

**Lecture 10 objectives**

- Review periodic interrupts with OC
- SPI interface
- DAC

**Why Periodic Interrupts?**

- **Perform software tasks at a fixed rate. E.g., sound**
- **System with multiple loosely-coupled tasks**

**Sequence of operations for OC3**

address	msb																lsb	Name
<b>\$0044</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	<b>TCNT</b>	
\$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	
<b>\$0056</b>	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	<b>TC3</b>	
\$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	<b>15</b>	<b>14</b>	<b>13</b>	<b>12</b>	<b>11</b>	<b>10</b>	<b>9</b>	<b>8</b>	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	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
\$0240	PT7	PT6	PT5	PT4	PT3	PT2	PT1	PT0	PTT
\$0242	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0	DDRT
\$0046	<b>TEN</b>	TSWAI	TSBCK	TFFCA	0	0	0	0	TSCR1
\$004D	TOI	0	0	0	TCRE	<b>PR2</b>	<b>PR1</b>	<b>PR0</b>	TSCR2
\$0040	IOS7	IOS6	IOS5	IOS4	<b>IOS3</b>	IOS2	IOS1	IOS0	TIOS
\$004C	C7I	C6I	C5I	C4I	<b>C3I</b>	C2I	C1I	C0I	TIE
\$004E	C7F	C6F	C5F	C4F	<b>C3F</b>	C2F	C1F	C0F	TFLG1
\$004F	TOF	0	0	0	0	0	0	0	TFLG2
\$0048	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4	TCTL1
\$0049	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0	TCTL2
\$004A	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A	TCTL3
\$004B	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A	TCTL4

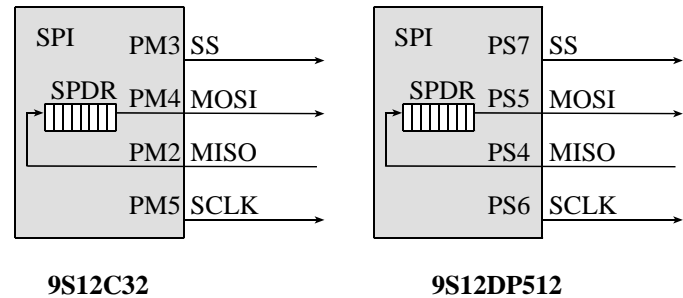
Table 9.8. 9S12 registers used for timer overflow, input capture, and output compare.

- 1) **reset**      **I=1, PC=> start12**
- 2) **start12**    **sets SP, initializes globals, calls your main**
- 3) **ritual**      **enable TCNT**  
`TSCR1= 0x80; //9S12DP512`
- 4) **ritual**      **set TCNT rate**  
`TSCR2= 0x03; PACTL = 0; //9S12DP512`
- 5) **ritual**      **activate OC3**  
`TIOS |= 0x08;`
- 6) **ritual**      **arms, C3I=1**  
`TIE |= 0x08; //9S12DP512`
- 7) **ritual**      **specify time for first interrupt**  
`TC3 = TCNT+50;`
- 8) **ritual**      **enables, I=0**  
`asm cli`
- 9) **main**        **executes foreground actions**
- 10) **trigger**    **C3F=1, set when TCNT equals TC3**
- 11) **switch**    **from foreground to background**  
`finish instruction`  
`push PC,Y,X,A,B,CCR`

```

I=1 (disable)
PC = [FFE8] (address of OC3han)
12) Executes 6 instructions (17 cycles) in the debugger
    bsr   ISRHandler   [4]
ISRHandler: pulx      [3]
           ldy   constant,X  [4]
           cpy   #$FFFF    [2]
           beq   BadVector  [1]
           jmp   ,Y         [3]
13) TC3han   ISR must ack, C3F=0
    TFLG1 = 0x08;
14) TC3han   ISR specifies time for next interrupt
    TC3 = TC3+10000;
15) TC3han   execute something useful
16) TC3han   ISR returns
    rti   pulls CCR,B,A,X,Y,PC
Interrupt period = TCNTperiod*10000
    
```

**7.7. Synchronous Transmission and Receiving using SPI**  
**7.7.1. SPI Fundamentals**



When software writes to the data register, this 8-bit register is serially shifted eight bit positions by the SCLK clock

**Common control features for the SPI module include:**

- a baud rate control register, used to select the transmission rate,
- a mode bits in the control register to select
  - master versus slave
  - clock polarity
  - clock phase
- interrupt arm bit

Common status bits for the SPI module include:  
 SPIF, transmission complete

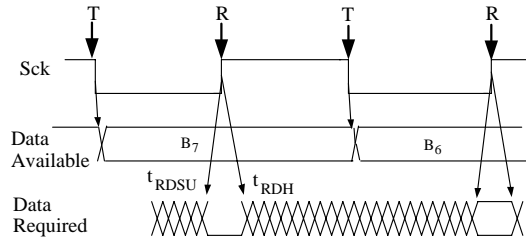


Figure 7.44. Synchronous serial timing showing the data available interval overlaps the data required interval.

