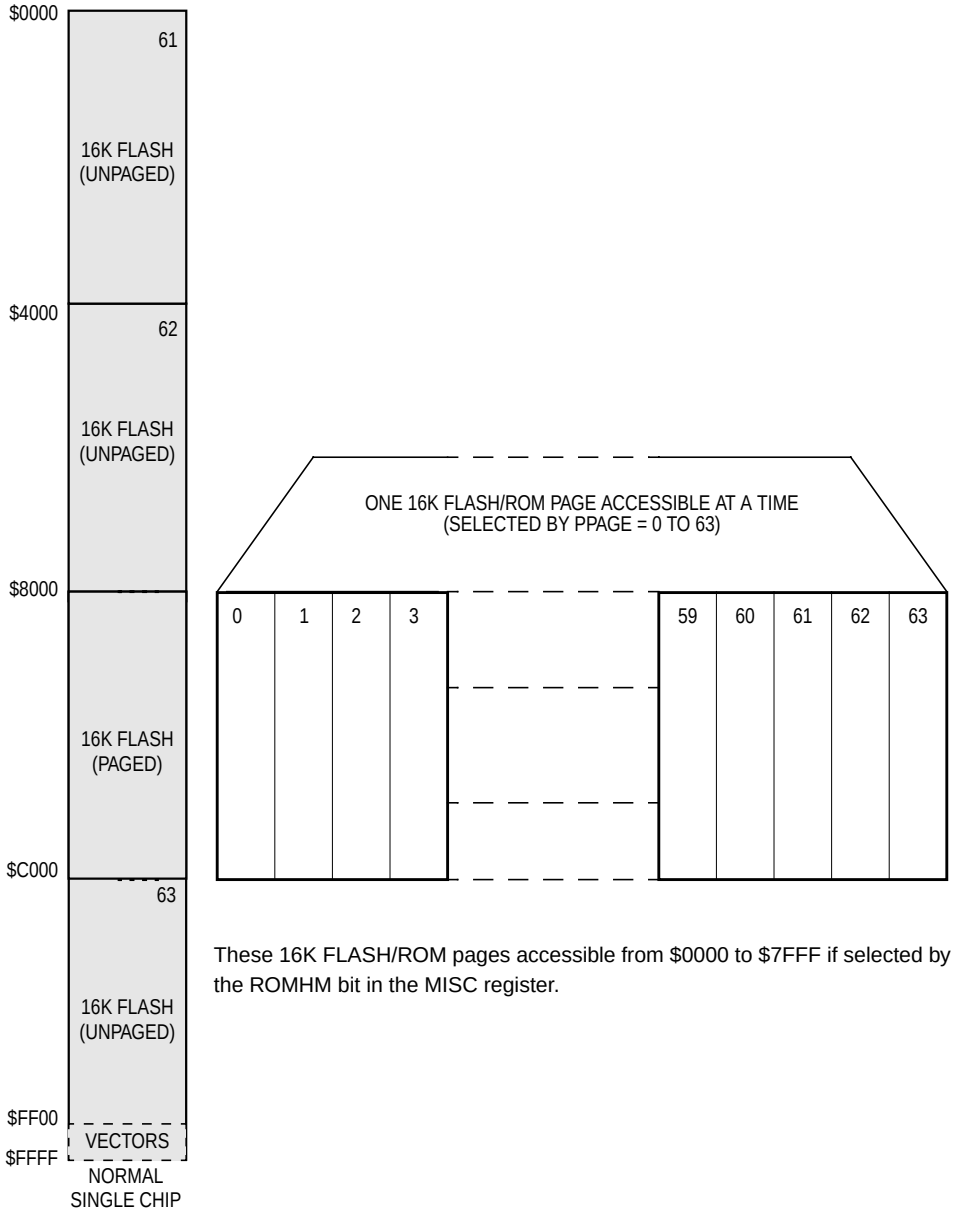


Figure 4-1 Memory Paging Example: 1M Byte On-Chip FLASH/ROM, 64K Allocation

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

JAPAN:
Motorola Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
Silicon Harbour Centre
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

HOME PAGE:
http://motorola.com/semiconductors

MOTOROLA

S12MMCV4/D
Rev. 4.00
5/2003

MSCAN
Block Guide
V02.15

Original Release Date: 19 MAY 1998
Revised: 15 JUL 2004

Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
V02.08	17 JUL 2001	17 JUL 2001		- 1st offical version by Technical Publishing
V02.09	10 JUL 2001	10 JUL 2001		- Updated according to requirements of SRSv3 - Corrected footnote 1 in INITRQ description.
V02.10	10 OCT 2001	10 OCT 2001		- Replaced all references w.r.t. new family name HCS12.
V02.11	22 OCT 2001	22 OCT 2001		- Corrected figure title and note of CANTIER. - Corrected local enable register names in table 4-4 ’CRG Interrupt Vectors’. - Updated block diagram. - Corrected section ’Description of Interrupt Operation’.
V02.12	04 MAR 2002	04 MAR 2002		- Document format updates.
V02.13	22 JUL 2002	22 JUL 2002		- Corrected TBPR register offset. - Corrected Table ’Message Buffer Organization’. - Corrected SLPRQ bit description. - Corrected MSCAN Sleep Mode description. - Updated WUPE bit description. - Updated Simplified State Transitions figure. - Updated Recovery from STOP or WAIT description and CPU vs. MSCAN Modes table.
V02.14	18 SEP 2002	18 SEP 2002		- Added Initialization/Application information. - Replaced ’MCU’ with ’CPU’ in several places. - Cleaned up Mode descriptions. - General cleanup.
V02.15	15 JUL 2004	15 JUL 2004		- Corrected buffer read/write access definitions. - Corrected bit time equation.

Table of Contents

Section 5 Initialization/Application Information

List of Figures

List of Tables

Preface

Terminology

Acronyms and Abbreviations	
ACK	Acknowledge
CAN	Controller Area Network
CRC	Cyclic Redundancy Code
EOF	End of Frame
FIFO	First-In-First-Out Memory
IFS	Inter-Frame Sequence
MSCAN	Motorola Scalable CAN Module
SOF	Start of Frame

Section 1 Introduction

1.1 Overview

The Motorola Scalable Controller Area Network (MSCAN) definition is based on the MSCAN12 definition which is the specific implementation of the Motorola Scalable CAN concept targeted for the Motorola MC68HC12 Microcontroller Family.

The module is a communication controller implementing the CAN 2.0 A/B protocol as defined in the BOSCH specification dated September 1991. For users to fully understand the MSCAN specification, it is recommended that the Bosch specification be read first to familiarize the reader with the terms and concepts contained within this document.

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth.

MSCAN utilizes an advanced buffer arrangement resulting in a predictable real-time behavior and simplifies the application software.

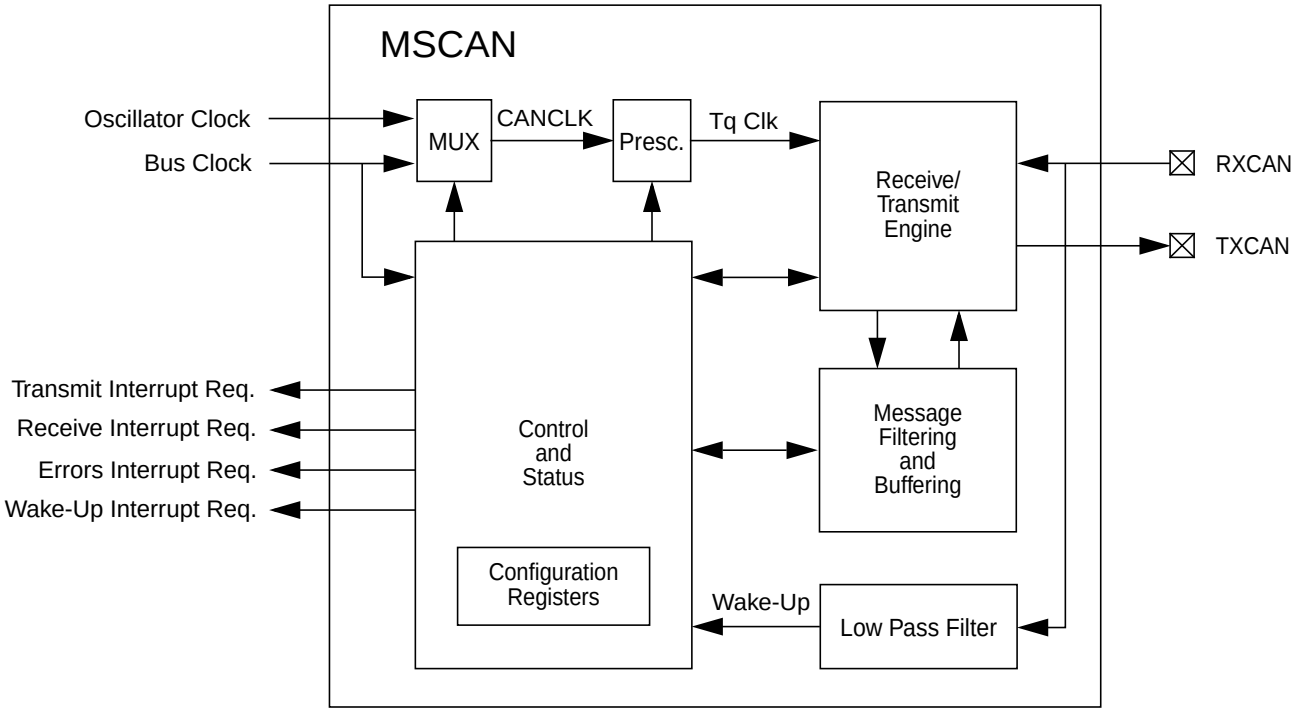


Figure 1-1 MSCAN Block Diagram

1.2 Features

The basic features of the MSCAN are as follows:

- Implementation of the CAN protocol - Version 2.0A/B
  - Standard and extended data frames
  - 0 - 8 bytes data length
  - Programmable bit rate up to 1 Mbps[1]
  - Support for remote frames
  - 5 receive buffers with FIFO storage scheme
- 3 transmit buffers with internal prioritization using a “local priority” concept
- Flexible maskable identifier filter supports two full size extended identifier filters (two 32-bit) or four 16-bit filters or eight 8-bit filters
- Programmable wake-up functionality with integrated low-pass filter
- Programmable loop back mode supports self-test operation
- Programmable listen-only mode for monitoring of CAN bus
- Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states (Warning, Error Passive, Bus-Off)
- Programmable MSCAN clock source either Bus Clock or Oscillator Clock
- Internal timer for time-stamping of received and transmitted messages
- Three low power modes: Sleep, Power Down and MSCAN Enable
- Global initialization of configuration registers

NOTES:
1. Depending on the actual bit timing and the clock jitter of the PLL.

Section 2 External Signal Description

2.1 Overview

This section lists and describes the signals that connect off chip.

2.2 Detailed Signal Description

The MSCAN uses two external pins.

2.2.1 RXCAN — CAN Receiver Input Pin

RXCAN is the MSCAN receiver input pin.

2.2.2 TXCAN — CAN Transmitter Output Pin

TXCAN is the MSCAN transmitter output pin. The TXCAN output pin represents the logic level on the CAN bus:

0 = Dominant state
1 = Recessive state

2.3 CAN System

A typical CAN system with MSCAN is shown in **Figure 2-1**. Each CAN station is connected physically to the CAN bus lines through a transceiver chip. The transceiver is capable of driving the large current needed for the CAN bus and has current protection against defected CAN or defected stations.

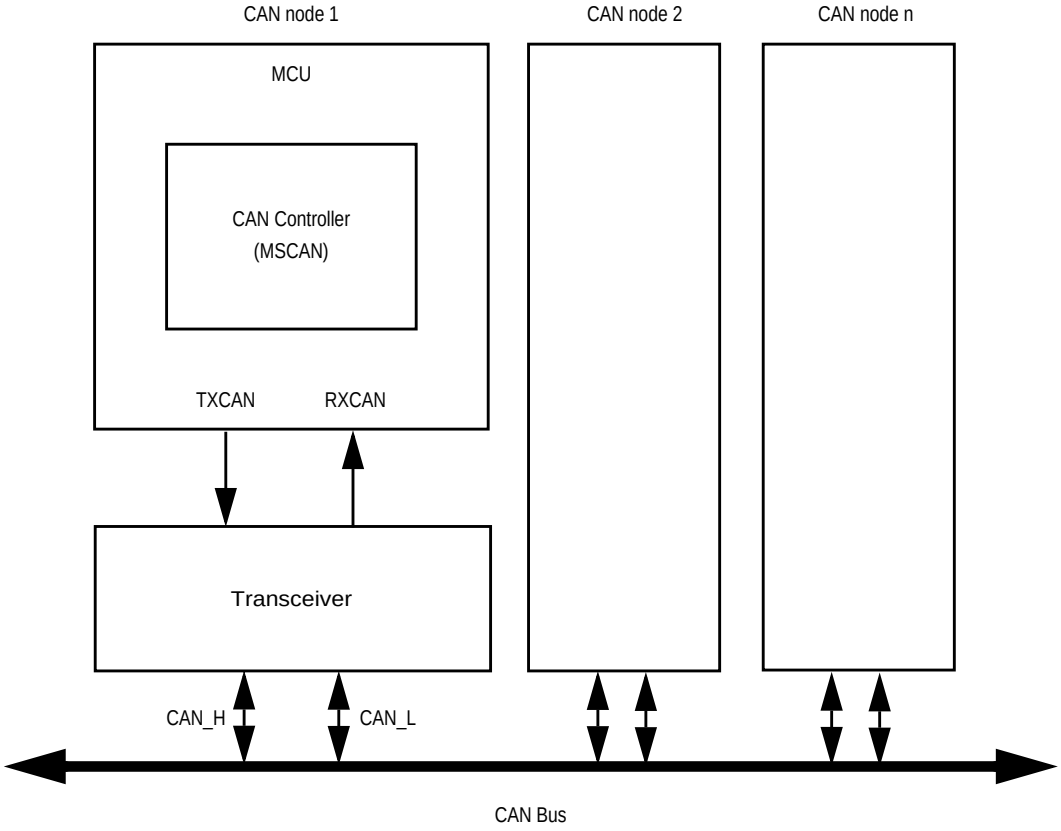


Figure 2-1 The CAN System

Section 3 Memory Map/Register Definition

3.1 Overview

This section provides a detailed description of all registers accessible in the MSCAN.

3.2 Module Memory Map

Table 3-1 and **Table 3-2** give an overview on all registers and their individual bits in the MSCAN memory map. The *register address* results from the addition of *base address* and *address offset*. The *base address* is determined at the MCU level. The *address offset* is defined at the module level.

The MSCAN occupies 64 bytes in the memory space. The base address of the MSCAN module is determined at the MCU level when the MCU is defined. The register decode map is fixed and begins at the first address of the module address offset.

Address Offset	
$__00 $__0B	CONTROL REGISTERS 12 BYTES
$__0C $__0D	RESERVED 2 BYTES
$__0E $__0F	ERROR COUNTERS 2 BYTES
$__10 $__1F	IDENTIFIER FILTER 16 BYTES
$__20 $__2F	RECEIVE BUFFER 16 BYTES (Window)
$__30 $__3F	TRANSMIT BUFFER 16 BYTES (Window)

Table 3-1 MSCAN Register Organization

Table 3-1 shows the individual registers associated with the MSCAN and their relative offset from the base address. The detailed register descriptions follow in the order they appear in the register map (see **Table 3-2**).

Table 3-2 Module Memory Map

Address	Use	Access
$__00	MSCAN Control Register 0 (CANCTL0)	R/W[1]
$__01	MSCAN Control Register 1 (CANCTL1)	R/W[1]
$__02	MSCAN Bus Timing Register 0 (CANBTR0)	R/W
$__03	MSCAN Bus Timing Register 1 (CANBTR1)	R/W

$__04	MSCAN Receiver Flag Register (CANRFLG)	R/W[1]
$__05	MSCAN Receiver Interrupt Enable Register (CANRIER)	R/W
$__06	MSCAN Transmitter Flag Register (CANTFLG)	R/W[1]
$__07	MSCAN Transmitter Interrupt Enable Register (CANTIER)	R/W[1]
$__08	MSCAN Transmitter Message Abort Control (CANTARQ)	R/W[1]
$__09	MSCAN Transmitter Message Abort Control (CANTAAK)	R
$__0A	MSCAN Transmit Buffer Selection (CANTBSEL)	R/W[1]
$__0B	MSCAN Identifier Acceptance Control Register (CANIDAC)	R/W[1]
$__0C -$__0D	RESERVED	
$__0E	MSCAN Receive Error Counter Register (CANRXERR)	R
$__0F	MSCAN Transmit Error Counter Register (CANTXERR)	R
$__10	MSCAN Identifier Acceptance Register 0 (CANIDAR0)	R/W
$__11	MSCAN Identifier Acceptance Register 1 (CANIDAR1)	R/W
$__12	MSCAN Identifier Acceptance Register 2 (CANIDAR2)	R/W
$__13	MSCAN Identifier Acceptance Register 3 (CANIDAR3)	R/W
$__14	MSCAN Identifier Mask Register 0 (CANIDMR0)	R/W
$__15	MSCAN Identifier Mask Register 1 (CANIDMR1)	R/W
$__16	MSCAN Identifier Mask Register 2 (CANIDMR2)	R/W
$__17	MSCAN Identifier Mask Register 3 (CANIDMR3)	R/W
$__18	MSCAN Identifier Acceptance Register 4 (CANIDAR4)	R/W
$__19	MSCAN Identifier Acceptance Register 5 (CANIDAR5)	R/W
$__1A	MSCAN Identifier Acceptance Register 6 (CANIDAR6)	R/W
$__1B	MSCAN Identifier Acceptance Register 7 (CANIDAR7)	R/W
$__1C	MSCAN Identifier Mask Register 4 (CANIDMR4)	R/W
$__1D	MSCAN Identifier Mask Register 5 (CANIDMR5)	R/W
$__1E	MSCAN Identifier 6 Mask Register 6 (CANIDMR6)	R/W
$__1F	MSCAN Identifier Mask Register 7 (CANIDMR7)	R/W
$__20 -$__2F	Foreground Receive Buffer (CANRXFG)	R[2]
$__30 -$__3F	Foreground Transmit Buffer (CANTXFG)	R[2]/W

NOTES:
1. Refer to detailed register description for write access restrictions on per bit basis.
2. Reserved bits and unused bits within the TX- and RX-Buffers (CANTXFG, CANRXFG) will be read as “X”, because of RAM based implementation.

3.3 Register Descriptions

This section describes in detail all the registers and register bits in the MSCAN module. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order. All bits of all registers in this module are completely synchronous to internal clocks during a register read.

3.3.1 Programmer's Model of Control Registers

The programmer's model is laid out for maximum simplicity and efficiency. **Table 3-2** provides an overview of the control registers for the MSCAN.

3.3.1.1 MSCAN Control 0 Register (CANCTL0)

The CANCTL0 register provides for various control of the MSCAN module as described below.

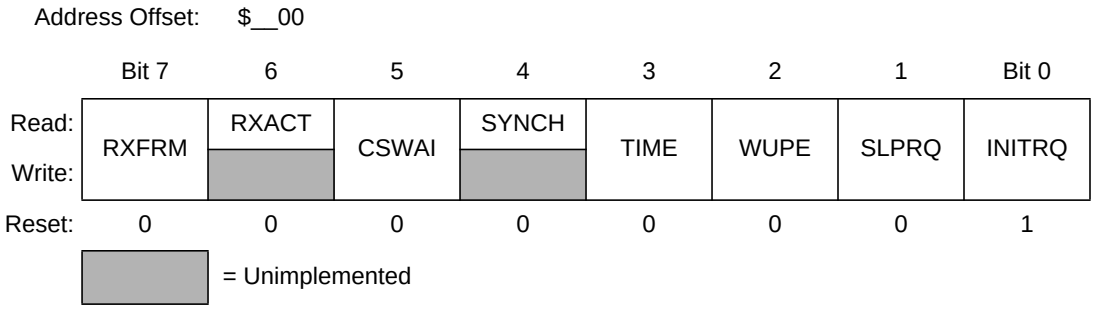


Figure 3-1 MSCAN Control 0 Register (CANCTL0)

NOTE: *The CANCTL0 register, except the WUPE, INITRQ and SLPRQ bits, is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: Anytime

Write: Anytime when out of Initialization Mode; exceptions are read-only bits RXACT and SYNCH, bit RXFRM which is set by the module only and bit INITRQ which is also writable in Initialization Mode.

RXFRM — Received Frame Flag

This bit is read and clear only. It is set when a receiver has received a valid message correctly, independently of the filter configuration. Once set, it remains set until cleared by software or reset. Clearing is done by writing a '1' to the bit. A write '0' is ignored. This bit is not valid in loop back mode.

1 = A valid message was received since last clearing of this flag
0 = No valid message was received since last clearing this flag.

NOTE: *The MSCAN must be in Normal Mode for this bit to become set.*

RXACT — Receiver Active Status

This read-only flag indicates the MSCAN is receiving a message. The flag is controlled by the receiver front end. This bit is not valid in loop back mode.

1 = MSCAN is receiving a message (including when arbitration is lost)[1]
0 = MSCAN is transmitting or idle[1].

CSWAI — CAN Stops in Wait Mode

Enabling this bit allows for lower power consumption in Wait Mode by disabling all the clocks at the bus interface to the MSCAN module.

1 = The module ceases to be clocked during Wait Mode.
0 = The module is not affected during Wait Mode.

NOTE: *In order to protect from accidentally violating the CAN protocol the TXCAN pin is immediately forced to a recessive state when the CPU enters Wait (CSWAI=1) or Stop Mode (see **4.6.2 CPU Wait Mode** and **4.6.3 CPU Stop Mode**)*

SYNCH — Synchronized Status

This read-only flag indicates whether the MSCAN is synchronized to the CAN bus and, as such, can participate in the communication process. It is set and cleared by the MSCAN.

1 = MSCAN is synchronized to the CAN bus.
0 = MSCAN is not synchronized to the CAN bus.

TIME - Timer Enable

This bit activates an internal 16-bit wide free running timer which is clocked by the bit clock. If the timer is enabled, a 16-bit time stamp will be assigned to each transmitted/received message within the active TX/RX buffer. As soon as a message is acknowledged on the CAN bus, the time stamp will be written to the highest bytes ($_E, $_F) in the appropriate buffer **3.3.2 Programmer's Model of Message Storage**. The internal timer is reset (all bits set to "0") when Initialization Mode is active.

1 = Enable internal MSCAN timer.
0 = Disable internal MSCAN timer.

WUPE — Wake-Up Enable

This configuration bit allows the MSCAN to restart from Sleep Mode when traffic on CAN is detected (see **4.6.4 MSCAN Sleep Mode**).

1 = Wake-Up enabled– The MSCAN is able to restart.
0 = Wake-Up disabled– The MSCAN ignores traffic on CAN.

NOTE: *The CPU has to make sure that the WUPE register and the WUPIE Wake-Up interrupt enable register **3.3.1.6 MSCAN Receiver Interrupt Enable Register (CANRIER)** is enabled, if the recovery mechanism from STOP or WAIT is required.*

SLPRQ — Sleep Mode Request

This bit requests the MSCAN to enter Sleep Mode, which is an internal power saving mode (see **4.6.4 MSCAN Sleep Mode**). The Sleep Mode request is serviced when the CAN bus is idle, i.e. the module is not receiving a message and all transmit buffers are empty. The module indicates entry to Sleep Mode by setting SLPAK=1 (**3.3.1.2 MSCAN Control 1 Register (CANCTL1)**). Sleep Mode will be active until SLPRQ is cleared by the CPU or, depending on the setting of WUPE bit, the MSCAN detects bus activity on CAN and clears the SLPRQ itself.

NOTES:
1. See the Bosch CAN 2.0A/B protocol specification dated September 1991 for a detailed definition of transmitter and receiver states.

1 = Sleep Mode Request – The MSCAN enters Sleep Mode when CAN bus idle.
0 = Running – The MSCAN functions normally.

NOTE: *The CPU cannot clear the SLPRQ bit before the MSCAN has entered Sleep Mode (SLPRQ=1 and SLPAK=1)*

INITRQ — Initialization Mode Request

When this bit is set by the CPU, the MSCAN skips to Initialization Mode (see **4.6.5 MSCAN Initialization Mode**). Any ongoing transmission or reception is aborted and synchronization to the bus is lost. The module indicates entry to Initialization Mode by setting INITAK=1 (**3.3.1.2 MSCAN Control 1 Register (CANCTL1)**).

The following registers enter their hard reset state and restore their default values: CANCTL0[1], CANRFLG[2], CANRIER[3], CANTFLG, CANTIER, CANTARQ, CANTAAK, CANTBSEL.

The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0-7, CANIDMR0-7 can only be written by the CPU when the MSCAN is in Initialization Mode (INITRQ=1 and INITAK=1). The values of the error counters are not affected by Initialization Mode.

When this bit is cleared by the CPU, the MSCAN restarts and then tries to synchronize to the CAN bus. If the MSCAN is not in Bus-Off state, it synchronizes after 11 consecutive recessive bits on the bus; if the MSCAN is in Bus-Off state it continues to wait for 128 occurrences of 11 consecutive recessive bits.

Writing to other bits in CANCTL0, CANRFLG, CANRIER, CANTFLG or CANTIER must only be done after Initialization Mode is left, which is INITRQ=0 and INITAK=0.

1 = MSCAN in Initialization Mode.
0 = Normal operation.

NOTE: *The CPU cannot clear the INITRQ bit before the MSCAN has entered Initialization Mode (INITRQ=1 and INITAK=1)*

NOTE: *In order to protect from accidentally violating the CAN protocol the TXCAN pin is immediately forced to a recessive state when the Initialization Mode is requested by the CPU. Thus the recommended procedure is to bring the MSCAN into Sleep Mode (SLPRQ=1 and SLPAK=1) before.*

3.3.1.2 MSCAN Control 1 Register (CANCTL1)

The CANCTL1 register provides for various control and handshake status information of the MSCAN module as described below.

Address Offset: $__01

NOTES:
1. Except the WUPE, INITRQ and SLPRQ bits
2. The TSTAT1, TSTAT0 bits are not affected by Initialization Mode
3. The RSTAT1, RSTAT0 bits are not affected by Initialization Mode

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK
Write:								
Reset:	0	0	0	1	0	0	0	1

= Unimplemented

Figure 3-2 MSCAN Control 1 Register (CANCTL1)

Read: Anytime

Write: Anytime when INITRQ=1 and INITAK=1, except CANE which is write once in normal and anytime in special system operation modes when the MSCAN is in Initialization Mode (INITRQ=1 and INITAK=1).

CANE — MSCAN Enable

1 = The MSCAN module is enabled.
0 = The MSCAN module is disabled.

CLKSRC — MSCAN Clock Source

This bit defines the clock source for the MSCAN module (only for systems with a clock generation module; **4.3.2 Clock System** and **Figure 4-5**).

1 = The MSCAN clock source is the Bus Clock.
0 = The MSCAN clock source is the Oscillator Clock.

LOOPB — Loop Back Self Test Mode

When this bit is set, the MSCAN performs an internal loop back which can be used for self test operation. The bit stream output of the transmitter is fed back to the receiver internally. The RXCAN input pin is ignored and the TXCAN output goes to the recessive state (logic ‘1’). The MSCAN behaves as it does normally when transmitting and treats its own transmitted message as a message received from a remote node. In this state, the MSCAN ignores the bit sent during the ACK slot in the CAN frame Acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

1 = Loop Back Self Test enabled
0 = Loop Back Self Test disabled

LISTEN — Listen Only Mode

This bit configures the MSCAN as a bus monitor. When the bit is set, all valid CAN messages with matching ID are received, but no acknowledgement or error frames are sent out **4.5.4 Listen-Only Mode**. In addition the error counters are frozen.

Listen Only Mode supports applications which require “hot plugging” or throughput analysis. The MSCAN is unable to transmit any messages, when Listen Only Mode is active.

1 = Listen Only Mode activated
0 = Normal operation

WUPM — Wake-Up Mode

This bit defines whether the integrated low-pass filter is applied to protect the MSCAN from spurious wake-up **4.6.4 MSCAN Sleep Mode**.

1 = MSCAN wakes-up the CPU only in case of a dominant pulse on the bus which has a length of T_{wup} and WUPE=1 in CANCTL0 (see **3.3.1.1 MSCAN Control 0 Register (CANCTL0)**).

0 = MSCAN wakes-up the CPU after any recessive to dominant edge on the CAN bus and WUPE=1 in CANCTL0.

SLPAK — Sleep Mode Acknowledge

This flag indicates whether the MSCAN module has entered Sleep Mode **4.6.4 MSCAN Sleep Mode**. It is used as a handshake flag for the SLPRQ Sleep Mode request. Sleep Mode is active when SLPRQ=1 and SLPAK=1. Depending on the setting of the WUPE bit the MSCAN will clear the flag if it detects bus activity on CAN while in Sleep Mode.

1 = Sleep Mode Active – The MSCAN has entered Sleep Mode.

0 = Running – The MSCAN operates normally.

INITAK — Initialization Mode Acknowledge

This flag indicates whether the MSCAN module is in Initialization Mode **4.6.5 MSCAN Initialization Mode**. It is used as a handshake flag for the INITRQ Initialization Mode request. Initialization Mode is active when INITRQ=1 and INITAK=1.

The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0-7, CANIDMR0-7 can only be written by the CPU when the MSCAN is in Initialization Mode.

1 = Initialization Mode Active – The MSCAN has entered Initialization Mode.

0 = Running – The MSCAN operates normally.

3.3.1.3 MSCAN Bus Timing Register 0 (CANBTR0)

The CANBTR0 register provides for various bus timing control of the MSCAN module as described below.

Address Offset: $__02

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Reset:	0	0	0	0	0	0	0	0

Figure 3-3 MSCAN Bus Timing Register 0 (CANBTR0)

Read: Anytime

Write: Anytime in Initialization Mode (INITRQ=1 and INITAK=1)

SJW1, SJW0 — Synchronization Jump Width

The synchronization jump width defines the maximum number of time quanta (Tq) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the bus (see **Table 3-3**).

Table 3-3 Synchronization Jump Width

SJW1	SJW0	Synchronization jump width
0	0	1 Tq clock cycle
0	1	2 Tq clock cycles
1	0	3 Tq clock cycles
1	1	4 Tq clock cycles

BRP[5-0] — Baud Rate Prescaler

These bits determine the time quanta (Tq) clock which is used to build up the individual bit timing, as shown in **Table 3-4**.

Table 3-4 Baud Rate Prescaler

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Prescaler value (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
:	:	:	:	:	:	:
1	1	1	1	1	0	63
1	1	1	1	1	1	64

3.3.1.4 MSCAN Bus Timing Register 1 (CANBTR1)

The CANBTR1 register provides for various bus timing control of the MSCAN module as described below.

Address Offset: $__03

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10

Figure 3-4 MSCAN Bus Timing Register 1 (CANBTR1)

Reset:	0	0	0	0	0	0	0	0

Figure 3-4 MSCAN Bus Timing Register 1 (CANBTR1)

Read: Anytime

Write: Anytime in Initialization Mode (INITRQ=1 and INITAK=1)

SAMP — Sampling

This bit determines the number of samples of the serial bus to be taken per bit time. If set, three samples per bit are taken; the regular one (sample point) and two preceding samples using a majority rule. For higher bit rates, it is recommended that SAMP be cleared which means that only one sample is taken per bit.

1 = Three samples per bit[1].
0 = One sample per bit.

TSEG22 – TSEG20 — Time Segment 2

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point (see **Figure 4-6 Segments within the Bit Time**).

Time segment 2 (TSEG2) values are programmable as shown in **Table 3-5**.

Table 3-5 Time Segment 2 Values

TSEG22	TSEG21	TSEG20	Time segment 2
0	0	0	1 Tq clock cycle[1]
0	0	1	2 Tq clock cycles
.	.	.	.
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

NOTES:
1. This setting is not valid. Please refer to **Table 4-2 CAN Standard Compliant Bit Time Segment Settings** for valid settings.

TSEG13 – TSEG10 — Time Segment 1

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point (see **Figure 4-6 Segments within the Bit Time**).

Time segment 1 (TSEG1) values are programmable as shown in **Table 3-6**.

NOTES:
1. In this case, PHASE_SEG1 must be at least 2 Time Quanta.

Table 3-6 Time Segment 1 Values

TSEG13	TSEG12	TSEG11	TSEG10	Time segment 1
0	0	0	0	1 Tq clock cycle[1]
0	0	0	1	2 Tq clock cycles[1]
0	0	1	0	3 Tq clock cycles[1]
0	0	1	1	4 Tq clock cycles
.	.	.	.	.
1	1	1	0	15 Tq clock cycles
1	1	1	1	16 Tq clock cycles

NOTES:
1. This setting is not valid. Please refer to **Table 4-2 CAN Standard Compliant Bit Time Segment Settings** for valid settings.

The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (Tq) clock cycles per bit (as shown in **Table 3-5** and **Table 3-6** above).

$$\text{Bit Time} = \frac{(\text{Prescaler value})}{f_{CANCLK}} \bullet (1 + \text{TimeSegment1} + \text{TimeSegment2})$$

3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)

A flag can only be cleared when the condition which caused the setting is no longer valid and can only be cleared by software (writing a ‘1’ to the corresponding bit position). Every flag has an associated interrupt enable bit in the CANRIER register.

Address Offset: $__04

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
Write:								
Reset:	0	0	0	0	0	0	0	0

Figure 3-5 MSCAN Receiver Flag Register (CANRFLG)

NOTE: *The CANRFLG register is held in the reset state[1] when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: Anytime

Write: Anytime when out of Initialization Mode, except RSTAT[1:0] and TSTAT[1:0] flags which are read-only; write of '1' clears flag; write of '0' ignored

WUPIF — Wake-Up Interrupt Flag

If the MSCAN detects bus activity while in Sleep Mode **4.6.4 MSCAN Sleep Mode** and the WUPE=1 in CANTCTL0 (see **3.3.1.1 MSCAN Control 0 Register (CANCTL0)**), it will set the WUPIF flag. If not masked, a Wake-Up interrupt is pending while this flag is set.

1 = MSCAN detected activity on the bus and requested wake-up.
0 = No wake-up activity observed while in Sleep Mode.

CSCIF — CAN Status Change Interrupt Flag

This flag is set when the MSCAN changes its current bus status due to the actual value of the Transmit Error Counter (TEC) and the Receive Error Counter (REC). An additional 4-bit (RSTAT[1:0], TSTAT[1:0]) status register, which is split into separate sections for TEC/REC, informs the system on the actual bus status **3.3.1.6 MSCAN Receiver Interrupt Enable Register (CANRIER)**. If not masked, an Error interrupt is pending while this flag is set. CSCIF provides a blocking interrupt. That guarantees that the Receiver / Transmitter status bits (RSTAT/TSTAT) are only updated when no CAN Status Change interrupt is pending. If the TECs/RECs change their current value after the CSCIF is asserted and therefore would cause an additional state change in the RSTAT/TSTAT bits, these bits keep their old state bits until the current CSCIF interrupt is cleared again.

1 = MSCAN changed current bus status.
0 = No change in bus status occurred since last interrupt.

RSTAT1, RSTAT0 — Receiver Status Bits

The values of the error counters control the actual bus status of the MSCAN. As soon as the Status Change Interrupt Flag (CSCIF) is set these bits indicate the appropriate receiver related bus status of the MSCAN. The coding for the bits RSTAT1, RSTAT0 is:

00 = RxOK: $0 \leq$ Receive Error Counter ≤ 96
01 = RxWRN: $96 <$ Receive Error Counter ≤ 127
10 = RxERR: $127 <$ Receive Error Counter
11 = Bus-Off[2]: Transmit Error Counter > 255

TSTAT1, TSTAT0 — Transmitter Status Bits

The values of the Error Counters control the actual bus status of the MSCAN. As soon as the Status Change Interrupt Flag (CSCIF) is set these bits indicate the appropriate transmitter related bus status of the MSCAN. The coding for the bits TSTAT1, TSTAT0 is:

NOTES:
1. The RSTAT[1:0], TSTAT[1:0] bits are not affected by Initialization Mode
2. Redundant Information for the most critical bus status which is "CAN Bus-Off". This only occurs if the Tx Error Counter exceeds a number of 255 errors. CAN Bus-Off affects the receiver state. As soon as the transmitter leaves its Bus-Off state the receiver state skips to RxOK too. Refer also to TSTAT[1:0] coding.

00 = TxOK: 0 ≤ Transmit Error Counter ≤ 96
01 = TxWRN: 96 < Transmit Error Counter ≤ 127
10 = TxERR: 127 < Transmit Error Counter ≤ 255
11 = Bus-Off: Transmit Error Counter > 255

OVRIF — Overrun Interrupt Flag

This flag is set when a data overrun condition occurs. If not masked, an Error interrupt is pending while this flag is set.

1 = A data overrun detected.
0 = No data overrun condition.

RXF — Receive Buffer Full Flag

The RXF flag is set by the MSCAN when a new message is shifted in the receiver FIFO. This flag indicates whether the shifted buffer is loaded with a correctly received message (matching identifier, matching Cyclic Redundancy Code (CRC) and no other errors detected). After the CPU has read that message from the RxFG buffer in the receiver FIFO, the RXF flag must be cleared to release the buffer. A set RXF flag prohibits the shifting of the next FIFO entry into the foreground buffer (RxFG). If not masked, a Receive interrupt is pending while this flag is set.

1 = The receiver FIFO is not empty. A new message is available in the RxFG.
0 = No new message available within the RxFG.

NOTE: *To ensure data integrity, do not read the receive buffer registers while the RXF flag is cleared.*

For MCUs with dual CPUs, reading the receive buffer registers while the RXF flag is cleared may result in a CPU fault condition.

3.3.1.6 MSCAN Receiver Interrupt Enable Register (CANRIER)

This register contains the interrupt enable bits for the interrupt flags described above.

Address Offset: $__05

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
Reset:	0	0	0	0	0	0	0	0

Figure 3-6 MSCAN Receiver Interrupt Enable Register (CANRIER)

NOTE: *The CANRIER register is held in the reset state[1] when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

NOTES:
1. The RSTATE[1:0], TSTATE[1:0] bits are not affected by Initialization Mode

Read: Anytime

Write: Anytime when out of Initialization Mode

WUPIE — Wake-Up Interrupt Enable

1 = A wake-up event causes a Wake-Up interrupt request.
0 = No interrupt request is generated from this event.

NOTE: *The CPU has to make sure that the Wake-Up interrupt register and the WUPE register **3.3.1.1 MSCAN Control 0 Register (CANCTL0)** is enabled, if the recovery mechanism from STOP or WAIT is required.*

CSCIE — CAN Status Change Interrupt Enable

1 = A CAN Status Change event causes an error interrupt request.
0 = No interrupt request is generated from this event.

RSTATE1, RSTATE0— Receiver Status Change Enable

These RSTAT enable bits control the sensitivity level in which receiver state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level the RSTAT flags still indicate the actual receiver state and are only updated if no CSCIF interrupt is pending.

11 = generate CSCIF interrupt on all state changes
10 = generate CSCIF interrupt only if the receiver enters or leaves “RxErr” or “Bus-Off”[1] state. Discard other receiver state changes for generating CSCIF interrupt.
01 = generate CSCIF interrupt only if the receiver enters or leaves “Bus-Off” state. Discard other receiver state changes for generating CSCIF interrupt.
00 = do not generate any CSCIF interrupt caused by receiver state changes.

TSTATE1, TSTATE0— Transmitter Status Change Enable

These TSTAT enable bits control the sensitivity level in which transmitter state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level the TSTAT flags still indicate the actual transmitter state and are only updated if no CSCIF interrupt is pending.

11 = generate CSCIF interrupt on all state changes
10 = generate CSCIF interrupt only if the transmitter enters or leaves “TxErr” or “Bus-Off” state. Discard other transmitter state changes for generating CSCIF interrupt.
01 = generate CSCIF interrupt only if the transmitter enters or leaves “Bus-Off” state. Discard other transmitter state changes for generating CSCIF interrupt.
00 = do not generate any CSCIF interrupt caused by transmitter state changes.

