

Lecture 6 objectives

- Software latency and real-time systems
- Blind-cycle versus busy-wait synchronization
- Output compare interrupts

Software Latency is the time between when the I/O device needs service, and the time when service is initiated.

For an input device,

latency is the time between new input data ready, and the software reading the data.

For an output device,

latency is the time between output device idle, and the software giving the device new data to output.

Periodic events (sampling ADC, outputting to DAC).

latency is the time between when it is supposed to be run, and when it is actually run.

A **real time** system is one that can guarantee a worst case software latency. In other words, there is an upper bound on the software response time.

Hardware Latency or **device latency** is the time between when an I/O device is given a command, and the time when command is completed.

bandwidth is maximum data flow that can be processed

Sometimes the bandwidth is limited by the I/O device, while other times it is limited by computer software.

can be reported as an overall average or a short-term maximum.

Priority determines the order of service when two or more requests are made simultaneously.

a "soft-real-time" system supports priority.

Five ways to synchronize

- Blind cycle** **Fixed time delay**
- Busy Waiting** **Software checks flag over and over**
- Interrupt** **Flag causes ISR to run**
- Periodic Polling** **Software check flag in periodic interrupt**
- DMA** **I/O device reads/writes directly into RAM**

I/O bound is defined as

- Bandwidth is limited by speed of I/O device
- Making the I/O device faster will increase bandwidth
- Making the software run faster will not increase bandwidth
- Software often waits for the I/O device

CPU bound is defined as

- Bandwidth is limited by speed of executing software
- Making the I/O device faster will not increase bandwidth
- Making the software run faster will increase bandwidth
- Software does not have to wait for the I/O device

Synchronizing the software with an input device

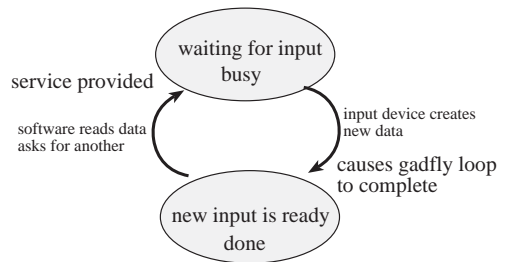
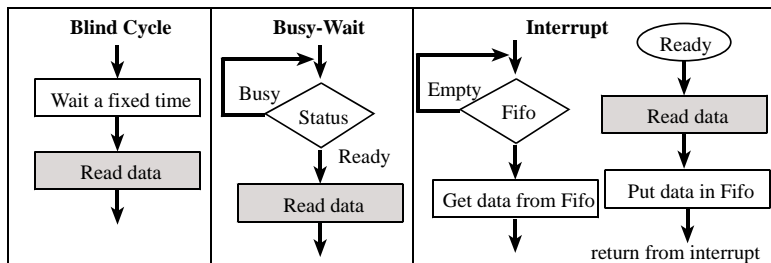
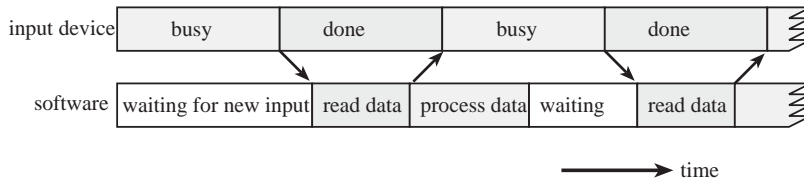


Figure 3.1. The input device sets a flag when it has new data.

Synchronizing with an input device





Is this I/O bound or CPU bound?

Answer: a little bit of both because in the done state hardware waits for software. In the busy state software waits for hardware.

