

Review from EE345L

- PLL
- Output compare interrupts
- SPI

Introduction to solid state disk lab

- EEPROM interface
- Linked allocation of disk blocks
- Simple directory
- Internal fragmentation
- External fragmentation

Set 9S12C32/9SDP512 PLL to run at 24 MHz

```

//***** PLL_Init *****
// Set PLL clock to 24 MHz, and run at this rate
// Inputs: none
// Outputs: none
// Errors: will hang if PLL does not stabilize
void PLL_Init(void){
    SYNCR = 0x02;
// PLLCLK = 2 * OSCCLK * (SYNR + 1) / (REFDV + 1)
    REFDV = 0x00; // for C32 with 8 MHz crystal
// PLLCLK = 2 * 8 MHz * (2 + 1) / (0 + 1) = 24 MHz
// REFDV = 0x01; // for DP512 with 16 MHz crystal
// PLLCLK = 2 * 16 MHz * (2 + 1) / (1 + 1) = 24 MHz

    CLKSEL = 0x00;
// Meaning for CLKSEL:
//   Bit 7: PLLSEL = 0 Keep OSCCLK until we are ready
//   Bit 6: PSTP    = 0 Do not need to go to Pseudo-Stop
//   Bit 5: SYSWAI = 0 In wait mode system clocks stop.
//   Bit 4: ROAWAI = 0 Do not reduce oscillator
//   Bit 3: PLLWAI = 0 Do not turn off PLL in wait mode
//   Bit 2: CWAI   = 0 Do not stop the core during wait
//   Bit 1: RTIWAI = 0 Do not stop the RTI in wait mode
//   Bit 0: COPWAI = 0 Do not stop the COP in wait mode

```

```

    PLLCTL = 0xD1;
// Meaning for PLLCTL:
// Bit 7: CME = 1; Clock monitor enable-reset if bad
// Bit 6: PLLON= 1; PLL On bit
// Bit 5: AUTO = 0; No auto control of bandwidth
// Bit 4: ACQ = 1; 1 for acquisition; 0 for tracking
// Bit 3:          (Not Used by 9s12)
// Bit 2: PRE = 0; RTI stops during Pseudo Stop Mode
// Bit 1: PCE = 0; COP disabled during Pseudo STOP
// Bit 0: SCME = 1; Crystal Failure->Self Clock mode

    while((CRGFLG&0x08) == 0){
// Wait for PLLCLK to stabilize.
    }
    CLKSEL_PLLSEL = 1; // Switch to PLL clock
}

```

Periodic interrupts using OC5

Discuss concept of race condition

When to allow first interrupt?

```

//-----OC_Init-----
// arm output compare 5 for 1000Hz periodic interrupt
// Input: none
// Output: none
void OC_Init(void){
    asm sei // make ritual atomic
    TIOS |= 0x20; // activate TC5 as output compare
    TSCR1 = 0x80; // Enable TCNT, assume PLL = 24MHz
    TSCR2 = 0x04; // TCNT prescale, TOI disarm
    PACTL = 0; // timer prescale used for TCNT
/* TSCR2 (PR2,PR1,PR0) determine TCNT period
    divide FastMode(24MHz)
000 1 42ns TOF 2.73ms
001 2 84ns TOF 5.46ms
010 4 167ns TOF 10.9ms
011 8 333ns TOF 21.8ms
100 16 667ns TOF 43.7ms
101 32 1.33us TOF 87.4ms
110 64 2.67us TOF 174.8ms
111 128 5.33us TOF 349.5ms */

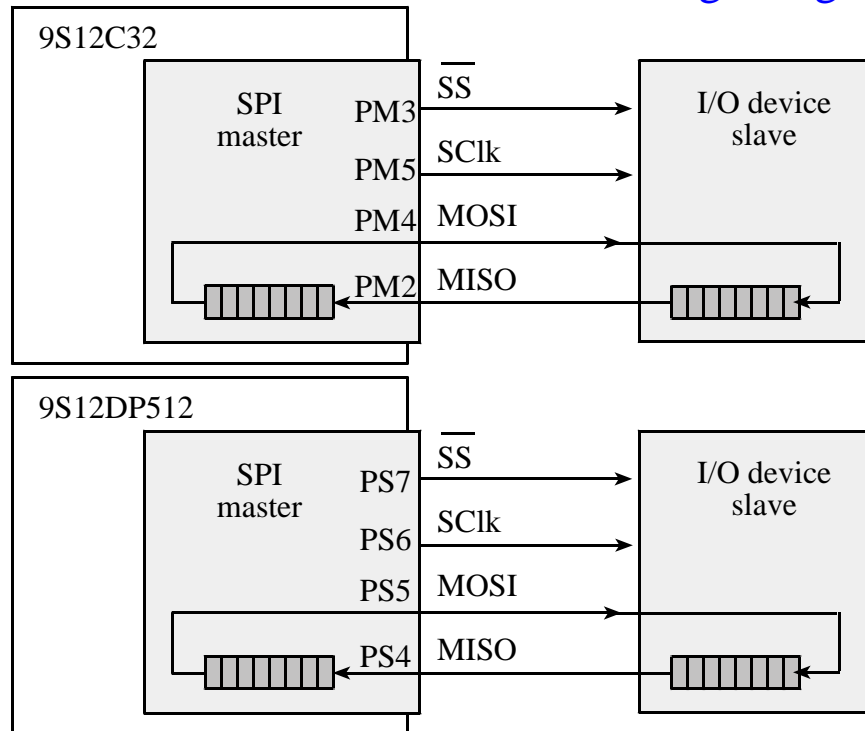
```

```

TIE |= 0x20; // arm OC5
TC5 = TCNT+50; // first interrupt right away
TFLG1 = 0x20; // clear OC5
asm cli
}
interrupt 13 void TC5handler(void){
// executes at 1000 Hz
TFLG1 = 0x20; // acknowledge OC5
TC5 = TC5+1500; // 1 ms
}
    
```