OVRIE — Overrun Interrupt Enable

1 = An overrun event causes an error interrupt request.
0 = No interrupt request is generated from this event.

RXFIE — Receiver Full Interrupt Enable

NOTES:

1. Bus-Off state is only defined by the CAN standard for transmitters. Because the only possible state change for the transmitter from Bus-Off to TxOK also forces the receiver to skip its current state to RxOK, the coding of the RXSTAT[1:0] flags define an additional Bus-Off state for the receiver **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)**

1 = A receive buffer full (successful message reception) event causes a receiver interrupt request.
0 = No interrupt request is generated from this event.

3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)

The Transmit Buffer Empty flags each have an associated interrupt enable bit in the CANTIER register.

Address Offset: $__06

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	TXE2	TXE1	TXE0
Write:								
Reset:	0	0	0	0	0	1	1	1

= Unimplemented

Figure 3-7 MSCAN Transmitter Flag Register (CANTFLG)

NOTE: *The CANTFLG register is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: Anytime

Write: Anytime for TXEx flags when not in Initialization Mode; write of ‘1’ clears flag, write of ‘0’ ignored

TXE2 - TXE0 —Transmitter Buffer Empty

This flag indicates that the associated transmit message buffer is empty, and thus not scheduled for transmission. The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission. The MSCAN sets the flag after the message is sent successfully. The flag is also set by the MSCAN when the transmission request is successfully aborted due to a pending abort request (see **3.3.1.9 MSCAN Transmitter Message Abort Control (CANTARQ)**). If not masked, a Transmit interrupt is pending while this flag is set.

Clearing a TXEx flag also clears the corresponding ABTAKx (see **3.3.1.10 MSCAN Transmitter Message Abort Control (CANTAAK)**). When a TXEx flag is set, the corresponding ABTRQx bit is cleared (see **3.3.1.9 MSCAN Transmitter Message Abort Control (CANTARQ)**).

When Listen-Mode is active (see **3.3.1.2 MSCAN Control 1 Register (CANCTL1)**) the TXEx flags cannot be cleared and no transmission is started.

Read and write accesses to the transmit buffer will be blocked, if the corresponding TXEx bit is cleared (TXEx=’0’) and the buffer is scheduled for transmission.

1 = The associated message buffer is empty (not scheduled).
0 = The associated message buffer is full (loaded with a message due for transmission).

3.3.1.8 MSCAN Transmitter Interrupt Enable Register (CANTIER)

This register contains the interrupt enable bits for the Transmit Buffer Empty interrupt flags.

Address Offset: $__07

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-8 MSCAN Transmitter Interrupt Enable Register (CANTIER)

NOTE: *The CANTIER register is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: Anytime

Write: Anytime when not in Initialization Mode

TXEIE2 - TXEIE0 — Transmitter Empty Interrupt Enable

1 = A transmitter empty (transmit buffer available for transmission) event causes a transmitter empty interrupt request.

0 = No interrupt request is generated from this event.

3.3.1.9 MSCAN Transmitter Message Abort Control (CANTARQ)

The CANTARQ register provides for abort request of queued messages as described below.

Address Offset: $__08

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-9 MSCAN Transmitter Message Abort Request (CANTARQ)

NOTE: *The CANTARQ register is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: Anytime

Write: Anytime when not in Initialization Mode

ABTRQ2 - ABTRQ0 — Abort Request

The CPU sets the ABTRQx bit to request that a scheduled message buffer (TXEx=0) be aborted. The MSCAN grants the request if the message has not already started transmission, or if the transmission is not successful (lost arbitration or error). When a message is aborted, the associated TXE (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and Abort Acknowledge flags (ABTAK, see **3.3.1.10 MSCAN Transmitter Message Abort Control (CANTAAK)**) are set and a transmit interrupt occurs if enabled. The CPU cannot reset ABTRQx. ABTRQx is reset whenever the associated TXE flag is set.

1 = Abort request pending.
0 = No abort request.

3.3.1.10 MSCAN Transmitter Message Abort Control (CANTAAK)

The CANTAAK register indicates the successful abort of a queued message, if requested by the appropriate bits in the CANTARQ register

Address Offset: $__09

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-10 MSCAN Transmitter Message Abort Control (CANTAAK)

NOTE: *The CANTAAK register is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1).*

Read: Anytime

Write: Unimplemented for ABTAKx flags;

ABTAK2 - ABTAK0 — Abort Acknowledge

This flag acknowledges that a message was aborted due to a pending abort request from the CPU. After a particular message buffer is flagged empty, this flag can be used by the application software to identify whether the message was aborted successfully or was sent anyway. The ABTAKx flag is cleared whenever the corresponding TXE flag is cleared.

1 = The message was aborted.
0 = The message was not aborted.

3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)

The CANTBSEL register allows the selection of the actual transmit message buffer, which will be then accessible in the CANTXFG register space (**3.3.1 Programmer's Model of Control Registers**).

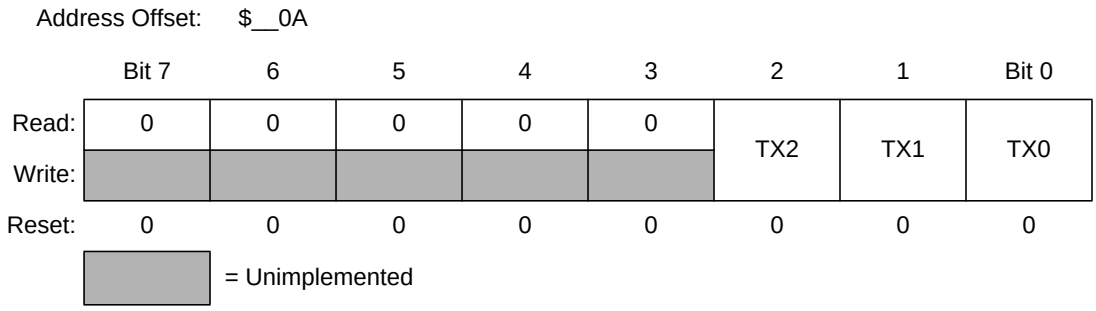


Figure 3-11 MSCAN Transmitter Flag Register (CANTBSEL)

NOTE: *The CANTBSEL register is held in the reset state when the Initialization Mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the Initialization Mode is left (INITRQ=0 and INITAK=0).*

Read: find the lowest ordered bit set to "1", all other bits will be read as "0"

Write: Anytime when not in Initialization Mode

TX2 - TX0 — Transmit Buffer Select

The lowest numbered bit places the respective transmit buffer in the CANTXFG register space (e.g. TX1=1 and TX0=1 selects transmit buffer TX0, TX1=1 and TX0=0 selects transmit buffer TX1)

Read and write accesses to the selected transmit buffer will be blocked, if the corresponding TXEx bit is cleared and the buffer is scheduled for transmission **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**.

1 = The associated message Buffer is selected, if lowest numbered bit.
0 = The associated message buffer is deselected

NOTE: *The following gives a short programming example of the usage of the CANTBSEL register:*

The application software wants to get the next available transmit buffer. It reads the CANTFLG register and writes this value back into the CANTBSEL register. In this example Tx buffers TX1 and TX2 are available. The value read from CANTFLG is therefore 0b0000_0110. When writing this value back to CANTBSEL the Tx buffer TX1 is selected in the CANTXFG because the lowest numbered bit set to "1" is at bit position 1. Reading back this value out of CANTBSEL results in 0b0000_0010, because only the lowest numbered bit position set to "1" is presented. This mechanism eases the application software the selection of the next available Tx buffer.

LDD CANTFLG; value read is 0b0000_0110

STD CANTBSEL; value written is 0b0000_0110

LDD CANTBSEL; value read is 0b0000_0010

If all transmit message buffers are deselected no accesses are allowed to the CANTXFG registers.

3.3.1.12 MSCAN Identifier Acceptance Control Register (CANIDAC)

The CANIDAC register provides for identifier acceptance control as described below.

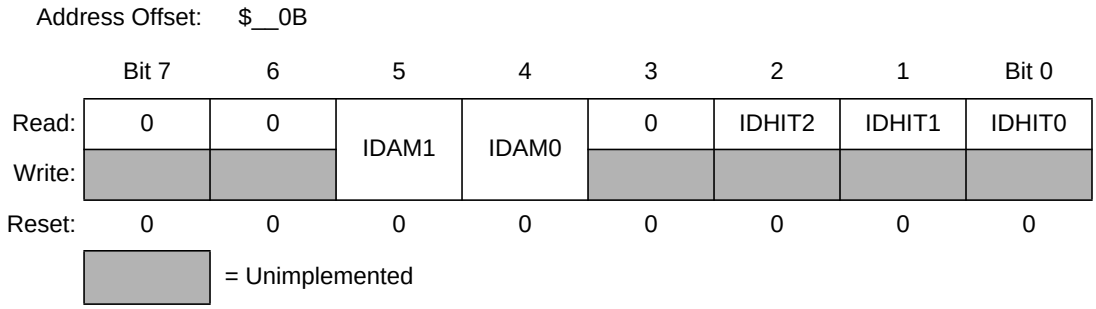


Figure 3-12 MSCAN Identifier Acceptance Control Register (CANIDAC)

Read: Anytime

Write: Anytime in Initialization Mode (INITRQ=1 and INITAK=1), except bits IDHITx which are read-only

IDAM1 - IDAM0 — Identifier Acceptance Mode

The CPU sets these flags to define the identifier acceptance filter organization **4.3 Identifier Acceptance Filter**. **Table 3-7** summarizes the different settings. In Filter Closed mode, no message is accepted such that the foreground buffer is never reloaded.

Table 3-7 Identifier Acceptance Mode Settings

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	Two 32 bit Acceptance Filters
0	1	Four 16 bit Acceptance Filters
1	0	Eight 8 bit Acceptance Filters
1	1	Filter Closed

IDHIT2 - IDHIT0 — Identifier Acceptance Hit Indicator

The MSCAN sets these flags to indicate an identifier acceptance hit **4.3 Identifier Acceptance Filter**. **Table 3-8** summarizes the different settings.

Table 3-8 Identifier Acceptance Hit Indication

IDHIT2	IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	0	Filter 0 Hit
0	0	1	Filter 1 Hit
0	1	0	Filter 2 Hit
0	1	1	Filter 3 Hit
1	0	0	Filter 4 Hit
1	0	1	Filter 5 Hit
1	1	0	Filter 6 Hit
1	1	1	Filter 7 Hit

The IDHITx indicators are always related to the message in the foreground buffer (RxFG). When a message gets shifted into the foreground buffer of the receiver FIFO the indicators are updated as well.

3.3.1.13 Reserved Registers

These registers are reserved for factory testing of the MSCAN module and are not available in normal system operation modes.

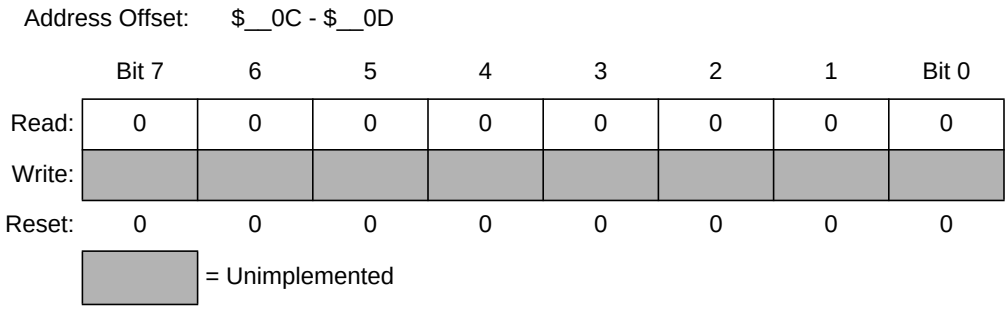


Figure 3-13 Reserved Registers

Read: always read $00 in normal system operation modes

Write: Unimplemented in normal system operation modes

NOTE: *Writing to these registers when in special modes can alter the MSCAN functionality.*

3.3.1.14 MSCAN Receive Error Counter Register (CANRXERR)

This register reflects the status of the MSCAN receive error counter.

Address Offset: $__0E

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-14 MSCAN Receive Error Counter Register (CANRXERR)

Read: only when in Sleep Mode (SLPRQ=1 and SLPAK=1) or Initialization Mode (INITRQ=1 and INITAK=1)

Write: Unimplemented

NOTE: *Reading this register when in any other mode other than Sleep or Initialization Mode, may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.*

NOTE: *Writing to this register when in special modes can alter the MSCAN functionality.*

3.3.1.15 MSCAN Transmit Error Counter Register (CANTXERR)

This register reflects the status of the MSCAN transmit error counter.

Address Offset: $__0F

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
Write:								
Reset:	0	0	0	0	0	0	0	0

= Unimplemented

Figure 3-15 MSCAN Transmit Error Counter Register (CANTXERR)

Read: only when in Sleep Mode (SLPRQ=1 and SLPAK=1) or Initialization Mode (INITRQ=1 and INITAK=1)

Write: Unimplemented

NOTE: *Reading this register when in any other mode other than Sleep or Initialization Mode, may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.*

NOTE: *Writing to this register when in special modes can alter the MSCAN functionality.*

3.3.1.16 MSCAN Identifier Acceptance Registers (CANIDAR0-7)

On reception, each message is written into the background receive buffer. The CPU is only signalled to read the message if it passes the criteria in the identifier acceptance and identifier mask registers (accepted); otherwise, the message is overwritten by the next message (dropped).

The acceptance registers of the MSCAN are applied on the IDR0 to IDR3 registers **3.3.2.1 Identifier Registers (IDR0-3)** of incoming messages in a bit by bit manner **4.3 Identifier Acceptance Filter**.

For extended identifiers, all four acceptance and mask registers are applied. For standard identifiers, only the first two (CANIDAR0/1, CANIDMR0/1) are applied.

	Bit 7	6	5	4	3	2	1	Bit 0
Address Offset:	$__10							CANIDAR0
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__11							CANIDAR1
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__12							CANIDAR2
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__13							CANIDAR3
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0

Figure 3-16 MSCAN Identifier Acceptance Registers (1st Bank)

	Bit 7	6	5	4	3	2	1	Bit 0
Address Offset:	$__18							CANIDAR4

Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__19							CANIDAR5
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__1A							CANIDAR6
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__1B							CANIDAR7
Read: Write:	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset:	0	0	0	0	0	0	0	0

Figure 3-17 MSCAN Identifier Acceptance Registers (2nd Bank)

Read: Anytime

Write: Anytime in Initialization Mode (INITRQ=1 and INITAK=1)

AC7 – AC0 — Acceptance Code Bits

AC7 – AC0 comprise a user defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is then masked with the corresponding identifier mask register.

3.3.1.17 MSCAN Identifier Mask Registers (CANIDMR0-7)

The identifier mask register specifies which of the corresponding bits in the identifier acceptance register are relevant for acceptance filtering. To receive standard identifiers in 32 bit filter mode, it is required to program the last three bits (AM2 - AM0) in the mask registers CANIDMR1 and CANIDMR5 to “don’t care”. To receive standard identifiers in 16 bit filter mode, it is required to program the last three bits (AM2 - AM0) in the mask registers CANIDMR1, CANIDMR3, CANIDMR5 and CANIDMR7 to “don’t care”.

Bit 7	6	5	4	3	2	1	Bit 0
Address Offset:	$__14						CANIDMR0

Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__15							CANIDMR1
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__16							CANIDMR2
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__17							CANIDMR3
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0

Figure 3-18 MSCAN Identifier Mask Registers (1st Bank)

	Bit 7	6	5	4	3	2	1	Bit 0
Address Offset:	$__1C							CANIDMR4
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__1D							CANIDMR5
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__1E							CANIDMR6
Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0
Address Offset:	$__1F							CANIDMR7

Figure 3-19 MSCAN Identifier Mask Registers (2nd Bank)

Read: Write:	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset:	0	0	0	0	0	0	0	0

Figure 3-19 MSCAN Identifier Mask Registers (2nd Bank)

Read: Anytime

Write: Anytime in Initialization Mode (INITRQ=1 and INITAK=1)

AM7 – AM0 — Acceptance Mask Bits

If a particular bit in this register is cleared, this indicates that the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates that the state of the corresponding bit in the identifier acceptance register does not affect whether or not the message is accepted.

1 = Ignore corresponding acceptance code register bit.

0 = Match corresponding acceptance code register and identifier bits.

3.3.2 Programmer's Model of Message Storage

The following section details the organization of the receive and transmit message buffers and the associated control registers.

For reasons of programmer interface simplification, the receive and transmit message buffers have the same outline. Each message buffer allocates 16 bytes in the memory map containing a 13 byte data structure.

An additional Transmit Buffer Priority Register (TBPR) is defined for the transmit buffers. Within the last two bytes of this memory map the MSCAN stores a special 16-bit time stamp, which is sampled from an internal timer after successful transmission or reception of a message. This feature is only available for transmit and receiver buffers, if the TIME bit is set (**3.3.1.1 MSCAN Control 0 Register (CANCTL0)**).

The Time Stamp register is written by the MSCAN. The CPU can only read these registers.

Addr	Register Name
$__x0	Identifier Register 0
$__x1	Identifier Register 1
$__x2	Identifier Register 2
$__x3	Identifier Register 3
$__x4	Data Segment Register 0
$__x5	Data Segment Register 1
$__x6	Data Segment Register 2
$__x7	Data Segment Register 3
$__x8	Data Segment Register 4
$__x9	Data Segment Register 5
$__xA	Data Segment Register 6
$__xB	Data Segment Register 7
$__xC	Data Length Register
$__xD	Transmit Buffer Priority Register[1]
$__xE	Time Stamp Register (High Byte)[2]
$__xF	Time Stamp Register (Low Byte)[3]

Table 3-9 Message Buffer Organization

NOTES:
1. Not Applicable for Receive Buffers
2. Read-Only for CPU
3. Read-Only for CPU

Figure 3-20 shows the common 13 byte data structure of receive and transmit buffers for extended identifiers. The mapping of standard identifiers into the IDR registers is shown in **Figure 3-21**.

All bits of the receive and transmit buffers are ‘x’ out of reset because of RAM based implementation[1]. All reserved or unused bits of the receive and transmit buffers are always read ‘x’.

Register name		Bit 7	6	5	4	3	2	1	Bit 0	ADDR
IDR0	Read: Write:	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21	$__x0
IDR1	Read: Write:	ID20	ID19	ID18	SRR (=1)	IDE (=1)	ID17	ID16	ID15	$__x1
IDR2	Read: Write:	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	$__x2
IDR3	Read: Write:	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR	$__x3

= Unused[1]

Figure 3-20 Receive / Transmit Message Buffer Extended Identifier

NOTES:
1. Exception: The Transmit Priority Registers are “0” out of reset

Register name		Bit 7	6	5	4	3	2	1	Bit 0	ADDR
DSR0	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x4
DSR1	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x5
DSR2	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x6
DSR3	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x7
DSR4	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x8
DSR5	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__x9
DSR6	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__xA
DSR7	Read: Write:	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	$__xB
DLR	Read: Write:					DLC3	DLC2	DLC1	DLC0	$__xC

= Unused[1]

Figure 3-20 Receive / Transmit Message Buffer Extended Identifier

NOTES:
1. Unused bits are always read ‘x’

Read: Anytime for transmit buffers when TXEx flag is set (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and the corresponding transmit buffer is selected in CANTBSEL (see **3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**); only when RXF flag is set for receive buffers (see **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)**).

Write: Anytime for transmit buffers when TXEx flag is set (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and the corresponding transmit buffer is selected in CANTBSEL (see **3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**); unimplemented for receive buffers

Reset: $xx because of RAM based implementation

Register name		Bit 7	6	5	4	3	2	1	Bit 0	ADDR
IDR0	Read: Write:	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	$__x0
IDR1	Read: Write:	ID2	ID1	ID0	RTR	IDE (=0)				$__x1
IDR2	Read: Write:									$__x2
IDR3	Read: Write:									$__x3

= Unused[1]

Figure 3-21 Standard Identifier Mapping

NOTES:
1. Unused bits are always read ‘x’

3.3.2.1 Identifier Registers (IDR0-3)

The identifier registers for an extended format identifier consist of a total of 32 bits; ID28 - ID0, SRR, IDE, and RTR bits. The identifier registers for a standard format identifier consist of a total of 13 bits; ID10 - ID0, RTR, and IDE bits.

ID28 - ID0 — Extended format identifier

The identifiers consist of 29 bits (ID28 - ID0) for the extended format. ID28 is the most significant bit and is transmitted first on the bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

ID10 - ID0 — Standard format identifier

The identifiers consist of 11 bits (ID10 – ID0) for the standard format. ID10 is the most significant bit and is transmitted first on the bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

SRR — Substitute Remote Request

This fixed recessive bit is used only in extended format. It must be set to 1 by the user for transmission buffers and is stored as received on the CAN bus for receive buffers.

IDE — ID Extended

This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the CPU how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send.

1 = Extended format (29 bit)
0 = Standard format (11 bit)

RTR — Remote Transmission Request

This flag reflects the status of the Remote Transmission Request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent.

1 = Remote frame
0 = Data frame

3.3.2.2 Data Segment Registers (DSR0-7)

The eight data segment registers, each with bits DB7-DB0, contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the corresponding DLR register.

DB7 - DB0 — Data Bits 7-0

3.3.2.3 Data Length Register (DLR)

This register keeps the data length field of the CAN frame.

DLC3 - DLC0 — Data Length Code bits

The data length code contains the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0. The data byte count ranges from 0 to 8 for a data frame. **Table 3-10** shows the effect of setting the DLC bits.

Table 3-10 Data length codes

Data length code				Data byte count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

3.3.2.4 Transmit Buffer Priority Register (TBPR)

This register defines the local priority of the associated message buffer. The local priority is used for the internal prioritization process of the MSCAN and is defined to be highest for the smallest binary number. The MSCAN implements the following internal prioritization mechanisms:

- All transmission buffers with a cleared TXEx flag participate in the prioritization immediately before the SOF (Start of Frame) is sent.

- The transmission buffer with the lowest local priority field wins the prioritization.

In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins.

Address Offset: $xxxD

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
Reset:	0	0	0	0	0	0	0	0

Figure 3-22 Transmit Buffer Priority Register (TBPR)

Read: Anytime when TXEx flag is set (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and the corresponding transmit buffer is selected in CANTBSEL (see **3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**)

Write: Anytime when TXEx flag is set (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and the corresponding transmit buffer is selected in CANTBSEL (see **3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**)

3.3.2.5 Time Stamp Register (TSRH, TSRL)

If the TIME bit is enabled, the MSCAN will write a special time stamp to the respective registers in the active transmit or receive buffer as soon as a message has been acknowledged on the CAN bus (**3.3.1.1 MSCAN Control 0 Register (CANCTL0)**). The time stamp is written on the bit sample point for the recessive bit of the ACK delimiter in the CAN frame. In case of a transmission, the CPU can only read the time stamp after the respective transmit buffer has been flagged empty.

The timer value, which is used for stamping, is taken from a free running internal CAN bit clock. A timer overrun is not indicated by the MSCAN. The timer is reset (all bits set to “0”) during Initialization Mode. The CPU can only read the Time Stamp registers.

Address Offset: $xxxE

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
Write:								
Reset:	X	X	X	X	X	X	X	X

Figure 3-23 Time Stamp Register (TSRH - High Byte)

Address Offset: $xxxF

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
Write:								
Reset:	X	X	X	X	X	X	X	X

Figure 3-24 Time Stamp Register (TSRL - Low Byte)

Read: Anytime when TXEx flag is set (see **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**) and the corresponding transmit buffer is selected in CANTBSEL (see **3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**)

Write: Unimplemented

Section 4 Functional Description

4.1 General

This section provides a complete functional description of the MSCAN. It describes each of the features and modes listed in the introduction.

4.2 Message Storage

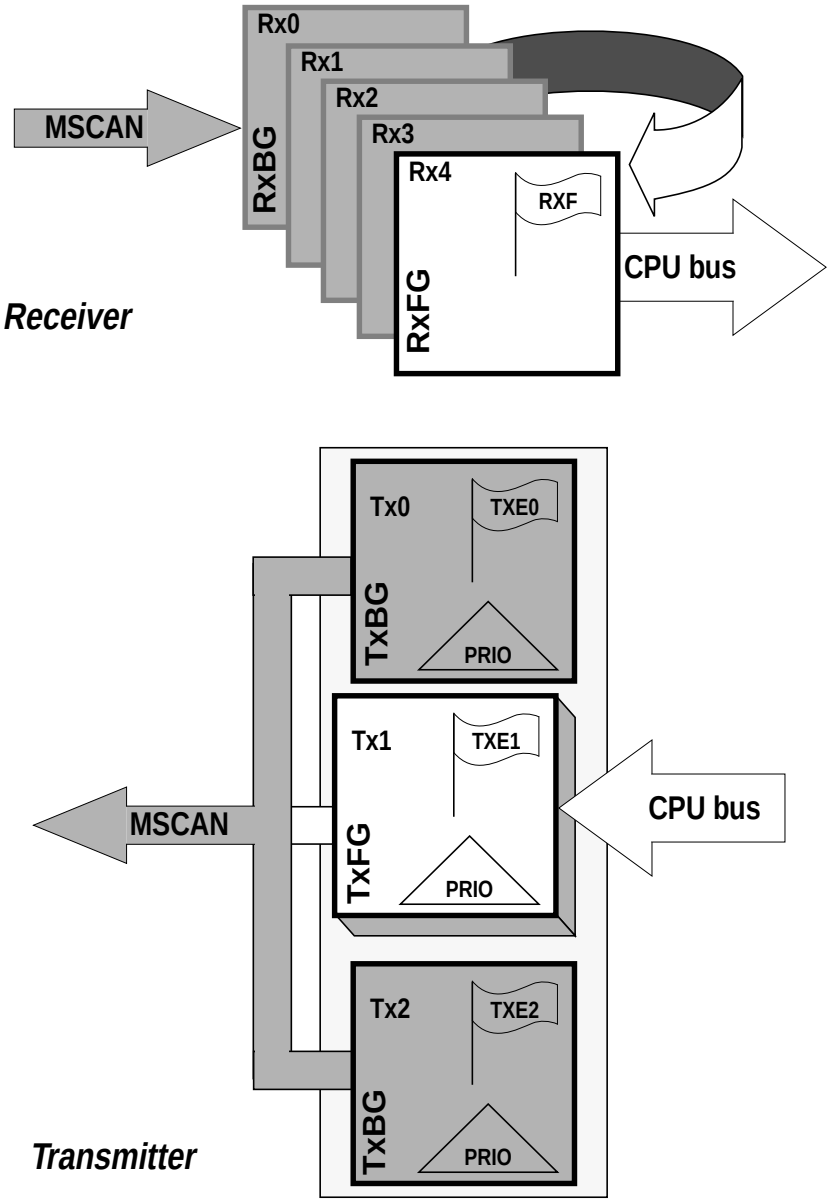


Figure 4-1 User Model for Message Buffer Organization

MSCAN facilitates a sophisticated message storage system which addresses the requirements of a broad range of network applications.

4.2.1 Message Transmit Background

Modern application layer software is built upon two fundamental assumptions:

- Any CAN node is able to send out a stream of scheduled messages without releasing the bus between the two messages. Such nodes arbitrate for the bus immediately after sending the previous message and only release the bus in case of lost arbitration.
- The internal message queue within any CAN node is organized such that the highest priority message is sent out first, if more than one message is ready to be sent.

The above behavior cannot be achieved with a single transmit buffer. That buffer must be reloaded right after the previous message is sent. This loading process lasts a finite amount of time and has to be completed within the Inter-Frame Sequence (IFS)[1] to be able to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires that the CPU react with short latencies to the transmit interrupt.

A double buffer scheme de-couples the reloading of the transmit buffer from the actual message sending and, as such, reduces the reactiveness requirements on the CPU. Problems can arise if the sending of a message is finished while the CPU re-loads the second buffer. No buffer would then be ready for transmission and the bus would be released.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The MSCAN has three transmit buffers.

The second requirement calls for some sort of internal prioritization which the MSCAN implements with the “local priority” concept described in **4.2.2 Transmit Structures**.

4.2.2 Transmit Structures

The MSCAN has a triple transmit buffer scheme which allows multiple messages to be set up in advance and achieve an optimized real-time performance. The three buffers are arranged as shown in **Figure 4-1 User Model for Message Buffer Organization**.

All three buffers have a 13 byte data structure similar to the outline of the receive buffers **3.3.2 Programmer’s Model of Message Storage**. An additional **3.3.2.4 Transmit Buffer Priority Register (TBPR)** contains an 8-bit “Local Priority” field (PRIO) (see **3.3.2.4 Transmit Buffer Priority Register (TBPR)**). The remaining two bytes are used for time stamping of a message, if required (see **3.3.2.5 Time Stamp Register (TSRH, TSRL)**).

To transmit a message, the CPU has to identify an available transmit buffer which is indicated by a set Transmitter Buffer Empty (TXEx) flag **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**. If a transmit buffer is available, the CPU has to set a pointer to this buffer by writing to the CANTBSEL register (**3.3.1.11 MSCAN Transmit Buffer Selection (CANTBSEL)**). This makes the respective buffer

NOTES:
1. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991.

accessible within the CANTXFG address space **3.3.2 Programmer's Model of Message Storage**. The algorithmic feature associated with the CANTBSEL register simplifies the transmit buffer selection. In addition this scheme makes the handler software simpler as only one address area is applicable for the transmit process. In addition the required address space is minimized.

The CPU then stores the identifier, the control bits and the data content into one of the transmit buffers. Finally, the buffer is flagged as ready for transmission by clearing the associated TXE flag.

The MSCAN then schedules the message for transmission and signals the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt **4.9.1 Transmit Interrupt** is generated[1] when TXEx is set and can be used to drive the application software to re-load the buffer.

In case more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the MSCAN uses the "local priority" setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit local priority field (PRIO). The application software programs this field when the message is set up. The local priority reflects the priority of this particular message relative to the set of messages being transmitted from this node. The lowest binary value of the PRIO field is defined to be the highest priority. The internal scheduling process takes place whenever the MSCAN arbitrates for the bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message in one of the three transmit buffers. As messages that are already in transmission cannot be aborted, the user has to request the abort by setting the corresponding Abort Request bit (ABTRQ) **3.3.1.9 MSCAN Transmitter Message Abort Control (CANTARQ)**. The MSCAN then grants the request, if possible, by: 1) setting the corresponding Abort Acknowledge flag (ABTAK) in the CANTAAK register, 2) setting the associated TXE flag to release the buffer, and 3) generating a transmit interrupt. The transmit interrupt handler software can tell from the setting of the ABTAK flag whether the message was aborted (ABTAK=1) or sent (ABTAK=0).

4.2.3 Receive Structures

The received messages are stored in a five stage input FIFO. The five message buffers are alternately mapped into a single memory area **Figure 4-1 User Model for Message Buffer Organization**. While the background receive buffer (RxBG) is exclusively associated with the MSCAN, the foreground receive buffer (RxFG) is addressable by the CPU **Figure 4-1 User Model for Message Buffer Organization**. This scheme simplifies the handler software as only one address area is applicable for the receive process.

All receive buffers have a size of 15 bytes to store the CAN control bits, the identifier (standard or extended), the data contents and a time stamp, if enabled (for details **3.3.2 Programmer's Model of Message Storage**)[2].

The Receiver Full flag (RXF) **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)** signals the status of the foreground receive buffer. When the buffer contains a correctly received message with a matching identifier, this flag is set.

NOTES:
1. The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.
2. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for details.

On reception, each message is checked to see if it passes the filter (**4.3 Identifier Acceptance Filter**) and in parallel, is written into the active RxBG. After successful reception of a valid message the MSCAN shifts the content of RxBG into the receiver FIFO[1], sets the RXF flag, and generates a receive interrupt **4.9.2 Receive Interrupt** to the CPU[2]. The user's receive handler has to read the received message from the RxFG and then reset the RXF flag to acknowledge the interrupt and to release the foreground buffer. A new message, which can follow immediately after the IFS field of the CAN frame, is received into the next available RxBG. If the MSCAN receives an invalid message in its RxBG (wrong identifier, transmission errors etc.) the actual contents of the buffer will be over-written by the next message. The buffer will then not be shifted into the FIFO.

When the MSCAN module is transmitting, the MSCAN receives its own transmitted messages into the background receive buffer, RxBG, but does not shift it into the receiver FIFO, generate a receive interrupt, or acknowledge its own messages on the CAN bus. The exception to this rule is in loop back mode **3.3.1.2 MSCAN Control 1 Register (CANCTL1)** where the MSCAN treats its own messages exactly like all other incoming messages. The MSCAN receives its own transmitted messages in the event that it loses arbitration[3]. If arbitration is lost, the MSCAN must be prepared to become a receiver.

An overrun condition occurs when all receive message buffers in the FIFO are filled with correctly received messages with accepted identifiers and another message is correctly received from the bus with an accepted identifier. The latter message is discarded and an error interrupt with overrun indication is generated if enabled **4.9.4 Error Interrupt**. The MSCAN is still able to transmit messages while the receiver FIFO being filled, but all incoming messages are discarded. As soon as a receive buffer in the FIFO is available again, new valid messages will be accepted.

4.3 Identifier Acceptance Filter

The MSCAN Identifier Acceptance Registers (**3.3.1.12 MSCAN Identifier Acceptance Control Register (CANIDAC)**) define the acceptable patterns of the standard or extended identifier (ID10 - ID0 or ID28 - ID0). Any of these bits can be marked 'don't care' in the MSCAN Identifier Mask Registers **3.3.1.17 MSCAN Identifier Mask Registers (CANIDMR0-7)**.

A filter hit is indicated to the application software by a set Receive Buffer Full flag (RXF=1) and three bits in the CANIDAC register **3.3.1.12 MSCAN Identifier Acceptance Control Register (CANIDAC)**. These Identifier Hit flags (IDHIT2-0) clearly identify the filter section that caused the acceptance. They simplify the application software's task to identify the cause of the receiver interrupt. In case more than one hit occurs (two or more filters match), the lower hit has priority.

A very flexible programmable generic identifier acceptance filter has been introduced to reduce the CPU interrupt loading. The filter is programmable to operate in four different modes[4]:

- Two identifier acceptance filters, each to be applied to a) the full 29 bits of the extended identifier and to the following bits of the CAN 2.0B frame: Remote Transmission Request (RTR), Identifier Extension (IDE), and Substitute Remote Request (SRR) or b)[5] the 11 bits of the standard identifier

NOTES:
1. Only if the RXF flag is not set.
2. The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.
3. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for details.
4. For a better understanding of references made within the filter mode description, reference the Bosch specification dated September 1991 which details the CAN 2.0A/B protocol.

plus the RTR and IDE bits of the CAN 2.0A/B messages. This mode implements two filters for a full length CAN 2.0B compliant extended identifier. **Figure 4-2** shows how the first 32-bit filter bank (CANIDAR0-3, CANIDMR0-3) produces a filter 0 hit. Similarly, the second filter bank (CANIDAR4-7, CANIDMR4-7) produces a filter 1 hit.

- Four identifier acceptance filters, each to be applied to a) the 14 most significant bits of the extended identifier plus the SRR and IDE bits of CAN 2.0B messages or b) the 11 bits of the standard identifier, the RTR and IDE bits of CAN 2.0A/B messages. **Figure 4-3** shows how the first 32-bit filter bank (CANIDAR0-3, CANIDMR0-3) produces filter 0 and 1 hits. Similarly, the second filter bank (CANIDAR4-7, CANIDMR4-7) produces filter 2 and 3 hits.
- Eight identifier acceptance filters, each to be applied to the first 8 bits of the identifier. This mode implements eight independent filters for the first 8 bits of a CAN 2.0A/B compliant standard identifier or a CAN 2.0B compliant extended identifier. **Figure 4-4** shows how the first 32-bit filter bank (CANIDAR0-3, CANIDMR0-3) produces filter 0 to 3 hits. Similarly, the second filter bank (CANIDAR4-7, CANIDMR4-7) produces filter 4 to 7 hits.
- Closed filter. No CAN message is copied into the foreground buffer RxFG, and the RXF flag is never set.