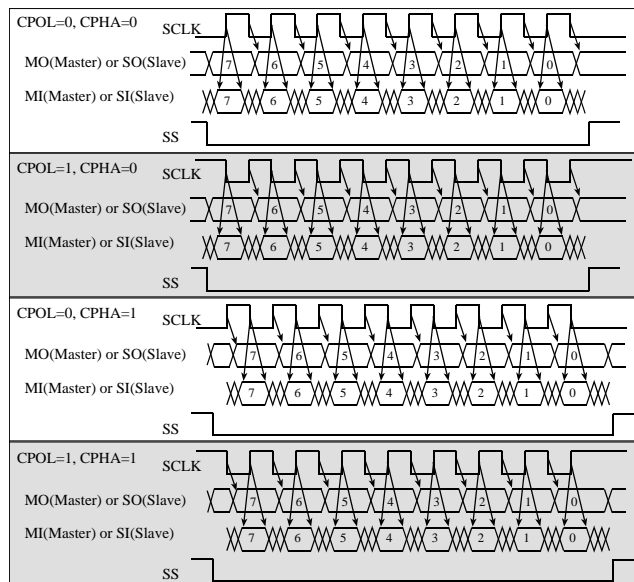


Figure 7.45. Synchronous serial modes of the Motorola SPI interface.

7.7.5. 9S12 SPI Details (see S12SPIV3.pdf)

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

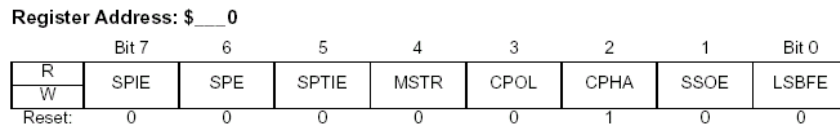


Figure 3-1 SPI Control Register 1 (SPICR1)

SPE — SPI System Enable

0 = SPI internal hardware is initialized and SPI system is in a low-power disabled state.

1 = four pins in above table are dedicated to the SPI function

MSTR — SPI Master/Slave Mode Select

0 = Slave mode

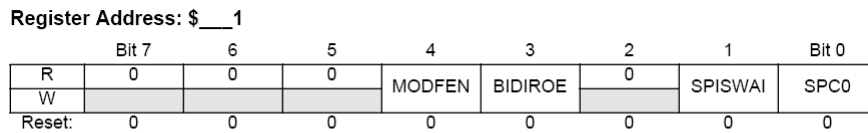
1 = Master mode

CPOL, CPHA — SPI Clock Polarity, Clock Phase

These two bits are used to specify the clock format to be used in SPI operations.

SSOE — Slave Select Output Enable

The  $\overline{SS}$  output feature is enabled only in the master mode by asserting the SSOE and the corresponding data direction bit for that pin.

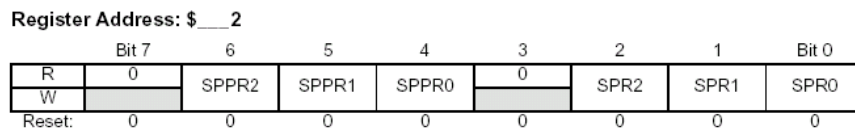


 = Reserved

Figure 3-2 SPI Control Register 2 (SPICR2)

SSOE=0, MODFEN=0 to make  $\overline{SS}$  a regular output

The SPIBR register determines the transfer rate.



 = Reserved

Figure 3-3 SPI Baud Rate Register (SPIBR)

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) \cdot 2^{(\text{SPR} + 1)}$$

$$\text{Baud Rate} = \text{BusClock} / \text{BaudRateDivisor}$$

Register Address: \$\_\_3

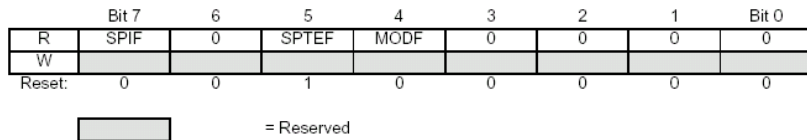


Figure 3-4 SPI Status Register (SPISR)

SPIF — SPI Interrupt Request

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 an access (read or write) to the SPI data register.