What if **TC5= TCNT+50;**
 but only execute **cli** until all initializations are complete?

7.7. Synchronous Transmission and Receiving using SPI



Microcomputer	pin for \overline{SS}	pin for SCK	pin for MOSI	pin for MISO
9S12C32	PM3	PM5	PM4	PM2
9S12DP512 SPI0	PS7	PS6	PS5	PS4
9S12DP512 SPI1	PH3	PH2	PH1	PH0
9S12DP512 SPI2	PH7	PH6	PH5	PH4

7.7.5. 6812 SPI Details

	7	6	5	4	3	2	1	0
SPICR1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBF

SPIE — SPI Interrupt Enable

1 = Hardware interrupt sequence is requested each time the SPIF

0 = SPI interrupts are inhibited

SPE — SPI System Enable

0 = SPI system is in a low-power disabled state.

1 = PS[7:4], PM[2:5] are dedicated to the SPI function

SPTIE — SPI Transmit Interrupt Enable

MSTR — SPI Master/Slave Mode Select

0 = Slave mode

1 = Master mode

CPOL, CPHA — SPI Clock Polarity, Clock Phase

SSOE=0 — Slave Select Output Enable (also set MODFEN=0, in SPICR2)

LSBF=0 — SPI LSB First enable

	7	6	5	4	3	2	1	0
SPISR	SPIF		SPTEF	MODF	0	0	0	0

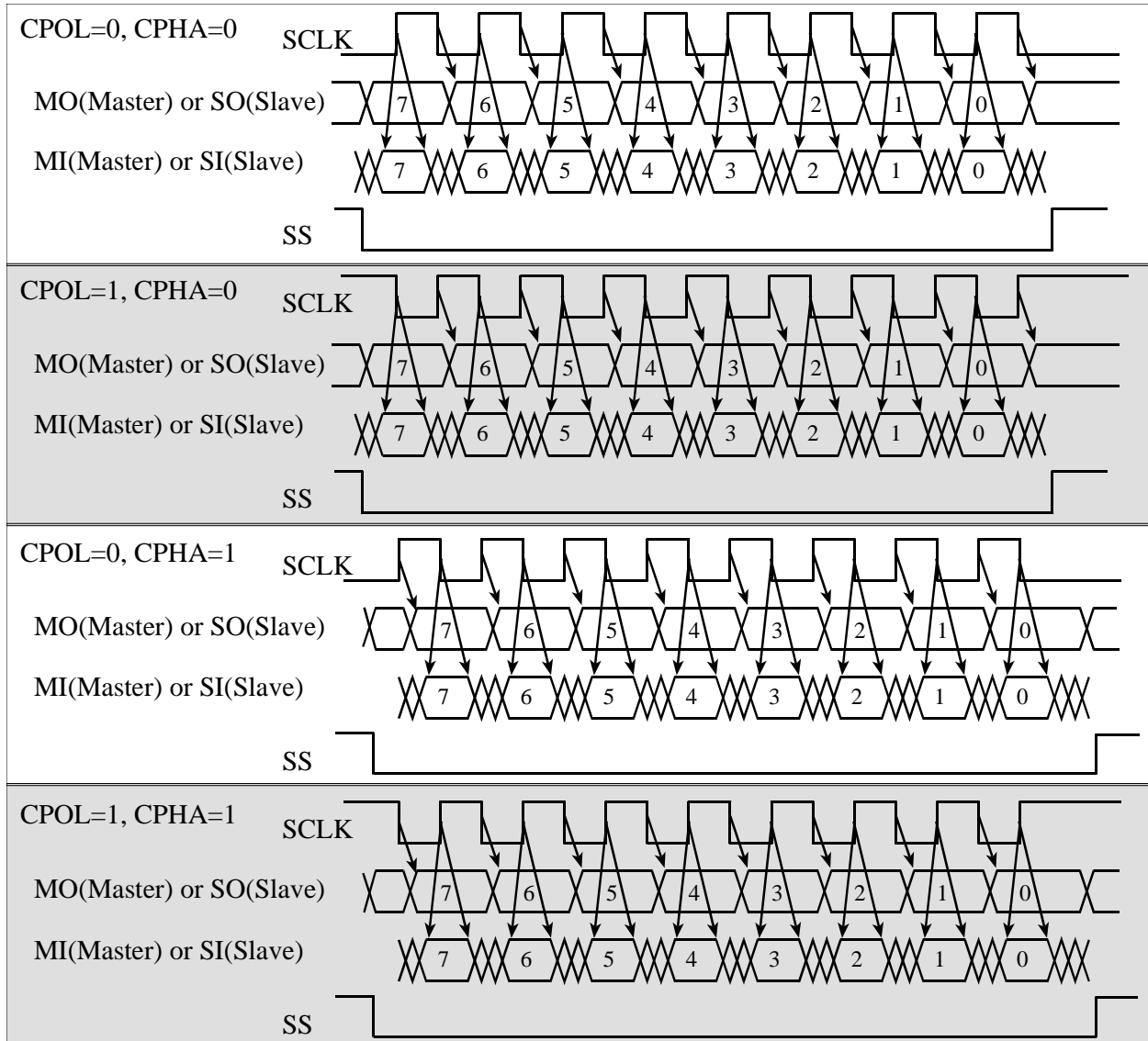
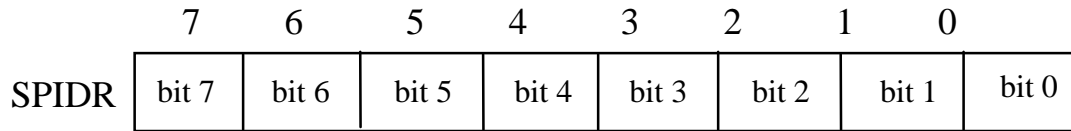
SPIF — SPI Flag

SPIF is set after the eighth SCK cycle in a data transfer and it is cleared by reading the SPISR register (with SPIF set) followed by a read access to the SPI data register.

SPTEF — SPI transmit Flag

SPTEF occurs when the SPI Data Register is ready to accept new data.

After SPTEF is set, it does not clear until it is serviced. SPTEF has an automatic clearing process. 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.



As with any SPI interface, there are basic interfacing issues to consider.

- 1) *Word size.*
- 2) *Bit order.* The EEPROM requires the most significant bits first.
- 3) *Clock phase, clock polarity.* There are two issues to resolve. Since the EEPROM samples its serial input data on the rising edge of the clock, the SPI

must changes the data on the falling edge. CPOL=CPHA=0 and CPOL=CPHA=1 both satisfy this requirement. The second issue is which edge comes first the rise or the fall. In this interface it probably doesn't matter.

- 4) *Bandwidth.* We look at the timing specifications of the EEPROM. The setup time of 45 ns means the shortest SPI period we can use is 90ns.