NOTES:

5. Although this mode can be used for standard identifiers, it is recommended to use the four or eight identifier acceptance filters for standard identifiers

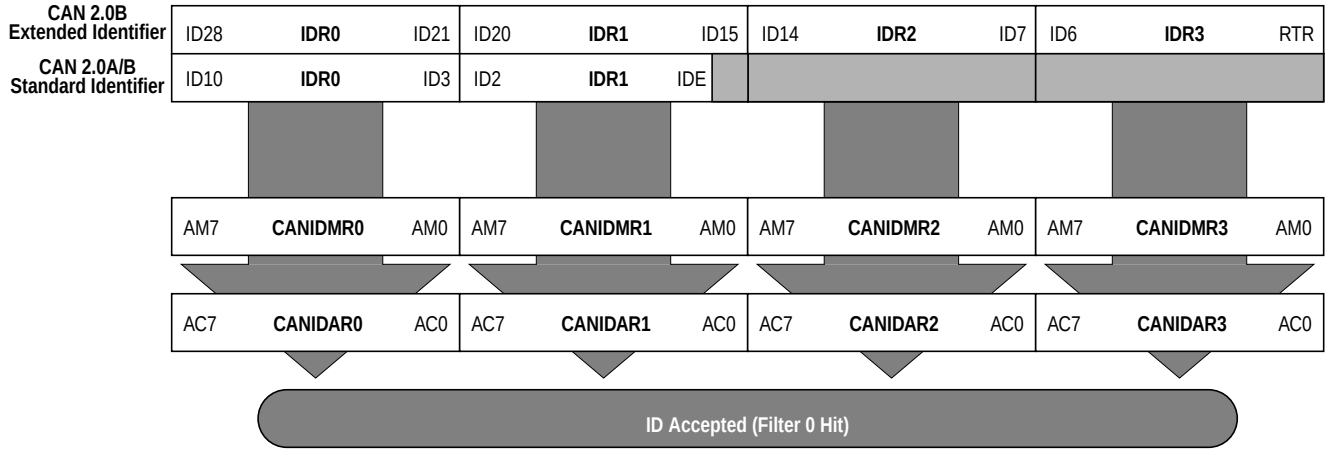


Figure 4-2 32-bit Maskable Identifier Acceptance Filter

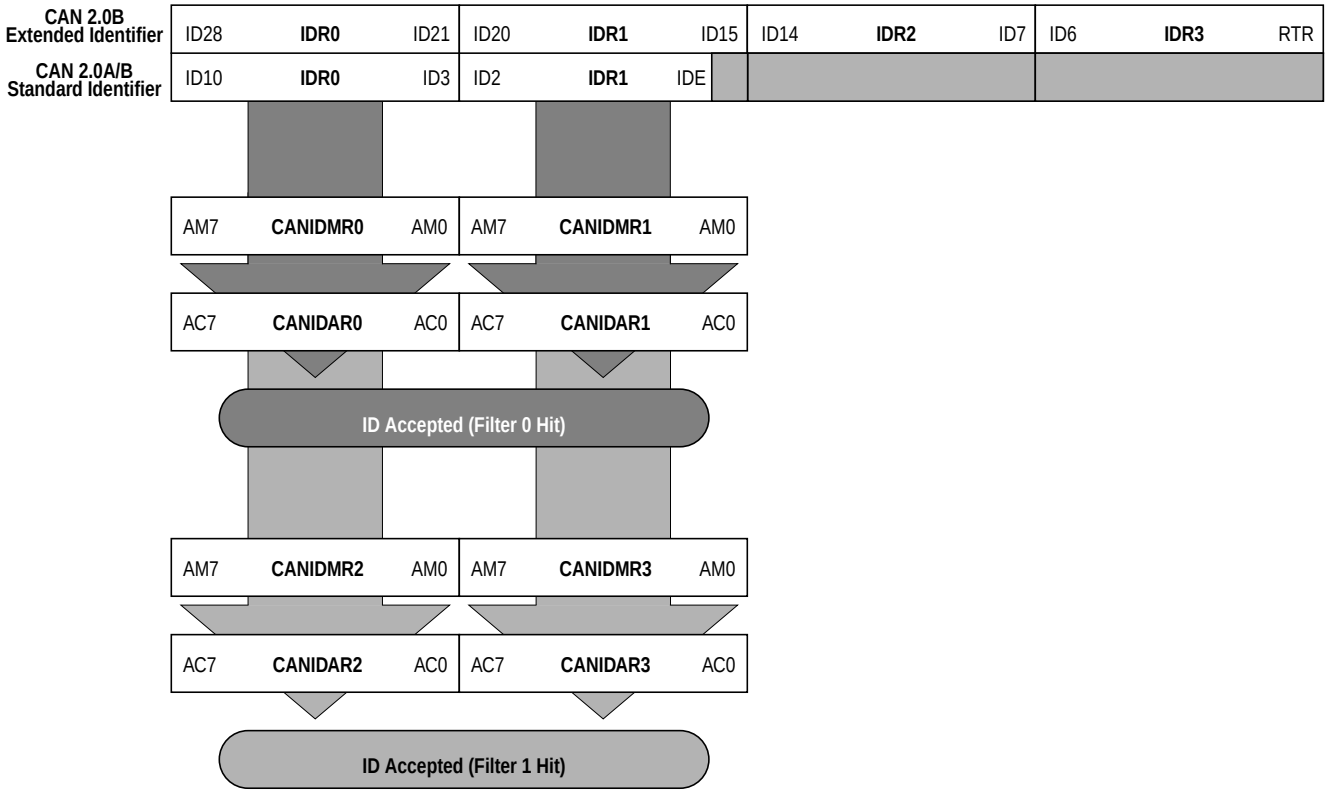


Figure 4-3 16-bit Maskable Identifier Acceptance Filters

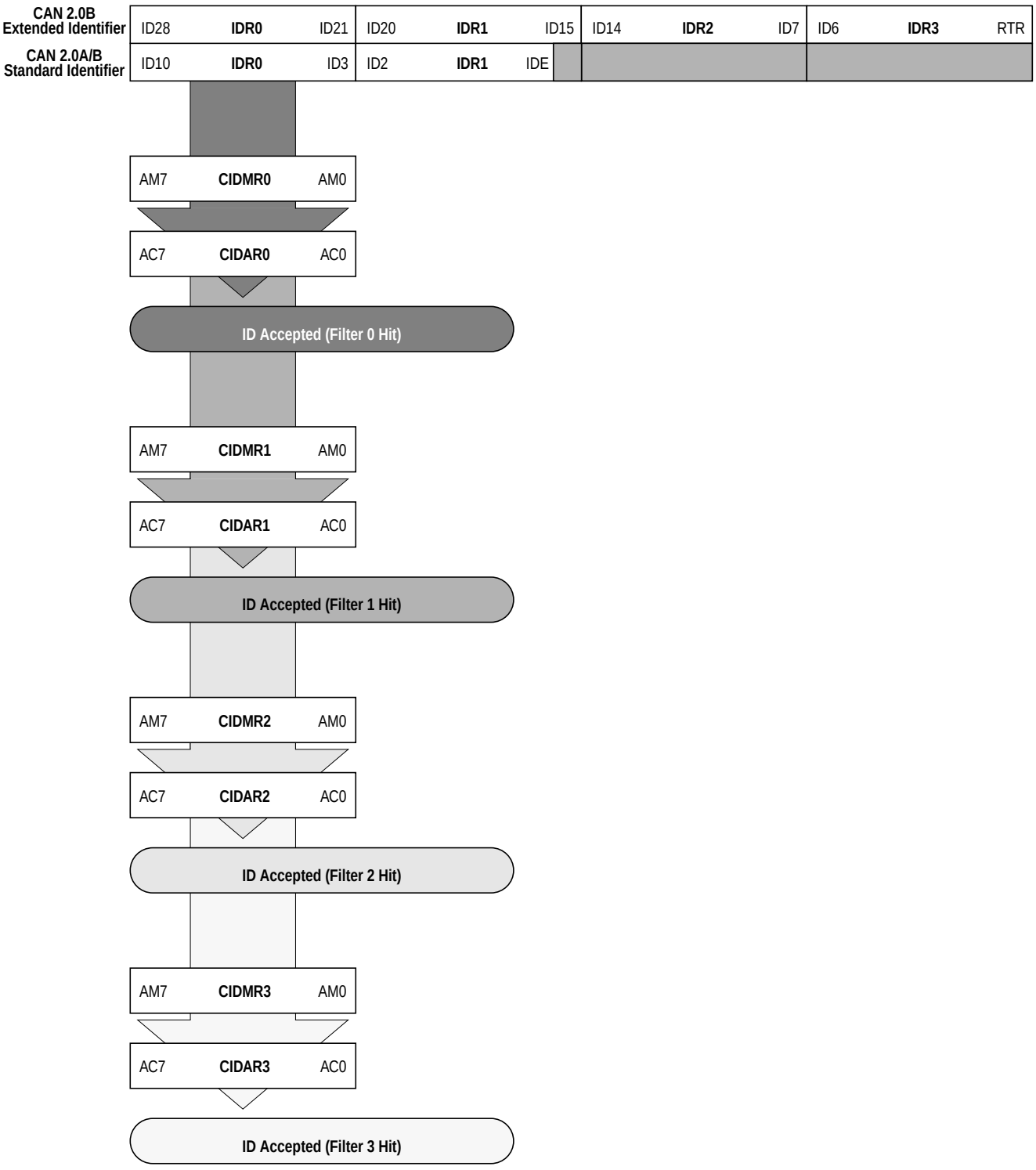


Figure 4-4 8-bit Maskable Identifier Acceptance Filters

4.3.1 Protocol Violation Protection

The MSCAN protects the user from accidentally violating the CAN protocol through programming errors. The protection logic implements the following features:

- The receive and transmit error counters cannot be written or otherwise manipulated.
- All registers which control the configuration of the MSCAN cannot be modified while the MSCAN is on-line. The MSCAN has to be in Initialization Mode. The corresponding INITRQ/INITAK handshake bits in the CANCTL0/CANCTL1 registers **3.3.1.1 MSCAN Control 0 Register (CANCTL0)** serve as a lock to protect the following registers:
  - MSCAN Control 1 Register (CANCTL1)
  - MSCAN Bus Timing Registers 0 and 1 (CANBTR0, CANBTR1)
  - MSCAN Identifier Acceptance Control Register (CANIDAC)
  - MSCAN Identifier Acceptance Registers (CANIDAR0-7)
  - MSCAN Identifier Mask Registers (CANIDMR0-7)
- The TXCAN pin is immediately forced to a recessive state when the MSCAN goes into the Power Down Mode or Initialization Mode (see **4.6.6 MSCAN Power Down Mode** and **4.6.5 MSCAN Initialization Mode**).
- The MSCAN enable bit (CANE) is only writable once in normal system operation modes as further protection against inadvertently disabling the MSCAN.

4.3.2 Clock System

Figure 4-5 shows the structure of the MSCAN clock generation circuitry. With this flexible clocking scheme, the MSCAN is able to handle CAN bus rates ranging from 10 Kbps up to 1 Mbps.

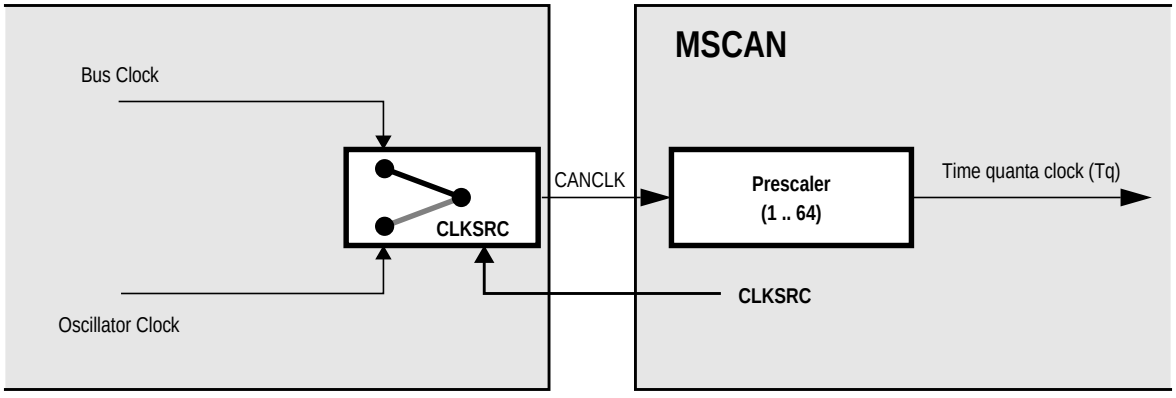


Figure 4-5 MSCAN Clocking Scheme

The clock source bit (CLKSRC) in the CANCTL1 register **3.3.1.2 MSCAN Control 1 Register (CANCTL1)** defines whether the internal CANCLK is connected to the output of a crystal oscillator (Oscillator Clock) or to the Bus Clock.

The clock source has to be chosen such that the tight oscillator tolerance requirements (up to 0.4%) of the CAN protocol are met. Additionally, for high CAN bus rates (1 Mbps), a 45%-55% duty cycle of the clock is required.

If the Bus Clock is generated from a PLL, it is recommended to select the Oscillator Clock rather than the Bus Clock due to jitter considerations, especially at the faster CAN bus rates.

For microcontrollers without a clock and reset generator (CRG), CANCLK is driven from the crystal oscillator (Oscillator Clock).

A programmable prescaler generates the time quanta (Tq) clock from CANCLK. A time quantum is the atomic unit of time handled by the MSCAN.

$$f_{Tq} = \frac{f_{CANCLK}}{(\text{Prescaler value})}$$

A bit time is subdivided into three segments[1] [2] (reference **Figure 4-6**):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the PROP_SEG and the PHASE_SEG1 of the CAN standard. It can be programmed by setting the parameter TSEG1 to consist of 4 to 16 time quanta.
- Time Segment 2: This segment represents the PHASE_SEG2 of the CAN standard. It can be programmed by setting the TSEG2 parameter to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})}$$

NOTES:
1. For further explanation of the under-lying concepts please refer to ISO/DIS 11519-1, Section 10.3.
2. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

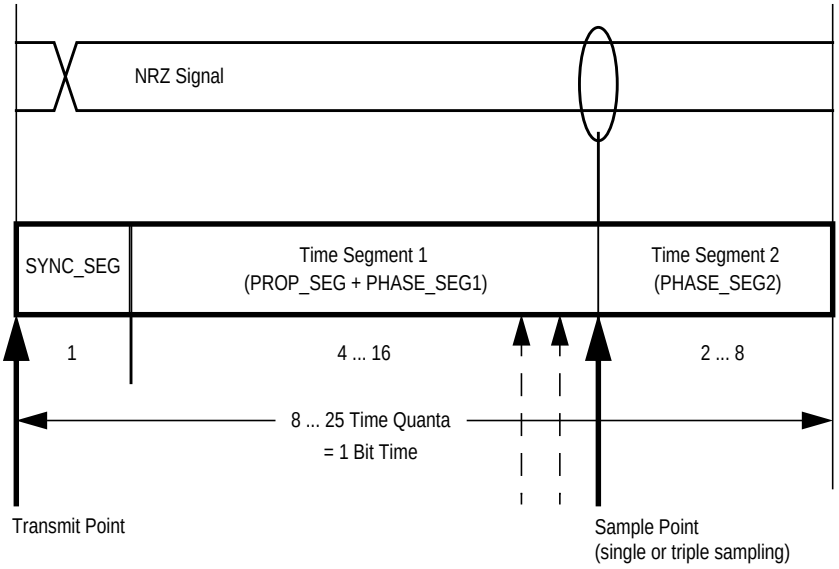


Figure 4-6 Segments within the Bit Time

Table 4-1 Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

The Synchronization Jump Width[1] can be programmed in a range of 1 to 4 time quanta by setting the SJW parameter.

The above parameters are set by programming the MSCAN Bus Timing Registers (CANBTR0, CANBTR1) (see **3.3.1.3 MSCAN Bus Timing Register 0 (CANBTR0)** and **3.3.1.4 MSCAN Bus Timing Register 1 (CANBTR1)**).

Table 4-2 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE: *It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard.*

NOTES:
1. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Time Segment 1	TSEG1	Time Segment 2	TSEG2	Synchronization Jump Width	SJW
5 .. 10	4 .. 9	2	1	1 .. 2	0 .. 1
4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
5 .. 12	4 .. 11	4	3	1 .. 4	0 .. 3
6 .. 13	5 .. 12	5	4	1 .. 4	0 .. 3
7 .. 14	6 .. 13	6	5	1 .. 4	0 .. 3
8 .. 15	7 .. 14	7	6	1 .. 4	0 .. 3
9 .. 16	8 .. 15	8	7	1 .. 4	0 .. 3

Table 4-2 CAN Standard Compliant Bit Time Segment Settings

4.4 Timer Link

The MSCAN generates an internal time stamp whenever a valid frame is received or transmitted and the TIME bit is enabled. Because the CAN specification defines a frame to be valid if no errors occur before the End of Frame (EOF) field is transmitted successfully, the actual value of an internal timer is written at EOF to the appropriate time stamp position within the transmit buffer. For receive frames the time stamp is written to the receive buffer.

4.5 Modes of Operation

4.5.1 Normal Modes

The MSCAN module behaves as described within this specification in all normal system operation modes.

4.5.2 Special Modes

The MSCAN module behaves as described within this specification in all special system operation modes.

4.5.3 Emulation Modes

In all emulation modes, the MSCAN module behaves just like normal system operation modes as described within this specification.

4.5.4 Listen-Only Mode

In an optional bus monitoring mode (Listen-Only), the CAN node is able to receive valid data frames and valid remote frames, but it sends only “recessive” bits on the CAN bus. In addition it cannot start a transmision. If the MAC sub-layer is required to send a “dominant” bit (ACK bit, overload flag, active

error flag), the bit is rerouted internally so that the MAC sub-layer monitors this “dominant” bit, although the CAN bus may remain in recessive state externally.

4.5.5 Security Modes

The MSCAN module has no security features.

4.6 Low Power Options

If the MSCAN is disabled (CANE=0), the MSCAN clocks are stopped for power savings.

If the MSCAN is enabled (CANE=1), the MSCAN has two additional modes with reduced power consumption, compared to Normal Mode: Sleep and Power Down Mode. In Sleep Mode power consumption is reduced by stopping all clocks except those to access the registers from the CPU side. In Power Down Mode, all clocks are stopped and no power is consumed.

Table 4-3 summarizes the combinations of MSCAN and CPU modes. A particular combination of modes is entered by the given settings on the CSWAI and SLPRQ/SLPAK bits.

For all modes, an MSCAN Wake-Up interrupt can only occur if the MSCAN is in Sleep Mode (SLPRQ=1 and SLPAK=1), wake-up functionality is enabled (WUPE=1) and the Wake-Up interrupt is enabled (WUPIE=1).

Table 4-3 CPU vs. MSCAN Operating Modes

CPU Mode	MSCAN Mode			
	Normal	Reduced Power Consumption		
		Sleep	Power Down	Disabled (CANE=0)
RUN	CSWAI = X[1] SLPRQ = 0 SLPAK = 0	CSWAI = X SLPRQ = 1 SLPAK = 1		CSWAI = X SLPRQ = X SLPAK = X
WAIT	CSWAI = 0 SLPRQ = 0 SLPAK = 0	CSWAI = 0 SLPRQ = 1 SLPAK = 1	CSWAI = 1 SLPRQ = X SLPAK = X	CSWAI = X SLPRQ = X SLPAK = X
STOP			CSWAI = X SLPRQ = X SLPAK = X	CSWAI = X SLPRQ = X SLPAK = X

NOTES:
1. ‘X’ means don’t care.

4.6.1 CPU Run Mode

As can be seen in **Table 4-3 CPU vs. MSCAN Operating Modes**, only MSCAN Sleep Mode is available as low power option, when CPU is in Run Mode.

4.6.2 CPU Wait Mode

The WAI instruction puts the MCU in a low power consumption stand-by mode. If the CSWAI bit is set, then additional power can be saved in Power Down Mode since the CPU clocks are stopped. After leaving this Power Down Mode the MSCAN restarts its internal controllers and enters Normal Mode again.

While the CPU is in Wait Mode, the MSCAN can be operated in Normal Mode and generate interrupts (registers can be accessed via background debug mode). The MSCAN can also operate in any of the low power modes depending on the values of the SLPRQ/SLPAK and CSWAI bits as seen in **Table 4-3 CPU vs. MSCAN Operating Modes**.

4.6.3 CPU Stop Mode

The STOP instruction puts the MCU in a low power consumption stand-by mode. In Stop Mode, the MSCAN set in Power Down mode regardless of the value of the SLPRQ/SLPAK and CSWAI bits **Table 4-3**.

4.6.4 MSCAN Sleep Mode

The CPU can request the MSCAN to enter this low power mode by asserting the SLPRQ bit in the CANCTL0 register. The time when the MSCAN enters Sleep Mode depends on a fixed synchronization delay and its current activity:

- If there are one or more message buffers scheduled for transmission (TXEx = 0), the MSCAN will continue to transmit until all transmit message buffers are empty (TXEx = 1, transmitted successfully or aborted) and then goes into Sleep Mode.
- If it is receiving, it continues to receive and goes into Sleep Mode as soon as the CAN bus next becomes idle.
- If it is neither transmitting nor receiving, it immediately goes into Sleep Mode.

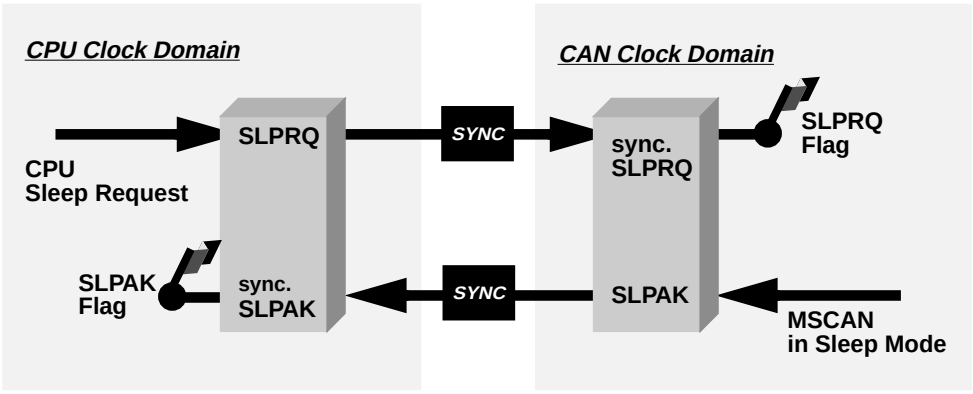


Figure 4-7 Sleep Request / Acknowledge Cycle

NOTE: *The application software must avoid setting up a transmission (by clearing one or more TXEx flag(s)) and immediately request Sleep Mode (by setting SLPRQ). It depends on the exact sequence of operations whether the MSCAN starts transmitting or goes into Sleep Mode directly.*

If Sleep Mode is active, the SLPRQ and SLPAK bits are set (**Figure 4-7**). The application software must use SLPAK as a handshake indication for the request (SLPRQ) to go into Sleep Mode.

When in Sleep Mode (SLPRQ=1 and SLPAK=1), the MSCAN stops its internal clocks. However, clocks to allow register accesses from the CPU side still run.

If the MSCAN is in Bus-Off state, it stops counting the 128*11 consecutive recessive bits due to the stopped clocks. The TXCAN pin remains in a recessive state. If RXF=1, the message can be read and RXF can be cleared. Shifting a new message into the foreground buffer of the receiver FIFO (RxFG) does not take place while in Sleep Mode.

It is possible to access the transmit buffers and to clear the associated TXE flags. No message abort takes place while in Sleep Mode.

If the WUPE bit in CANCLT0 is not asserted, the MSCAN will mask any activity it detects on CAN. The RXCAN pin is therefore held internally in a recessive state. This locks the MSCAN in Sleep Mode (**Figure 4-8 Simplified State Transitions for Entering/Leaving Sleep Mode**).

The MSCAN is only able to leave Sleep Mode (wake up) when

- bus activity occurs and WUPE=1 or
- the CPU clears the SLPRQ bit

NOTE: *The CPU cannot clear the SLPRQ bit before Sleep Mode (SLPRQ=1 and SLPAK=1) is active.*

After wake-up, the MSCAN waits for 11 consecutive recessive bits to synchronize to the bus. As a consequence, if the MSCAN is woken-up by a CAN frame, this frame is not received.

The receive message buffers (RxFG and RxBG) contain messages if they were received before Sleep Mode was entered. All pending actions will be executed upon wake-up; copying of RxBG into RxFG, message aborts and message transmissions. If the MSCAN is still in Bus-Off state after Sleep Mode was left, it continues counting the 128*11 consecutive recessive bits.

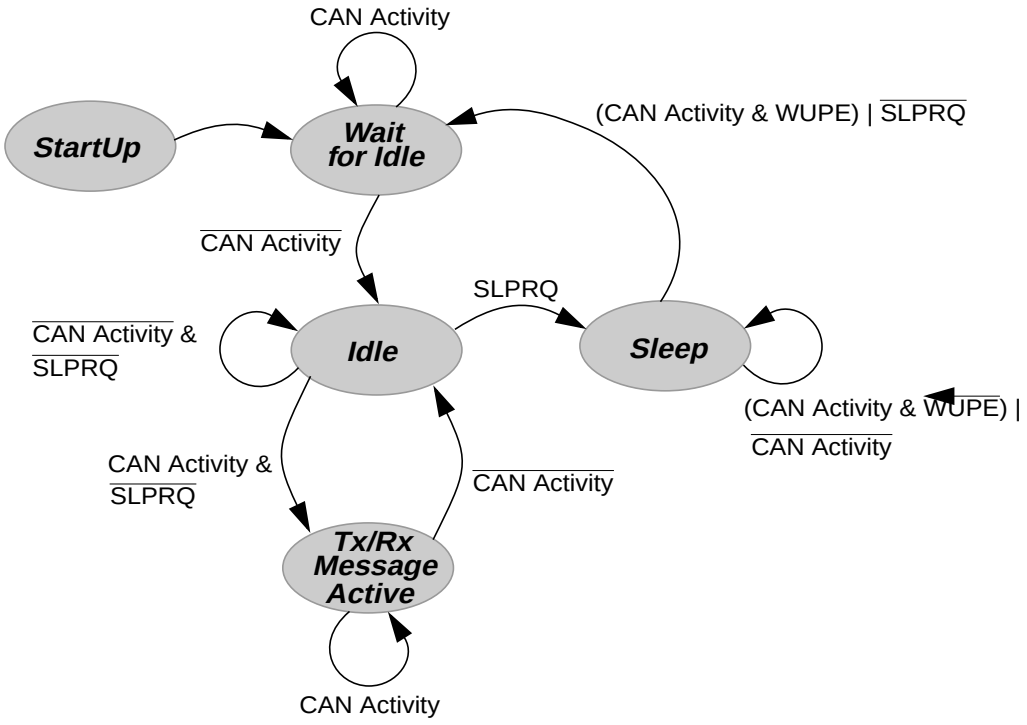


Figure 4-8 Simplified State Transitions for Entering/Leaving Sleep Mode

4.6.5 MSCAN Initialization Mode

In Initialization Mode, any ongoing transmission or reception is immediately aborted and synchronization to the bus is lost potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE: *The user is responsible for ensuring that the MSCAN is not active when Initialization Mode is entered. The recommended procedure is to bring the MSCAN into Sleep Mode (SLPRQ=1 and SLPAK=1) before setting the INITRQ bit in the CANCTL0 register. Otherwise the abort of an ongoing message can cause an error condition and can have an impact on the other bus devices.*

In Initialization Mode, the MSCAN is stopped. However, interface registers can still be accessed. This mode is used to reset the CANCTL0, CANRFLG, CANRIER, CANTFLG, CANTIER, CANTARQ, CANTAAK, CANTBSEL registers to their default values. In addition it enables the configuration of the CANBTR0, CANBTR1 bit timing registers, CANIDAC and the CANIDAR, CANIDMR message filters. **3.3.1.1 MSCAN Control 0 Register (CANCTL0)** for a detailed description of the Initialization Mode.

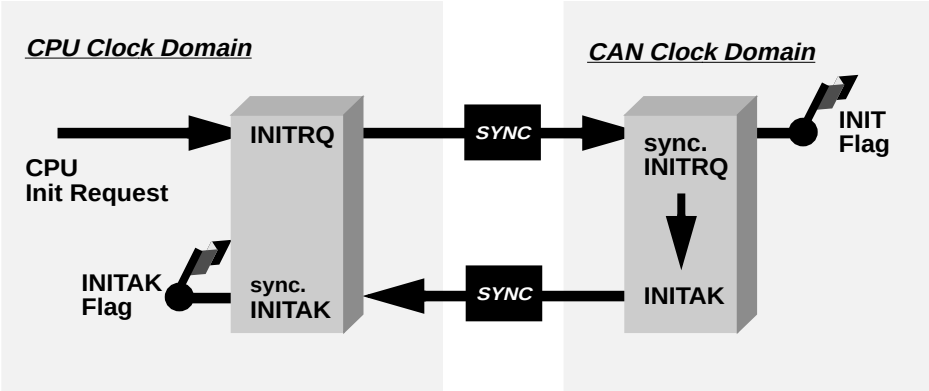


Figure 4-9 Initialization Request/Acknowledge Cycle

Due to independent clock domains within the MSCAN the INITRQ has to be synchronized to all domains by using a special handshake mechanism. This handshake causes additional synchronization delay (**Figure 4-9 Initialization Request/Acknowledge Cycle**).

If there is no message transfer ongoing on the CAN bus, the minimum delay will be two additional bus clocks and three additional CAN clocks. When all parts of the MSCAN are in Initialization Mode the INITAK flag is set. The application software must use INITAK as a handshake indication for the request (INITRQ) to go into Initialization Mode.

NOTE: *The CPU cannot clear the INITRQ bit before Initialization Mode (INITRQ=1 and INITAK=1) is active.*

4.6.6 MSCAN Power Down Mode

The MSCAN is in Power Down Mode (**Table 4-3**) when

- the CPU is in Stop Mode or
- the CPU is in Wait Mode and the CSWAI bit is set.

When entering the Power Down Mode, the MSCAN immediately stops all ongoing transmissions and receptions, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations to the above rule, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE: *The user is responsible for ensuring that the MSCAN is not active when Power Down Mode is entered. The recommended procedure is to bring the MSCAN into Sleep Mode before the STOP or WAI instruction (if CSWAI is set) is executed. Otherwise the abort of an ongoing message can cause an error condition and can have an impact on the other bus devices.*

In Power Down Mode, all clocks are stopped and no registers can be accessed. If the MSCAN was not in Sleep Mode before Power Down Mode became active, the module would perform an internal recovery cycle after powering up. This causes some fixed delay before the module enters Normal Mode again.

4.6.7 Programmable Wake-Up Function

The MSCAN can be programmed to wake up the MSCAN as soon as bus activity is detected (see control bit WUPE in **3.3.1.1 MSCAN Control 0 Register (CANCTL0)**). The sensitivity to existing bus action can be modified by applying a low-pass filter function to the RXCAN input line while in Sleep Mode (see control bit WUPM in **3.3.1.2 MSCAN Control 1 Register (CANCTL1)**).

This feature can be used to protect the MSCAN from wake-up due to short glitches on the CAN bus lines. Such glitches can result e.g. from electromagnetic interference within noisy environments.

4.7 Reset Initialization

The reset state of each individual bit is listed within the Register Description section **3.3 Register Descriptions** which details all the registers and their bit-fields.

4.8 General

This section describes all interrupts originated by the MSCAN. It documents the enable bits and generated flags, if applicable. Each interrupt is listed and described separately. The description explains what causes the interrupt, the registers affected, and how the interrupt request is provided to the core.

Table 4-4 CRG Interrupt Vectors

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
1	Wake-Up Interrupt (WUPIF)	I bit	CANRIER (WUPIE)	2
3	Error Interrupts Interrupt (CSCIF, OVRIF)	I bit	CANRIER (CSCIE, OVRIE)	4
5	Receive Interrupt (RXF)	I bit	CANRIER (RXFIE)	6
7	Transmit Interrupts (TXE2 - TXE0)	I bit	CANTIER (TXEIE2 - TXEIE0)	8

NOTES:
1. Wake-Up interrupt vector address is specific to MCU, refer to MCU specification.
2. Wake-Up interrupt HPRIO value is specific to MCU, refer to MCU specification.
3. Error interrupt vector address is specific to MCU, refer to MCU specification.
4. Error interrupt HPRIO value is specific to MCU, refer to MCU specification.
5. Receive interrupt vector address is specific to MCU, refer to MCU specification.
6. Receive interrupt HPRIO value is specific to MCU, refer to MCU specification.
7. Transmit interrupt vector address is specific to MCU, refer to MCU specification.
8. Transmit interrupt HPRIO value is specific to MCU, refer to MCU specification.

4.9 Description of Interrupt Operation

The MSCAN supports four interrupt vectors, any of which can be individually masked (for details see sections **3.3.1.6 MSCAN Receiver Interrupt Enable Register (CANRIER)** to **3.3.1.8 MSCAN Transmitter Interrupt Enable Register (CANTIER)**):

4.9.1 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. The TXEx flag of the empty message buffer is set.

4.9.2 Receive Interrupt

A message is successfully received and shifted into the foreground buffer (RxFG) of the receiver FIFO. This interrupt is generated immediately after receiving the EOF symbol. The RXF flag is set. If there are multiple messages in the receiver FIFO, the RXF flag is set as soon as the next message is shifted to the foreground buffer.

4.9.3 Wake-Up Interrupt

Activity on the CAN bus occurred during MSCAN internal Sleep Mode and WUPE **3.3.1.1 MSCAN Control 0 Register (CANCTL0)** enabled.

4.9.4 Error Interrupt

An overrun of the receiver FIFO, error, warning or Bus-Off condition occurred. The **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)** indicates one of the following conditions:

- Overrun

  An overrun condition of the receiver FIFO as described in **4.2.3 Receive Structures** occurred.

- CAN Status Change

  The actual value of the Transmit and Receive Error Counters control the bus state of the MSCAN. As soon as the error counters skip into a critical range (Tx/Rx-Warning, Tx/Rx-Error, Bus-Off) the MSCAN flags an error condition. The status change, which caused the error condition, is indicated by the TSTAT and RSTAT flags (see section **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)** and **3.3.1.6 MSCAN Receiver Interrupt Enable Register (CANRIER)**).

4.10 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in either the **3.3.1.5 MSCAN Receiver Flag Register (CANRFLG)** or the **3.3.1.7 MSCAN Transmitter Flag Register (CANTFLG)**. Interrupts are pending as long as one of the corresponding flags is set. The flags in the above registers must be reset within the interrupt handler to handshake the interrupt. The flags are reset by writing a “1” to the corresponding bit position. A flag cannot be cleared if the respective condition still prevails.

NOTE: *It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.*

4.11 Recovery from STOP or WAIT

The MSCAN can recover from STOP or WAIT via the Wake-Up interrupt. This interrupt can only occur if the MSCAN was in Sleep Mode (SLPRQ=1 and SLPAK=1) before entering Power Down Mode, the wake-up option is enabled (WUPE=1) and the Wake-Up interrupt is enabled (WUPIE=1).

Section 5 Initialization/Application Information

5.1 MSCAN initialization

The procedure to initially start up the MSCAN module out of reset is as follows:

1. Assert CANE
2. Write to the configuration registers in Initialization Mode
3. Clear INITRQ to leave Initialization Mode and enter Normal Mode

If the configuration of registers which are writable in Initialization Mode only needs to be changed when the MSCAN module is in Normal Mode:

1. Make sure that the MSCAN transmission queue gets empty and bring the module into Sleep Mode by asserting SLPRQ and awaiting SLPAK
2. Enter Initialization Mode: Assert INITRQ and await INITAK
3. Write to the configuration registers in Initialization Mode
4. Clear INITRQ to leave Initialization Mode and continue in Normal Mode

Index

Block Guide End Sheet

FINAL PAGE OF
70
PAGES

OSC
Block User Guide
V02.03

Original Release Date: 19 July 2002
Revised: 12 February 2003

Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
02.00	19-Jul-02	19-Jul-02		Initial Release for low power Colpitts plus full swing Pierce
02.01	04-Oct-02	04-Oct-02		Correct document number was inserted.
02.02	04-Feb-03	04-Feb-03		'user guide end sheet' text added.
02.03	12-Feb-03	12-Feb-03		-Disclaimer for overtone resonators and crystals was added to Figure 2-1 and Figure 2-2. - Recommendation for evaluation was added.

Table of Contents

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

The OSC module provides two alternative oscillator concepts:

1. a low noise and low power Colpitts oscillator with amplitude limitation control (ALC).
2. a robust full swing Pierce oscillator with the possibility to feed in an external square wave.

1.2 Features

The Colpitts OSC option provides the following features:

- Amplitude Limitation Control (ALC) Loop:
  - low power consumption and low current induced RF emission.
  - sinusoidal waveform with low RF emission.
  - low crystal stress. An external damping resistor is not required.
  - normal and low amplitude mode for further reduction of power and emission.
- an external biasing resistor is not required.

The Pierce OSC option provides the following features:

- wider high frequency operation range.
- no DC voltage applied across the crystal.
- full rail-to-rail (2.5V nominal) swing oscillation with low EM susceptibility.
- fast start up.

Common features:

- clock monitor (CM).
- operation from the VDDPLL 2.5 V (nominal) supply rail.

1.3 Modes of Operation

Two modes of operation exist:

- amplitude limitation controlled Colpitts oscillator mode suitable for power and emission critical applications.
- full swing Pierce oscillator mode that can also be used to feed in an externally generated square wave suitable for high frequency operation and harsh environments.

Section 2 Signal Description

2.1 Overview

This section lists and describes the signals that connect off chip.

2.2 Detailed Signal Descriptions

2.2.1 VDDPLL, VSSPLL

These pins provide the operating voltage (VDDPLL) and ground (VSSPLL) for the OSC circuitry. This allows the supply voltage to the OSC to be independently bypassed.

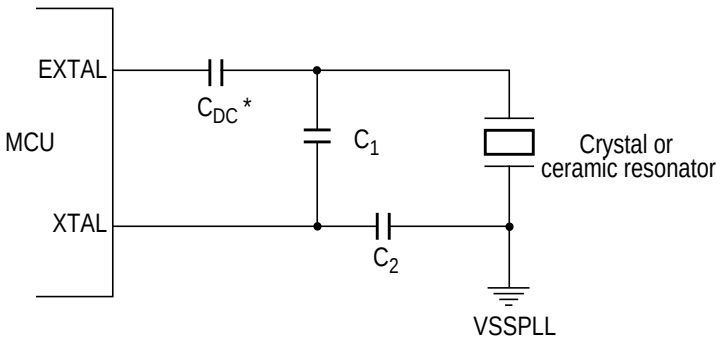
2.2.2 EXTAL, XTAL

These pins provide the interface for either a crystal or a CMOS compatible clock to control the internal clock generator circuitry. EXTAL is the external clock input or the input to the crystal oscillator amplifier. XTAL is the output of the crystal oscillator amplifier. All the MCU internal system clocks are derived from the EXTAL input frequency. In Full Stop Mode (PSTP=0) the EXTAL pin is pulled down by an internal resistor of typical 200k Ohms.

NOTE: *Motorola recommends an evaluation of the application board and chosen resonator or crystal by the resonator or crystal supplier!*

NOTE: *Crystal circuit is changed from standard!*

Colpitts circuit is not suited for overtone resonators and crystals!

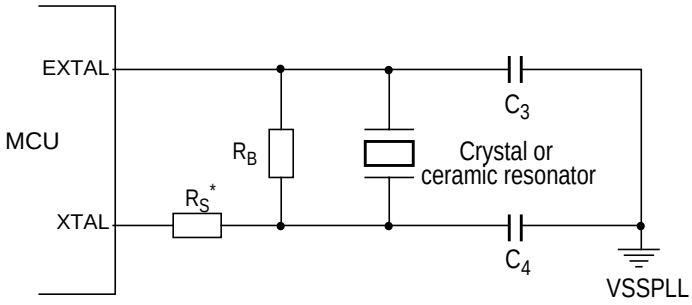


* Due to the nature of a translated ground Colpitts oscillator a DC voltage bias is applied to the crystal

Please contact the crystal manufacturer for crystal DC bias conditions and recommended capacitor value C_{DC}.

Figure 2-1 Colpitts Oscillator Connections (XCLKS=0)

NOTE: *Pierce circuit is not suited for overtone resonators and crystals without a careful component selection!*



* Rs can be zero (shorted) when used with higher frequency crystals.
Refer to manufacturer's data.

Figure 2-2 Pierce Oscillator Connections (XCLKS=1)

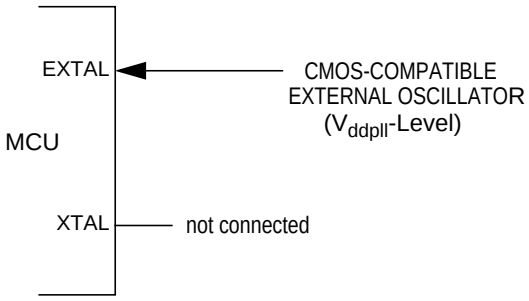


Figure 2-3 External Clock Connections (XCLKS=1)

2.2.3 XCLKS

The XCLKS is an input signal which controls whether a crystal in combination with the internal Colpitts (low power) oscillator is used or whether Pierce oscillator/external clock circuitry is used. The XCLKS signal is sampled during reset with the rising edge of $\overline{\text{RESET}}$. **Table 2-1** lists the state coding of the sampled XCLKS signal. **Refer to device specification for polarity of the XCLKS pin.**

Table 2-1 Clock Selection Based on XCLKS

XCLKS	Description
0	Colpitts Oscillator selected
1	Pierce Oscillator/external clock selected

Section 3 Memory Map and Registers

The CRG contains the registers and associated bits for controlling and monitoring the OSC module.

Section 4 Functional Description

4.1 General

The OSC block has two external pins, EXTAL and XTAL. The oscillator input pin, EXTAL, is intended to be connected to either a crystal or an external clock source. The selection of Colpitts oscillator or Pierce Oscillator/external clock depends on the XCLKS signal which is sampled during reset. The XTAL pin is an output signal that provides crystal circuit feedback.

A buffered EXTAL signal, OSCCLK, becomes the internal reference clock. To improve noise immunity, the oscillator is powered by the VDDPLL and VSSPLL power supply pins.

The Pierce oscillator can be used for higher frequencies compared to the low power Colpitts oscillator.

4.2 Amplitude Limitation Control (ALC)

The Colpitts oscillator is equipped with a feedback system which does not waste current by generating harmonics. Its configuration is “Colpitts oscillator with translated ground”. The transconductor used is driven by a current source under the control of a peak detector which will measure the amplitude of the AC signal appearing on EXTAL node in order to implement an Amplitude Limitation Control (ALC) loop. The ALC loop is in charge of reducing the quiescent current in the transconductor as a result of an increase in the oscillation amplitude. The oscillation amplitude can be limited to two values. The normal amplitude which is intended for non power saving modes and a small amplitude which is intended for low power operation modes. Please refer to the CRG block user guide for the control and assignment of the amplitude value to operation modes.

4.3 Clock Monitor (CM)

The clock monitor circuit is based on an internal resistor-capacitor (RC) time delay so that it can operate without any MCU clocks. If no OSCCLK edges are detected within this RC time delay, the clock monitor indicates a failure which asserts self clock mode or generates a system reset depending on the state of SCME bit. If the clock monitor is disabled or the presence of clocks is detected no failure is indicated.The clock monitor function is enabled/disabled by the CME control bit, described in the CRG block user guide.

Section 5 Interrupts

OSC contains a clock monitor, which can trigger an interrupt or reset. The control bits and status bits for the clock monitor are described in the CRG block user guide.