Register Address: \$\_\_5

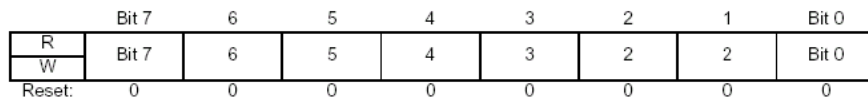
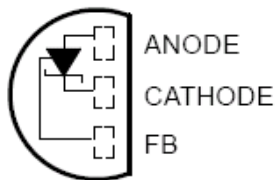
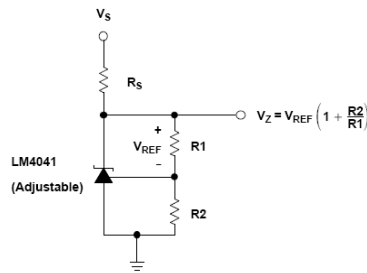


Figure 3-5 SPI Data Register (SPIDR)

LM4041CILPR (LM4041C.pdf)



$$R_s = \frac{(V_s - V_z)}{(I_L + I_z)}$$



What is  $V_s$ ?

What is  $V_{REF}$ ? (page 10)

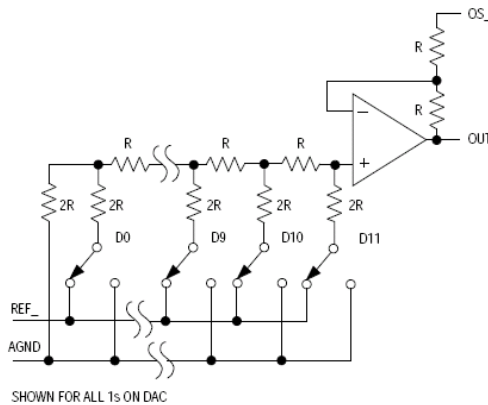
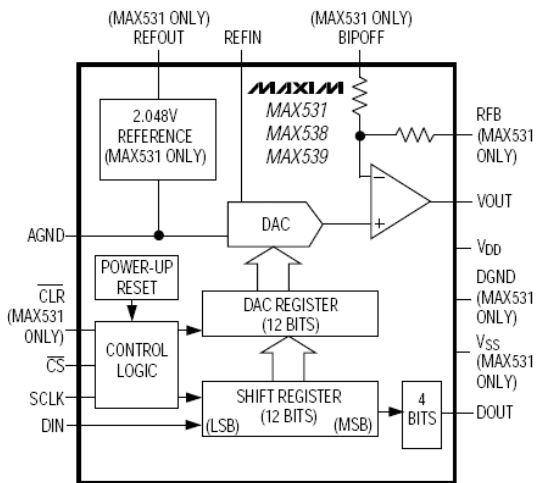
Design for  $V_z$  about 2.5V (choose R1,R2)

How much current needed for  $V_z$  output? (Max539 data sheet)

How much current needed for LM4041? (page 10)

Choose  $R_s$  to provide the current

Interface Max539 12 bit DAC



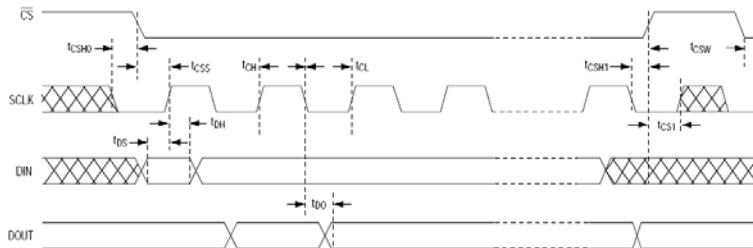
INPUT	OUTPUT	Input (n)	Output (V <sub>out</sub> )
1111 1111 1111	$+2 (V_{REFIN}) \frac{4095}{4096}$	4095	4.990 V
1000 0000 0001	$+2 (V_{REFIN}) \frac{2049}{4096}$	2049	2.498 V
1000 0000 0000	$+2 (V_{REFIN}) \frac{2048}{4096} = +V_{REFIN}$	2048	2.497 V
1000 0000 0000	$+2 (V_{REFIN}) \frac{2047}{4096}$	2047	2.496 V
0111 1111 1111	$+2 (V_{REFIN}) \frac{2047}{4096}$	2	0.003 V
0000 0000 0001	$+2 (V_{REFIN}) \frac{1}{4096}$	1	0.002 V
0000 0000 0000	0V	0	0.001 V

Show how to look up Max539 data sheet to see

Timing diagram  
Setup and hold

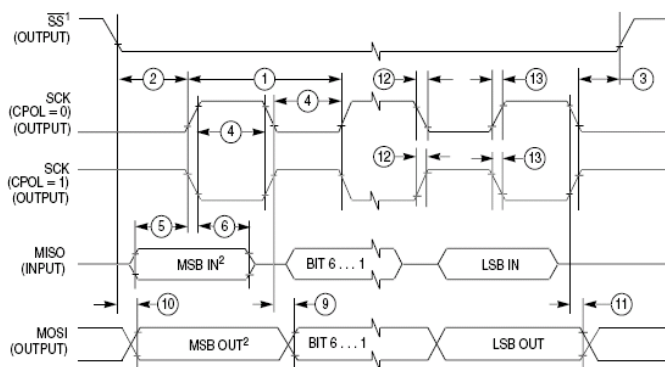
As with any SPI interface, there are basic interfacing issues to consider.