Synchronizing the software with an output device

Synchronizing with an output device

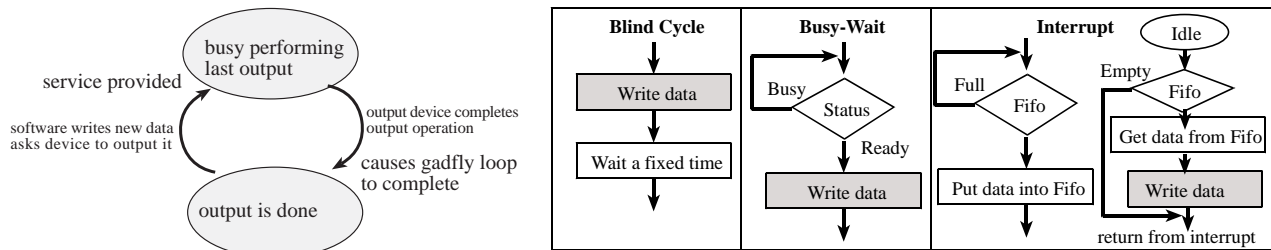


Figure 3.3. The output device sets a flag when it has finished outputting the last data.

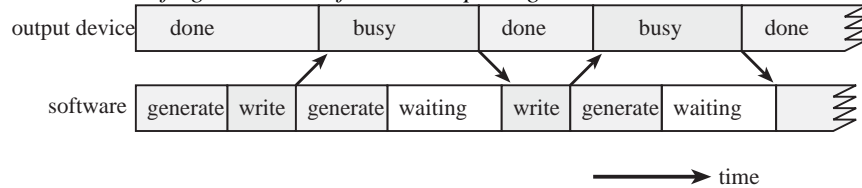


Figure 3.4. The software must wait for the output device to finish the previous operation.

3.2. Blind Cycle Counting Synchronization

Blind cycle counting is appropriate when the I/O delay is fixed and known. This type of synchronization is blind because it provides no feedback from the I/O back to the computer.

From Starter File **LCD_DP512**

```
//-----wait-----
// time delay
// Input: time in 0.667usec
// Output: none
void static wait(unsigned short cycles){
    unsigned short startTime = TCNT;
    while((TCNT-startTime) <= cycles){
    }
}
//-----outCsrNibble-----
// sends one command to LCD control/status
// Input: command is 4-bit function
// Output: none
static void outCsrNibble(unsigned char command){
    PTH = (PTH&0x80)+(command&0x0F); //E=0, RS=0
    PTH |= 0x10; // E goes 0,1
    PTH &= ~0x10; // E goes 1,0
}
//-----outCsr-----
// sends one command to LCD control/status
// Input: command is 8-bit function
// Output: none
```

```
static void outCsr(unsigned char command){
    outCsrNibble(command>>4); // ms,E=0,RS=0
    outCsrNibble(command&0x0F); // ls,E=0,RS=0
    wait(135); // blind cycle 90 us wait
}
```

3.3. Busy Waiting Synchronization

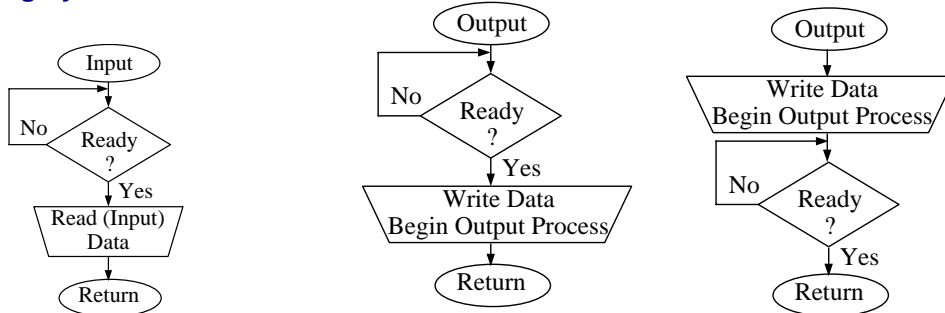


Figure 3.7. A software flowchart for busy-wait input. Figure 3.8. Two software flowcharts for busy-wait output.