User Guide End Sheet

FINAL PAGE OF
20
PAGES

PWM_8B8C

Block User Guide

V01.17

Original Release Date: 12 MAR 1998
Revised: 1 Aug 2004

Motorola Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
00.00		3-12-98		First pass release
00.01		3-15-98		- Updates of Section 1 based on Nancy Thomas peer review and internal spec review. - Added initial information into Section 2.
00.02		4-7-98		- Updates of Section 1 and Section 2 per MSIL review. - Updated cover page per latest spec template review.
01.00		4-15-98		- Updated per Rev. 3.0 TSCS Module Spec Template. - Changed the Port and DDR register names to match the latest HCS12 naming convention. - Added the reset state under each counter, period, and duty register in the Register Description sections. - Added Design for Testability sub-section in Section 2 to describe scan implementation. - Updated Module I/O signal names in Section 2 per the latest HCS12 signal naming convention. - Frozen PWM spec sent to Delco.
01.01		4-30-98		- Added Document Number (12MRE31052W) to cover page of spec per QS9000 requirements. - Changed PWMEN register to PWME and also changed the bit names from PWENx to PWMEx to follow the enable naming convention. - Changed PWMCEN register to PWMCAE and also changed the bit names from CENx to CAEx to avoid having the same register name as the PWMC module. - Section 2 Module I/O list changed to have only 1 input buffer enable signal (pwm_ibe_t4) for the entire port. The reset signal name was also changed to vsc_reset_t4 per the latest bus definition document. - Added further clarification on DISCRW test bit in Section 2. If set, duty and period registers are not loaded with the buffer value.
01.02		5-14-98		- Updated per Rev. 3.1, 3.2 and 3.3 TSCS Module Spec Template. - Removed Table 1-1 PWM Register Address Summary. Added the Address Offset along side the registers in Figure 1-2 PWM Register Map. - Added WARNING regarding writing to the test registers in special modes. - Added footnote regarding the counter value in the Period=0 boundary case. - Removed “weasel” words--may and should.

Version Number	Revision Date	Effective Date	Author	Description of Changes
01.03		5-27-98		Summary of changes: - Added clarification on how the counter counts in left and center aligned output modes. - Added further clarification on the Period=0 boundary case. Added that the counter=$00. - Added further clarification on what occurs on writes to the counter--output is changed according to the polarity bit. - Added Caution regarding the first PWM cycle after the channel is enabled can be irregular. - Replaced bit 'RDP' with 'RDPPWM' and bit 'PUPP' with 'PUPPWME' to match port control bit naming conventions. - Added 'iam8bit' signal in Table 2-1. - Added Table 2-2, Engineering Electrical Specs. - Added statement in Section 2 regarding DISCRW bit in PWMTST register. When bit is set, the output is not changed according to the polarity bit. - Added statement in Section 2 regarding DISCRM bit in PWMTST register. When bit is set, the duty and period registers do not get loaded with the buffer value. - Corrected left and center aligned max PWM output frequencies in Table A-2. - Created Table A-3 for the PWM Period/Duty Resolution Characteristics. - Miscellaneous clean up.
01.04		7-1-98		* Changed reset state of PWMPERx and PWMDTYx registers to FF. * In section 2: changed some bus interface signal names: vsc_wait_t2 changed to vsc_wait_t3 vsc_bdmact_t2 changed to vsc_bdmact_t4 vsc_smod_t2 changed to vsc_smod_t4 * In section 2: Added a note that in concatenated, left aligned, DISCRW=1 writing 16 bit (high-byte-data, low-byte-data) to the counter causes the high byte of the counter to start counting from (high-byte-data) and the low byte of the counter to start counting from (low-byte-data + 1).

Version Number	Revision Date	Effective Date	Author	Description of Changes
01.05		1-6-99		Includes spec tagging in conditional text. There are 3 conditional text tags: Tested- Functional Test (Blue), STATEMENT- Statement (Green), Test Outside Submodule (red). Summary of changes: * Section 1: - Added the following sentence in section 1.6.2.4: When the channel is disabled, writing “0” to the period register will cause the counter to reset on the next selected clock. - Added a caution in section 1.7.15.1 about reading from port register after changing pin to input. - In section 1.9 hanged “Reset state: The prescale free running counter begins to increment” to: “All the channels are disabled and all the counters don’t count”. * Section 2: - Section 2.2.3 Design for Test: Scan is not implemented on the PWM module. During scan mode (vsc_scanmod = 1) the module is not selected (vsc_pwmsel_t3 = 0) and the module’s internal clocks stop. - Table 2-1 PWM Module I/O Signals: Removed scane, changed pwm_purst_plug to pwm_puerst_plug, changed vsc_wait_t3 to vsc_wait_t2, changed pwm_outdata_t4[7:0] to pwm_do_t4[7:0], changed pad_indata[7:0] to pwmp_ind[7:0], removed iam8bit, added vsc_en2drv, changed vsc_bdmact_t4 to vsc_bdmact_t2, changed vsc_smod_t4 to vsc_smod_t2. - Added section 2.3.3 and table 2-2 Port Pin Connections. - Section 2.4 table 2-3: Changed Vdd Value to 3 to 3.6v (it was 2.7 to 3.6). Changed System Clock Value to dc to 16MHz (it was 20). - Section 2.4: added Figure 2-7: PWM Timing Diagram. It’s like A.6 in the appendix, but includes more details. * Appendix A: Table A-1: System Clock dc to 16MHz (it was 20). Table A-2: E-clock 16MHz (it was 20). A.6 added PWM Timing Diagram.
01.06	09-02-99			* In section 1, Added section 1.7.15 shutdown register(PWMSDN), changed the sec # for the subsequent sections for the registers. * Added emergency shutdown feature in the feature list (sec 1.3) * Added the PWMSDN in the register map (sec 1.5) * Removed $_24 from resvd. reg list in sec 1.7.17 * Modified sec 1.12, Interrupt Op., to support the intr. for emergency shutdown, to The PWM module has only one interrupt which is generated at the time of emergency shutdown, if the corresponding enable bit (PWMSDN[6]) is set. * Section 2.2.3: Design for Test: The PWM module will be fully scannable as per the project DFT guidelines * removed GPIO note from section 1.2, 1.3, 1.6.2.1, 1.6.2.7, 1.7.1 * renamed PSBCK to PFRZ in PWMCTL, and removed RDPPWM & PUPPWME bits * changed the interface signals for IP bus. * removed electrical spec details.

Version Number	Revision Date	Effective Date	Author	Description of Changes
01.07		10-25-99		updated the specs after feedback from Munich (removed redundant port signals: ibe, offval, obe[6:0])
01.08		11-25-99		added Global Clock signal to the I/O list. the same is used for reg. writes wherever possible.
01.09		12-08-99		Restart(from shutdown) functionality clarified
01.10		01-17-2000		Tagging Done for Barracuda
01.11		01-25-2000		Tagging for wait mode and freeze mode.
01.12		10-09-2000		format converted for SRS 2.0 compliance.
01.13		19-01-2001		updated section 6.1 and 3.3.1.15 for the shutdown feature changes.
01.14		05-04-2001		Made SRS 2.0 Compliant
01.15		07-19-2001		Document names have been added, Names and Variable definitions have been hidden
01.16		03-14-2002		Syntax corrections, document formal updates
01.17		08-01-2004		Added clarification of PWMIF operation in STOP and WAIT mode. Added notes on minimum pulse width of emergency shutdown signal.

Table 0-1 Revision History

Table of Contents

Section 4 Functional Description

Section 5 Resets

Section 6 Interrupts

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

The PWM_8B8C definition is based on the HC12 PWM definitions. This PWM_8B8C contains the basic features from the HC11 with some of the enhancements incorporated on the HC12., that is center aligned output mode and four available clock sources.The PWM_8B8C module has eight channels with independent control of left and center aligned outputs on each channel.

Each of the eight channels has a programmable period and duty cycle as well as a dedicated counter. A flexible clock select scheme allows a total of four different clock sources to be used with the counters. Each of the modulators can create independent continuous waveforms with software-selectable duty rates from 0% to 100%. The PWM outputs can be programmed as left aligned outputs or center aligned outputs.

1.2 Features

The block includes these distinctive features:

- Eight independent PWM channels with programmable period and duty cycle.
- Dedicated counter for each PWM channel.
- Programmable PWM enable/disable for each channel.
- Software selection of PWM duty pulse polarity for each channel.
- Period and duty cycle are double buffered. Change takes effect when the end of the effective period is reached (PWM counter reaches zero) or when the channel is disabled.
- Programmable center or left aligned outputs on individual channels.
- Eight 8-bit channel or four 16-bit channel PWM resolution.
- Four clock sources (A, B, SA and SB) provide for a wide range of frequencies.
- Programmable Clock Select Logic.
- Emergency shutdown.

1.3 Modes of Operation

There is a software programmable option for low power consumption in Wait mode that disables the input clock to the prescaler.

In freeze mode there is a software programmable option to disable the input clock to the prescaler. This is useful for emulation.

1.4 Block Diagram

Figure 1-1 shows the block diagram for the 8-bit 8-channel PWM_8B8C block.

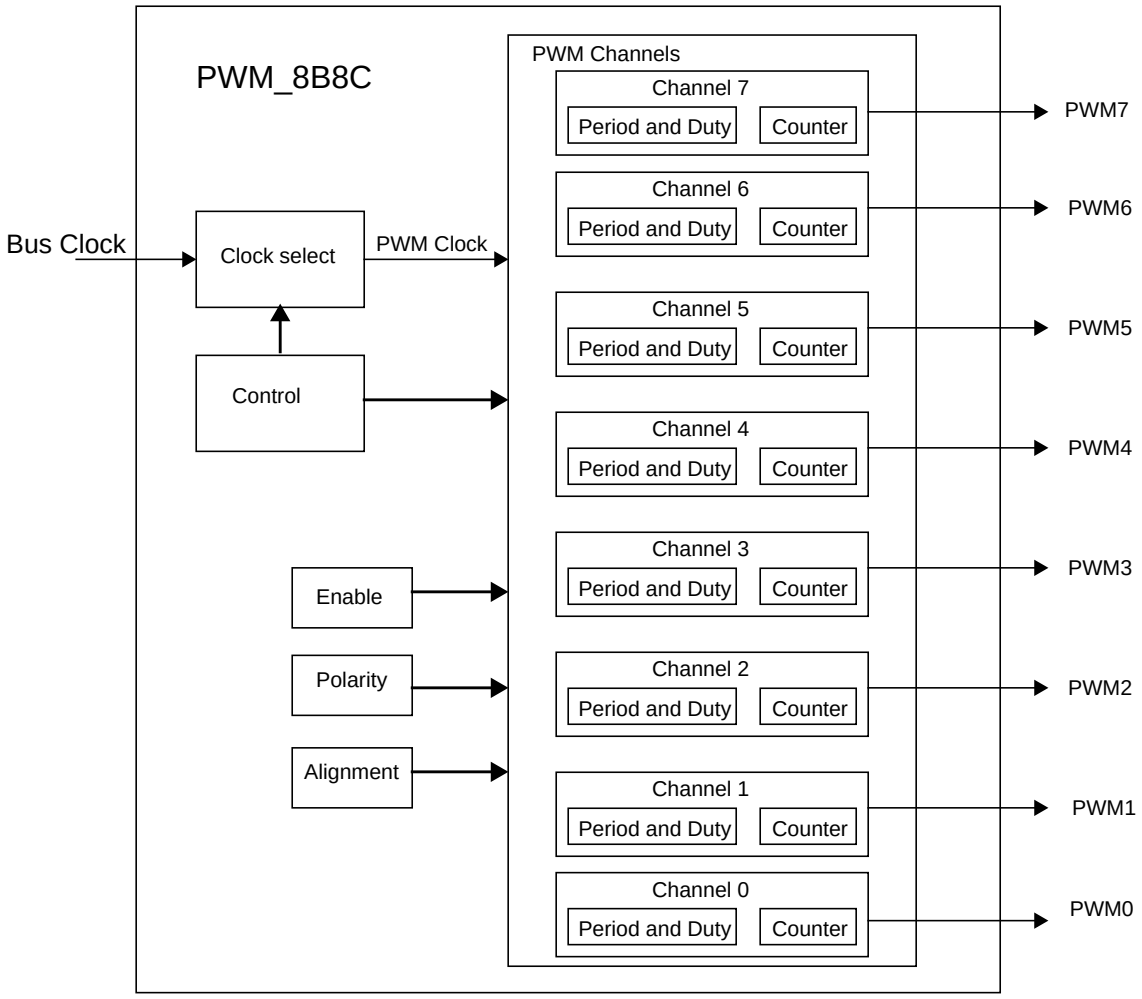


Figure 1-1 PWM_8B8C Block Diagram

Section 2 PWM8b8cSignal Description

2.1 Overview

The PWM_8B8C module has a total of 8 external pins.

2.2 Detailed Signal Descriptions

2.2.1 PWM7 — PWM8b8c Channel 7

This pin serves as waveform output of PWM channel 7 and as an input for the emergency shutdown feature.

2.2.2 PWM6 — PWM8b8c Channel 6

This pin serves as waveform output of PWM channel 6.

2.2.3 PWM5 — PWM8b8c Channel 5

This pin serves as waveform output of PWM channel 5.

2.2.4 PWM4 — PWM8b8c Channel 4

This pin serves as waveform output of PWM channel 4.

2.2.5 PWM3 — PWM8b8c Channel 3

This pin serves as waveform output of PWM channel 3.

2.2.6 PWM2 — PWM8b8c Channel 2

This pin serves as waveform output of PWM channel 2.

2.2.7 PWM1 — PWM8b8c Channel 1

This pin serves as waveform output of PWM channel 1.

2.2.8 PWM0 — PWM8b8c Channel 0

This pin serves as waveform output of PWM channel 0.

Section 3 Memory Map and Register Definition

3.1 Overview

This section describes in detail all the registers and register bits in the PWM_8B8C module.

The special-purpose registers and register bit functions that would not normally made available to device end users, such as factory test control registers and reserved registers are clearly identified by means of shading the appropriate portions of address maps and register diagrams. Notes explaining the reasons for restricting access to the registers and functions are also explained in the individual register descriptions.

3.2 Module Memory Map

This section describes the content of the registers in the PWM_8B8C module. The base address of the PWM_8B8C module is determined at the MCU level when the MCU is defined. The register decode map is fixed and begins at the first address of the module address offset. The figure below shows the registers associated with the PWM and their relative offset from the base address. The register detail description follows the order they appear in the register map.

Reserved bits within a register will always read as 0 and the write will be unimplemented. Unimplemented functions are indicated by shading the bit.

Table 3-1 shows the memory map for the PWM_8B8C module

Table 3-1 PWM_8B8C Memory Map

Address	Use	Access
$_00	PWM Enable Register (PWME)	R/W
$_01	PWM Polarity Register (PWMPOL)	R/W
$_02	PWM Clock Select Register (PWMCLK)	R/W
$_03	PWM Prescale Clock Select Register (PWMPRCLK)	R/W
$_04	PWM Center Align Enable Register (PWMCAE)	R/W
$_05	PWM Control Register (PWMCTL)	R/W
$_06	PWM Test Register (PWMTST)[1]	R/W
$_07	PWM Prescale Counter Register (PWMPRSC)[2]	R/W
$_08	PWM Scale A Register (PWMSCLA)	R/W
$_09	PWM Scale B Register (PWMSCLB)	R/W
$_0A	PWM Scale A Counter Register (PWMSCNTA)[3]	R/W
$_0B	PWM Scale B Counter Register (PWMSCNTB)[4]	R/W
$_0C	PWM Channel 0 Counter Register (PWMCNT0)	R/W
$_0D	PWM Channel 1 Counter Register (PWMCNT1)	R/W
$_0E	PWM Channel 2 Counter Register (PWMCNT2)	R/W
$_0F	PWM Channel 3 Counter Register (PWMCNT3)	R/W
$_10	PWM Channel 4 Counter Register (PWMCNT4)	R/W
$_11	PWM Channel 5 Counter Register (PWMCNT5)	R/W
$_12	PWM Channel 6 Counter Register (PWMCNT6)	R/W

Table 3-1 PWM_8B8C Memory Map

$_13	PWM Channel 7 Counter Register (PWMCNT7)	R/W
$_14	PWM Channel 0 Period Register (PWMPER0)	R/W
$_15	PWM Channel 1 Period Register (PWMPER1)	R/W
$_16	PWM Channel 2 Period Register (PWMPER2)	R/W
$_17	PWM Channel 3 Period Register (PWMPER3)	R/W
$_18	PWM Channel 4 Period Register (PWMPER4)	R/W
$_19	PWM Channel 5 Period Register (PWMPER5)	R/W
$_1A	PWM Channel 6 Period Register (PWMPER6)	R/W
$_1B	PWM Channel 7 Period Register (PWMPER7)	R/W
$_1C	PWM Channel 0 Duty Register (PWMDTY0)	R/W
$_1D	PWM Channel 1 Duty Register (PWMDTY1)	R/W
$_1E	PWM Channel 2 Duty Register (PWMDTY2)	R/W
$_1F	PWM Channel 3 Duty Register (PWMDTY3)	R/W
$_20	PWM Channel 4 Duty Register (PWMDTY4)	R/W
$_21	PWM Channel 5 Duty Register (PWMDTY5)	R/W
$_22	PWM Channel 6 Duty Register (PWMDTY6)	R/W
$_23	PWM Channel 7 Duty Register (PWMDTY7)	R/W
$_24	PWM Shutdown Register (PWMSDN)	R/W
$_25	Reserved	R
$_26	Reserved	R
$_27	Reserved	R

NOTES:
1. PWMTST is intended for factory test purposes only.
2. PWMPRSC is intended for factory test purposes only.
3. PWMSCNTA is intended for factory test purposes only.
4. PWMSCNTB is intended for factory test purposes only.

NOTE: *Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.*

3.3 Register Descriptions

This section describes in detail all the registers and register bits in the PWM_8B8C module.

3.3.1 PWM Enable Register (PWME)

Each PWM channel has an enable bit (PWMEx) to start its waveform output. When any of the PWMEx bits are set (PWMEx=1), the associated PWM output is enabled immediately. However, the actual PWM waveform is not available on the associated PWM output until its clock source begins its next cycle due to the synchronization of PWMEx and the clock source.

NOTE: *The first PWM cycle after enabling the channel can be irregular.*

An exception to this is when channels are concatenated. Once concatenated mode is enabled (CONxx bits set in PWMCTL register) then enabling/disabling the corresponding 16-bit PWM channel is controlled by

the low order PWMEx bit.In this case, the high order bytes PWMEx bits have no effect and their corresponding PWM output lines are disabled.

While in run mode, if all eight PWM channels are disabled (PWME7-0=0), the prescaler counter shuts off for power savings.

$_00

	7	6	5	4	3	2	1	0
R W	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-1 PWM Enable Register (PWME)

Read: anytime

Write: anytime

PWME7 — Pulse Width Channel 7 Enable

1 = Pulse Width channel 7 is enabled. The pulse modulated signal becomes available at PWM output bit7 when its clock source begins its next cycle.

0 = Pulse Width channel 7 is disabled.

PWME6 — Pulse Width Channel 6 Enable

1 = Pulse Width channel 6 is enabled. The pulse modulated signal becomes available at PWM output bit6 when its clock source begins its next cycle. If CON67=1, then bit has no effect and PWM output line6 is disabled.

0 = Pulse Width channel 6 is disabled.

PWME5 — Pulse Width Channel 5 Enable

1 = Pulse Width channel 5 is enabled. The pulse modulated signal becomes available at PWM output bit 5 when its clock source begins its next cycle.

0 = Pulse Width channel 5 is disabled.

PWME4 — Pulse Width Channel 4 Enable

1 = Pulse Width channel 4 is enabled. The pulse modulated signal becomes available at PWM, output bit 4 when its clock source begins its next cycle. If CON45=1, then bit has no effect and PWM output bit4 is disabled.

0 = Pulse Width channel 4 is disabled.

PWME3 — Pulse Width Channel 3 Enable

1 = Pulse Width channel 3 is enabled. The pulse modulated signal becomes available at PWM, output bit 3 when its clock source begins its next cycle.

0 = Pulse Width channel 3 is disabled.

PWME2 — Pulse Width Channel 2 Enable

1 = Pulse Width channel 2 is enabled. The pulse modulated signal becomes available at PWM, output bit 2 when its clock source begins its next cycle. If CON23=1, then bit has no effect and PWM output bit2 is disabled.
0 = Pulse Width channel 2 is disabled.

PWME1 — Pulse Width Channel 1 Enable
1 = Pulse Width channel 1 is enabled. The pulse modulated signal becomes available at PWM, output bit 1 when its clock source begins its next cycle.
0 = Pulse Width channel 1 is disabled.

PWME0 — Pulse Width Channel 0 Enable
1 = Pulse Width channel 0 is enabled. The pulse modulated signal becomes available at PWM, output bit 0 when its clock source begins its next cycle. If CON01=1, then bit has no effect and PWM output line0 is disabled.
0 = Pulse Width channel 0 is disabled.

3.3.2 PWM Polarity Register (PWMPOL)

The starting polarity of each PWM channel waveform is determined by the associated PPOLx bit in the PWMPOL register. If the polarity bit is one, the PWM channel output is high at the beginning of the cycle and then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

$_01

	7	6	5	4	3	2	1	0
R W	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-2 PWM Polarity Register (PWMPOL)

Read: anytime

Write: anytime

NOTE: *PPOLx register bits can be written anytime. If the polarity is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition*

PPOL7 — Pulse Width Channel 7 Polarity
1 = PWM channel 7 output is high at the beginning of the period, then goes low when the duty count is reached.
0 = PWM channel 7 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL6 — Pulse Width Channel 6 Polarity

1 = PWM channel 6 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 6 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL5 — Pulse Width Channel 5 Polarity

1 = PWM channel 5 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 5 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL4 — Pulse Width Channel 4 Polarity

1 = PWM channel 4 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 4 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL3 — Pulse Width Channel 3 Polarity

1 = PWM channel 3 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 3 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL2 — Pulse Width Channel 2 Polarity

1 = PWM channel 2 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 2 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL1 — Pulse Width Channel 1 Polarity

1 = PWM channel 1 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 1 output is low at the beginning of the period, then goes high when the duty count is reached.

PPOL0 — Pulse Width Channel 0 Polarity

1 = PWM channel 0 output is high at the beginning of the period, then goes low when the duty count is reached.

0 = PWM channel 0 output is low at the beginning of the period, then goes high when the duty count is reached

3.3.3 PWM Clock Select Register (PWMCLK)

Each PWM channel has a choice of two clocks to use as the clock source for that channel as described below.

$_02

	7	6	5	4	3	2	1	0
R W	PCLK7	PCLKL6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-3 PWM Clock Select Register (PWMCLK)

Read: anytime

Write: anytime

NOTE: *Register bits PCLK0 to PCLK7 can be written anytime. If a clock select is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.*

PCLK7 — Pulse Width Channel 7 Clock Select
1 = Clock SB is the clock source for PWM channel 7.
0 = Clock B is the clock source for PWM channel 7.

PCLK6 — Pulse Width Channel 6 Clock Select
1 = Clock SB is the clock source for PWM channel 6.
0 = Clock B is the clock source for PWM channel 6.

PCLK5 — Pulse Width Channel 5 Clock Select
1 = Clock SA is the clock source for PWM channel 5.
0 = Clock A is the clock source for PWM channel 5.

PCLK4 — Pulse Width Channel 4 Clock Select
1 = Clock SA is the clock source for PWM channel 4.
0 = Clock A is the clock source for PWM channel 4.

PCLK3 — Pulse Width Channel 3 Clock Select
1 = Clock SB is the clock source for PWM channel 3.
0 = Clock B is the clock source for PWM channel 3.

PCLK2 — Pulse Width Channel 2 Clock Select
1 = Clock SB is the clock source for PWM channel 2.
0 = Clock B is the clock source for PWM channel 2.

PCLK1 — Pulse Width Channel 1 Clock Select
1 = Clock SA is the clock source for PWM channel 1.
0 = Clock A is the clock source for PWM channel 1.

PCLK0 — Pulse Width Channel 0 Clock Select
1 = Clock SA is the clock source for PWM channel 0.

0 = Clock A is the clock source for PWM channel 0.

3.3.4 PWM Prescale Clock Select Register (PWMPRCLK)

This register selects the prescale clock source for clocks A and B independently.

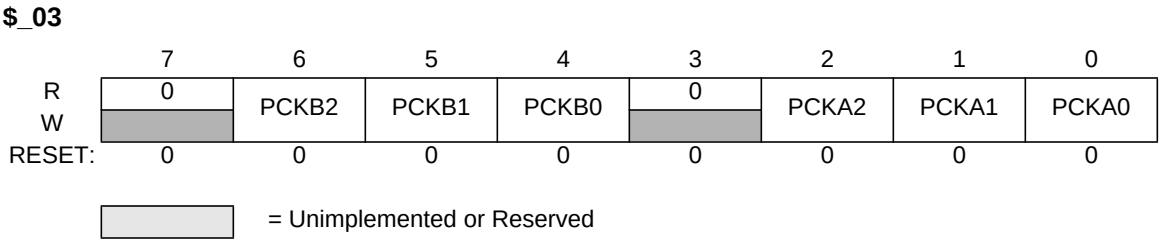


Figure 3-4 PWM Prescale Clock Select Register (PWMPRCLK)

Read: anytime

Write: anytime

NOTE: *PCKB2-0 and PCKA2-0 register bits can be written anytime. If the clock pre-scale is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.*

PCKB2 - PCKB0 — Prescaler Select for Clock B

Clock B is one of two clock sources which can be used for channels 2, 3, 6 or 7. These three bits determine the rate of clock B, as shown in the following table.

Table 3-2 Clock B Prescaler Selects

PCKB2	PCKB1	PCKB0	Value of Clock B
0	0	0	bus clock
0	0	1	bus clock / 2
0	1	0	bus clock / 4
0	1	1	bus clock / 8
1	0	0	bus clock / 16
1	0	1	bus clock / 32
1	1	0	bus clock / 64
1	1	1	bus clock / 128

PCKA2 - PCKA0 — Prescaler Select for Clock A

Clock A is one of two clock sources which can be used for channels 0, 1, 4 or 5. These three bits determine the rate of clock A, as shown in the following table.

Table 3-3 Clock A Prescaler Selects

PCKA2	PCKA1	PCKA0	Value of Clock A
0	0	0	bus clock
0	0	1	bus clock / 2
0	1	0	bus clock / 4
0	1	1	bus clock / 8
1	0	0	bus clock / 16
1	0	1	bus clock / 32
1	1	0	bus clock / 64
1	1	1	bus clock / 128

3.3.5 PWM Center Align Enable Register (PWMCAE)

The PWMCAE register contains eight control bits for the selection of center aligned outputs or left aligned outputs for each PWM channel. If the CAEx bit is set to a one, the corresponding PWM output will be center aligned. If the CAEx bit is cleared, the corresponding PWM output will be left aligned. Reference **4.2.5 Left Aligned Outputs** and **4.2.6 Center Aligned Outputs** for a more detailed description of the PWM output modes.

$_04

	7	6	5	4	3	2	1	0
R W	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-5 PWM Center Align Enable Register (PWMCAE)

Read: anytime

Write: anytime

NOTE: *Write these bits only when the corresponding channel is disabled.*

CAE7 — Center Aligned Output Mode on channel 7

1 = Channel 7 operates in Center Aligned Output Mode.

0 = Channel 7 operates in Left Aligned Output Mode.

CAE6 — Center Aligned Output Mode on channel 6

1 = Channel 6 operates in Center Aligned Output Mode.
0 = Channel 6 operates in Left Aligned Output Mode.

CAE5 — Center Aligned Output Mode on channel 5
1 = Channel 5 operates in Center Aligned Output Mode.
0 = Channel 5 operates in Left Aligned Output Mode.

CAE4 — Center Aligned Output Mode on channel 4
1 = Channel 4 operates in Center Aligned Output Mode.
0 = Channel 4 operates in Left Aligned Output Mode.

CAE3 — Center Aligned Output Mode on channel 3
1 = Channel 3 operates in Center Aligned Output Mode.
0 = Channel 3 operates in Left Aligned Output Mode.

CAE2 — Center Aligned Output Mode on channel 2
1 = Channel 2 operates in Center Aligned Output Mode.
0 = Channel 2 operates in Left Aligned Output Mode.

CAE1 — Center Aligned Output Mode on channel 1
1 = Channel 1 operates in Center Aligned Output Mode.
0 = Channel 1 operates in Left Aligned Output Mode.

CAE0 — Center Aligned Output Mode on channel 0
1 = Channel 0 operates in Center Aligned Output Mode.
0 = Channel 0 operates in Left Aligned Output Mode.

3.3.6 PWM Control Register (PWMCTL)

The PWMCTL register provides for various control of the PWM module.

$_05

	7	6	5	4	3	2	1	0
R W	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-6 PWM Control Register (PWMCTL)

Read: anytime

Write: anytime

There are three control bits for concatenation, each of which is used to concatenate a pair of PWM channels into one 16-bit channel. When channels 6 and 7are concatenated, channel 6 registers become the high order bytes of the double byte channel. When channels 4 and 5 are concatenated, channel 4 registers

become the high order bytes of the double byte channel. When channels 2 and 3 are concatenated, channel 2 registers become the high order bytes of the double byte channel. When channels 0 and 1 are concatenated, channel 0 registers become the high order bytes of the double byte channel.

Reference **4.2.7 PWM 16-Bit Functions** for a more detailed description of the concatenation PWM Function.

NOTE: *Change these bits only when both corresponding channels are disabled.*

CON67 — Concatenate channels 6 and 7

1 = Channels 6 and 7 are concatenated to create one 16-bit PWM channel. Channel 6 becomes the high order byte and channel 7 becomes the low order byte. Channel 7 output pin is used as the output for this 16-bit PWM (bit 7 of port PWMP). Channel 7 clock select control-bit determines the clock source, channel 7 polarity bit determines the polarity, channel 7 enable bit enables the output and channel 7 center aligned enable bit determines the output mode.

0 = Channels 6 and 7 are separate 8-bit PWMs.

CON45 — Concatenate channels 4 and 5

1 = Channels 4 and 5 are concatenated to create one 16-bit PWM channel. Channel 4 becomes the high order byte and channel 5 becomes the low order byte. Channel 5 output pin is used as the output for this 16-bit PWM (bit 5 of port PWMP). Channel 5 clock select control-bit determines the clock source, channel 5 polarity bit determines the polarity, channel 5 enable bit enables the output and channel 5 center aligned enable bit determines the output mode.

0 = Channels 4 and 5 are separate 8-bit PWMs.

CON23 — Concatenate channels 2 and 3

1 = Channels 2 and 3 are concatenated to create one 16-bit PWM channel. Channel 2 becomes the high order byte and channel 3 becomes the low order byte. Channel 3 output pin is used as the output for this 16-bit PWM (bit 3 of port PWMP). Channel 3 clock select control-bit determines the clock source, channel 3 polarity bit determines the polarity, channel 3 enable bit enables the output and channel 3 center aligned enable bit determines the output mode.

0 = Channels 2 and 3 are separate 8-bit PWMs.

CON01 — Concatenate channels 0 and 1

1 = Channels 0 and 1 are concatenated to create one 16-bit PWM channel. Channel 0 becomes the high order byte and channel 1 becomes the low order byte. Channel 1 output pin is used as the output for this 16-bit PWM (bit 1 of port PWMP). Channel 1 clock select control-bit determines the clock source, channel 1 polarity bit determines the polarity, channel 1 enable bit enables the output and channel 1 center aligned enable bit determines the output mode.

0 = Channels 0 and 1 are separate 8-bit PWMs.

PSWAI — PWM Stops in Wait Mode

Enabling this bit allows for lower power consumption in Wait Mode by disabling the input clock to the prescaler.

1 = Stop the input clock to the prescaler whenever the MCU is in Wait Mode.

0 = Allow the clock to the prescaler to continue while in wait mode.

PFRZ — PWM Counters Stop in Freeze Mode

In Freeze Mode, there is an option to disable the input clock to the prescaler by setting the PFRZ bit in the PWMCTL register. If this bit is set, whenever the MCU is in freeze mode the input clock to the prescaler is disabled. This feature is useful during emulation as it allows the PWM function to be suspended. In this way, the counters of the PWM can be stopped while in freeze mode so that once normal program flow is continued, the counters are re-enabled to simulate real-time operations. Since the registers can still be accessed in this mode, to re-enable the prescaler clock, either disable the PFRZ bit or exit freeze mode.

1 = Disable PWM input clock to the prescaler whenever the part is in freeze mode. This is useful for emulation.

0 = Allow PWM to continue while in freeze mode.

3.3.7 Reserved Register (PWMTST)

This register is reserved for factory testing of the PWM module and is not available in normal modes.

$_06

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-7 Reserved Register (PWMTST)

Read: always read $00 in normal modes

Write: unimplemented in normal modes

NOTE: *Writing to this register when in special modes can alter the PWM functionality.*

3.3.8 Reserved Register (PWMPRSC)

This register is reserved for factory testing of the PWM module and is not available in normal modes.

$_07

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-8 Reserved Register (PWMPRSC)

Read: always read $00 in normal modes

Write: unimplemented in normal modes

NOTE: *Writing to this register when in special modes can alter the PWM functionality.*

3.3.9 PWM Scale A Register (PWMSCLA)

PWMSCLA is the programmable scale value used in scaling clock A to generate clock SA. Clock SA is generated by taking clock A, dividing it by the value in the PWMSCLA register and dividing that by two.

Clock SA = Clock A / (2 * PWMSCLA)

NOTE: *When PWMSCLA = $00, PWMSCLA value is considered a full scale value of 256. Clock A is thus divided by 512.*

Any value written to this register will cause the scale counter to load the new scale value (PWMSCLA).

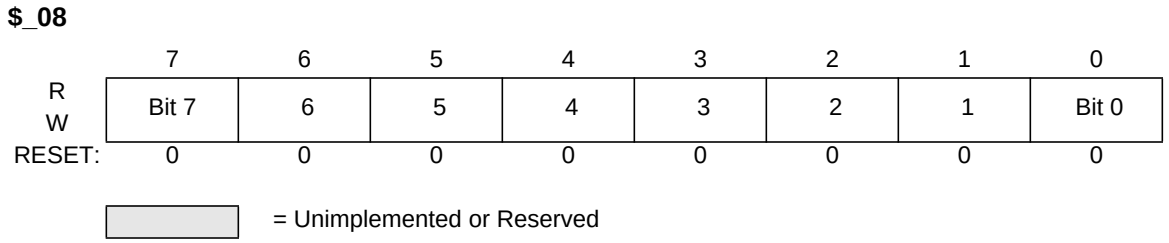


Figure 3-9 PWM Scale A Register (PWMSCLA)

Read: anytime

Write: anytime (causes the scale counter to load the PWMSCLA value)

3.3.10 PWM Scale B Register (PWMSCLB)

PWMSCLB is the programmable scale value used in scaling clock B to generate clock SB. Clock SB is generated by taking clock B, dividing it by the value in the PWMSCLB register and dividing that by two.

Clock SB = Clock B / (2 * PWMSCLB)

NOTE: *When PWMSCLB = $00, PWMSCLB value is considered a full scale value of 256. Clock B is thus divided by 512.*

Any value written to this register will cause the scale counter to load the new scale value (PWMSCLB).

$_09

	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-10 PWM Scale B Register (PWMSCLB)

Read: anytime

Write: anytime (causes the scale counter to load the PWMSCLB value).

3.3.11 Reserved Registers (PWMSCNTx)

The registers PWMSCNTA and PWMSCNTB are reserved for factory testing of the PWM module and are not available in normal modes.

$_0A, $_0B

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-11 Reserved Registers (PWMSCNTx)

Read: always read $00 in normal modes

Write: unimplemented in normal modes

NOTE: *Writing to these registers when in special modes can alter the PWM functionality.*

3.3.12 PWM Channel Counter Registers (PWMCNTx)

Each channel has a dedicated 8-bit up/down counter which runs at the rate of the selected clock source. The counter can be read at any time without affecting the count or the operation of the PWM channel. In left aligned output mode, the counter counts from 0 to the value in the period register - 1. In center aligned output mode, the counter counts from 0 up to the value in the period register and then back down to 0.

Any value written to the counter causes the counter to reset to $00, the counter direction to be set to up, the immediate load of both duty and period registers with values from the buffers, and the output to change according to the polarity bit. The counter is also cleared at the end of the effective period (see Sections **4.2.5 Left Aligned Outputs** and **4.2.6 Center Aligned Outputs** for more details). When the channel is disabled (PWMEx=0), the PWMCNTx register does not count. When a channel becomes enabled

(PWMEx=1), the associated PWM counter starts at the count in the PWMCNTx register. For more detailed information on the operation of the counters, reference **4.2.4 PWM Timer Counters**.

In concatenated mode, writes to the 16-bit counter by using a 16-bit access or writes to either the low or high order byte of the counter will reset the 16-bit counter. Reads of the 16-bit counter must be made by 16-bit access to maintain data coherency.

NOTE:*Writing to the counter while the channel is enabled can cause an irregular PWM cycle to occur.*

$_0C = PWMCNT0
$_0D = PWMCNT1
$_0E = PWMCNT2
$_0F = PWMCNT3
$_10 = PWMCNT4
$_11 = PWMCNT5
$_12 = PWMCNT6
$_13 = PWMCNT7

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W	0	0	0	0	0	0	0	0
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 3-12 PWM Channel Counter Registers (PWMCNTx)

Read: anytime

Write: anytime (any value written causes PWM counter to be reset to $00).

3.3.13 PWM Channel Period Registers (PWMPERx)

There is a dedicated period register for each channel. The value in this register determines the period of the associated PWM channel.

The period registers for each channel are double buffered so that if they change while the channel is enabled, the change will NOT take effect until one of the following occurs:

- The effective period ends
- The counter is written (counter resets to $00)
- The channel is disabled

In this way, the output of the PWM will always be either the old waveform or the new waveform, not some variation in between. If the channel is not enabled, then writes to the period register will go directly to the latches as well as the buffer.

NOTE: *Reads of this register return the most recent value written. Reads do not necessarily return the value of the currently active period due to the double buffering scheme.*

Reference **4.2.3 PWM Period and Duty** for more information.

To calculate the output period, take the selected clock source period for the channel of interest (A, B, SA, or SB) and multiply it by the value in the period register for that channel:

- Left Aligned Output (CAEx=0)
- PWMx Period = Channel Clock Period * PWMPERx Center Aligned Output (CAEx=1)

  PWMx Period = Channel Clock Period * (2 * PWMPERx)

  For Boundary Case programming values, please refer to Section **4.2.8 PWM Boundary Cases**

$_14 = PWMPER0
$_15 = PWMPER1
$_16 = PWMPER2
$_17 = PWMPER3
$_18 = PWMPER4
$_19 = PWMPER5
$_1A = PWMPER6
$_1B = PWMPER7

	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
RESET:	1	1	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 3-13 PWM Channel Period Registers (PWMPERx)

Read: anytime

Write: anytime

3.3.14 PWM Channel Duty Registers (PWMDTYx)

There is a dedicated duty register for each channel. The value in this register determines the duty of the associated PWM channel. The duty value is compared to the counter and if it is equal to the counter value a match occurs and the output changes state.

The duty registers for each channel are double buffered so that if they change while the channel is enabled, the change will NOT take effect until one of the following occurs:

- The effective period ends
- The counter is written (counter resets to $00)
- The channel is disabled

In this way, the output of the PWM will always be either the old duty waveform or the new duty waveform, not some variation in between. If the channel is not enabled, then writes to the duty register will go directly to the latches as well as the buffer.

NOTE: *Reads of this register return the most recent value written. Reads do not necessarily return the value of the currently active duty due to the double buffering scheme.*

Reference **4.2.3 PWM Period and Duty** for more information.

NOTE: *Depending on the polarity bit, the duty registers will contain the count of either the high time or the low time. If the polarity bit is one, the output starts high and then goes low when the duty count is reached, so the duty registers contain a count of the high time. If the polarity bit is zero, the output starts low and then goes high when the duty count is reached, so the duty registers contain a count of the low time.*

To calculate the output duty cycle (high time as a% of period) for a particular channel:

- Polarity = 0 (PPOLx=0)

  Duty Cycle = [(PWMPERx-PWMDTYx)/PWMPERx] * 100%
- Polarity = 1 (PPOLx=1)

  Duty Cycle = [PWMDTYx / PWMPERx] * 100%
- For Boundary Case programming values, please refer to Section **4.2.8 PWM Boundary Cases**.