FIGURE 1-2: SERIAL INPUT TIMING

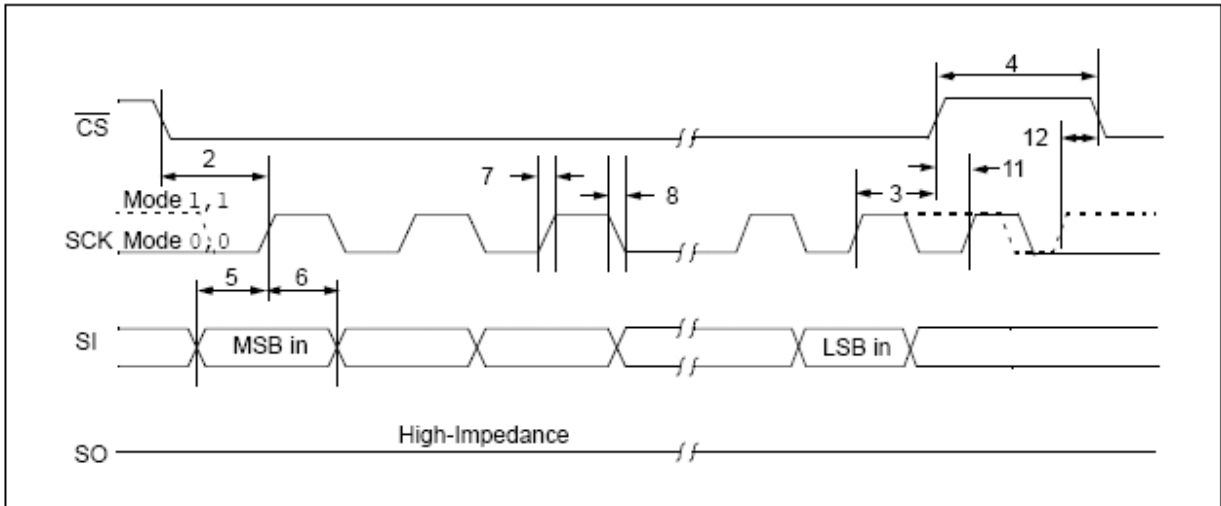
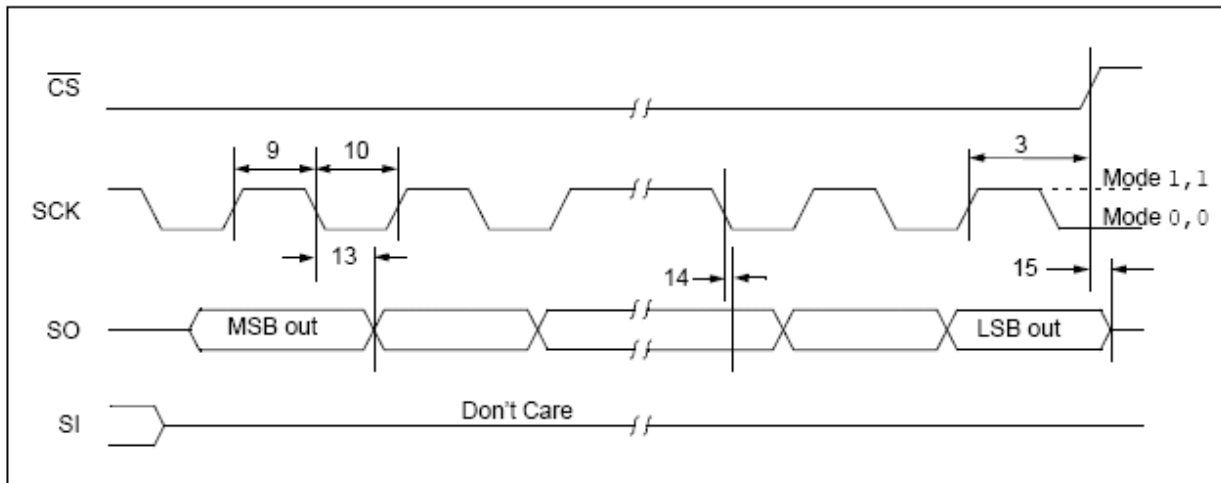