Busy Waiting with three devices

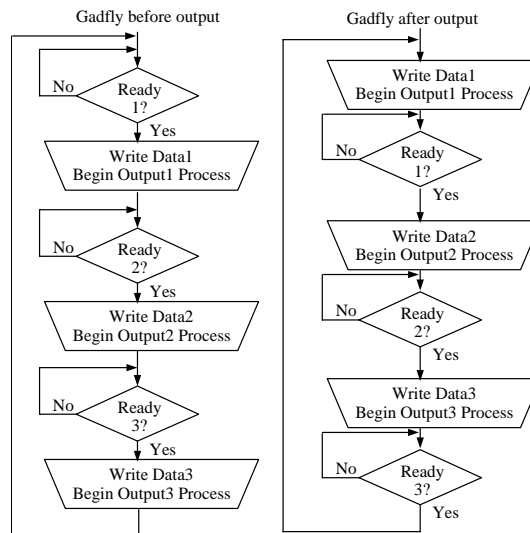


Figure 3.9. Two software flowcharts for multiple busy-wait outputs.

Time(ms)	Wait Before Event	Wait After Event
0	Start 1,2,3	Start1
from 0 to 1	Wait for 1	Wait for 1
1	Start 1,2,3	Start 2
from 1 to 2	Wait for 1	Wait for 2
2	Start 1,2,3	Start 3
from 2 to 3	Wait for 1	Wait for 3
3	Start 1,2,3	Start1
from 3 to 4	Wait for 1	Wait for 1
4	Start 1,2,3	Start 2
from 4 to 5	Wait for 1	Wait for 2

Performance Tip: Whenever we can establish concurrent I/O operations, we can expect an improvement in the overall system bandwidth.

Busy Waiting with multiple devices

Figure 3.10. A software flowchart for multiple busy-wait inputs and outputs.

Example of a three thread system

- 1) The foreground thread usually is running
- 2) The "key" background thread is invoked whenever a key is touched
- 3) The "time" background thread is invoked every 1ms in a periodic fashion.

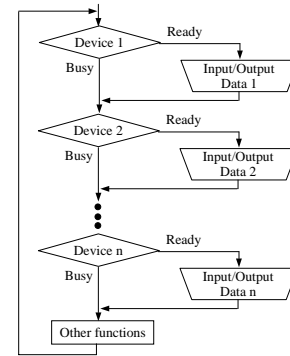
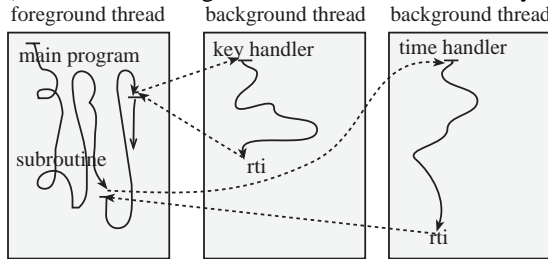


Figure 2.14. Interrupts allow us to have multiple background threads.

The running thread must be suspended in order to execute another thread.

- save registers on stack
- set PC to beginning of the interrupt handler
- created with an "empty" stack and uninitialized registers.

When the **keyhandler** is done it executes **rti** to relinquish control
original stack and registers of the main program will be restored

We can think of each thread as having its own registers and its own stack area.

Three conditions must be true for an interrupt to occur:

- 1) Initialization software will set the arm bit, E.g., **C0I**
individual control bit for each possible flag that can interrupt
- 2) When it is convenient, the software will enable, I=0
asm cli
- 3) Hardware action (busy to done) sets a flag E.g., **C0F**
C0F is set at a periodic rate by an output compare internal clock

What happens when an interrupt is processed?