$_1C = PWMDTY0
$_1D = PWMDTY1
$_1E = PWMDTY2
$_1F = PWMDTY3
$_20 = PWMDTY4
$_21 = PWMDTY5
$_22 = PWMDTY6
$_23 = PWMDTY7

	7	6	5	4	3	2	1	0
R W	Bit 7	6	5	4	3	2	1	Bit 0
RESET:	1	1	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 3-14 PWM Channel Duty Registers (PWMDTYx)

Read: anytime

Write: anytime

3.3.15 PWM Shutdown Register (PWMSDN)

The PWMSDN register provides for the shutdown functionality of the PWM module in the emergency cases. For proper operation channel 7 must be driven to the active level for a minimum of two bus clocks.

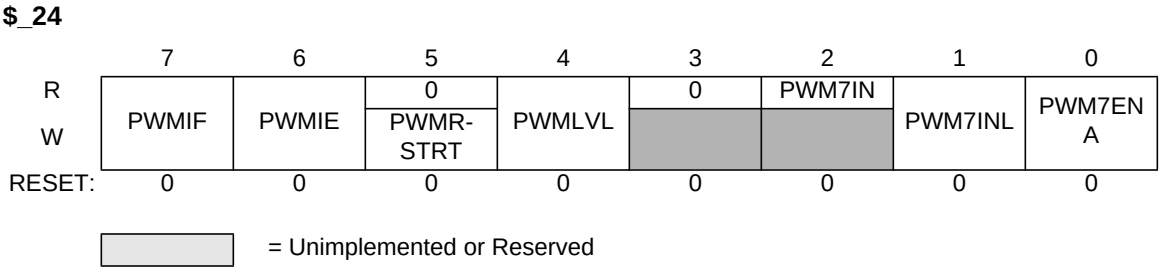


Figure 3-15 PWM Shutdown Register (PWMSDN)

Read: anytime

Write: anytime

PWM7ENA — PWM emergency shutdown Enable

If this bit is logic 1 the pin associated with channel 7 is forced to input and the emergency shutdown feature is enabled.All the other bits in this register are meaningful only if PWM7ENA = 1.

1 = PWM emergency feature is enabled.
0 = PWM emergency feature disabled.

PWM7INL — PWM shutdown active input level for channel 7.

If the emergency shutdown feature is enabled (PWM7ENA = 1), this bit determines the active level of the PWM7channel.

1 = Active level is high
0 = Active level is low

PWM7IN — PWM channel 7 input status.

This reflects the current status of the PWM7 pin.

PWMLVL — PWM shutdown output Level.

If active level as defined by the PWM7IN input, gets asserted all enabled PWM channels are immediately driven to the level defined by PWMLVL.

1 = PWM outputs are forced to 1.
0 = PWM outputs are forced to 0

PWMRSTRT — PWM Restart.

The PWM can only be restarted if the PWM channel input 7 is de-asserted. After writing a logic 1 to the PWMRSTRT bit (trigger event) the PWM channels start running after the corresponding counter passes next “counter == 0” phase.

Also if the PWM7ENA bit is reset to 0, the PWM do not start before the counter passes $00. The bit is always read as “0”.

PWMIE — PWM Interrupt Enable

If interrupt is enabled an interrupt to the CPU is asserted.

1 = PWM interrupt is enabled.

0 = PWM interrupt is disabled.

PWMIF — PWM Interrupt Flag

Any change from passive to asserted (active) state or from active to passive state will be flagged by setting the PWMIF flag = 1. The flag is cleared by writing a logic 1 to it. Writing a 0 has no effect.

1 = change on PWM7IN input

0 = No change on PWM7IN input.

Section 4 Functional Description

4.1 PWM Clock Select

There are four available clocks called clock A, clock B, clock SA (Scaled A), and clock SB (Scaled B). These four clocks are based on the bus clock.

Clock A and B can be software selected to be 1, 1/2, 1/4, 1/8,..., 1/64, 1/128 times the bus clock. Clock SA uses clock A as an input and divides it further with a reloadable counter. Similarly, Clock SB uses clock B as an input and divides it further with a reloadable counter. The rates available for clock SA are software selectable to be clock A divided by 2, 4, 6, 8,..., or 512 in increments of divide by 2. Similar rates are available for clock SB. Each PWM channel has the capability of selecting one of two clocks, either the pre-scaled clock (clock A or B) or the scaled clock (clock SA or SB).

The block diagram in **Figure 4-1** shows the four different clocks and how the scaled clocks are created.

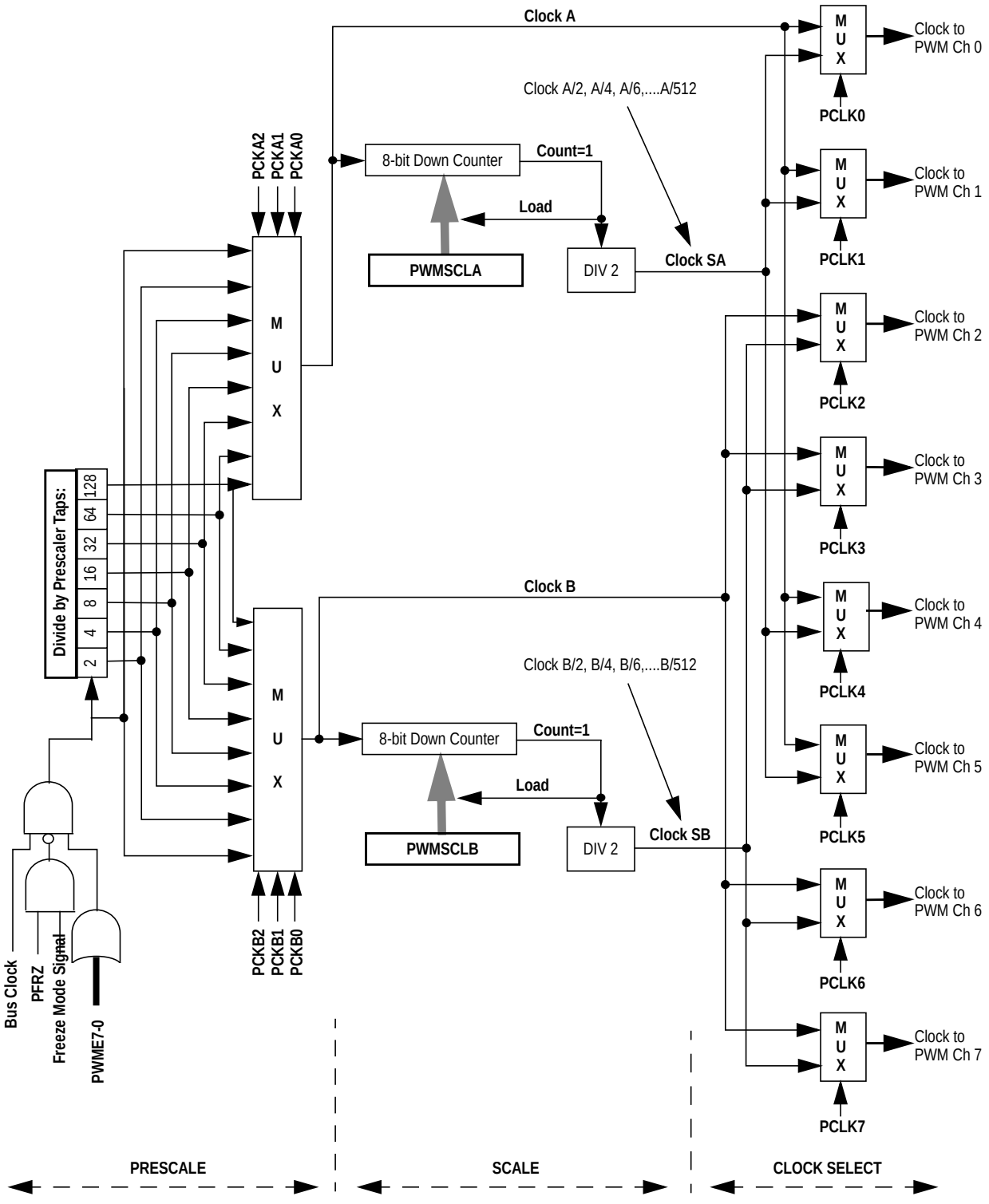


Figure 4-1 PWM Clock Select Block Diagram

4.1.1 Prescale

The input clock to the PWM prescaler is the bus clock. It can be disabled whenever the part is in freeze mode by setting the PFRZ bit in the PWMCTL register. If this bit is set, whenever the MCU is in freeze mode (Freeze Mode Signal active) the input clock to the prescaler is disabled. This is useful for emulation in order to freeze the PWM. The input clock can also be disabled when all eight PWM channels are disabled (PWME7-0=0). This is useful for reducing power by disabling the prescale counter.

Clock A and clock B are scaled values of the input clock. The value is software selectable for both clock A and clock B and has options of 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, or 1/128 times the bus clock. The value selected for clock A is determined by the PCKA2, PCKA1, PCKA0 bits in the PWMPRCLK register. The value selected for clock B is determined by the PCKB2, PCKB1, PCKB0 bits also in the PWMPRCLK register.

4.1.2 Clock Scale

The scaled A clock uses clock A as an input and divides it further with a user programmable value and then divides this by 2. The scaled B clock uses clock B as an input and divides it further with a user programmable value and then divides this by 2. The rates available for clock SA are software selectable to be clock A divided by 2, 4, 6, 8,..., or 512 in increments of divide by 2. Similar rates are available for clock SB.

Clock A is used as an input to an 8-bit down counter. This down counter loads a user programmable scale value from the scale register (PWMSCLA). When the down counter reaches one, two things happen; a pulse is output and the 8-bit counter is re-loaded. The output signal from this circuit is further divided by two. This gives a greater range with only a slight reduction in granularity. Clock SA equals Clock A divided by two times the value in the PWMSCLA register.

NOTE: *Clock SA = Clock A / (2 * PWMSCLA)*

When PWMSCLA = $00, PWMSCLA value is considered a full scale value of 256. Clock A is thus divided by 512.

Similarly, Clock B is used as an input to an 8-bit down counter followed by a divide by two producing clock SB. Thus, clock SB equals Clock B divided by two times the value in the PWMSCLB register.

NOTE: *Clock SB = Clock B / (2 * PWMSCLB)*

When PWMSCLB = $00, PWMSCLB value is considered a full scale value of 256. Clock B is thus divided by 512.

As an example, consider the case in which the user writes $FF into the PWMSCLA register. Clock A for this case will be E divided by 4. A pulse will occur at a rate of once every 255x4 E cycles. Passing this through the divide by two circuit produces a clock signal at an E divided by 2040 rate. Similarly, a value of $01 in the PWMSCLA register when clock A is E divided by 4 will produce a clock at an E divided by 8 rate.

Writing to PWMSCLA or PWMSCLB causes the associated 8-bit down counter to be re-loaded. Otherwise, when changing rates the counter would have to count down to $01 before counting at the proper rate. Forcing the associated counter to re-load the scale register value every time PWMSCLA or PWMSCLB is written prevents this.

NOTE: *Writing to the scale registers while channels are operating can cause irregularities in the PWM outputs.*

4.1.3 Clock Select

Each PWM channel has the capability of selecting one of two clocks. For channels 0, 1, 4, and 5 the clock choices are clock A or clock SA. For channels 2, 3, 6, and 7 the choices are clock B or clock SB. The clock selection is done with the PCLKx control bits in the PWMCLK register.

NOTE: *Changing clock control bits while channels are operating can cause irregularities in the PWM outputs.*

4.2 PWM Channel Timers

The main part of the PWM module are the actual timers. Each of the timer channels has a counter, a period register and a duty register (each are 8-bit). The waveform output period is controlled by a match between the period register and the value in the counter. The duty is controlled by a match between the duty register and the counter value and causes the state of the output to change during the period. The starting polarity of the output is also selectable on a per channel basis. Shown below in **Figure 4-2** is the block diagram for the PWM timer.

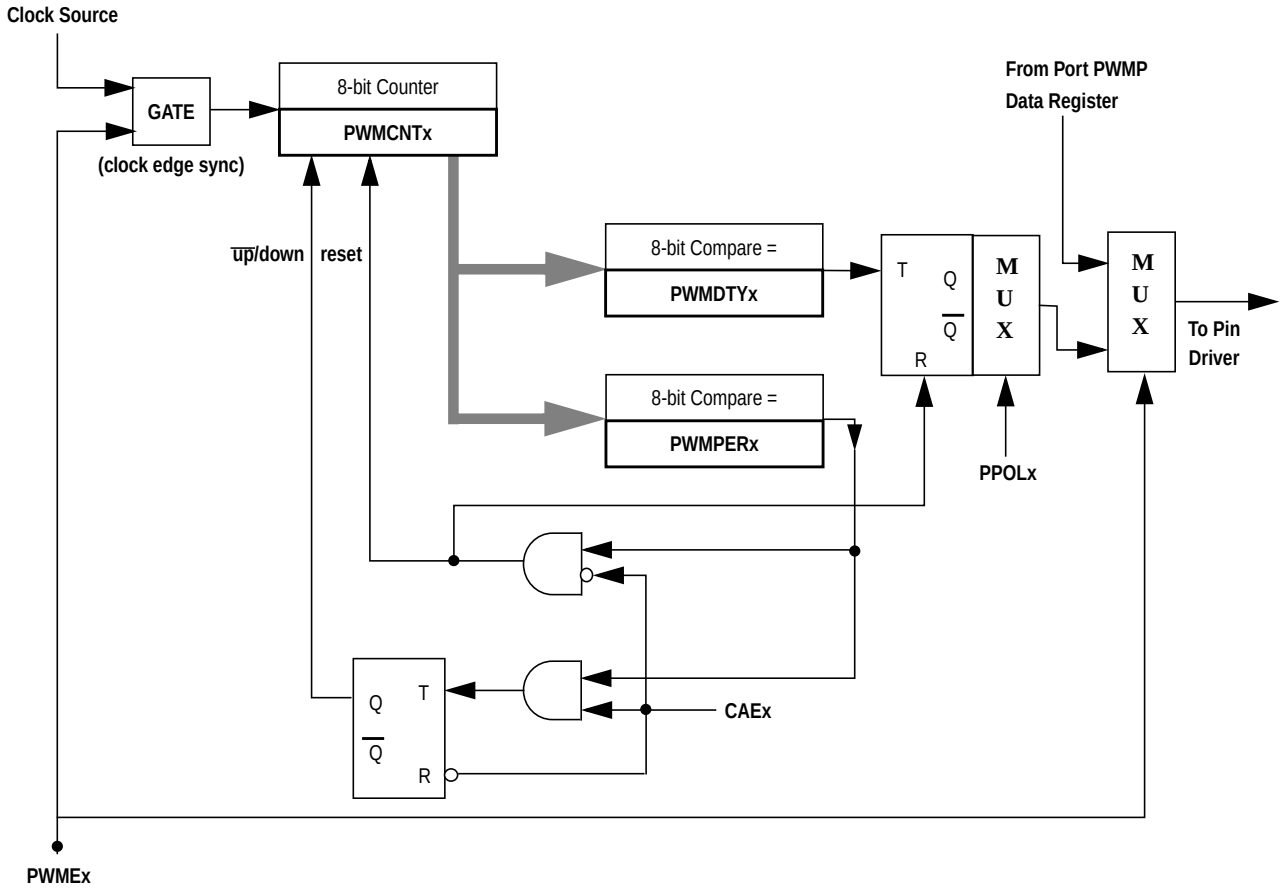


Figure 4-2 PWM Timer Channel Block Diagram

4.2.1 PWM Enable

Each PWM channel has an enable bit (PWMEx) to start its waveform output. When any of the PWMEx bits are set (PWMEx=1), the associated PWM output signal is enabled immediately. However, the actual PWM waveform is not available on the associated PWM output until its clock source begins its next cycle due to the synchronization of PWMEx and the clock source. An exception to this is when channels are concatenated. Refer to **Section 4.2.7 PWM 16-Bit Functions** for more detail.

NOTE: *The first PWM cycle after enabling the channel can be irregular.*

On the front end of the PWM timer, the clock is enabled to the PWM circuit by the PWMEx bit being high. There is an edge-synchronizing circuit to guarantee that the clock will only be enabled or disabled at an edge. When the channel is disabled (PWMEx=0), the counter for the channel does not count.

4.2.2 PWM Polarity

Each channel has a polarity bit to allow starting a waveform cycle with a high or low signal. This is shown on the block diagram as a Mux select of either the Q output or the $\overline{Q}$ output of the PWM output flip flop.

When one of the bits in the PWMPOL register is set, the associated PWM channel output is high at the beginning of the waveform, then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

4.2.3 PWM Period and Duty

Dedicated period and duty registers exist for each channel and are double buffered so that if they change while the channel is enabled, the change will NOT take effect until one of the following occurs:

- The effective period ends
- The counter is written (counter resets to $00)
- The channel is disabled

In this way, the output of the PWM will always be either the old waveform or the new waveform, not some variation in between. If the channel is not enabled, then writes to the period and duty registers will go directly to the latches as well as the buffer.

A change in duty or period can be forced into effect “immediately” by writing the new value to the duty and/or period registers and then writing to the counter. This forces the counter to reset and the new duty and/or period values to be latched. In addition, since the counter is readable it is possible to know where the count is with respect to the duty value and software can be used to make adjustments

NOTE: *When forcing a new period or duty into effect immediately, an irregular PWM cycle can occur.*

NOTE: *Depending on the polarity bit, the duty registers will contain the count of either the high time or the low time.*

4.2.4 PWM Timer Counters

Each channel has a dedicated 8-bit up/down counter which runs at the rate of the selected clock source (reference **Section 4.1 PWM Clock Select** for the available clock sources and rates). The counter compares to two registers, a duty register and a period register as shown in **Figure 4-2 PWM Timer Channel Block Diagram**. When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register behaves differently depending on what output mode is selected as shown in **Figure 4-2 PWM Timer Channel Block Diagram** and described in **Section 4.2.5 Left Aligned Outputs** and **Section 4.2.6 Center Aligned Outputs**.

Each channel counter can be read at anytime without affecting the count or the operation of the PWM channel.

Any value written to the counter causes the counter to reset to $00, the counter direction to be set to up, the immediate load of both duty and period registers with values from the buffers, and the output to change according to the polarity bit. When the channel is disabled (PWMEx=0), the counter stops. When a channel becomes enabled (PWMEx=1), the associated PWM counter continues from the count in the PWMCNTx register. This allows the waveform to continue where it left off when the channel is

re-enabled. When the channel is disabled, writing “0” to the period register will cause the counter to reset on the next selected clock.

NOTE: *If the user wants to start a new “clean” PWM waveform without any “history” from the old waveform, the user must write to channel counter (PWMCNTx) prior to enabling the PWM channel (PWMEx=1).*

Generally, writes to the counter are done prior to enabling a channel in order to start from a known state. However, writing a counter can also be done while the PWM channel is enabled (counting). The effect is similar to writing the counter when the channel is disabled except that the new period is started immediately with the output set according to the polarity bit.

NOTE: *Writing to the counter while the channel is enabled can cause an irregular PWM cycle to occur.*

The counter is cleared at the end of the effective period (see **Section 4.2.5 Left Aligned Outputs** and **Section 4.2.6 Center Aligned Outputs** for more details).

Table 4-1 PWM Timer Counter Conditions

Counter Clears ($00)	Counter Counts	Counter Stops
When PWMCNTx register written to any value	When PWM channel is enabled (PWMEx=1). Counts from last value in PWMCNTx.	When PWM channel is disabled (PWMEx=0)
Effective period ends		

4.2.5 Left Aligned Outputs

The PWM timer provides the choice of two types of outputs, Left Aligned or Center Aligned outputs. They are selected with the CAEx bits in the PWMCAE register. If the CAEx bit is cleared (CAEx=0), the corresponding PWM output will be left aligned.

In left aligned output mode, the 8-bit counter is configured as an up counter only. It compares to two registers, a duty register and a period register as shown in the block diagram in **Figure 4-2 PWM Timer Channel Block Diagram**. When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register resets the counter and the output flip-flop as shown in **Figure 4-2 PWM Timer Channel Block Diagram** as well as performing a load from the double buffer period and duty register to the associated registers as described in **Section 4.2.3 PWM Period and Duty**. The counter counts from 0 to the value in the period register - 1.

NOTE: *Changing the PWM output mode from Left Aligned Output to Center Aligned Output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.*

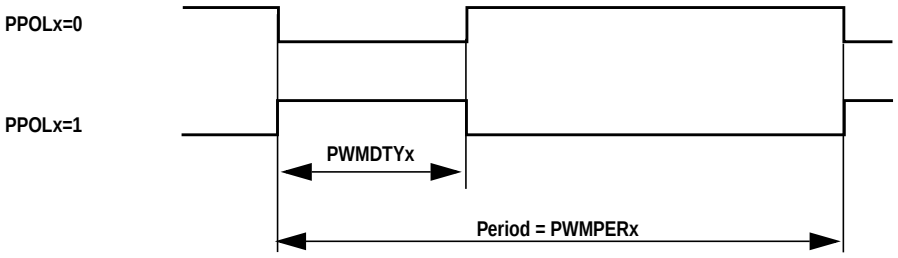


Figure 4-3 PWM Left Aligned Output Waveform

To calculate the output frequency in left aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by the value in the period register for that channel.

- PWMx Frequency = Clock(A, B, SA, or SB) / PWMPERx
- PWMx Duty Cycle (high time as a% of period):
  - Polarity = 0 (PPOLx=0)

    Duty Cycle = [(PWMPERx-PWMDTYx)/PWMPERx] * 100%
  - Polarity = 1 (PPOLx=1)

    Duty Cycle = [PWMDTYx / PWMPERx] * 100%

As an example of a left aligned output, consider the following case:

Clock Source = E, where E=10MHz (100ns period)

PPOLx = 0

PWMPERx = 4

PWMDTYx = 1

PWMx Frequency = 10MHz/4 = 2.5MHz

PWMx Period = 400ns

PWMx Duty Cycle = 3/4 *100% = 75%

Shown below is the output waveform generated.

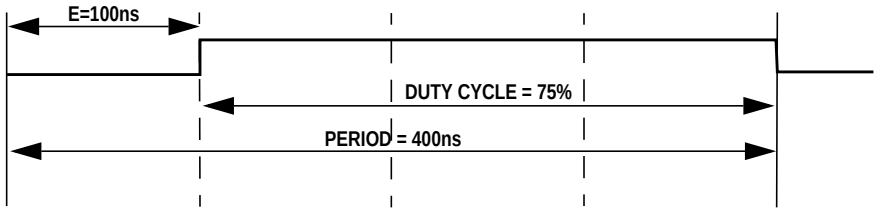


Figure 4-4 PWM Left Aligned Output Example Waveform

4.2.6 Center Aligned Outputs

For Center Aligned Output Mode selection, set the CAEx bit (CAEx=1) in the PWMCAE register and the corresponding PWM output will be center aligned.

The 8-bit counter operates as an up/down counter in this mode and is set to up whenever the counter is equal to $00. The counter compares to two registers, a duty register and a period register as shown in the block diagram in **Figure 4-2 PWM Timer Channel Block Diagram**. When the PWM counter matches the duty register the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register changes the counter direction from an up-count to a down-count. When the PWM counter decrements and matches the duty register again, the output flip-flop changes state causing the PWM output to also change state. When the PWM counter decrements and reaches zero, the counter direction changes from a down-count back to an up-count and a load from the double buffer period and duty registers to the associated registers is performed as described in **Section 4.2.3 PWM Period and Duty**. The counter counts from 0 up to the value in the period register and then back down to 0. Thus the effective period is PWMPERx*2.

NOTE: *Changing the PWM output mode from Left Aligned Output to Center Aligned Output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.*

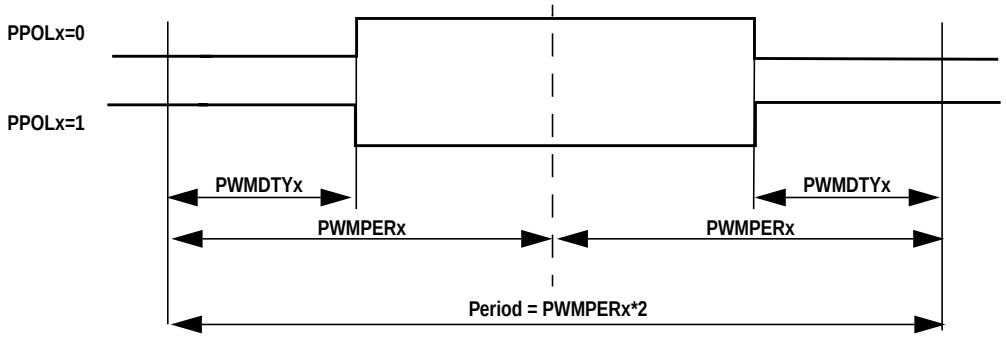


Figure 4-5 PWM Center Aligned Output Waveform

To calculate the output frequency in center aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by twice the value in the period register for that channel.

- PWMx Frequency = Clock(A, B, SA, or SB) / (2*PWMPERx)
- PWMx Duty Cycle (high time as a% of period):
  - Polarity = 0 (PPOLx=0)

    Duty Cycle = [(PWMPERx-PWMDTYx)/PWMPERx] * 100%
  - Polarity = 1 (PPOLx=1)

Duty Cycle = [PWMDTYx / PWMPERx] * 100%

As an example of a center aligned output, consider the following case:

Clock Source = E, where E=10MHz (100ns period)

PPOLx = 0

PWMPERx = 4

PWMDTYx = 1

PWMx Frequency = 10MHz/8 = 1.25MHz

PWMx Period = 800ns

PWMx Duty Cycle = 3/4 *100% = 75%

Shown below is the output waveform generated.

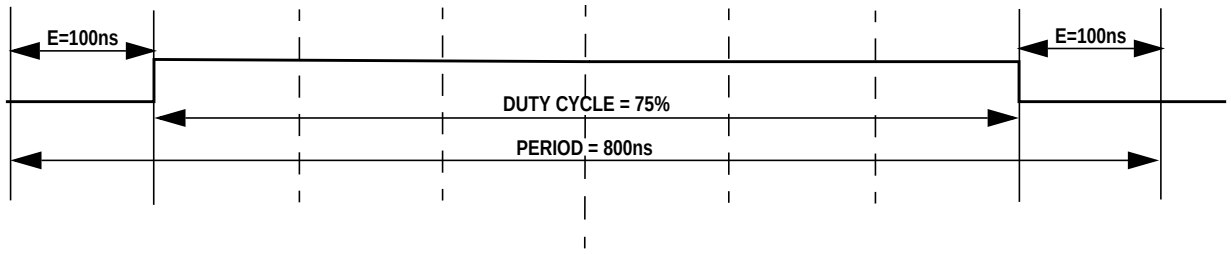


Figure 4-6 PWM Center Aligned Output Example Waveform

4.2.7 PWM 16-Bit Functions

The PWM timer also has the option of generating 8-channels of 8-bits or 4-channels of 16-bits for greater PWM resolution. This 16-bit channel option is achieved through the concatenation of two 8-bit channels.

The PWMCTL register contains four control bits, each of which is used to concatenate a pair of PWM channels into one 16-bit channel. Channels 6 and 7 are concatenated with the CON67 bit, channels 4 and 5 are concatenated with the CON45 bit, channels 2 and 3 are concatenated with the CON23 bit, and channels 0 and 1 are concatenated with the CON01 bit.

NOTE: *Change these bits only when both corresponding channels are disabled.*

When channels 6 and 7 are concatenated, channel 6 registers become the high order bytes of the double byte channel as shown in **Figure 4-7 PWM 16-Bit Mode**. Similarly, when channels 4 and 5 are concatenated, channel 4 registers become the high order bytes of the double byte channel. When channels 2 and 3 are concatenated, channel 2 registers become the high order bytes of the double byte channel. When channels 0 and 1 are concatenated, channel 0 registers become the high order bytes of the double

byte channel.

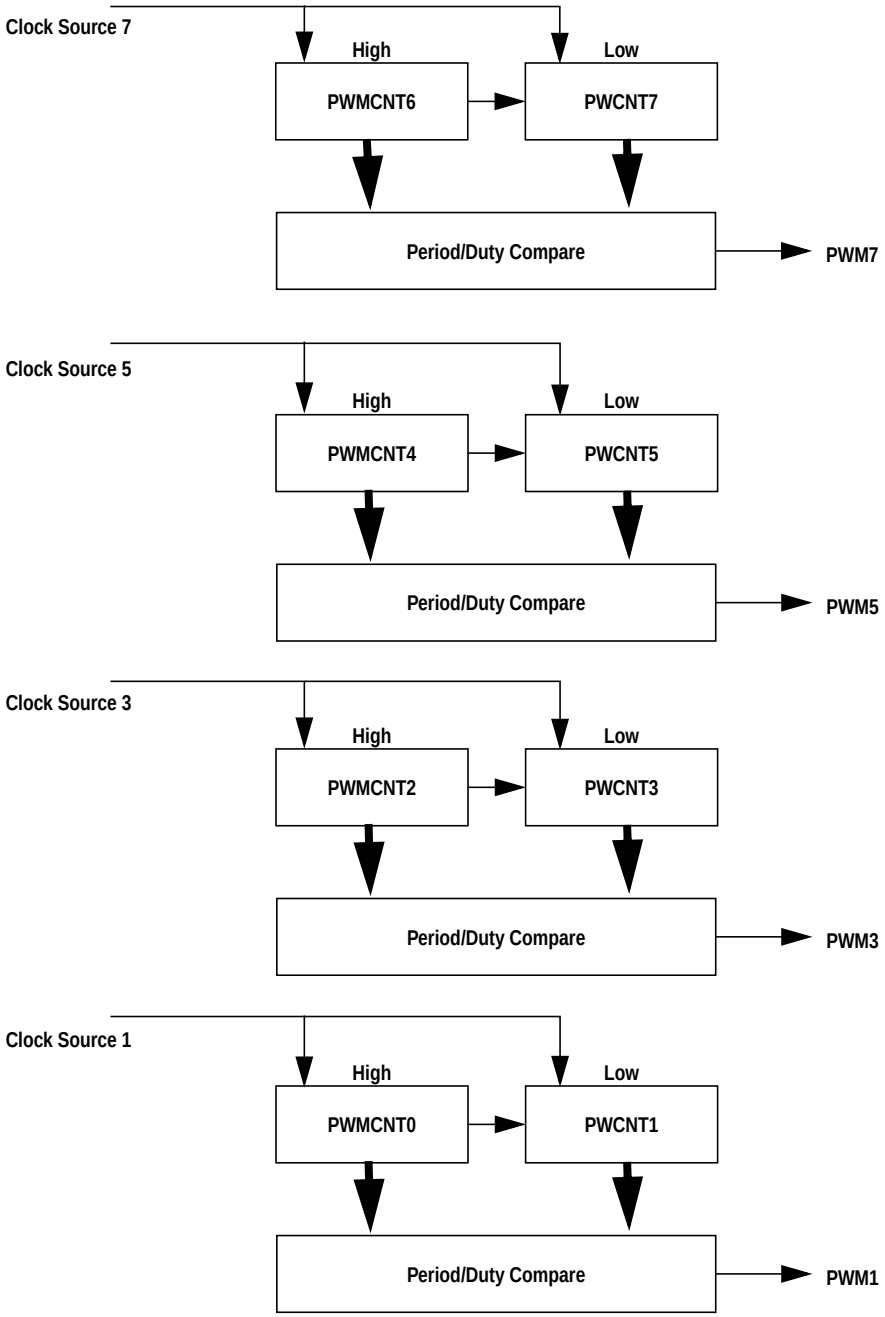


Figure 4-7 PWM 16-Bit Mode

When using the 16-bit concatenated mode, the clock source is determined by the low order 8-bit channel clock select control bits. That is channel 7 when channels 6 and 7 are concatenated, channel 5 when channels 4 and 5 are concatenated, channel 3 when channels 2 and 3 are concatenated, and channel 1 when channels 0 and 1 are concatenated. The resulting PWM is output to the pins of the corresponding low order

8-bit channel as also shown in **Figure 4-7 PWM 16-Bit Mode**. The polarity of the resulting PWM output is controlled by the PPOLx bit of the corresponding low order 8-bit channel as well.

Once concatenated mode is enabled (CONxx bits set in PWMCTL register) then enabling/disabling the corresponding 16-bit PWM channel is controlled by the low order PWMEx bit. In this case, the high order bytes PWMEx bits have no effect and their corresponding PWM output is disabled.

In concatenated mode, writes to the 16-bit counter by using a 16-bit access or writes to either the low or high order byte of the counter will reset the 16-bit counter. Reads of the 16-bit counter must be made by 16-bit access to maintain data coherency.

Either left aligned or center aligned output mode can be used in concatenated mode and is controlled by the low order CAEx bit. The high order CAEx bit has no effect.

The table shown below is used to summarize which channels are used to set the various control bits when in 16-bit mode.

Table 4-2 16-bit Concatenation Mode Summary

CONxx	PWMEx	PPOLx	PCLKx	CAEx	PWMx OUTPUT
CON67	PWME7	PPOL7	PCLK7	CAE7	PWM7
CON45	PWME5	PPOL5	PCLK5	CAE5	PWM5
CON23	PWME3	PPOL3	PCLK3	CAE3	PWM3
CON01	PWME1	PPOL1	PCLK1	CAE1	PWM1

4.2.8 PWM Boundary Cases

The following table summarizes the boundary conditions for the PWM regardless of the output mode (Left Aligned or Center Aligned) and 8-bit (normal) or 16-bit (concatenation).

Table 4-3 PWM Boundary Cases

PWMDTYx	PWMPERx	PPOLx	PWMx Output
$00 (indicates no duty)	>$00	1	Always Low
$00 (indicates no duty)	>$00	0	Always High
XX	$00[1] (indicates no period)	1	Always High
XX	$00[1] (indicates no period)	0	Always Low
>= PWMPERx	XX	1	Always High
>= PWMPERx	XX	0	Always Low

NOTES:
1. Counter=$00 and does not count.

Section 5 Resets

5.1 General

The reset state of each individual bit is listed within the Register Description section **Section 3.3 Register Descriptions** which details the registers and their bit-fields. All special functions or modes which are initialized during or just following reset are described within this section.

- The 8-bit up/down counter is configured as an up counter out of reset.
- All the channels are disabled and all the counters don’t count.

Section 6 Interrupts

6.1 Interrupt Operation

The PWM module has only one interrupt which is generated at the time of emergency shutdown, if the corresponding enable bit (PWMIE) is set. This bit is the enable for the interrupt. The interrupt flag PWMIF is set whenever the input level of the PWM7 channel changes while PWM7ENA=1 or when PWMENA is being asserted while the level at PWM7 is active.

In STOP mode or WAIT mode(with the PSWAI bit set) the emergency shutdown feature will drive the PWM outputs to their shutdown output levels but the PWMIF flag will not be set.

A description of the registers involved and affected due to this interrupt is explained in **Section 3.3.15 PWM Shutdown Register (PWMSDN)**.

The PWM block only generates the interrupt and does not service it. The interrupt signal name is PWM Interrupt Signal.

User Guide End Sheet

FINAL PAGE OF
54
PAGES

Chapter 1
Serial Communications Interface (S12SCIV2) Block Description

1.1 Introduction

This block guide provide an overview of serial communication interface (SCI) module. The SCI allows asynchronous serial communications with peripheral devices and other CPUs.

1.1.1 Glossary

IRQ — Interrupt Request

LSB — Least Significant Bit

MSB — Most Significant Bit

NRZ — Non-Return-to-Zero

RZI — Return-to-Zero-Inverted

RXD — Receive Pin

SCI — Serial Communication Interface

TXD — Transmit Pin

1.1.2 Features

The SCI includes these distinctive features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable 8-bit or 9-bit data format
- Separately enabled transmitter and receiver
- Programmable transmitter output parity
- Two receiver wake up methods:
  - — Idle line wake-up
  - — Address mark wake-up
- Interrupt-driven operation with eight flags:
  - — Transmitter empty

  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- Hardware parity checking
- 1/16 bit-time noise detection

1.1.3 Modes of Operation

The SCI operation is the same independent of device resource mapping and bus interface mode. Different power modes are available to facilitate power saving.

1.1.3.1 Run Mode

Normal mode of operation.

1.1.3.2 Wait Mode

SCI operation in wait mode depends on the state of the SCISWAI bit in the SCI control register 1 (SCICR1).

- If SCISWAI is clear, the SCI operates normally when the CPU is in wait mode.
- If SCISWAI is set, SCI clock generation ceases and the SCI module enters a power-conservation state when the CPU is in wait mode. Setting SCISWAI does not affect the state of the receiver enable bit, RE, or the transmitter enable bit, TE.
- If SCISWAI is set, any transmission or reception in progress stops at wait mode entry. The transmission or reception resumes when either an internal or external interrupt brings the CPU out of wait mode. Exiting wait mode by reset aborts any transmission or reception in progress and resets the SCI.

1.1.3.3 Stop Mode

The SCI is inactive during stop mode for reduced power consumption. The STOP instruction does not affect the SCI register states, but the SCI module clock will be disabled. The SCI operation resumes from where it left off after an external interrupt brings the CPU out of stop mode. Exiting stop mode by reset aborts any transmission or reception in progress and resets the SCI.

1.1.4 Block Diagram

Figure 1-1 is a high level block diagram of the SCI module, showing the interaction of various functional blocks.

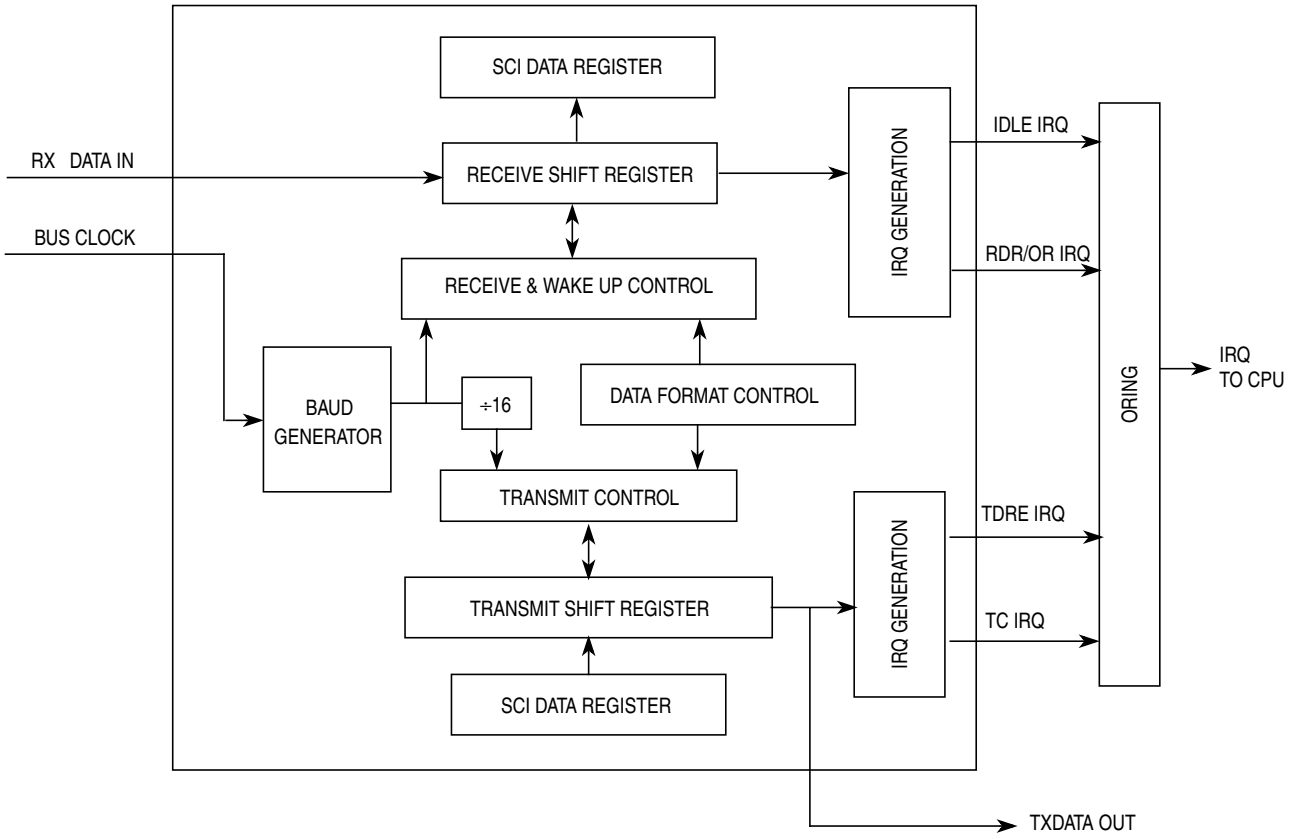


Figure 1-1. SCI Block Diagram

1.2 External Signal Description

The SCI module has a total of two external pins:

1.2.1 TXD-SCI Transmit Pin

This pin serves as transmit data output of SCI.