FIGURE 1-3: SERIAL OUTPUT TIMING

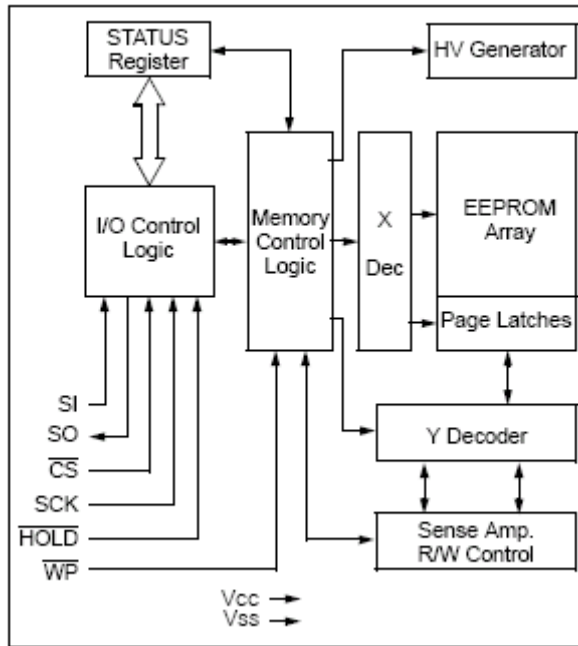


25LC1024

1 Mbit SPI Bus Serial EEPROM

512 pages, each page is 256 bytes

BLOCK DIAGRAM



PDIP/SOIJ (P, SM)

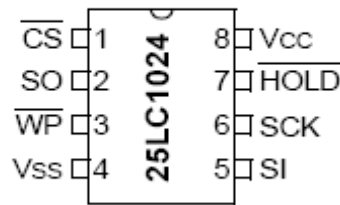
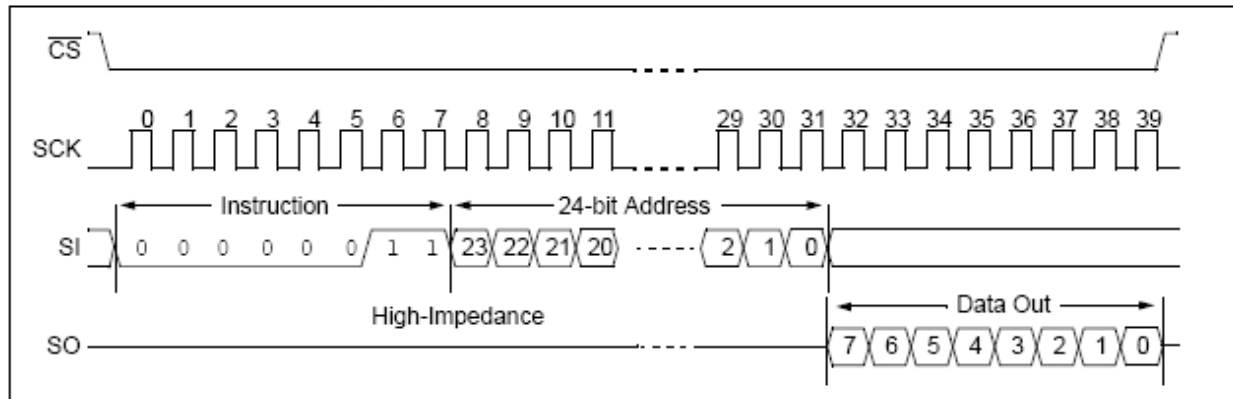


TABLE 2-1: INSTRUCTION SET

Instruction Name	Instruction Format	Description
READ	0000 0011	Read data from memory array beginning at selected address
WRITE	0000 0010	Write data to memory array beginning at selected address
WREN	0000 0110	Set the write enable latch (enable write operations)
WRDI	0000 0100	Reset the write enable latch (disable write operations)
RDSR	0000 0101	Read STATUS register
WRSR	0000 0001	Write STATUS register
PE	0100 0010	Page Erase – erase one page in memory array
SE	1101 1000	Sector Erase – erase one sector in memory array
CE	1100 0111	Chip Erase – erase all sectors in memory array
RDID	1010 1011	Release from Deep power-down and read electronic signature
DPD	1011 1001	Deep Power-Down mode