- 1) execution of main program is suspended (current instruction is finished),
pushes registers on the stack,
sets the I bit,
gets the vector address from high memory, E.g., **0xFFEE**
- 2) the interrupt service routine, or background thread is executed,
clears the flag that requested the interrupt, **C0F** cleared by writing a 1 to it
performs necessary operations
communicates using global variables
- 3) the main program is resumed when the interrupt service routine executes **rti**.
pulls the registers from the stack

9S12DP512 interrupts

Vector Address	CW num	Interrupt Source or Trigger	En	Local Arm	HPRIO Elevate
\$FFFE	0	Reset	none	none	—
\$FFFC	1	COP Clock Monitor Fail Reset	none none	COPCTL.CME COPCTL.FCME	— —
\$FFFA	2	COP Failure Reset	none	COP rate selected	—
\$FFF8	3	Unimplemented Inst Trap	none	none	—
\$FFF6	4	SWI	none	none	—
\$FFF4	5	XIRQ	X bit	none	—
\$FFF2	6	IRQ	I bit	INTCR.IRQEN	\$F2
\$FFF0	7	Real Time Interrupt, RTIF	I bit	CRGINT.RTIE	\$F0
\$FFEE	8	Timer Channel 0, C0F	I bit	TIE.C0I	\$EE
\$FFEC	9	Timer Channel 1, C1F	I bit	TIE.C1I	\$EC
\$FFEA	10	Timer Channel 2, C2F	I bit	TIE.C2I	\$EA

\$FFE8	11	Timer Channel 3, C3F	I bit	TIE.C3I	\$E8
\$FFE6	12	Timer Channel 4, C4F	I bit	TIE.C4I	\$E6
\$FFE4	13	Timer Channel 5, C5F	I bit	TIE.C5I	\$E4
\$FFE2	14	Timer Channel 6, C6F	I bit	TIE.C6I	\$E2
\$FFE0	15	Timer Channel 7, C7F	I bit	TIE.C7I	\$E0
\$FFDE	16	Timer Overflow, TOF	I bit	TIE.TOI	\$DE
\$FFDC	17	Pulse Acc. Overflow, PAOVF	I bit	PACTL.PAOVI	\$DC
\$FFDA	18	Pulse Acc. Input Edge, PAIF	I bit	PACTL.PAI	\$DA
\$FFD8	19	SPI0 Transfer Complete, SPIF SPI0 Transmit Empty, SPTEF	I bit	SPI0CR1.SPIE SPI0CR1.SPTIE	\$D8
\$FFD6	20	SCI0 Tx Buff Empty, TDRE SCI0 Tx Complete, TC SCI0 Rx Buffer Full, RDRF SCI0 Rx Idle, IDLE	I bit	SCI0CR2.TIE SCI0CR2.TCIE SCI0CR2.RIE SCI0CR2.ILIE	\$D6
\$FFD4	21	SCI1 Tx Buff Empty, TDRE SCI1 Tx Complete, TC SCI1 Rx Buffer Full, RDRF SCI1 Rx Idle, IDLE	I bit	SCI1CR2.TIE SCI1CR2.TCIE SCI1CR2.RIE SCI1CR2.ILIE	\$D4
\$FFD2	22	ATD0 Seq Complete, ASCIF	I bit	ATD0CTL2.ASCIE	\$D2
\$FFD0	23	ATD1 Seq Complete, ASCIF	I bit	ATD1CTL2.ASCIE	\$D0
\$FFCE	24	Key Wakeup J, PIFJ.[7:6],[1,0]	I bit	PIEJ.[7:6],[1:0]	\$CE
\$FFCC	25	Key Wakeup H, PIFH.[7:0]	I bit	PIEH.[7:0]	\$CC
\$FFC8	27	Pulse Acc. Overflow, PBOVF	I bit	PBCTL.PBOVI	\$DC
\$FFC0	31	I2C	I bit	IBCR.IBIE	\$C0
\$FFBE	32	SPI1 Transfer Complete, SPIF SPI1 Transmit Empty, SPTEF	I bit	SPI1CR1.SPIE SPI1CR1.SPTIE	\$BE
\$FFBC	33	SPI2 Transfer Complete, SPIF SPI2 Transmit Empty, SPTEF	I bit	SPI2CR1.SPIE SPI2CR1.SPTIE	\$BC
\$FFB6	36	CAN wakeup	I bit	CANRIER.WUPIE	\$B6
\$FFB4	37	CAN errors	I bit	CANRIER.CSCIE CANRIER.OVRIE	\$B4
\$FFB2	38	CAN receive	I bit	CANRIER.RXFIE	\$B2
\$FFB0	39	CAN transmit	I bit	CANTIER.TXEIE[2:0]	\$B0
\$FF8E	56	Key Wakeup P, PIFP[7:0]	I bit	PIEP.[7:0]	\$8E