1.2.2 RXD-SCI Receive Pin

This pin serves as receive data input of the SCI.

1.3 Memory Map and Registers

This section provides a detailed description of all memory and registers.

1.3.1 Module Memory Map

The memory map for the SCI module is given below in Figure 1-2. The Address listed for each register is the address offset. The total address for each register is the sum of the base address for the SCI module and the address offset for each register.

Address	Name		Bit 7	6	5	4	3	2	1	Bit 0
0x0000	SCIBDH	R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
		W								
0x0001	SCIBDL	R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
		W								
0x0002	SCICR1	R	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
		W								
0x0003	SCICR2	R	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
		W								
0x0004	SCISR1	R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
		W								
0x0005	SCISR2	R	0	0	0	0	0	BRK13	TXDIR	RAF
		W								
0x0006	SCIDRH	R	R8	T8	0	0	0	0	0	0
		W								
0x0007	SCIDRL	R	R7	R6	R5	R4	R3	R2	R1	R0
		W	T7	T6	T5	T4	T3	T2	T1	T0

= Unimplemented or Reserved

Figure 1-2. SCI Register Summary

1.3.2 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a zero. Details of register bit and field function follow the register diagrams, in bit order.

1.3.2.1 SCI Baud Rate Registers (SCIBDH and SCHBDL)

Module Base + 0x_0000

	7	6	5	4	3	2	1	0
R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
W								
Reset	0	0	0	0	0	0	0	0

Module Base + 0x_0001

	7	6	5	4	3	2	1	0
R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
W								
Reset	0	0	0	0	0	1	0	0

= Unimplemented or Reserved

Figure 1-3. SCI Baud Rate Registers (SCIBDH and SCIBDL)

The SCI Baud Rate Register is used by the counter to determine the baud rate of the SCI. The formula for calculating the baud rate is:

SCI baud rate = SCI module clock / (16 x BR)

where:

BR is the content of the SCI baud rate registers, bits SBR12 through SBR0. The baud rate registers can contain a value from 1 to 8191.

Read: Anytime. If only SCIBDH is written to, a read will not return the correct data until SCIBDL is written to as well, following a write to SCIBDH.

Write: Anytime

Table 1-1. SCIBDH AND SCIBDL Field Descriptions

Field	Description
4–0 7–0 SBR[12:0]	**SCI Baud Rate Bits** — The baud rate for the SCI is determined by these 13 bits. **Note:** The baud rate generator is disabled until the TE bit or the RE bit is set for the first time after reset. The baud rate generator is disabled when BR = 0. Writing to SCIBDH has no effect without writing to SCIBDL, since writing to SCIBDH puts the data in a temporary location until SCIBDL is written to.

1.3.2.2 SCI Control Register 1 (SCICR1)

Module Base + 0x_0002

	7	6	5	4	3	2	1	0
R W	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

Figure 1-4. SCI Control Register 1 (SCICR1)

Read: Anytime

Write: Anytime

Table 1-2. SCICR1 Field Descriptions

Field	Description
7 LOOPS	**Loop Select Bit** — LOOPS enables loop operation. In loop operation, the RXD pin is disconnected from the SCI and the transmitter output is internally connected to the receiver input. Both the transmitter and the receiver must be enabled to use the loop function.See Table 1-3. 0 Normal operation enabled 1 Loop operation enabled **Note:** The receiver input is determined by the RSRC bit.
6 SCISWAI	**SCI Stop in Wait Mode Bit** — SCISWAI disables the SCI in wait mode. 0 SCI enabled in wait mode 1 SCI disabled in wait mode
5 RSRC	**Receiver Source Bit** — When LOOPS = 1, the RSRC bit determines the source for the receiver shift register input. 0 Receiver input internally connected to transmitter output 1 Receiver input connected externally to transmitter
4 M	**Data Format Mode Bit** — MODE determines whether data characters are eight or nine bits long. 0 One start bit, eight data bits, one stop bit 1 One start bit, nine data bits, one stop bit
3 WAKE	**Wakeup Condition Bit** — WAKE determines which condition wakes up the SCI: a logic 1 (address mark) in the most significant bit position of a received data character or an idle condition on the RXD. 0 Idle line wakeup 1 Address mark wakeup
2 ILT	**Idle Line Type Bit** — ILT determines when the receiver starts counting logic 1s as idle character bits. The counting begins either after the start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding the stop bit may cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires properly synchronized transmissions. 0 Idle character bit count begins after start bit 1 Idle character bit count begins after stop bit
1 PE	**Parity Enable Bit** — PE enables the parity function. When enabled, the parity function inserts a parity bit in the most significant bit position. 0 Parity function disabled 1 Parity function enabled
0 PT	**Parity Type Bit** — PT determines whether the SCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit. 0 Even parity 1 Odd parity

Table 1-3. Loop Functions

LOOPS	RSRC	Function
0	x	Normal operation
1	0	Loop mode with Rx input internally connected to Tx output
1	1	Single-wire mode with Rx input connected to TXD

1.3.2.3 SCI Control Register 2 (SCICR2)

Module Base + 0x_0003

	7	6	5	4	3	2	1	0
R W	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

Figure 1-5. SCI Control Register 2 (SCICR2)

Read: Anytime

Write: Anytime

Table 1-4. SCICR2 Field Descriptions

Field	Description
7 TIE	**Transmitter Interrupt Enable Bit** — TIE enables the transmit data register empty flag, TDRE, to generate interrupt requests. 0 TDRE interrupt requests disabled 1 TDRE interrupt requests enabled
6 TCIE	**Transmission Complete Interrupt Enable Bit** — TCIE enables the transmission complete flag, TC, to generate interrupt requests. 0 TC interrupt requests disabled 1 TC interrupt requests enabled
5 RIE	**Receiver Full Interrupt Enable Bit** — RIE enables the receive data register full flag, RDRF, or the overrun flag, OR, to generate interrupt requests. 0 RDRF and OR interrupt requests disabled 1 RDRF and OR interrupt requests enabled
4 ILIE	**Idle Line Interrupt Enable Bit** — ILIE enables the idle line flag, IDLE, to generate interrupt requests. 0 IDLE interrupt requests disabled 1 IDLE interrupt requests enabled
3 TE	**Transmitter Enable Bit** — TE enables the SCI transmitter and configures the TXD pin as being controlled by the SCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled 1 Transmitter enabled
2 RE	**Receiver Enable Bit** — RE enables the SCI receiver. 0 Receiver disabled 1 Receiver enabled

Table 1-4. SCICR2 Field Descriptions (continued)

Field	Description
1 RWU	**Receiver Wakeup Bit** — Standby state 0 Normal operation. 1 RWU enables the wakeup function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU.
0 SBK	**Send Break Bit** — Toggling SBK sends one break character (10 or 11 logic 0s, respectively 13 or 14 logics 0s if BRK13 is set). Toggling implies clearing the SBK bit before the break character has finished transmitting. As long as SBK is set, the transmitter continues to send complete break characters (10 or 11 bits, respectively 13 or 14 bits). 0 No break characters 1 Transmit break characters

1.3.2.4 SCI Status Register 1 (SCISR1)

The SCISR1 and SCISR2 registers provides inputs to the MCU for generation of SCI interrupts. Also, these registers can be polled by the MCU to check the status of these bits. The flag-clearing procedures require that the status register be read followed by a read or write to the SCI Data Register.It is permissible to execute other instructions between the two steps as long as it does not compromise the handling of I/O, but the order of operations is important for flag clearing.

Module Base + 0x_0004

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 1-6. SCI Status Register 1 (SCISR1)

Read: Anytime

Write: Has no meaning or effect

Table 1-5. SCISR1 Field Descriptions

Field	Description
7 TDRE	**Transmit Data Register Empty Flag** — TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is 1, the transmit data register (SCIDRH/L) is empty and can receive a new value to transmit.Clear TDRE by reading SCI status register 1 (SCISR1), with TDRE set and then writing to SCI data register low (SCIDRL). 0 No byte transferred to transmit shift register 1 Byte transferred to transmit shift register; transmit data register empty
6 TC	**Transmit Complete Flag** — TC is set low when there is a transmission in progress or when a preamble or break character is loaded. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted.When TC is set, the TXD out signal becomes idle (logic 1). Clear TC by reading SCI status register 1 (SCISR1) with TC set and then writing to SCI data register low (SCIDRL). TC is cleared automatically when data, preamble, or break is queued and ready to be sent. TC is cleared in the event of a simultaneous set and clear of the TC flag (transmission not complete). 0 Transmission in progress 1 No transmission in progress

Table 1-5. SCISR1 Field Descriptions (continued)

Field	Description
5 RDRF	**Receive Data Register Full Flag** — RDRF is set when the data in the receive shift register transfers to the SCI data register. Clear RDRF by reading SCI status register 1 (SCISR1) with RDRF set and then reading SCI data register low (SCIDRL). 0 Data not available in SCI data register 1 Received data available in SCI data register
4 IDLE	**Idle Line Flag** — IDLE is set when 10 consecutive logic 1s (if M=0) or 11 consecutive logic 1s (if M=1) appear on the receiver input. Once the IDLE flag is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag.Clear IDLE by reading SCI status register 1 (SCISR1) with IDLE set and then reading SCI data register low (SCIDRL). 0 Receiver input is either active now or has never become active since the IDLE flag was last cleared 1 Receiver input has become idle **Note:** When the receiver wakeup bit (RWU) is set, an idle line condition does not set the IDLE flag.
3 OR	**Overrun Flag** — OR is set when software fails to read the SCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame. The data in the shift register is lost, but the data already in the SCI data registers is not affected. Clear OR by reading SCI status register 1 (SCISR1) with OR set and then reading SCI data register low (SCIDRL). 0 No overrun 1 Overrun **Note:** OR flag may read back as set when RDRF flag is clear. This may happen if the following sequence of events occurs: 1. After the first frame is received, read status register SCISR1 (returns RDRF set and OR flag clear); 2. Receive second frame without reading the first frame in the data register (the second frame is not received and OR flag is set); 3. Read data register SCIDRL (returns first frame and clears RDRF flag in the status register); 4. Read status register SCISR1 (returns RDRF clear and OR set). Event 3 may be at exactly the same time as event 2 or any time after. When this happens, a dummy SCIDRL read following event 4 will be required to clear the OR flag if further frames are to be received.
2 NF	**Noise Flag** — NF is set when the SCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by reading SCI status register 1(SCISR1), and then reading SCI data register low (SCIDRL). 0 No noise 1 Noise
1 FE	**Framing Error Flag** — FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. FE inhibits further data reception until it is cleared. Clear FE by reading SCI status register 1 (SCISR1) with FE set and then reading the SCI data register low (SCIDRL). 0 No framing error 1 Framing error
0 PF	**Parity Error Flag** — PF is set when the parity enable bit (PE) is set and the parity of the received data does not match the parity type bit (PT). PF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear PF by reading SCI status register 1 (SCISR1), and then reading SCI data register low (SCIDRL). 0 No parity error 1 Parity error

1.3.2.5 SCI Status Register 2 (SCISR2)

Module Base + 0x_0005

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	BK13	TXDIR	RAF
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 1-7. SCI Status Register 2 (SCISR2)

Read: Anytime

Write: Anytime; writing accesses SCI status register 2; writing to any bits except TXDIR and BRK13 (SCISR2[1] & [2]) has no effect

Table 1-6. SCISR2 Field Descriptions

Field	Description
2 BK13	**Break Transmit Character Length** — This bit determines whether the transmit break character is 10 or 11 bit respectively 13 or 14 bits long. The detection of a framing error is not affected by this bit. 0 Break Character is 10 or 11 bit long 1 Break character is 13 or 14 bit long
1 TXDIR	**Transmitter Pin Data Direction in Single-Wire Mode.** — This bit determines whether the TXD pin is going to be used as an input or output, in the Single-Wire mode of operation. This bit is only relevant in the Single-Wire mode of operation. 0 TXD pin to be used as an input in Single-Wire mode 1 TXD pin to be used as an output in Single-Wire mode
0 RAF	**Receiver Active Flag** — RAF is set when the receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character. 0 No reception in progress 1 Reception in progress

1.3.2.6 SCI Data Registers (SCIDRH and SCIDRL)

Module Base + 0x_0006

	7	6	5	4	3	2	1	0
R	R8	T8	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Module Base + 0x_0007

	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W	T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	1	0	0

= Unimplemented or Reserved

Figure 1-8. SCI Data Registers (SCIDRH and SCIDRL)

Read: Anytime; reading accesses SCI receive data register

Write: Anytime; writing accesses SCI transmit data register; writing to R8 has no effect

Table 1-7. SCIDRH AND SCIDRL Field Descriptions

Field	Description
7 R8	**Received Bit 8** — R8 is the ninth data bit received when the SCI is configured for 9-bit data format (M = 1).
6 T8	**Transmit Bit 8** — T8 is the ninth data bit transmitted when the SCI is configured for 9-bit data format (M = 1).
7–0 R[7:0] T[7:0]	**Received Bits** — Received bits seven through zero for 9-bit or 8-bit data formats **Transmit Bits** — Transmit bits seven through zero for 9-bit or 8-bit formats

NOTE

If the value of T8 is the same as in the previous transmission, T8 does not have to be rewritten.The same value is transmitted until T8 is rewritten

In 8-bit data format, only SCI data register low (SCIDRL) needs to be accessed.

When transmitting in 9-bit data format and using 8-bit write instructions, write first to SCI data register high (SCIDRH), then SCIDRL.

1.4 Functional Description

This section provides a complete functional description of the SCI block, detailing the operation of the design from the end user perspective in a number of subsections.

Figure 1-9 shows the structure of the SCI module. The SCI allows full duplex, asynchronous, NRZ serial communication between the CPU and remote devices, including other CPUs. The SCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the SCI, writes the data to be transmitted, and processes received data.

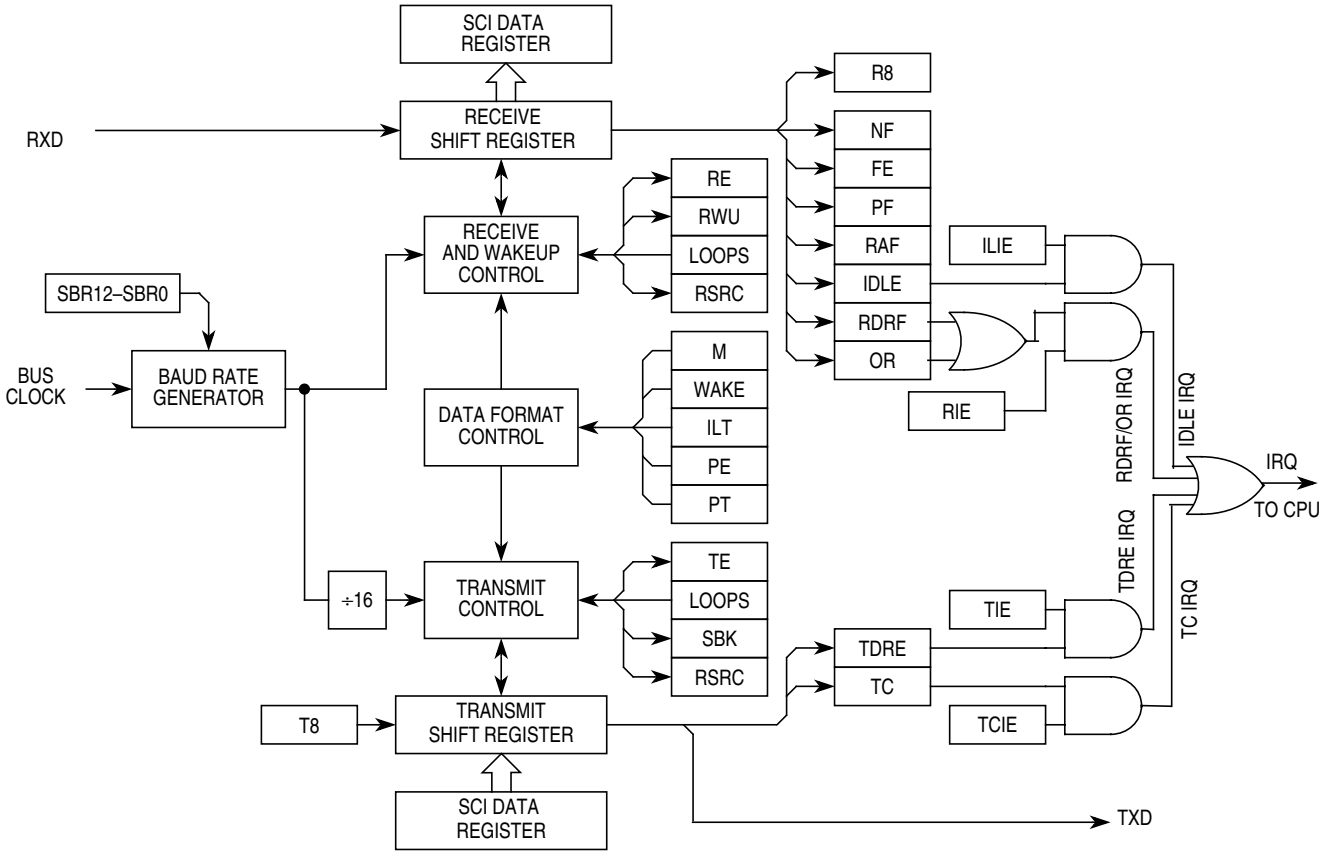


Figure 1-9. SCI Block Diagram

1.4.1 Data Format

The SCI uses the standard NRZ mark/space data format illustrated in Figure 1-10 below.

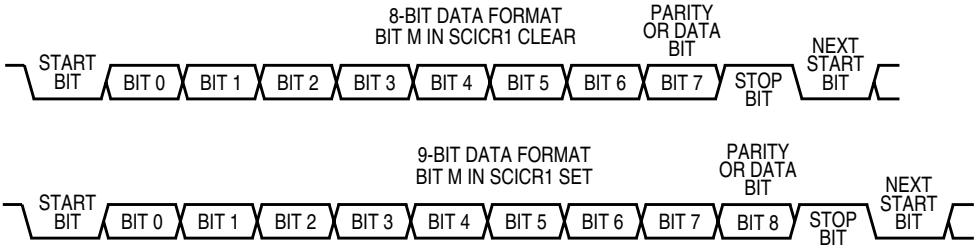


Figure 1-10. SCI Data Formats

Each data character is contained in a frame that includes a start bit, eight or nine data bits, and a stop bit. Clearing the M bit in SCI control register 1 configures the SCI for 8-bit data characters.A frame with eight

data bits has a total of 10 bits. Setting the M bit configures the SCI for nine-bit data characters. A frame with nine data bits has a total of 11 bits

Table 1-8. Example of 8-Bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	8	0	0	1
1	7	0	1	1
1	7	1[1]	0	1

[1] The address bit identifies the frame as an address character. See Section 1.4.4.6, “Receiver Wakeup”.

When the SCI is configured for 9-bit data characters, the ninth data bit is the T8 bit in SCI data register high (SCIDRH). It remains unchanged after transmission and can be used repeatedly without rewriting it. A frame with nine data bits has a total of 11 bits.

Table 1-9. Example of 9-Bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	9	0	0	1
1	8	0	1	1
1	8	1[1]	0	1

[1] The address bit identifies the frame as an address character. See Section 1.4.4.6, “Receiver Wakeup”.

1.4.2 Baud Rate Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value from 0 to 8191 written to the SBR12–SBR0 bits determines the module clock divisor. The SBR bits are in the SCI baud rate registers (SCIBDH and SCIBDL). The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

Baud rate generation is subject to one source of error:

Integer division of the module clock may not give the exact target frequency.

Table 1-10 lists some examples of achieving target baud rates with a module clock frequency of 25 MHz

SCI baud rate = SCI module clock / (16 * SCIBR[12:0])

Table 1-10. Baud Rates (Example: Module Clock = 25 MHz)

Bits SBR[12-0]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
41	609,756.1	38,109.8	38,400	.76
81	308,642.0	19,290.1	19,200	.47
163	153,374.2	9585.9	9600	.16
326	76,687.1	4792.9	4800	.15

Table 1-10. Baud Rates (Example: Module Clock = 25 MHz)

Bits SBR[12-0]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
651	38,402.5	2400.2	2400	.01
1302	19,201.2	1200.1	1200	.01
2604	9600.6	600.0	600	.00
5208	4800.0	300.0	300	.00
10417	2400.0	150.0	150	.00
14204	1760.1	110.0	110	.00

1.4.3 Transmitter

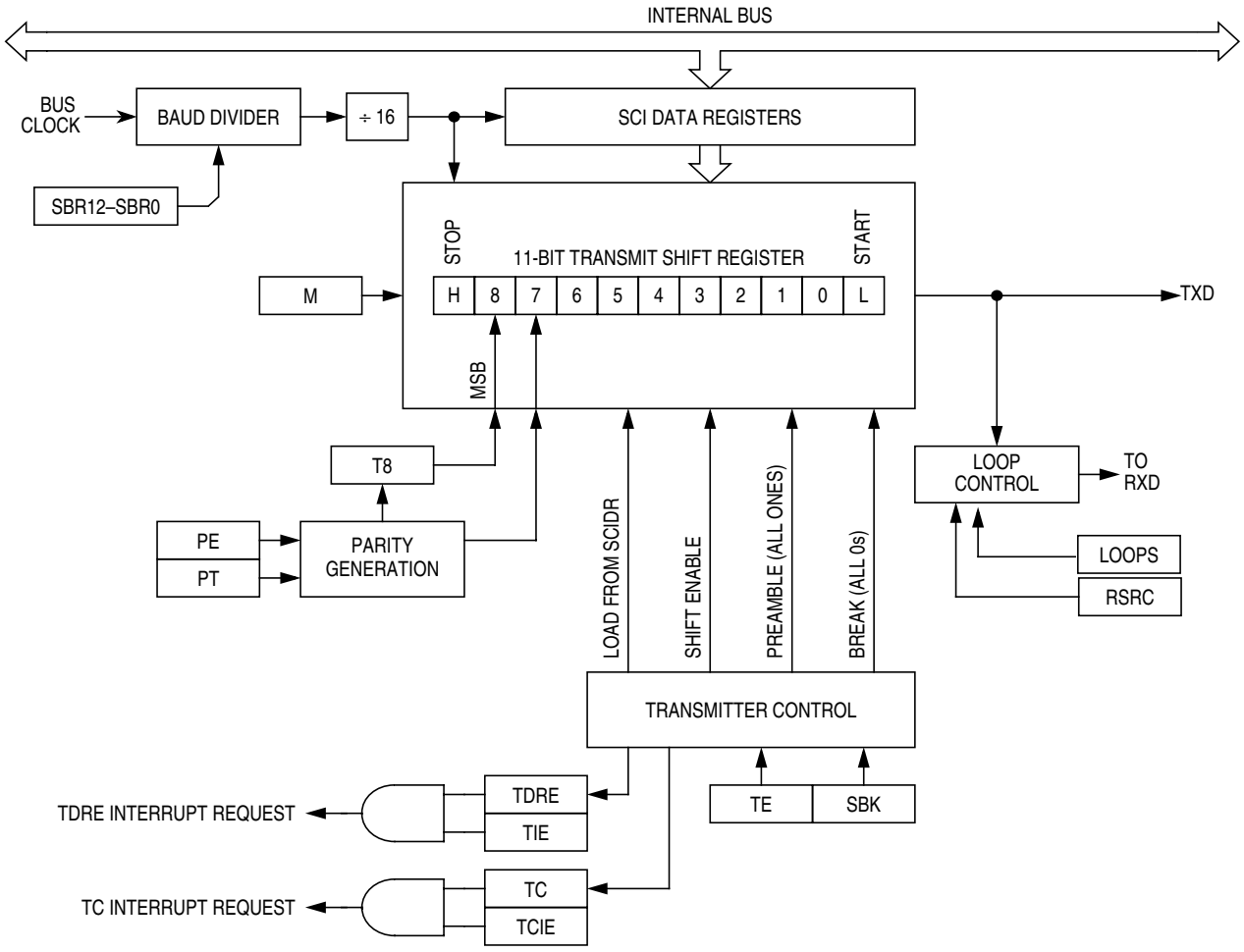


Figure 1-11. Transmitter Block Diagram

1.4.3.1 Transmitter Character Length

The SCI transmitter can accommodate either 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCICR1) determines the length of data characters. When transmitting 9-bit data, bit T8 in SCI data register high (SCIDRH) is the ninth bit (bit 8).

1.4.3.2 Character Transmission

To transmit data, the MCU writes the data bits to the SCI data registers (SCIDRH/SCIDRL), which in turn are transferred to the transmitter shift register. The transmit shift register then shifts a frame out through the **Tx output** signal, after it has prefaced them with a start bit and appended them with a stop bit. The SCI data registers (SCIDRH and SCIDRL) are the write-only buffers between the internal data bus and the transmit shift register.

The SCI also sets a flag, the transmit data register empty flag (TDRE), every time it transfers data from the buffer (SCIDRH/L) to the transmitter shift register.The transmit driver routine may respond to this flag by writing another byte to the Transmitter buffer (SCIDRH/SCIDRL), while the shift register is still shifting out the first byte.

To initiate an SCI transmission:

1. Configure the SCI:
   a) Select a baud rate. Write this value to the SCI baud registers (SCIBDH/L) to begin the baud rate generator. Remember that the baud rate generator is disabled when the baud rate is zero. Writing to the SCIBDH has no effect without also writing to SCIBDL.
   b) Write to SCICR1 to configure word length, parity, and other configuration bits (LOOPS,RSRC,M,WAKE,ILT,PE,PT).
   c) Enable the transmitter, interrupts, receive, and wake up as required, by writing to the SCICR2 register bits (TIE,TCIE,RIE,ILIE,TE,RE,RWU,SBK). A preamble or idle character will now be shifted out of the transmitter shift register.
2. Transmit Procedure for Each Byte:
   a. Poll the TDRE flag by reading the SCISR1 or responding to the TDRE interrupt. Keep in mind that the TDRE bit resets to one.
   d) If the TDRE flag is set, write the data to be transmitted to SCIDRH/L, where the ninth bit is written to the T8 bit in SCIDRH if the SCI is in 9-bit data format. A new transmission will not result until the TDRE flag has been cleared.
3. Repeat step 2 for each subsequent transmission.

NOTE

The TDRE flag is set when the shift register is loaded with the next data to be transmitted from SCIDRH/L, which happens, generally speaking, a little over half-way through the stop bit of the previous frame. Specifically, this transfer occurs 9/16ths of a bit time AFTER the start of the stop bit of the previous frame.

Writing the TE bit from 0 to a 1 automatically loads the transmit shift register with a preamble of 10 logic 1s (if M = 0) or 11 logic 1s (if M = 1). After the preamble shifts out, control logic transfers the data from

the SCI data register into the transmit shift register. A logic 0 start bit automatically goes into the least significant bit position of the transmit shift register. A logic 1 stop bit goes into the most significant bit position.

Hardware supports odd or even parity. When parity is enabled, the most significant bit (msb) of the data character is the parity bit.

The transmit data register empty flag, TDRE, in SCI status register 1 (SCISR1) becomes set when the SCI data register transfers a byte to the transmit shift register. The TDRE flag indicates that the SCI data register can accept new data from the internal data bus. If the transmit interrupt enable bit, TIE, in SCI control register 2 (SCICR2) is also set, the TDRE flag generates a transmitter interrupt request.

When the transmit shift register is not transmitting a frame, the **Tx output** signal goes to the idle condition, logic 1. If at any time software clears the TE bit in SCI control register 2 (SCICR2), the transmitter enable signal goes low and the transmit signal goes idle.

If software clears TE while a transmission is in progress (TC = 0), the frame in the transmit shift register continues to shift out. To avoid accidentally cutting off the last frame in a message, always wait for TDRE to go high after the last frame before clearing TE.

To separate messages with preambles with minimum idle line time, use this sequence between messages:

1. Write the last byte of the first message to SCIDRH/L.
2. Wait for the TDRE flag to go high, indicating the transfer of the last frame to the transmit shift register.
3. Queue a preamble by clearing and then setting the TE bit.
4. Write the first byte of the second message to SCIDRH/L.

1.4.3.3 Break Characters

Writing a logic 1 to the send break bit, SBK, in SCI control register 2 (SCICR2) loads the transmit shift register with a break character. A break character contains all logic 0s and has no start, stop, or parity bit. Break character length depends on the M bit in SCI control register 1 (SCICR1). As long as SBK is at logic 1, transmitter logic continuously loads break characters into the transmit shift register. After software clears the SBK bit, the shift register finishes transmitting the last break character and then transmits at least one logic 1. The automatic logic 1 at the end of a break character guarantees the recognition of the start bit of the next frame.

The SCI recognizes a break character when a start bit is followed by eight or nine logic 0 data bits and a logic 0 where the stop bit should be. Receiving a break character has these effects on SCI registers:

- Sets the framing error flag, FE
- Sets the receive data register full flag, RDRF
- Clears the SCI data registers (SCIDRH/L)
- May set the overrun flag, OR, noise flag, NF, parity error flag, PE, or the receiver active flag, RAF (see Section 1.3.2.4, “SCI Status Register 1 (SCISR1)” and Section 1.3.2.5, “SCI Status Register 2 (SCISR2)”

1.4.3.4 Idle Characters

An idle character contains all logic 1s and has no start, stop, or parity bit. Idle character length depends on the M bit in SCI control register 1 (SCICR1). The preamble is a synchronizing idle character that begins the first transmission initiated after writing the TE bit from 0 to 1.

If the TE bit is cleared during a transmission, the **Tx output** signal becomes idle after completion of the transmission in progress. Clearing and then setting the TE bit during a transmission queues an idle character to be sent after the frame currently being transmitted.

NOTE

When queueing an idle character, return the TE bit to logic 1 before the stop bit of the current frame shifts out through the **Tx output** signal. Setting TE after the stop bit appears on **Tx output signal** causes data previously written to the SCI data register to be lost. Toggle the TE bit for a queued idle character while the TDRE flag is set and immediately before writing the next byte to the SCI data register.

NOTE

If the TE bit is clear and the transmission is complete, the SCI is not the master of the TXD pin

1.4.4 Receiver

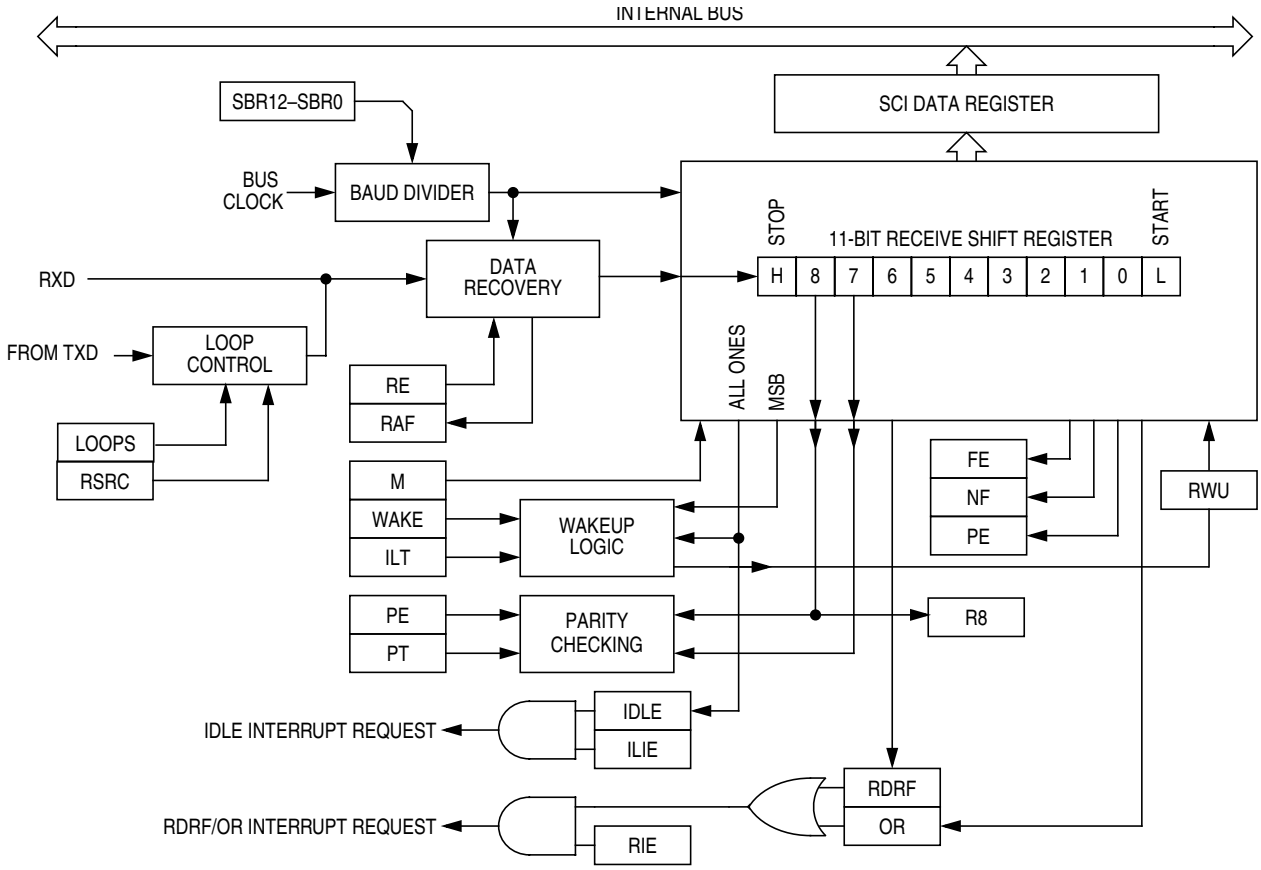


Figure 1-12. SCI Receiver Block Diagram

1.4.4.1 Receiver Character Length

The SCI receiver can accommodate either 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCICR1) determines the length of data characters. When receiving 9-bit data, bit R8 in SCI data register high (SCIDRH) is the ninth bit (bit 8).

1.4.4.2 Character Reception

During an SCI reception, the receive shift register shifts a frame in from the **Rx input** signal. The SCI data register is the read-only buffer between the internal data bus and the receive shift register.

After a complete frame shifts into the receive shift register, the data portion of the frame transfers to the SCI data register. The receive data register full flag, RDRF, in SCI status register 1 (SCISR1) becomes set, indicating that the received byte can be read. If the receive interrupt enable bit, RIE, in SCI control register 2 (SCICR2) is also set, the RDRF flag generates an RDRF interrupt request.

1.4.4.3 Data Sampling

The receiver samples the **Rx input** signal at the RT clock rate. The RT clock is an internal signal with a frequency 16 times the baud rate. To adjust for baud rate mismatch, the RT clock (see Figure 1-13) is re-synchronized:

- After every start bit
- After the receiver detects a data bit change from logic 1 to logic 0 (after the majority of data bit samples at RT8, RT9, and RT10 returns a valid logic 1 and the majority of the next RT8, RT9, and RT10 samples returns a valid logic 0)

To locate the start bit, data recovery logic does an asynchronous search for a logic 0 preceded by three logic 1s.When the falling edge of a possible start bit occurs, the RT clock begins to count to 16.

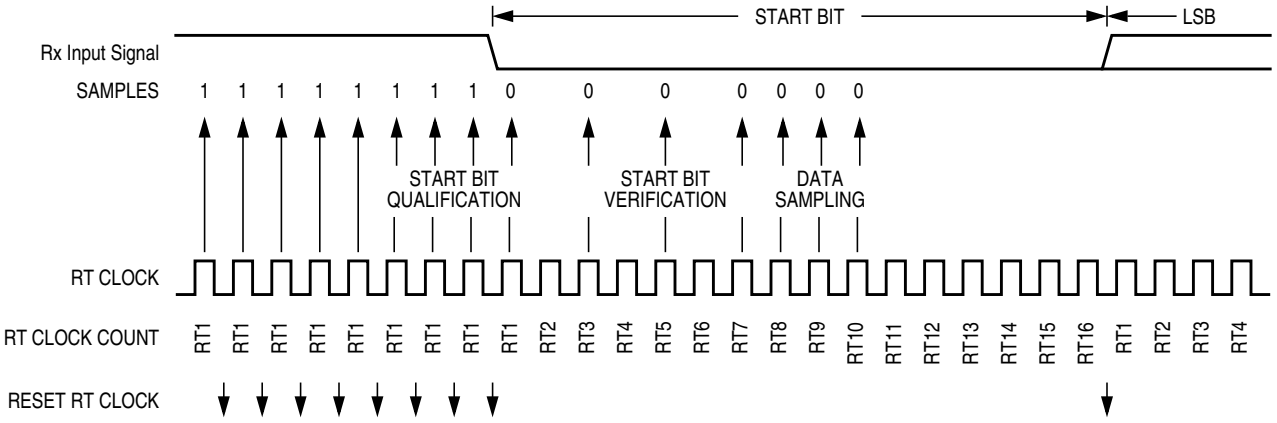


Figure 1-13. Receiver Data Sampling

To verify the start bit and to detect noise, data recovery logic takes samples at RT3, RT5, and RT7. Table 1-11 summarizes the results of the start bit verification samples.

Table 1-11. Start Bit Verification

RT3, RT5, and RT7 Samples	Start Bit Verification	Noise Flag
000	Yes	0
001	Yes	1
010	Yes	1
011	No	0
100	Yes	1
101	No	0
110	No	0
111	No	0

If start bit verification is not successful, the RT clock is reset and a new search for a start bit begins.

To determine the value of a data bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. Table 1-12 summarizes the results of the data bit samples.

Table 1-12. Data Bit Recovery

RT8, RT9, and RT10 Samples	Data Bit Determination	Noise Flag
000	0	0
001	0	1
010	0	1
011	1	1
100	0	1
101	1	1
110	1	1
111	1	0

NOTE

The RT8, RT9, and RT10 samples do not affect start bit verification. If any or all of the RT8, RT9, and RT10 start bit samples are logic 1s following a successful start bit verification, the noise flag (NF) is set and the receiver assumes that the bit is a start bit (logic 0).

To verify a stop bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. Table 1-13 summarizes the results of the stop bit samples.

Table 1-13. Stop Bit Recovery

RT8, RT9, and RT10 Samples	Framing Error Flag	Noise Flag
000	1	0
001	1	1
010	1	1
011	0	1
100	1	1
101	0	1
110	0	1
111	0	0

In Figure 1-14 the verification samples RT3 and RT5 determine that the first low detected was noise and not the beginning of a start bit. The RT clock is reset and the start bit search begins again. The noise flag is not set because the noise occurred before the start bit was found.

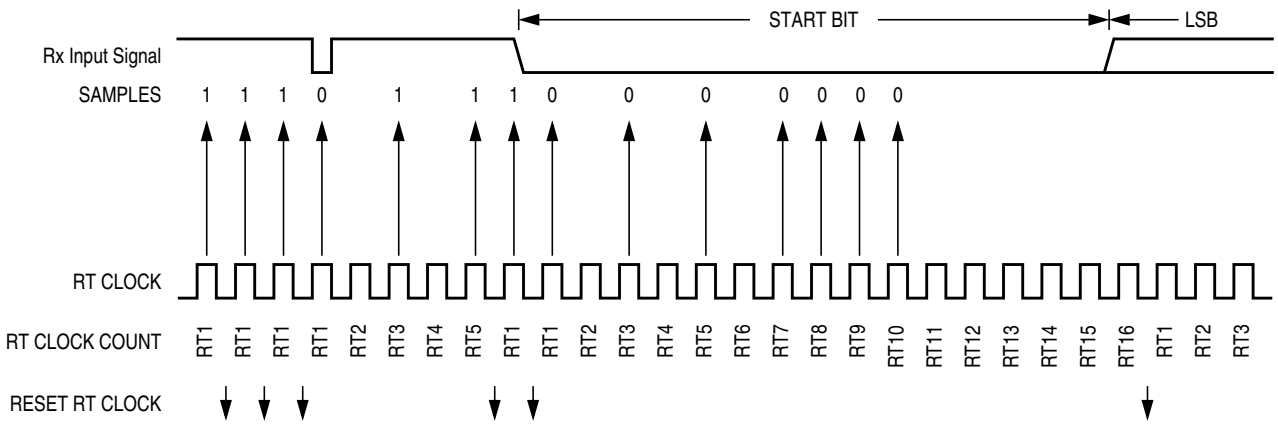


Figure 1-14. Start Bit Search Example 1

In Figure 1-15, verification sample at RT3 is high. The RT3 sample sets the noise flag. Although the perceived bit time is misaligned, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.