FIGURE 2-1: READ SEQUENCE



Read one or more bytes

- 1) Set CS=0
- 2) Send READ \$03
 - a) Wait for SPTEF to be 1
 - b) Write to SPIDR (\$03)
 - c) Wait for SPIF to be 1
 - d) Read SPIDR (to clear SPIF)
- 3) Send address 23-16
 - a) Wait for SPTEF to be 1
 - b) Write to SPIDR (address 23-16)
 - c) Wait for SPIF to be 1
 - d) Read SPIDR (to clear SPIF)
- 4) Send address 15-8
 - a) Wait for SPTEF to be 1
 - b) Write to SPIDR (address 15-8)
 - c) Wait for SPIF to be 1
 - d) Read SPIDR (to clear SPIF)
- 5) Send address 7-0
 - a) Wait for SPTEF to be 1
 - b) Write to SPIDR (address 7-0)
 - c) Wait for SPIF to be 1
 - d) Read SPIDR (to clear SPIF)
- 6) Receive data at that address
 - a) Wait for SPTEF to be 1
 - b) Write to SPIDR (0)
 - c) Wait for SPIF to be 1
 - d) Read SPIDR (this is the data at that address)
- 7) Optional repeat step 6 over and over to retrieve sequential data
- 8) Set CS=1

Max1247 12-bit Analog to Digital Converter

```
//***** Max1247_In *****
// perform 12-bit analog to digital
// input: specifying channel to sample
// output: 12-bit ADC sample
// analog input      digital output
// 0.0000           0
```

```

//      1.250                2048
//      2.500                4095
// uses busy-wait synchronization
// example data = ADC_In(CH2);
#define CH0 0x9F
#define CH1 0xDF
#define CH2 0xAF
#define CH3 0xEF
unsigned short Max1247_In(unsigned char chan){
unsigned char data;
unsigned short result;
    // PM3 low for one output, two inputs
    PTM &= ~0x08;    // clear PM3= /CS

    while(((SPISR&0x20)==0)){}; // wait SPTEF
    SPIDR = chan;           // command
    while(((SPISR&0x80)==0)){}; // wait SPIF
    data = SPIDR;          // clear SPIF

    while(((SPISR&0x20)==0)){}; // wait SPTEF
    SPIDR = 0;             // output
    while(((SPISR&0x80)==0)){}; // wait SPIF=1
    data = SPIDR;          // bits 11-5
    result = data<<5;

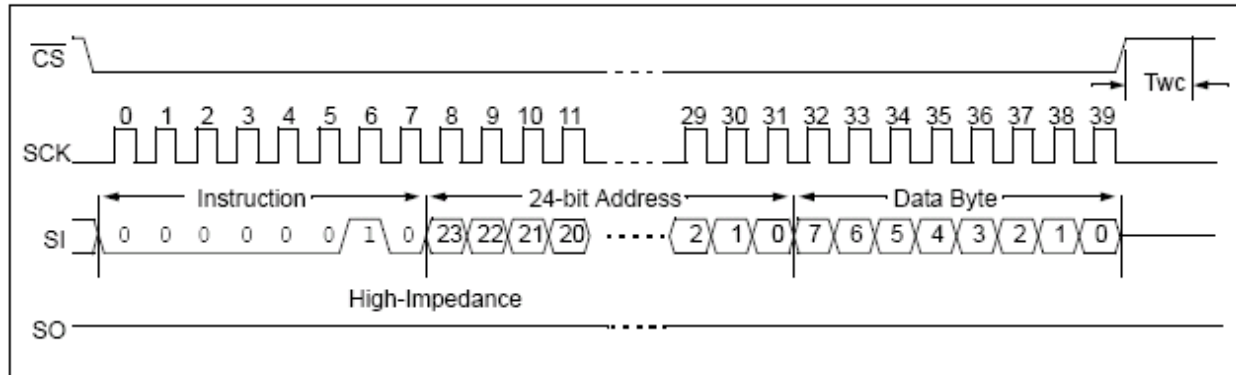
    while(((SPISR&0x20)==0)){}; // wait SPTEF
    SPIDR = 0;             // output
    while(((SPISR&0x80)==0)){}; // wait SPIF
    data = SPIDR;          // bits 4-0
    result += data>>3;
    PTM |= 0x08;           // /CS=1

```

```

return result;
}
    
```

FIGURE 2-2: BYTE WRITE SEQUENCE



Write one byte

- 1) Set CS=0
- 2) Send WREN \$06 (enable write latch)
- 3) Set CS=1
- 4) Set CS=0
- 5) Send WRITE \$02
- 6) Send address 23-16
- 7) Send address 15-8
- 8) Send address 7-0
- 9) Send data to be written at that address
 As long as address within same 256-byte page, repeat step 9
- 10) Set CS=1
- 11) Set CS=0
- 12) Send RDSR \$05 (read status register)
- 13) Receive status, bit 0 is WIP (write in progress)
- 14) Set CS=1
- 15) Repeat steps 11-14 until WIP=0 <<takes 6 ms for 1 to 256 bytes>>