1. *Word size.* In this case we need to transmit 12 bits (two 8-bit frames).
2. *Bit order.* The Max539 requires the most significant bits first.
3. *Clock phase, clock polarity.* There are two issues to resolve. Since the Max539 samples its serial input data on the rising edge of the clock, the SPI must change 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 Max539. t<sub>DS</sub> setup time is 45ns, t<sub>DH</sub> hold time 0ns. The t<sub>CH</sub> and t<sub>CL</sub> are both 35ns



Show how to look up 9S12DP512 data sheet to see  
9S12DP512DGV1.pdf, Section A.7, page 113

Timing diagram  
Data Valid after SCK time, t<sub>o</sub> = 30 ns  
Data hold after SCK time, t<sub>11</sub> = 20 ns



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)

```
// 9S12DP512 SPI0 interface to Max539
// PS6 (out) SCLK synchronous clock
```

```
// PS5 (out) MOSI serial data output
// PS7 (out) CS used to latch data into Max539
// PS4 (in) is associated with SPI0, but not used
void DAC_Init(void){
    1) make PS7, PS6, PS5 outputs, PS4 input
       DDRS
    2) enable SPI, no interrupts, master, CPOL, CPHA
       SPI0CR1
    3) set up PS7 as a regular output
       SSOE=0, MODFEN=0 SPI0CR1, SPI0CR2
    4) set the baud rate, SPI0BR
    5) make PS7=CS high
#define CS PTS_PTS7
```

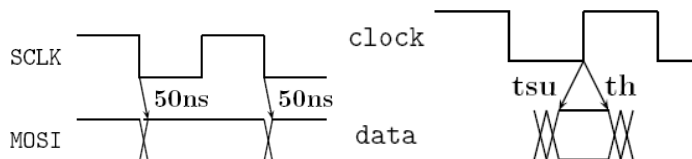
To transmit one SPI frame

```
1) wait for SPTEF to be 1, SPI0SR
2) write 8-bit data to SPI0DR
3) wait for SPIF to be 1, SPI0SR
4) clear the SPIF flag by reading the data
   dummy = SPI0DR; // clear SPIF
```

To send 12-bit data to the DAC

```
void DAC_Out(unsigned short data){
    1) set PS7=CS low
    2) transmit most significant 8-bit data to the DAC
    3) transmit least significant 8-bit data to the DAC
    4) set PS7=CS high
```

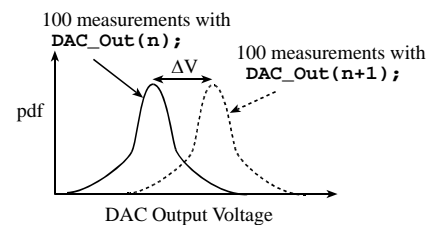
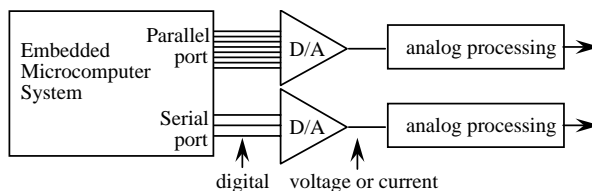
**Hints about the HW question**



**Falling edge + 50ns must occur before the rising edge-t<sub>su</sub>**

## 11.4. Digital to Analog Converters

### 11.4.1. DAC Converter Parameters



*DAC converters provide analog output for our embedded microcomputer systems.*

The DAC *precision* is the number of distinguishable DAC outputs (e.g., 4096 alternatives, 12 bits).

The DAC *range* is the maximum and minimum DAC output (0 to 5V).

The DAC *resolution* is the smallest distinguishable change in output. ( $5V/4096 = 1.2 \text{ mV}$ )

$$\text{Range(volts)} = \text{Precision(alternatives)} \cdot \text{Resolution(volts)}$$

The DAC *accuracy* is  $(\text{Actual} - \text{Ideal}) / \text{Ideal}$

#### 12-bit DAC

$$\text{Ideal } V_{\text{out}} = 5V * (n/4095)$$

$$\text{Ideal } V_{\text{out}} = 5V * (n/4096)$$