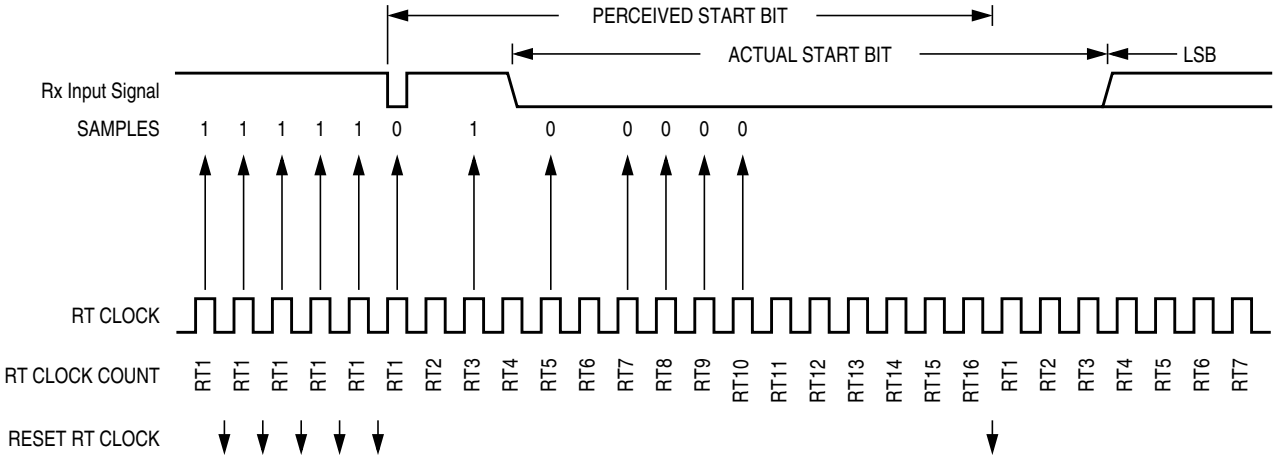


Figure 1-15. Start Bit Search Example 2

In Figure 1-16, a large burst of noise is perceived as the beginning of a start bit, although the test sample at RT5 is high. The RT5 sample sets the noise flag. Although this is a worst-case misalignment of perceived bit time, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.

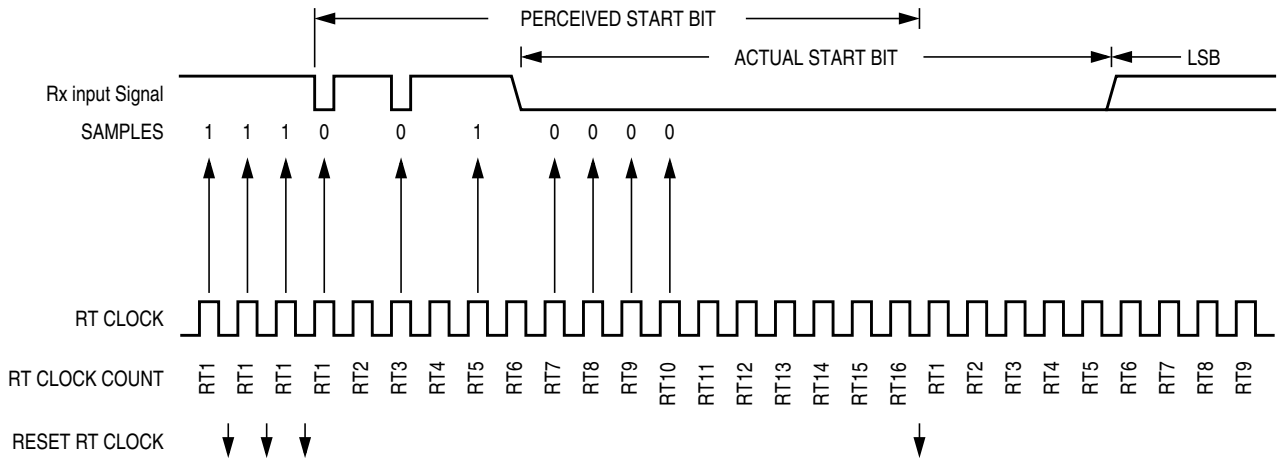


Figure 1-16. Start Bit Search Example 3

Figure 1-17 shows the effect of noise early in the start bit time. Although this noise does not affect proper synchronization with the start bit time, it does set the noise flag.

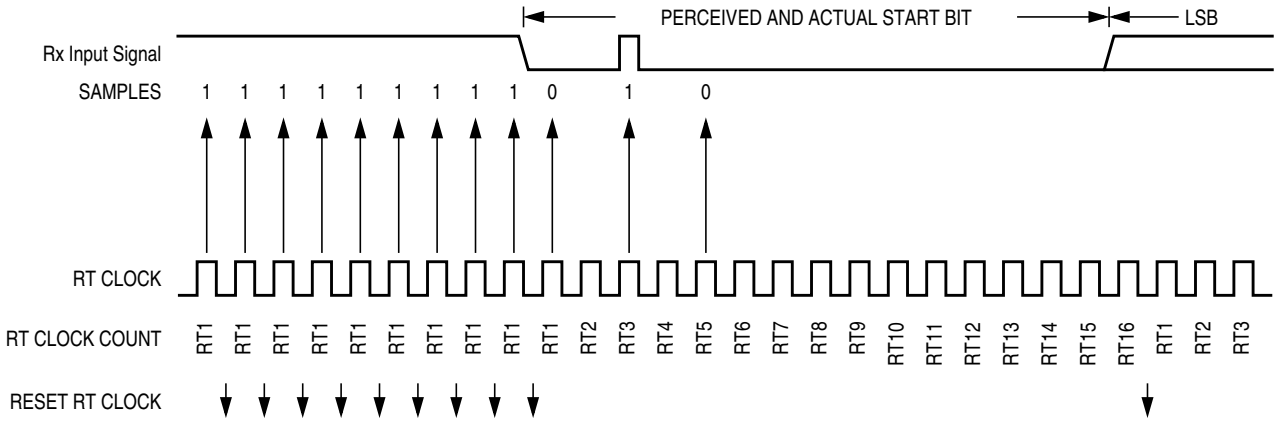


Figure 1-17. Start Bit Search Example 4

Figure 1-18 shows a burst of noise near the beginning of the start bit that resets the RT clock. The sample after the reset is low but is not preceded by three high samples that would qualify as a falling edge. Depending on the timing of the start bit search and on the data, the frame may be missed entirely or it may set the framing error flag.

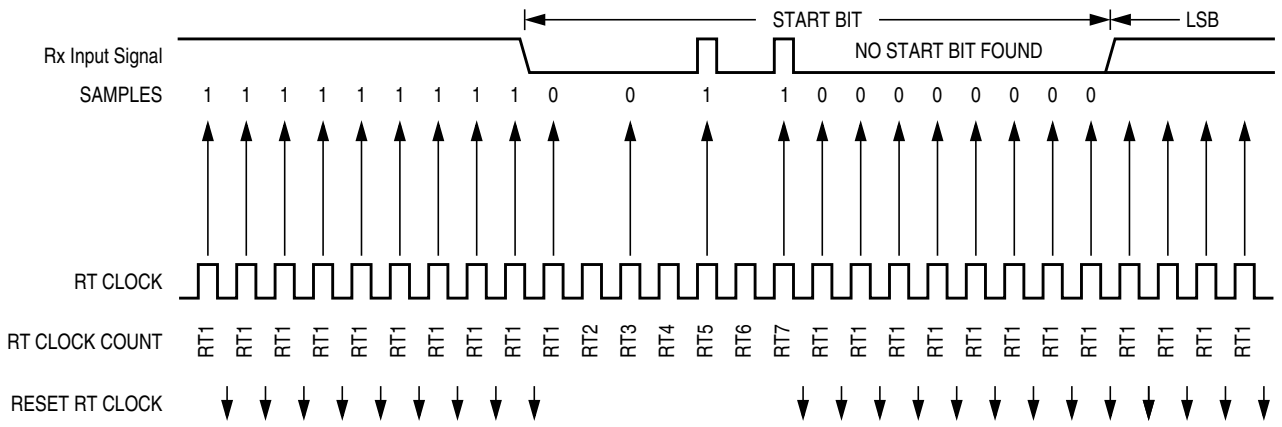


Figure 1-18. Start Bit Search Example 5

In Figure 1-19, a noise burst makes the majority of data samples RT8, RT9, and RT10 high. This sets the noise flag but does not reset the RT clock. In start bits only, the RT8, RT9, and RT10 data samples are ignored.

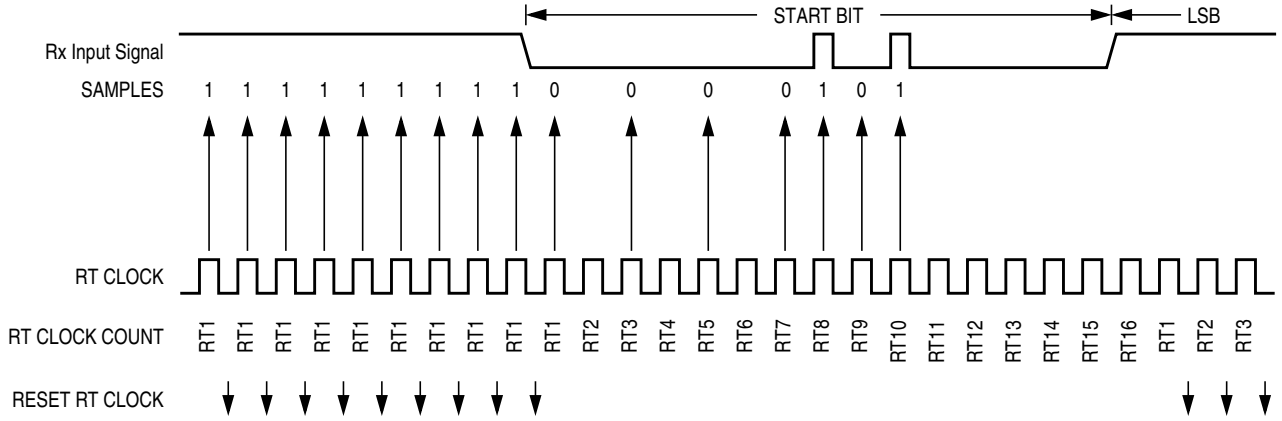


Figure 1-19. Start Bit Search Example 6

1.4.4.4 Framing Errors

If the data recovery logic does not detect a logic 1 where the stop bit should be in an incoming frame, it sets the framing error flag, FE, in SCI status register 1 (SCISR1). A break character also sets the FE flag because a break character has no stop bit. The FE flag is set at the same time that the RDRF flag is set.

1.4.4.5 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the actual stop bit.A noise error will occur if the RT8, RT9, and RT10 samples are not all the same logical values. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RT8, RT9, and RT10 stop bit samples are a logic zero.

As the receiver samples an incoming frame, it re-synchronizes the RT clock on any valid falling edge within the frame. Re synchronization within frames will correct a misalignment between transmitter bit times and receiver bit times.

1.4.4.5.1 Slow Data Tolerance

Figure 1-20 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

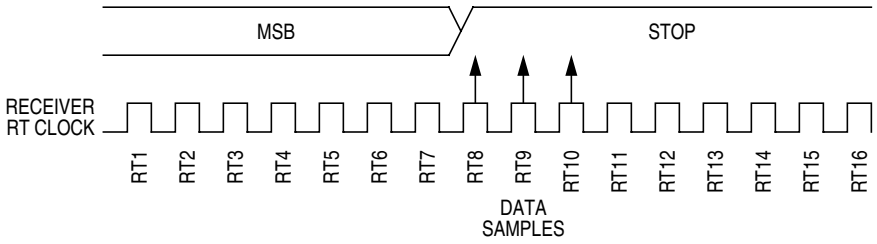


Figure 1-20. Slow Data

Let's take RTr as receiver RT clock and RTt as transmitter RT clock.

For an 8-bit data character, it takes the receiver 9 bit times x 16 RTr cycles +7 RTr cycles =151 RTr cycles to start data sampling of the stop bit.

With the misaligned character shown in Figure 1-20, the receiver counts 151 RTr cycles at the point when the count of the transmitting device is 9 bit times x 16 RTt cycles = 144 RTt cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is:

((151 – 144) / 151) x 100 = 4.63%

For a 9-bit data character, it takes the receiver 10 bit times x 16 RTr cycles + 7 RTr cycles = 167 RTr cycles to start data sampling of the stop bit.

With the misaligned character shown in Figure 1-20, the receiver counts 167 RTr cycles at the point when the count of the transmitting device is 10 bit times x 16 RTt cycles = 160 RTt cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is:

((167 – 160) / 167) X 100 = 4.19%

1.4.4.5.2 Fast Data Tolerance

Figure 1-21 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but is still sampled at RT8, RT9, and RT10.

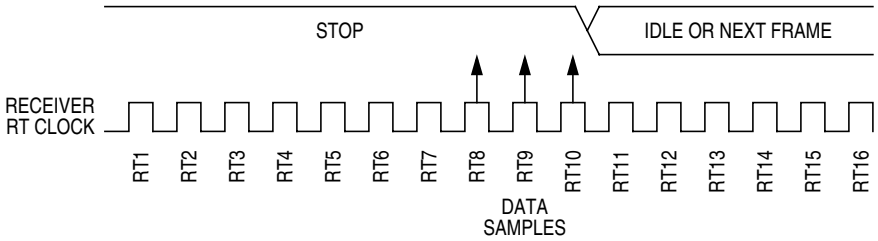


Figure 1-21. Fast Data

For an 8-bit data character, it takes the receiver 9 bit times x 16 RTr cycles + 10 RTr cycles = 154 RTr cycles to finish data sampling of the stop bit.

With the misaligned character shown in Figure 1-21, the receiver counts 154 RTr cycles at the point when the count of the transmitting device is 10 bit times x 16 RTt cycles = 160 RTt cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is:

((160 – 154) / 160) x 100 = 3.75%

For a 9-bit data character, it takes the receiver 10 bit times x 16 RTr cycles + 10 RTr cycles = 170 RTr cycles to finish data sampling of the stop bit.

With the misaligned character shown in Figure 1-21, the receiver counts 170 RTr cycles at the point when the count of the transmitting device is 11 bit times x 16 RTt cycles = 176 RTt cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is:

((176 – 170) / 176) x 100 = 3.40%

1.4.4.6 Receiver Wakeup

To enable the SCI to ignore transmissions intended only for other receivers in multiple-receiver systems, the receiver can be put into a standby state. Setting the receiver wakeup bit, RWU, in SCI control register 2 (SCICR2) puts the receiver into standby state during which receiver interrupts are disabled.The SCI will still load the receive data into the SCIDRH/L registers, but it will not set the RDRF flag.

The transmitting device can address messages to selected receivers by including addressing information in the initial frame or frames of each message.

The WAKE bit in SCI control register 1 (SCICR1) determines how the SCI is brought out of the standby state to process an incoming message. The WAKE bit enables either idle line wakeup or address mark wakeup.

1.4.4.6.1 Idle Input Line Wakeup (WAKE = 0)

In this wakeup method, an idle condition on the **Rx Input** signal clears the RWU bit and wakes up the SCI. The initial frame or frames of every message contain addressing information. All receivers evaluate the addressing information, and receivers for which the message is addressed process the frames that follow.

Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another idle character appears on the **Rx Input** signal.

Idle line wakeup requires that messages be separated by at least one idle character and that no message contains idle characters.

The idle character that wakes a receiver does not set the receiver idle bit, IDLE, or the receive data register full flag, RDRF.

The idle line type bit, ILT, determines whether the receiver begins counting logic 1s as idle character bits after the start bit or after the stop bit. ILT is in SCI control register 1 (SCICR1).

1.4.4.6.2 Address Mark Wakeup (WAKE = 1)

In this wakeup method, a logic 1 in the most significant bit (msb) position of a frame clears the RWU bit and wakes up the SCI. The logic 1 in the msb position marks a frame as an address frame that contains addressing information. All receivers evaluate the addressing information, and the receivers for which the message is addressed process the frames that follow.Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the **Rx Input** signal.

The logic 1 msb of an address frame clears the receiver's RWU bit before the stop bit is received and sets the RDRF flag.

Address mark wakeup allows messages to contain idle characters but requires that the msb be reserved for use in address frames.{sci_wake}

NOTE

With the WAKE bit clear, setting the RWU bit after the **Rx Input** signal has been idle can cause the receiver to wake up immediately.

1.4.5 Single-Wire Operation

Normally, the SCI uses two pins for transmitting and receiving. In single-wire operation, the RXD pin is disconnected from the SCI. The SCI uses the TXD pin for both receiving and transmitting.

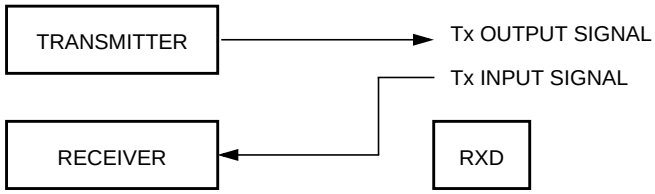


Figure 1-22. Single-Wire Operation (LOOPS = 1, RSRC = 1)

Enable single-wire operation by setting the LOOPS bit and the receiver source bit, RSRC, in SCI control register 1 (SCICR1). Setting the LOOPS bit disables the path from the **Rx Input** signal to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).The TXDIR bit (SCISR2[1]) determines whether the TXD pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation.

1.4.6 Loop Operation

In loop operation the transmitter output goes to the receiver input. The **Rx Input** signal is disconnected from the SCI

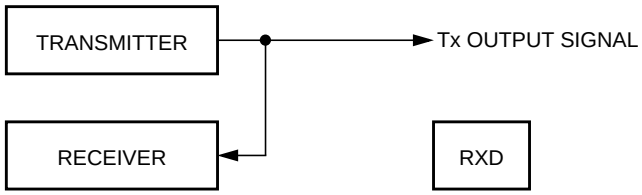


Figure 1-23. Loop Operation (LOOPS = 1, RSRC = 0)

Enable loop operation by setting the LOOPS bit and clearing the RSRC bit in SCI control register 1 (SCICR1). Setting the LOOPS bit disables the path from the **Rx Input** signal to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).

1.5 Initialization Information

1.5.1 Reset Initialization

The reset state of each individual bit is listed in Section 1.3, “Memory Map and Registers” which details the registers and their bit fields. All special functions or modes which are initialized during or just following reset are described within this section.

1.5.2 Interrupt Operation

1.5.2.1 System Level Interrupt Sources

There are five interrupt sources that can generate an SCI interrupt in to the CPU. They are listed in Table 1-14.

Table 1-14. SCI Interrupt Source

Interrupt Source	Flag	Local Enable
Transmitter	TDRE	TIE
Transmitter	TC	TCIE
Receiver	RDRF	RIE
	OR	
Receiver	IDLE	ILIE

1.5.2.2 Interrupt Descriptions

The SCI only originates interrupt requests. The following is a description of how the SCI makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt number are

chip dependent. The SCI only has a single interrupt line (**SCI Interrupt Signal**, active high operation) and all the following interrupts, when generated, are ORed together and issued through that port.

1.5.2.2.1 TDRE Description

The TDRE interrupt is set high by the SCI when the transmit shift register receives a byte from the SCI data register. A TDRE interrupt indicates that the transmit data register (SCIDRH/L) is empty and that a new byte can be written to the SCIDRH/L for transmission.Clear TDRE by reading SCI status register 1 with TDRE set and then writing to SCI data register low (SCIDRL).

1.5.2.2.2 TC Description

The TC interrupt is set by the SCI when a transmission has been completed.A TC interrupt indicates that there is no transmission in progress. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD pin becomes idle (logic 1). Clear TC by reading SCI status register 1 (SCISR1) with TC set and then writing to SCI data register low (SCIDRL).TC is cleared automatically when data, preamble, or break is queued and ready to be sent.

1.5.2.2.3 RDRF Description

The RDRF interrupt is set when the data in the receive shift register transfers to the SCI data register. A RDRF interrupt indicates that the received data has been transferred to the SCI data register and that the byte can now be read by the MCU. The RDRF interrupt is cleared by reading the SCI status register one (SCISR1) and then reading SCI data register low (SCIDRL).

1.5.2.2.4 OR Description

The OR interrupt is set when software fails to read the SCI data register before the receive shift register receives the next frame. The newly acquired data in the shift register will be lost in this case, but the data already in the SCI data registers is not affected. The OR interrupt is cleared by reading the SCI status register one (SCISR1) and then reading SCI data register low (SCIDRL).

1.5.2.3 IDLE Description

The IDLE interrupt is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M = 1) appear on the receiver input. Once the IDLE is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. Clear IDLE by reading SCI status register 1 (SCISR1) with IDLE set and then reading SCI data register low (SCIDRL).

1.5.3 Recovery from Wait Mode

The SCI interrupt request can be used to bring the CPU out of wait mode.

SPI
Block Guide
V03.06

Original Release Date: 21 JAN 2000
Revised: 04 FEB 2003

Motorola, Inc.

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
0.1	21 Jan 2000			This spec is based on the Barracuda, with modifications to change the module from 16 bit to 8 bit.
0.2	1 Mar 2000			Template of this document changed as per Version 2.0 SRS.
0.3	14 Jun 2000			- Signal names are changed as per the SRS2.0 - SPE bit remains set in the Mode Fault error case - Slave SPI does not support div2 and div4 cases
0.4	31 Aug 2000			- Electrical spec added - SPIF flag is cleared by a read access to the status register followed by read access to the data register.
0.5	13 Mar 2001	13 Mar 2001		- Incorporated feedback regarding format of the document.
0.6	13 Mar 2001	19 Mar 2001		- Incorporated changes as a result of internal discussions and clarification of SRS2
0.7	6 July 2001	6 July 2001		- Line is added with respect to SPTEF bit to make spec more clear. - Landscape pages have been removed from pdf. - Extra blank pages have been removed.
0.8	19 July 2001	19 July 2001		- Line is added with respect to SPE bit to make spec more clear.
V02.02	26 July 2001			-Added Document Names -variable definitions and Names have been hidden -Changed chapter 3.9 Errata to Note
V03.00	27 Sep 2001	27 Sep 2001		Based on the BUG version V02.02 an improved version was created. The specification counter has to be increased, because there is a difference in the behavior in SPI master mode from this specification to its predecessor. In SPI Master Mode, the change of a config bit during a transmission in progress, will abort the transmission and force the SPI into idle state.
V03.01	14 Dec 2001	14 Dec 2001		Section 4.4.2 - Changed description of transfer format CPHA=0 in slave mode Section 4.4.3 - Changed description of transfer format CPHA=1 in master mode - Changed Figure 4-3 Section 4.6.2 - Added note for mode fault in bidirectional master mode Section 4.7.1 - Changed description of bidirectional mode with mode fault Section 4.8.3 - Changed last sentence in stop mode description
V03.02	07 Jan 2002	07 Jan 2002		Section 3.3.4 - Changed description of SPTEF flag Section 4.1 - Changed description of SPTEF flag and SPIDR behaviour

Version Number	Revision Date	Effective Date	Author	Description of Changes
V03.03	09 Jan 2002	09 JAN 2002		Transferred document to SRS3.0 format
V03.04	18 Mar 2002	18 Mar 2002		Updated Document Format.
V03.05	03 Apr 2002	03 Apr 2002		Minor Document cleanup.
V03.06	04 Feb 2003	04 Feb 2003		Minor Document cleanup.

Table of Contents

Section 5 Initialization/Application Information

List of Figures

List of Tables

Preface

Terminology

Acronyms and Abbreviations	
SPI	Serial Parallel Interface
$\overline{SS}$	Slave Select
SCK	Serial Clock
MOSI	Master Output, Slave Input
MISO	Master Input, Slave Output
MOMI	Master Output, Master Input
SISO	Slave Input, Slave Output

Section 1 Introduction

Figure 1-1 gives an overview on the SPI architecture. The main parts of the SPI are status,control and data registers, shifter logic, baud rate generator, master/slave control logic and port control logic.

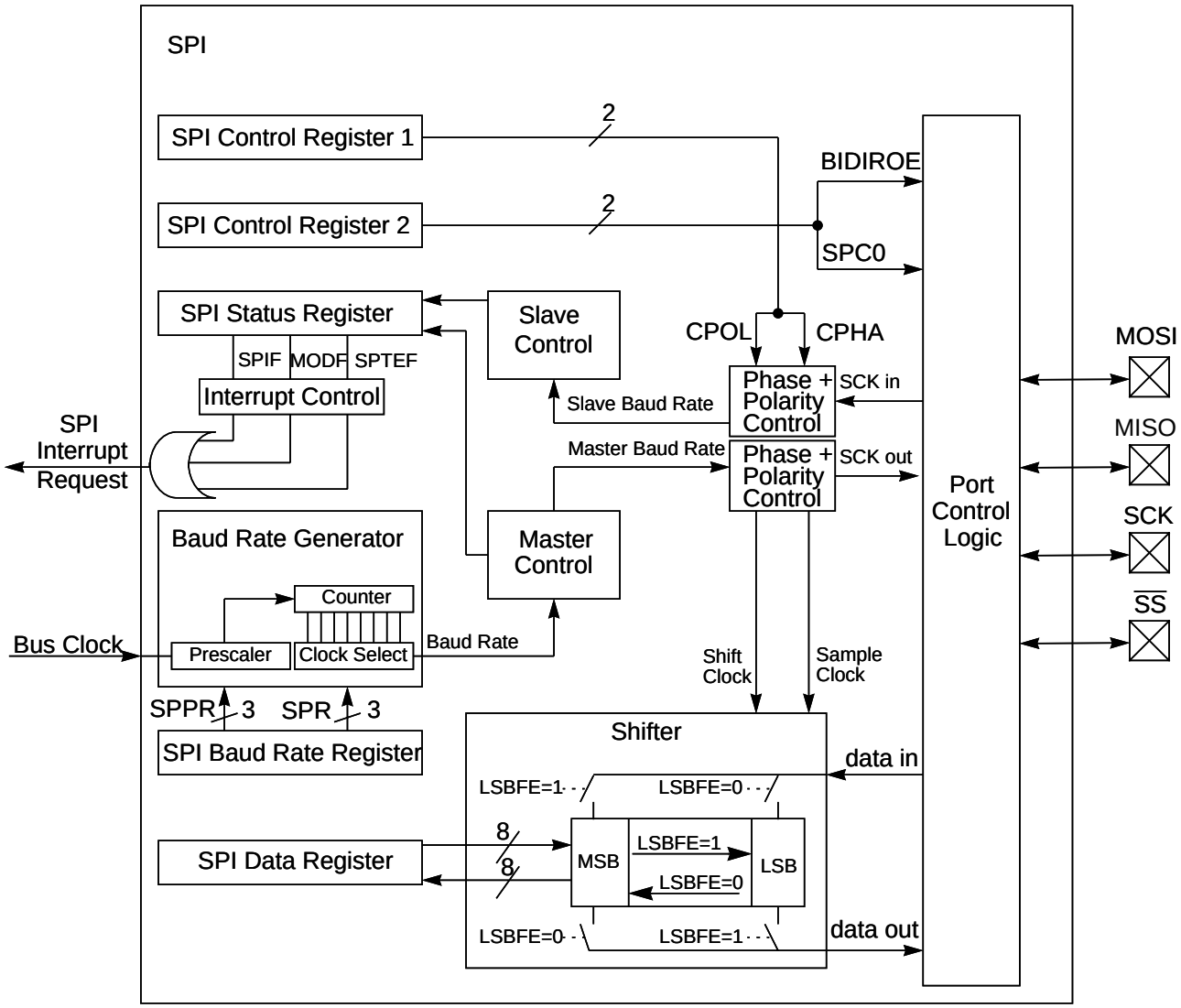


Figure 1-1 SPI Block Diagram

1.1 Overview

The SPI module allows a duplex, synchronous, serial communication between the MCU and peripheral devices. Software can poll the SPI status flags or the SPI operation can be interrupt driven.

1.2 Features

The SPI includes these distinctive features:

- Master mode and slave mode
- Bi-directional mode
- Slave select output
- Mode fault error flag with CPU interrupt capability
- Double-buffered data register
- Serial clock with programmable polarity and phase
- Control of SPI operation during wait mode

1.3 Modes of Operation

The SPI functions in three modes, run, wait, and stop.

- Run Mode

  This is the basic mode of operation.

- Wait Mode

  SPI operation in wait mode is a configurable low power mode, controlled by the SPISWAI bit located in the SPICR2 register. In wait mode, if the SPISWAI bit is clear, the SPI operates like in Run Mode. If the SPISWAI bit is set, the SPI goes into a power conservative state, with the SPI clock generation turned off. If the SPI is configured as a master, any transmission in progress stops, but is resumed after CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a byte continues, so that the slave stays synchronized to the master.

- Stop Mode

  The SPI is inactive in stop mode for reduced power consumption. If the SPI is configured as a master, any transmission in progress stops, but is resumed after CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a byte continues, so that the slave stays synchronized to the master.

This is a high level description only, detailed descriptions of operating modes are contained in section **4.8 Low Power Mode Options**.

Section 2 External Signal Description

2.1 Overview

This section lists the name and description of all ports including inputs and outputs that do, or may, connect off chip. The SPI module has a total of 4 external pins.

2.2 Detailed Signal Description

2.2.1 MOSI

This pin is used to transmit data out of the SPI module when it is configured as a Master and receive data when it is configured as Slave.

2.2.2 MISO

This pin is used to transmit data out of the SPI module when it is configured as a Slave and receive data when it is configured as Master.

2.2.3 $\overline{SS}$

This pin is used to output the select signal from the SPI module to another peripheral with which a data transfer is to take place when its configured as a Masterand its used as an input to receive the slave select signal when the SPI is configured as Slave.

2.2.4 SCK

This pin is used to output the clock with respect to which the SPI transfers data or receive clock in case of Slave.

Section 3 Memory Map/Register Definition

This section provides a detailed description of address space and registers used by the SPI.

The memory map for the SPI is given below in **Table 3-1**. The address listed for each register is the sum of a base address and an address offset. The base address is defined at the SoC level and the address offset is defined at the module level. Reads from the reserved bits return zeros and writes to the reserved bits have no effect.

Table 3-1 Module Memory Map

Address	Use	Access
$___0	SPI Control Register 1 (SPICR1)	Read / Write
$___1	SPI Control Register 2 (SPICR2)	Read / Write [1]
$___2	SPI Baud Rate Register (SPIBR)	Read / Write [1]

Table 3-1 Module Memory Map

$___3	SPI Status Register (SPISR)	Read [2]
$___4	Reserved	__ [2 3]
$___5	SPI Data Register (SPIDR)	Read / Write
$___6	Reserved	__ [2 3]
$___7	Reserved	__ [2 3]

NOTES:
1. Certain bits are non-writable.
2. Writes to this register are ignored.
3. Reading from this register returns all zeros.

3.1 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

3.1.1 SPI Control Register 1

Register Address: $___0

	Bit 7	6	5	4	3	2	1	Bit 0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset:	0	0	0	0	0	1	0	0

Figure 3-1 SPI Control Register 1 (SPICR1)

Read: anytime
Write: anytime

SPIE — SPI Interrupt Enable Bit

This bit enables SPI interrupt requests, if SPIF or MODF status flag is set.
1 = SPI interrupts enabled.
0 = SPI interrupts disabled.

SPE — SPI System Enable Bit

This bit enables the SPI system and dedicates the SPI port pins to SPI system functions. If SPE is cleared, SPI is disabled and forced into idle state, status bits in SPISR register are reseted
1 = SPI enabled, port pins are dedicated to SPI functions.
0 = SPI disabled (lower power consumption).

SPTIE — SPI Transmit Interrupt Enable

This bit enables SPI interrupt requests, if SPTEF flag is set.
1 = SPTEF interrupt enabled.

0 = SPTEF interrupt disabled.

MSTR — SPI Master/Slave Mode Select Bit

This bit selects, if the SPI operates in master or slave mode. Switching the SPI from master to slave or vice versa forces the SPI system into idle state.

1 = SPI is in Master mode
0 = SPI is in Slave mode

CPOL — SPI Clock Polarity Bit

This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.

1 = Active-low clocks selected. In idle state SCK is high.
0 = Active-high clocks selected. In idle state SCK is low.

CPHA — SPI Clock Phase Bit

This bit is used to select the SPI clock format. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.

1 = Sampling of data occurs at even edges (2,4,6,...,16) of the SCK clock
0 = Sampling of data occurs at odd edges (1,3,5,...,15) of the SCK clock

SSOE — Slave Select Output Enable

The $\overline{SS}$ output feature is enabled only in master mode, if MODFEN is set, by asserting the SSOE as shown in **Table 3-2**. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.

Table 3-2 $\overline{SS}$ Input / Output Selection

MOD FEN	SSOE	Master Mode	Slave Mode
0	0	$\overline{SS}$ not used by SPI	$\overline{SS}$ input
0	1	$\overline{SS}$ not used by SPI	$\overline{SS}$ input
1	0	$\overline{SS}$ input with MODF feature	$\overline{SS}$ input
1	1	$\overline{SS}$ is slave select output	$\overline{SS}$ input

LSBFE — LSB-First Enable

This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.

1 = Data is transferred least significant bit first.
0 = Data is transferred most significant bit first.

3.1.2 SPI Control Register 2

Register Address: $___1

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W								
Reset:	0	0	0	0	0	0	0	0

= Reserved

Figure 3-2 SPI Control Register 2 (SPICR2)

Read: anytime
Write: anytime; writes to the reserved bits have no effect

MODFEN — Mode Fault Enable Bit

This bit allows the MODF failure being detected. If the SPI is in Master mode and MODFEN is cleared, then the $\overline{SS}$ port pin is not used by the SPI. In Slave mode, the $\overline{SS}$ is available only as an input regardless of the value of MODFEN. For an overview on the impact of the MODFEN bit on the $\overline{SS}$ port pin configuration refer to **Table 3-2**. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.

1 = $\overline{SS}$ port pin with MODF feature
0 = $\overline{SS}$ port pin is not used by the SPI

BIDIROE — Output enable in the Bidirectional mode of operation

This bit controls the MOSI and MISO output buffer of the SPI, when in bidirectional mode of operation (SPC0 is set). In master mode this bit controls the output buffer of the MOSI port, in slave mode it controls the output buffer of the MISO port. In master mode, with SPC0 set, a change of this bit will abort a transmission in progress and force the SPI into idle state.

1 = Output buffer enabled
0 = Output buffer disabled

SPISWAI — SPI Stop in Wait Mode Bit

This bit is used for power conservation while in wait mode.

1 = Stop SPI clock generation when in wait mode
0 = SPI clock operates normally in wait mode

SPC0 — Serial Pin Control Bit 0

This bit enables bidirectional pin configurations as shown in **Table 3-3**. In master mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state

Table 3-3 Bidirectional Pin Configurations

Pin Mode	SPC0	BIDIROE	MISO	MOSI
Master Mode of Operation				

Table 3-3 Bidirectional Pin Configurations

Pin Mode	SPC0	BIDIROE	MISO	MOSI
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
Slave Mode of Operation				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

3.1.3 SPI Baud Rate Register

Register Address: $___2

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W								
Reset:	0	0	0	0	0	0	0	0

= Reserved

Figure 3-3 SPI Baud Rate Register (SPIBR)

Read: anytime
Write: anytime; writes to the reserved bits have no effect

SPPR2–SPPR0 — SPI Baud Rate Preselection Bits

SPR2–SPR0 — SPI Baud Rate Selection Bits

These bits specify the SPI baud rates as shown in the table below. In master mode, a change of these bits will abort a transmission in progress and force the SPI system into idle state.

The baud rate divisor equation is as follows:

$$BaudRateDivisor = (SPPR + 1) \bullet 2^{(SPR + 1)}$$

The baud rate can be calculated with the following equation:

$$Baud\ Rate = BusClock / BaudRateDivisor$$

Table 3-4 Example SPI Baud Rate Selection (25 MHz Bus Clock)

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRate Divisor	Baud Rate
0	0	0	0	0	0	2	12.5 MHz

Table 3-4 Example SPI Baud Rate Selection (25 MHz Bus Clock)

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRate Divisor	Baud Rate
0	0	0	0	0	1	4	6.25 MHz
0	0	0	0	1	0	8	3.125 MHz
0	0	0	0	1	1	16	1.5625 MHz
0	0	0	1	0	0	32	781.25 kHz
0	0	0	1	0	1	64	390.63 kHz
0	0	0	1	1	0	128	195.31 kHz
0	0	0	1	1	1	256	97.66 kHz
0	0	1	0	0	0	4	6.25 MHz
0	0	1	0	0	1	8	3.125 MHz
0	0	1	0	1	0	16	1.5625 MHz
0	0	1	0	1	1	32	781.25 kHz
0	0	1	1	0	0	64	390.63 kHz
0	0	1	1	0	1	128	195.31 kHz
0	0	1	1	1	0	256	97.66 kHz
0	0	1	1	1	1	512	48.83 kHz
0	1	0	0	0	0	6	4.16667 MHz
0	1	0	0	0	1	12	2.08333 MHz
0	1	0	0	1	0	24	1.04167 MHz
0	1	0	0	1	1	48	520.83 kHz
0	1	0	1	0	0	96	260.42 kHz
0	1	0	1	0	1	192	130.21 kHz
0	1	0	1	1	0	384	65.10 kHz
0	1	0	1	1	1	768	32.55 kHz
0	1	1	0	0	0	8	3.125 MHz
0	1	1	0	0	1	16	1.5625 MHz
0	1	1	0	1	0	32	781.25 kHz
0	1	1	0	1	1	64	390.63 kHz
0	1	1	1	0	0	128	195.31 kHz
0	1	1	1	0	1	256	97.66 kHz
0	1	1	1	1	0	512	48.83 kHz
0	1	1	1	1	1	1024	24.41 kHz
1	0	0	0	0	0	10	2.5 MHz
1	0	0	0	0	1	20	1.25 MHz
1	0	0	0	1	0	40	625 kHz
1	0	0	0	1	1	80	312.5 kHz
1	0	0	1	0	0	160	156.25 kHz
1	0	0	1	0	1	320	78.13 kHz
1	0	0	1	1	0	640	39.06 kHz
1	0	0	1	1	1	1280	19.53 kHz
1	0	1	0	0	0	12	2.08333 MHz
1	0	1	0	0	1	24	1.04167 MHz
1	0	1	0	1	0	48	520.83 kHz
1	0	1	0	1	1	96	260.42 kHz

Table 3-4 Example SPI Baud Rate Selection (25 MHz Bus Clock)

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRate Divisor	Baud Rate
1	0	1	1	0	0	192	130.21 kHz
1	0	1	1	0	1	384	65.10 kHz
1	0	1	1	1	0	768	32.55 kHz
1	0	1	1	1	1	1536	16.28 kHz
1	1	0	0	0	0	14	1.78571 MHz
1	1	0	0	0	1	28	892.86 kHz
1	1	0	0	1	0	56	446.43 kHz
1	1	0	0	1	1	112	223.21 kHz
1	1	0	1	0	0	224	111.61 kHz
1	1	0	1	0	1	448	55.80 kHz
1	1	0	1	1	0	896	27.90 kHz
1	1	0	1	1	1	1792	13.95 kHz
1	1	1	0	0	0	16	1.5625 MHz
1	1	1	0	0	1	32	781.25 kHz
1	1	1	0	1	0	64	390.63 kHz
1	1	1	0	1	1	128	195.31 kHz
1	1	1	1	0	0	256	97.66 kHz
1	1	1	1	0	1	512	48.83 kHz
1	1	1	1	1	0	1024	24.41 kHz
1	1	1	1	1	1	2048	12.21 kHz

NOTE: ***In slave mode of SPI S-clock speed DIV2 is not supported.***

3.1.4 SPI Status Register

Register Address: $___3

	Bit 7	6	5	4	3	2	1	Bit 0
R	SPIF	0	SPTEF	MODF	0	0	0	0
W								
Reset:	0	0	1	0	0	0	0	0

= Reserved

Figure 3-4 SPI Status Register (SPISR)

Read: anytime
Write: has no effect

SPIF — SPIF Interrupt Flag

This bit is set after a received data byte has been transferred into the SPI Data Register. This bit is cleared by reading the SPISR register (with SPIF set) followed by a read access to the SPI Data Register.