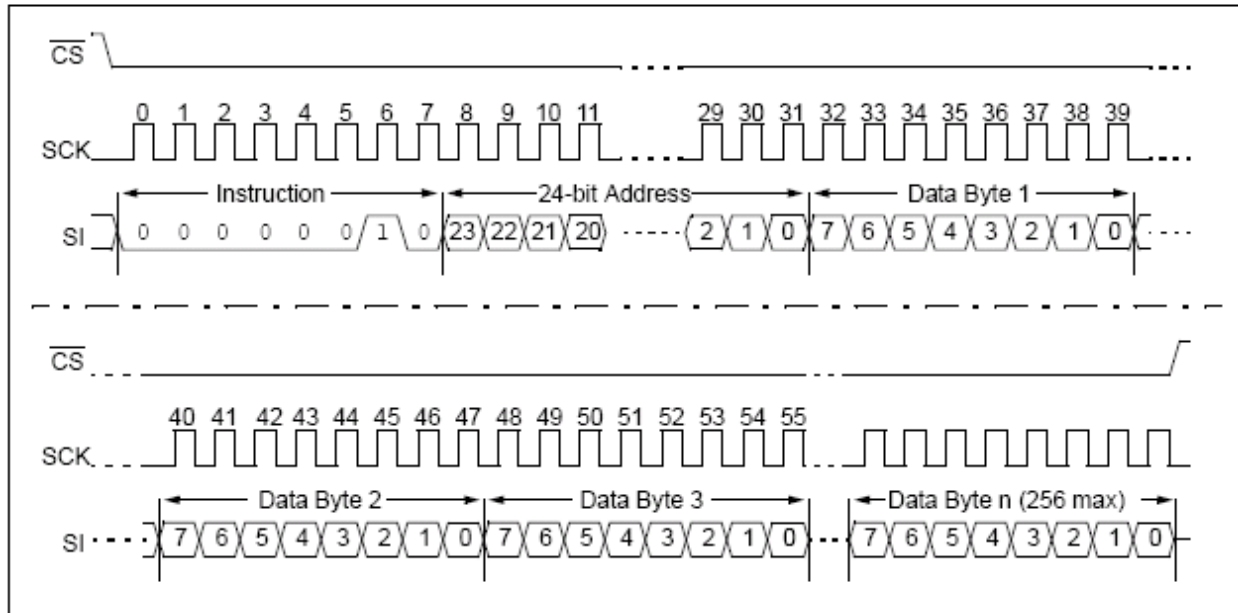
Lab 2h Estimate bandwidth?

- Simple 8-bit data stream (8-bit ADC)
- Unpacked 10-bit / 12-bit ADC
- Packed 10-bit / 12-bit ADC

Is the ADC sampling running simultaneously file storage?

- Waiting for ADC to complete
- Waiting for time to sample
- Waiting for EEPROM write to complete

FIGURE 2-3: PAGE WRITE SEQUENCE



Review Lab 1f, look at
 Data flow graph
 Call graph
 Linked allocation
 Directory
 Free space management

Review Lab 2h Data flow graph

Internal Fragmentation {Specific to Lab 2h}

- Spaced used by OS due to how the disk is organized
 - Storage used for the directory, {256 bytes of directory},
 - Storage used to manage free space that can not be allocated for data
 - Storage used to allow OS to access files (tables, pointers, counters)
 - {Every block uses 3 bytes for size and link to next}
- Caused by allocation method itself (e.g., must allocate an entire block)
 - 50% of the last block (on average) for each file
- Caused by the limited number of files allowed in the directory
 - If the directory is full, but there are free blocks,
 - then the wasted space is internal fragmentation

External Fragmentation

Let n be the total number of allocatable data bytes,
 $\{n=(number\ of\ free\ blocks)*253\}$
 I.e., n is the total available data space that is free

Let **m** be the largest file size that can be allocated

External fragmentation if **m < n**

No external fragmentation if **m equals n**

{either None in Lab 2h if 24/32-bit file size

50% in Lab 2h if 16-bit file size}

More on file systems will be presented in recitation on Friday