MC9S12DP512 Periodic Interrupt using Output Compare 0

address	msb																lsb	Name
\$0044	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TCNT	
\$0050	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC0	
\$0052	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC1	
\$0054	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC2	
\$0056	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC3	
\$0058	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC4	
\$005A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC5	
\$005C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC6	
\$005E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	TC7	

Address	Bit 7	6	5	4	3	2	1	Bit 0	Name
\$0046	TEN	TSWAI	TSBCK	TFFCA	0	0	0	0	TSCR1
\$004D	TOI	0	0	0	TCRE	PR2	PR1	PR0	TSCR2
\$0040	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	TIOS
\$004C	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	TIE
\$004E	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F	TFLG1
\$004F	TOF	0	0	0	0	0	0	0	TFLG2

Table MC9S12DP512 registers used to configure periodic interrupts.

The rate is dependent of the PLL, TSCR2, and 10000 in ISR

Things you must do in every interrupt service routine**Acknowledge (clear flag that requested interrupt)****Things you must do in an OC interrupt service routine****Acknowledge (make C0F become zero)****Set the timer to specify when to interrupt next**

```

interrupt 8 void TC0handler(void){
// executes at 100 Hz
    TFLG1 = 0x01;    // acknowledge OC0
    TC0 = TC0+10000; // 10 ms
// do something
    Count0++;
    PTT ^= 0x01;    // toggle LED
}

```

Things you must do in a ritual**Arm (specify a flag may interrupt)****Enable (allow all interrupts on the 9S12)****Things you must do in an OC ritual****Turn on TCNT (TEN=1)****Set channel to output compare (TIOS)****Specify TCNT rate (TSCR2, PACTL, PLL)****Arm (C0I=1)****When to generate first interrupt****Enable (I=0)**

```

//-----OC_Init0-----
// arm output compare 0 for 100Hz interrupt
// Input: none
// Output: none
void OC_Init0(){
    Count0 = 0;    // debugging monitor
    DDRT |= 0x01;  // debugging monitor
    TIOS |= 0x01;  // TC0 as output compare
    TIE |= 0x01;   // arm OC0
    TSCR1 = 0x80;  // Enable TCNT
    TSCR2 = 0x03;  // divide by 8 TCNT
    PACTL = 0;     // timer prescale
/* Bottom three bits of TSCR2 (PR2,PR1,PR0) set TCNT period
    divide  FastMode(24MHz)    Slow Mode (8MHz)
000  1      42ns   TOF  2.73ms  125ns TOF  8.192ms
001  2      84ns   TOF  5.46ms  250ns TOF  16.384ms
010  4     167ns   TOF  10.9ms  500ns TOF  32.768ms
011  8     333ns   TOF  21.8ms   1us  TOF  65.536ms
100 16     667ns   TOF  43.7ms   2us  TOF  131.072ms
101 32    1.33us   TOF  87.4ms   4us  TOF  262.144ns
110 64    2.67us   TOF  174.8ms  8us  TOF  524.288ms
111 128   5.33us   TOF  349.5ms  16us TOF  1.048576s */
    TC0 = TCNT+50; // first interrupt right away
}
void main(void) {
    OC_Init0();    // 100 Hz OC0
    OC_Init3();    // 1000 Hz OC3
    DDRP |= 0x80;  // LED
    asm cli
    for(;;) {
        PTP ^= 0x80; // flash LED
    }
}

```