1 = New data copied to SPIDR
0 = Transfer not yet complete

SPTEF — SPI Transmit Empty Interrupt Flag

If set, this bit indicates that the transmit data register is empty. To clear this bit and place data into the transmit data register, SPISR has to be read with SPTEF=1, followed by a write to SPIDR. Any write to the SPI Data Register without reading SPTEF=1, is effectively ignored.

1 = SPI Data register empty
0 = SPI Data register not empty

MODF — Mode Fault Flag

This bit is set if the $\overline{SS}$ input becomes low while the SPI is configured as a master and mode fault detection is enabled, MODFEN bit of SPICR2 register is set. Refer to MODFEN bit description in **3.1.2 SPI Control Register 2**. The flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to the SPI Control Register 1.

1 = Mode fault has occurred.
0 = Mode fault has not occurred.

3.1.5 SPI Data Register

Register Address: $___5

	Bit 7	6	5	4	3	2	1	Bit 0
R W	Bit 7	6	5	4	3	2	2	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 3-5 SPI Data Register (SPIDR)

Read: anytime; normally read only after SPIF is set
Write: anytime

The SPI Data Register is both the input and output register for SPI data. A write to this register allows a data byte to be queued and transmitted. For a SPI configured as a master, a queued data byte is transmitted immediately after the previous transmission has completed. The SPI Transmitter Empty Flag SPTEF in the SPISR register indicates when the SPI Data Register is ready to accept new data.

Reading the data can occur anytime from after the SPIF is set to before the end of the next transfer. If the SPIF is not serviced by the end of the successive transfers, those data bytes are lost and the data within the SPIDR retains the first byte until SPIF is serviced.

Section 4 Functional Description

4.1 General

The SPI module allows a duplex, synchronous, serial communication between the MCU and peripheral devices. Software can poll the SPI status flags or SPI operation can be interrupt driven.

The SPI system is enabled by setting the SPI enable (SPE) bit in SPI Control Register 1. While SPE bit is set, the four associated SPI port pins are dedicated to the SPI function as:

- Slave select ($\overline{SS}$)
- Serial clock (SCK)
- Master out/slave in (MOSI)
- Master in/slave out (MISO)

The main element of the SPI system is the SPI Data Register. The 8-bit data register in the master and the 8-bit data register in the slave are linked by the MOSI and MISO pins to form a distributed 16-bit register. When a data transfer operation is performed, this 16-bit register is serially shifted eight bit positions by the S-clock from the master, so data is exchanged between the master and the slave. Data written to the master SPI Data Register becomes the output data for the slave, and data read from the master SPI Data Register after a transfer operation is the input data from the slave.

A read of SPISR with SPTEF=1 followed by a write to SPIDR puts data into the transmit data register. When a transfer is complete, received data is moved into the receive data register. Data may be read from this double-buffered system any time before the next transfer has completed. This 8-bit data register acts as the SPI receive data register for reads and as the SPI transmit data register for writes. A single SPI register address is used for reading data from the read data buffer and for writing data to the transmit data register.

The clock phase control bit (CPHA) and a clock polarity control bit (CPOL) in the SPI Control Register 1 (SPICR1) select one of four possible clock formats to be used by the SPI system. The CPOL bit simply selects a non-inverted or inverted clock. The CPHA bit is used to accommodate two fundamentally different protocols by sampling data on odd numbered SCK edges or on even numbered SCK edges (see **4.4 Transmission Formats**).

The SPI can be configured to operate as a master or as a slave. When the MSTR bit in SPI Control Register1 is set, master mode is selected, when the MSTR bit is clear, slave mode is selected.

4.2 Master Mode

The SPI operates in master mode when the MSTR bit is set. Only a master SPI module can initiate transmissions. A transmission begins by writing to the master SPI Data Register. If the shift register is empty, the byte immediately transfers to the shift register. The byte begins shifting out on the MOSI pin under the control of the serial clock.

- S-clock

The SPR2, SPR1, and SPR0 baud rate selection bits in conjunction with the SPPR2, SPPR1, and SPPR0 baud rate preselection bits in the SPI Baud Rate register control the baud rate generator and determine the speed of the transmission. The SCK pin is the SPI clock output. Through the SCK pin, the baud rate generator of the master controls the shift register of the slave peripheral.

- MOSI, MISO pin

In master mode, the function of the serial data output pin (MOSI) and the serial data input pin (MISO) is determined by the SPC0 and BIDIROE control bits.

- $\overline{SS}$ pin

If MODFEN and SSOE bit are set, the $\overline{SS}$ pin is configured as slave select output. The $\overline{SS}$ output becomes low during each transmission and is high when the SPI is in idle state.

If MODFEN is set and SSOE is cleared, the $\overline{SS}$ pin is configured as input for detecting mode fault error. If the $\overline{SS}$ input becomes low this indicates a mode fault error where another master tries to drive the MOSI and SCK lines. In this case, the SPI immediately switches to slave mode, by clearing the MSTR bit and also disables the slave output buffer MISO (or SISO in bidirectional mode). So the result is that all outputs are disabled and SCK, MOSI and MISO are inputs. If a transmission is in progress when the mode fault occurs, the transmission is aborted and the SPI is forced into idle state.

This mode fault error also sets the mode fault (MODF) flag in the SPI Status Register (SPISR). If the SPI interrupt enable bit (SPIE) is set when the MODF flag gets set, then an SPI interrupt sequence is also requested.

When a write to the SPI Data Register in the master occurs, there is a half SCK-cycle delay. After the delay, SCK is started within the master. The rest of the transfer operation differs slightly, depending on the clock format specified by the SPI clock phase bit, CPHA, in SPI Control Register 1 (see **4.4 Transmission Formats**).

NOTE: *A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0, BIDIROE with SPC0 set, SPPR2-SPPR0 and SPR2-SPR0 in master mode will abort a transmission in progress and force the SPI into idle state. The remote slave cannot detect this, therefore the master has to ensure that the remote slave is set back to idle state.*

4.3 Slave Mode

The SPI operates in slave mode when the MSTR bit in SPI Control Register1 is clear.

- SCK clock

In slave mode, SCK is the SPI clock input from the master.

- MISO, MOSI pin

In slave mode, the function of the serial data output pin (MISO) and serial data input pin (MOSI) is determined by the SPC0 bit and BIDIROE bit in SPI Control Register 2.

- $\overline{SS}$ pin

The $\overline{SS}$ pin is the slave select input. Before a data transmission occurs, the $\overline{SS}$ pin of the slave SPI must be low. $\overline{SS}$ must remain low until the transmission is complete. If $\overline{SS}$ goes high, the SPI is forced into idle state.

The $\overline{SS}$ input also controls the serial data output pin, if $\overline{SS}$ is high (not selected), the serial data output pin is high impedance, and, if $\overline{SS}$ is low the first bit in the SPI Data Register is driven out of the serial data output pin. Also, if the slave is not selected ($\overline{SS}$ is high), then the SCK input is ignored and no internal shifting of the SPI shift register takes place.

Although the SPI is capable of duplex operation, some SPI peripherals are capable of only receiving SPI data in a slave mode. For these simpler devices, there is no serial data out pin.

NOTE: *When peripherals with duplex capability are used, take care not to simultaneously enable two receivers whose serial outputs drive the same system slave's serial data output line.*

As long as no more than one slave device drives the system slave's serial data output line, it is possible for several slaves to receive the same transmission from a master, although the master would not receive return information from all of the receiving slaves.

If the CPHA bit in SPI Control Register 1 is clear, odd numbered edges on the SCK input cause the data at the serial data input pin to be latched. Even numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

If the CPHA bit is set, even numbered edges on the SCK input cause the data at the serial data input pin to be latched. Odd numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

When CPHA is set, the first edge is used to get the first data bit onto the serial data output pin. When CPHA is clear and the $\overline{SS}$ input is low (slave selected), the first bit of the SPI data is driven out of the serial data output pin. After the eighth shift, the transfer is considered complete and the received data is transferred into the SPI Data Register. To indicate transfer is complete, the SPIF flag in the SPI Status Register is set.

NOTE: *A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0 and BIDIROE with SPC0 set in slave mode will corrupt a transmission in progress and has to be avoided.*

4.4 Transmission Formats

During an SPI transmission, data is transmitted (shifted out serially) and received (shifted in serially) simultaneously. The serial clock (SCK) synchronizes shifting and sampling of the information on the two serial data lines. A slave select line allows selection of an individual slave SPI device, slave devices that are not selected do not interfere with SPI bus activities. Optionally, on a master SPI device, the slave select line can be used to indicate multiple-master bus contention.

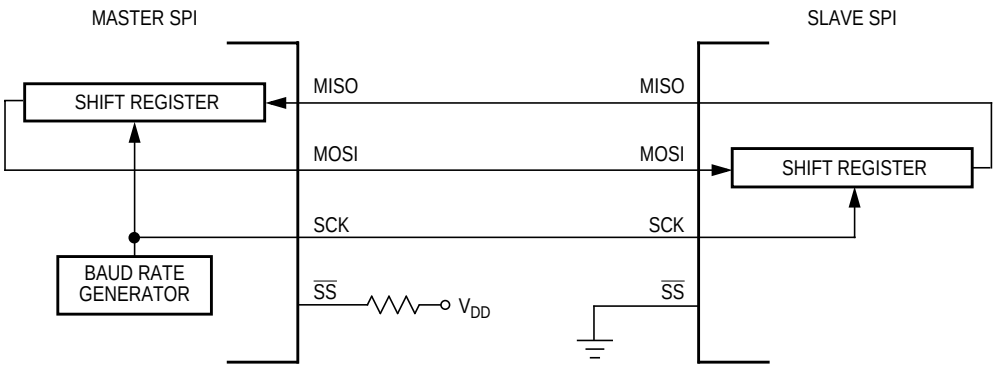


Figure 4-1 Master/Slave Transfer Block Diagram

4.4.1 Clock Phase and Polarity Controls

Using two bits in the SPI Control Register1, software selects one of four combinations of serial clock phase and polarity.

The CPOL clock polarity control bit specifies an active high or low clock and has no significant effect on the transmission format.

The CPHA clock phase control bit selects one of two fundamentally different transmission formats.

Clock phase and polarity should be identical for the master SPI device and the communicating slave device. In some cases, the phase and polarity are changed between transmissions to allow a master device to communicate with peripheral slaves having different requirements.

4.4.2 CPHA = 0 Transfer Format

The first edge on the SCK line is used to clock the first data bit of the slave into the master and the first data bit of the master into the slave. In some peripherals, the first bit of the slave's data is available at the slave's data out pin as soon as the slave is selected. In this format, the first SCK edge is issued a half cycle after $\overline{SS}$ has become low.

A half SCK cycle later, the second edge appears on the SCK line. When this second edge occurs, the value previously latched from the serial data input pin is shifted into the LSB or MSB of the shift register, depending on LSBFE bit.

After this second edge, the next bit of the SPI master data is transmitted out of the serial data output pin of the master to the serial input pin on the slave. This process continues for a total of 16 edges on the SCK line, with data being latched on odd numbered edges and shifted on even numbered edges.

Data reception is double buffered. Data is shifted serially into the SPI shift register during the transfer and is transferred to the parallel SPI Data Register after the last bit is shifted in.

After the 16th (last) SCK edge:

- Data that was previously in the master SPI Data Register should now be in the slave data register and the data that was in the slave data register should be in the master.
- The SPIF flag in the SPI Status Register is set indicating that the transfer is complete.

Figure 4-2 is a timing diagram of an SPI transfer where CPHA = 0. SCK waveforms are shown for CPOL = 0 and CPOL = 1. The diagram may be interpreted as a master or slave timing diagram since the SCK, MISO, and MOSI pins are connected directly between the master and the slave. The MISO signal is the output from the slave and the MOSI signal is the output from the master. The $\overline{SS}$ pin of the master must be either high or reconfigured as a general-purpose output not affecting the SPI.

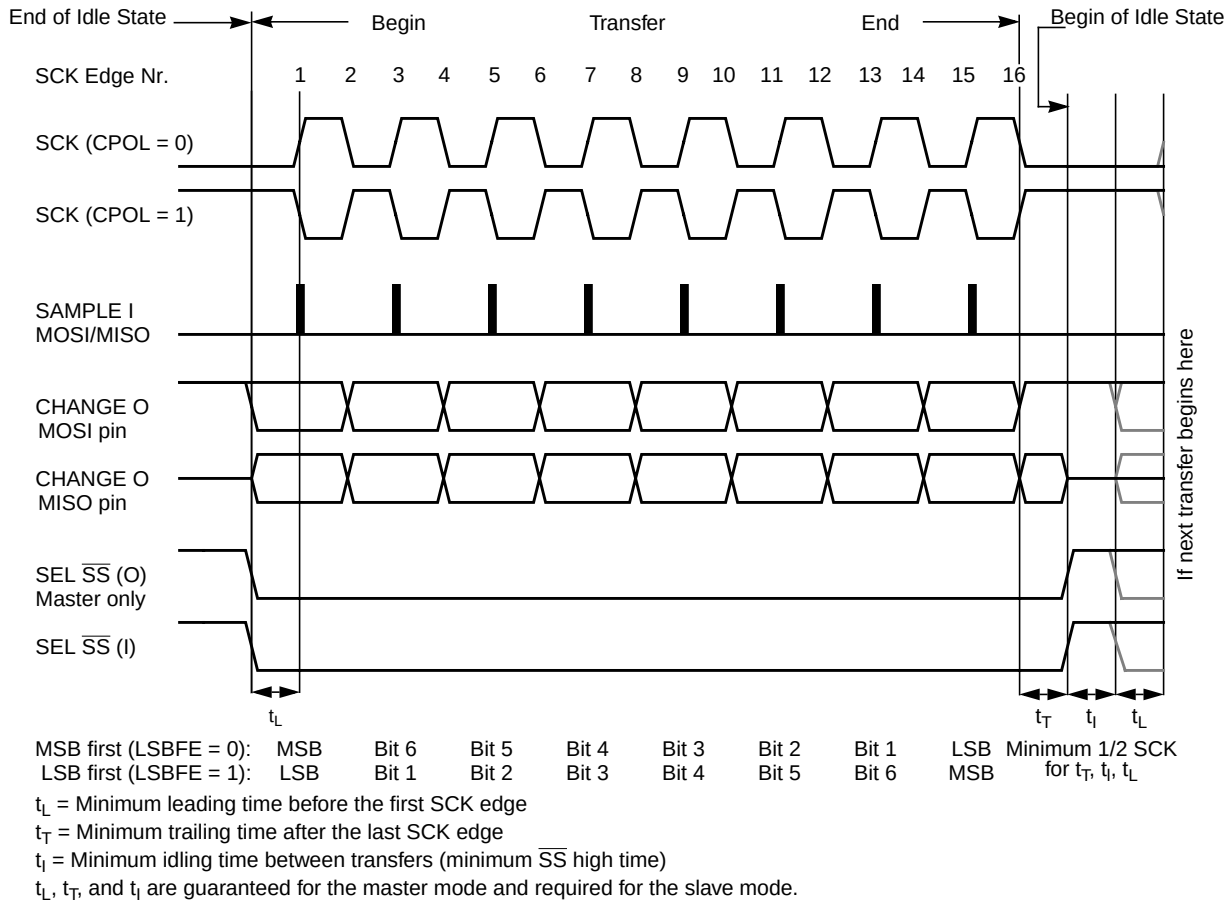


Figure 4-2 SPI Clock Format 0 (CPHA = 0)

In slave mode, if the $\overline{SS}$ line is not deasserted between the successive transmissions then the content of the SPI Data Register is not transmitted, instead the last received byte is transmitted. If the $\overline{SS}$ line is deasserted for at least minimum idle time (half SCK cycle) between successive transmissions then the content of the SPI Data Register is transmitted.

In master mode, with slave select output enabled the $\overline{SS}$ line is always deasserted and reasserted between successive transfers for at least minimum idle time.

4.4.3 CPHA = 1 Transfer Format

Some peripherals require the first SCK edge before the first data bit becomes available at the data out pin, the second edge clocks data into the system. In this format, the first SCK edge is issued by setting the CPHA bit at the beginning of the 8-cycle transfer operation.

The first edge of SCK occurs immediately after the half SCK clock cycle synchronization delay. This first edge commands the slave to transfer its first data bit to the serial data input pin of the master.

A half SCK cycle later, the second edge appears on the SCK pin. This is the latching edge for both the master and slave.

When the third edge occurs, the value previously latched from the serial data input pin is shifted into the LSB or MSB of the SPI shift register, depending on LSBFE bit. After this edge, the next bit of the master data is coupled out of the serial data output pin of the master to the serial input pin on the slave.

This process continues for a total of 16 edges on the SCK line with data being latched on even numbered edges and shifting taking place on odd numbered edges.

Data reception is double buffered, data is serially shifted into the SPI shift register during the transfer and is transferred to the parallel SPI Data Register after the last bit is shifted in.

After the 16th SCK edge:

- Data that was previously in the SPI Data Register of the master is now in the data register of the slave, and data that was in the data register of the slave is in the master.
- The SPIF flag bit in SPISR is set indicating that the transfer is complete.

Figure 4-3 shows two clocking variations for CPHA = 1. The diagram may be interpreted as a master or slave timing diagram since the SCK, MISO, and MOSI pins are connected directly between the master and the slave. The MISO signal is the output from the slave, and the MOSI signal is the output from the master. The $\overline{SS}$ line is the slave select input to the slave. The $\overline{SS}$ pin of the master must be either high or reconfigured as a general-purpose output not affecting the SPI.

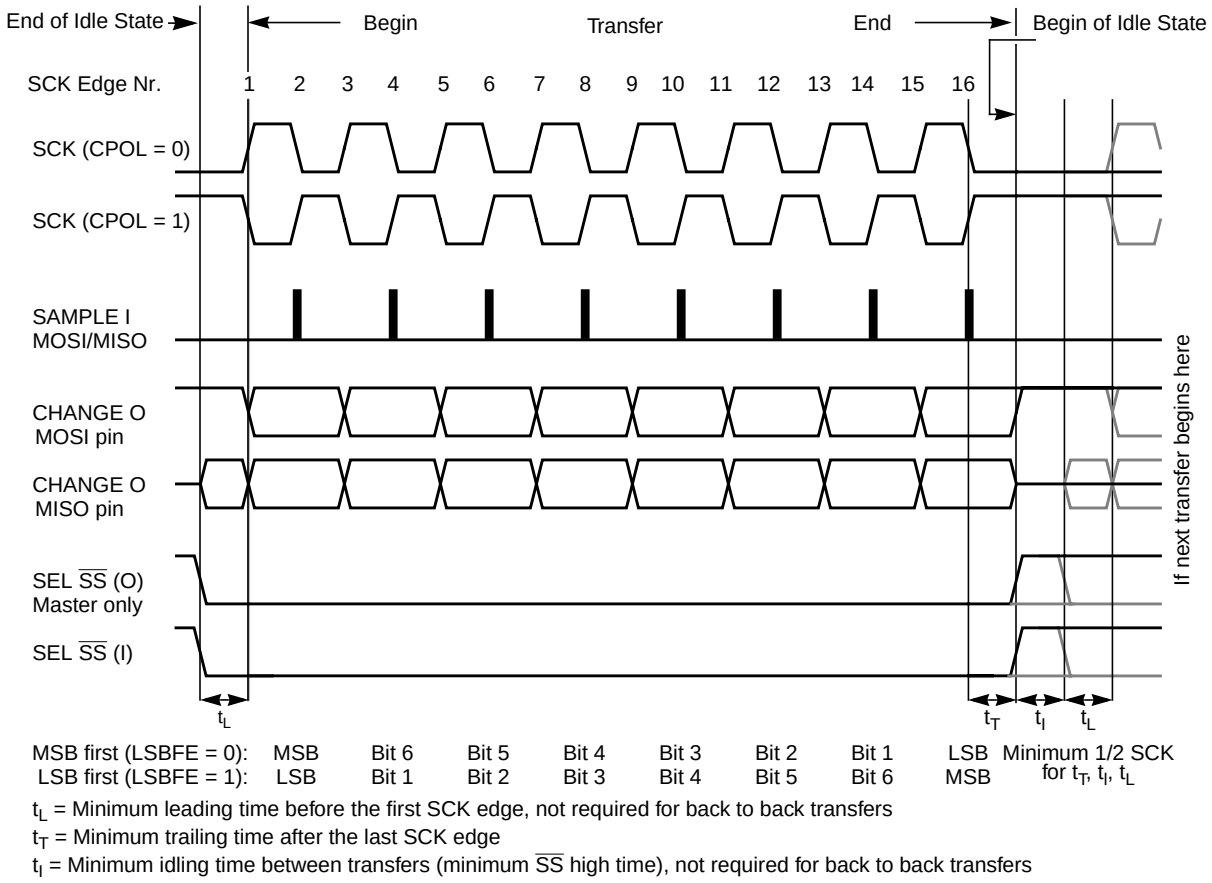


Figure 4-3 SPI Clock Format 1 (CPHA = 1)

The $\overline{SS}$ line can remain active low between successive transfers (can be tied low at all times). This format is sometimes preferred in systems having a single fixed master and a single slave that drive the MISO data line.

- Back to Back transfers in master mode

In master mode, if a transmission has completed and a new data byte is available in the SPI Data Register, this byte is send out immediately without a trailing and minimum idle time.

The SPI interrupt request flag (SPIF) is common to both the master and slave modes. SPIF gets set one half SCK cycle after the last SCK edge.

4.5 SPI Baud Rate Generation

Baud rate generation consists of a series of divider stages. Six bits in the SPI Baud Rate register (SPPR2, SPPR1, SPPR0, SPR2, SPR1, and SPR0) determine the divisor to the SPI module clock which results in the SPI baud rate.

The SPI clock rate is determined by the product of the value in the baud rate preselection bits (SPPR2–SPPR0) and the value in the baud rate selection bits (SPR2–SPR0). The module clock divisor equation is shown in **Figure 4-4**.

When all bits are clear (the default condition), the SPI module clock is divided by 2. When the selection bits (SPR2–SPR0) are 001 and the preselection bits (SPPR2–SPPR0) are 000, the module clock divisor becomes 4. When the selection bits are 010, the module clock divisor becomes 8 etc.

When the preselection bits are 001, the divisor determined by the selection bits is multiplied by 2. When the preselection bits are 010, the divisor is multiplied by 3, etc. See **Table 3-4** for baud rate calculations for all bit conditions, based on a 25 MHz Bus Clock. The two sets of selects allows the clock to be divided by a non-power of two to achieve other baud rates such as divide by 6, divide by 10, etc.

The baud rate generator is activated only when the SPI is in the master mode and a serial transfer is taking place. In the other cases, the divider is disabled to decrease I_{DD} current.

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) \bullet 2^{(\text{SPR} + 1)}$$

Figure 4-4 Baud Rate Divisor Equation

4.6 Special Features

4.6.1 $\overline{SS}$ Output

The $\overline{SS}$ output feature automatically drives the $\overline{SS}$ pin low during transmission to select external devices and drives it high during idle to deselect external devices. When $\overline{SS}$ output is selected, the $\overline{SS}$ output pin is connected to the $\overline{SS}$ input pin of the external device.

The $\overline{SS}$ output is available only in master mode during normal SPI operation by asserting SSOE and MODFEN bit as shown in **Table 3-2**.

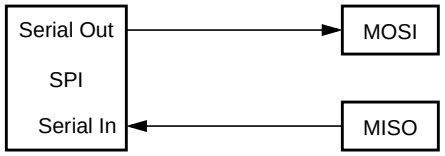
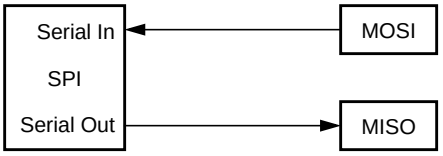
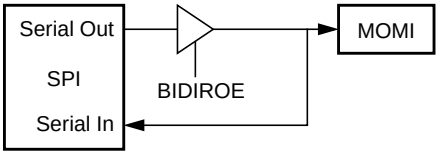
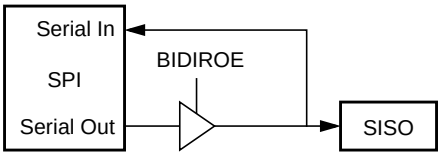
The mode fault feature is disabled while $\overline{SS}$ output is enabled.

NOTE: *Care must be taken when using the $\overline{SS}$ output feature in a multimaster system since the mode fault feature is not available for detecting system errors between masters.*

4.6.2 Bidirectional Mode (MOMI or SISO)

The bidirectional mode is selected when the SPC0 bit is set in SPI Control Register 2 (see **Table 4-1 Normal Mode and Bidirectional Mode**). In this mode, the SPI uses only one serial data pin for the interface with external device(s). The MSTR bit decides which pin to use. The MOSI pin becomes the serial data I/O (MOMI) pin for the master mode, and the MISO pin becomes serial data I/O (SISO) pin for the slave mode. The MISO pin in master mode and MOSI pin in slave mode are not used by the SPI.

Table 4-1 Normal Mode and Bidirectional Mode

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0		
Bidirectional Mode SPC0 = 1		

The direction of each serial I/O pin depends on the BIDIROE bit. If the pin is configured as an output, serial data from the shift register is driven out on the pin. The same pin is also the serial input to the shift register.

The SCK is output for the master mode and input for the slave mode.

The $\overline{SS}$ is the input or output for the master mode, and it is always the input for the slave mode.

The bidirectional mode does not affect SCK and $\overline{SS}$ functions.

NOTE: *In bidirectional master mode, with mode fault enabled, both data pins MISO and MOSI can be occupied by the SPI, though MOSI is normally used for transmissions in bidirectional mode and MISO is not used by the SPI. If a mode fault occurs, the SPI is automatically switched to slave mode, in this case MISO becomes occupied by the SPI and MOSI is not used. This has to be considered, if the MISO pin is used for other purpose.*

4.7 Error Conditions

The SPI has one error condition:

- Mode fault error

4.7.1 Mode Fault Error

If the $\overline{SS}$ input becomes low while the SPI is configured as a master, it indicates a system error where more than one master may be trying to drive the MOSI and SCK lines simultaneously. This condition is not

permitted in normal operation, the MODF bit in the SPI Status Register is set automatically provided the MODFEN bit is set.

In the special case where the SPI is in master mode and MODFEN bit is cleared, the $\overline{SS}$ pin is not used by the SPI. In this special case, the mode fault error function is inhibited and MODF remains cleared. In case the SPI system is configured as a slave, the $\overline{SS}$ pin is a dedicated input pin. Mode fault error doesn't occur in slave mode.

If a mode fault error occurs the SPI is switched to slave mode, with the exception that the slave output buffer is disabled. So SCK, MISO and MOSI pins are forced to be high impedance inputs to avoid any possibility of conflict with another output driver. A transmission in progress is aborted and the SPI is forced into idle state.

If the mode fault error occurs in the bidirectional mode for a SPI system configured in master mode, output enable of the MOMI (MOSI in bidirectional mode) is cleared if it was set. No mode fault error occurs in the bidirectional mode for SPI system configured in slave mode.

The mode fault flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to SPI Control Register 1. If the mode fault flag is cleared, the SPI becomes a normal master or slave again.

4.8 Low Power Mode Options

4.8.1 SPI in Run Mode

In run mode with the SPI system enable (SPE) bit in the SPI control register clear, the SPI system is in a low-power, disabled state. SPI registers can still be accessed, but clocks to the core of this module are disabled.

4.8.2 SPI in Wait Mode

SPI operation in wait mode depends upon the state of the SPISWAI bit in SPI Control Register 2.

- If SPISWAI is clear, the SPI operates normally when the CPU is in wait mode
- If SPISWAI is set, SPI clock generation ceases and the SPI module enters a power conservation state when the CPU is in wait mode.
  - If SPISWAI is set and the SPI is configured for master, any transmission and reception in progress stops at wait mode entry. The transmission and reception resumes when the SPI exits wait mode.
  - If SPISWAI is set and the SPI is configured as a slave, any transmission and reception in progress continues if the SCK continues to be driven from the master. This keeps the slave synchronized to the master and the SCK.

If the master transmits several bytes while the slave is in wait mode, the slave will continue to send out bytes consistent with the operation mode at the start of wait mode (i.e. If the slave is currently sending its SPIDR to the master, it will continue to send the same byte. Else if the slave is currently sending the last received byte from the master, it will continue to send each previous master byte).

NOTE: *Care must be taken when expecting data from a master while the slave is in wait or stop mode. Even though the shift register will continue to operate, the rest of the SPI is shut down (i.e. a SPIF interrupt will* ***not*** *be generated until exiting stop or wait mode). Also, the byte from the shift register will not be copied into the SPIDR register until after the slave SPI has exited wait or stop mode. A SPIF flag and SPIDR copy is only generated if wait mode is entered or exited during a tranmission. If the slave enters wait mode in idle mode and exits wait mode in idle mode, neither a SPIF nor a SPIDR copy will occur.*

4.8.3 SPI in Stop Mode

Stop mode is dependent on the system. The SPI enters stop mode when the module clock is disabled (held high or low). If the SPI is in master mode and exchanging data when the CPU enters stop mode, the transmission is frozen until the CPU exits stop mode. After stop, data to and from the external SPI is exchanged correctly. In slave mode, the SPI will stay synchronized with the master.

The stop mode is not dependent on the SPISWAI bit.

4.8.4 Reset

The reset values of registers and signals are described in the Memory Map and Registers section (see **Section 3 Memory Map/Register Definition**) which details the registers and their bit-fields.

- If a data transmission occurs in slave mode after reset without a write to SPIDR, it will transmit garbage, or the byte last received from the master before the reset.
- Reading from the SPIDR after reset will always read a byte of zeros.

4.8.5 Interrupts

The SPI only originates interrupt requests when SPI is enabled (SPE bit in SPICR1 set). The following is a description of how the SPI makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt priority are chip dependent.

The interrupt flags MODF, SPIF and SPTEF are logically ORed to generate an interrupt request.

4.8.5.1 MODF

MODF occurs when the master detects an error on the $\overline{SS}$ pin. The master SPI must be configured for the MODF feature (see **Table 3-2 SS Input / Output Selection**). Once MODF is set, the current transfer is aborted and the following bit is changed:

- MSTR=0, The master bit in SPICR1 resets.

The MODF interrupt is reflected in the status register MODF flag. Clearing the flag will also clear the interrupt. This interrupt will stay active while the MODF flag is set. MODF has an automatic clearing process which is described in **3.1.4 SPI Status Register**.

4.8.5.2 SPIF

SPIF occurs when new data has been received and copied to the SPI Data Register. Once SPIF is set, it does not clear until it is serviced. SPIF has an automatic clearing process which is described in **3.1.4 SPI Status Register**. In the event that the SPIF is not serviced before the end of the next transfer (i.e. SPIF remains active throughout another transfer), the latter transfers will be ignored and no new data will be copied into the SPIDR.

4.8.5.3 SPTEF

SPTEF occurs when the SPI Data Register is ready to accept new dataOnce SPTEF is set, it does not clear until it is serviced. SPTEF has an automatic clearing process which is described in **3.1.4 SPI Status Register**.

Section 5 Initialization/Application Information

Index

Block Guide End Sheet

FINAL PAGE OF
38
PAGES

VREG
Block User Guide
V01.01

Original Release Date: 21 FEB 2001
Revised: 4 MAR 2002

Motorola, Inc

Revision History

Version Number	Revision Date	Effective Date	Author	Description of Changes
0.1	2/21/01	2/21/01		VREG spec moved to SRS2.0 compliant format
1.0	4/05/01	4/05/01		Minor update, spec version number reflects clearcase label
V01.00	7/27/01	7/27/01		Document names have been added Names and Variable definitions have been hidden
V01.01	3/4/02	3/4/02		Changed Document Number

Table of Contents

List of Figures

List of Tables

Section 1 Introduction

1.1 Overview

The VREG block is used to generate the supply voltage (2.5V typ.) of the core logic and memory blocks out of the chip supply voltage (5V typ.).

1.2 Features

The block name includes these distinctive features:

- linear voltage regulator with two independent outputs
- power on reset signal generation

1.3 Modes of Operation

VREG can operate in three different modes

- RUN

  In run mode both regulating loops of the voltage regulator are active. This mode is selected whenever the CPU is neither in stop nor in pseudo stop mode and VREGEN is pulled high.

- STANDBY

  Standby mode is selected when the CPU is in stop or pseudo stop mode and VREGEN is pulled high. In standby mode the gates of the power transistors are directly connected to the reference voltage. In this case the voltage regulator acts as a voltage clamp. While in standby mode, the effective inner resistance of the regulator is increased, the quiescent current consumption of the regulator itself is heavily decreased.

- SHUTDOWN

  Shutdown mode is only available, when the device is equipped with a VREGEN bit. Shutdown mode is selected by tieing VREGEN to ground. In this case, the core logic must be supplied from external by applying 2.5V(+/-10%) on VDD and VDDPLL. The power on reset pulse generation circuit is not affected by selecting shutdown mode.

1.4 Block Diagram

Figure 1-1 is a block diagram of the VREG.

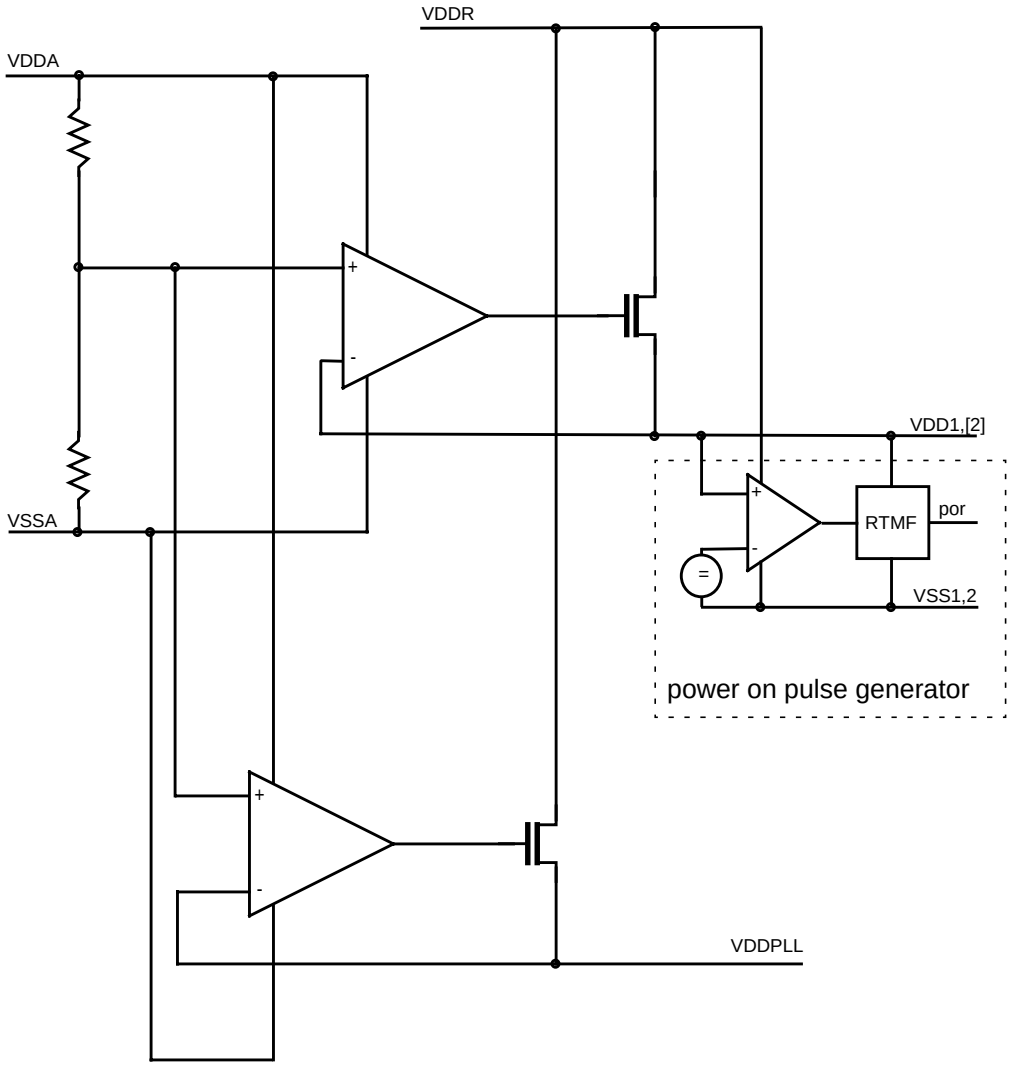


Figure 1-1 VREG Block Diagram

Section 2 Signal Description

2.1 Overview

Table 2-1 lists all pins associated with the VREG block.

Table 2-1 Signal Properties

Name	Function
VDDA	VREG positive reference and supply input
VSSA	VREG negative reference and supply input
VDDR	VREG power input
VDD1,[2]	VREG output 1st regulation loop
VSS1,[2]	VREG 1st regulation loop ground pin
VDDPLL	VREG output 2nd regulation loop
VSSPLL	VREG 2nd regulation loop ground pin
VREGEN	Selects Shutdown or Run/Standby mode

2.2 Detailed Signal Descriptions

2.2.1 VDDA, VSSA

VREG uses the VDDA/VSSA supply pin pair to supply the voltage regulator and to derive the reference voltage. The reference voltage VREG is regulating to is $(V_{DDA} - V_{SSA})/2$.

2.2.2 VDDR

VDDR is the power input to the voltage regulator. The output current of the two regulating loops is drawn out of this pin.

2.2.3 VDD1,[2], VSS1,[2]

VDD1, VSS1 and optional VDD2, VSS2 are the core logic supply pins. VDD1 and VDD2 are connected internally by metal as well as VSS1 and VSS2. Each power supply pin pair must be externally decoupled with a ceramic capacitor (100nF .. 220nF, X7R ceramic). VDD1,[2] is connected to the output of the first regulating loop of the voltage regulator.

2.2.4 VDDPLL, VSSPLL

VDDPLL and VSSPLL are the oscillator and pll supply pins. This supply pin pair must be externally decoupled with a ceramic capacitor (100nF .. 220nF, X7R ceramic). VDDPLL is connected to the output of the second regulating loop of the voltage regulator.

2.2.5 VREGEN

This optional pin is used to disable the voltage regulator if the core logic as well as the oscillators are supplied from external.

Section 3 Memory Map and Registers

3.1 Overview

The VREG block has no CPU accessible registers.

Section 4 Functional Description

4.1 General

The VREG block consists of a reference voltage generator, two operational amplifiers, two nmos power output stages and a power on reset pulse generation circuit.

4.1.1 Reference Generation

The reference generation is comprised of a resistor reference ladder between VDDA and VSSA. The output voltage of the reference ladder $(V_{DDA} - V_{SSA})/2$ is fed into both operational amplifiers as regulation reference.

4.1.2 Operational Amplifier

The operational amplifier compare the reference voltage $((V_{DDA} - V_{SSA})/2)$ with the actual output voltage (vdd or vddpll) to generate the gate voltage of the power output transistors.

In standby mode, the operational amplifiers are disabled and the gates of the power transistors are connected directly to the reference voltage in order to decrease the quiescent current.

4.1.3 Power Output Stage

Each power output stage consists of an nmos power transistor with its drain on VDDR and its source on VDD or VDDPLL.

4.1.4 Power On Reset Pulse Generation

A comparator monitors the actual value of VDD. If V_{DD} is below V_{POR}, the power on reset signal is asserted forcing the CPU in the power on reset state.

User Guide End Sheet

FINAL PAGE OF
18
PAGES