- **The 9S12 Serial Peripheral Inteface (SPI)**
- Huang Section 10.2 through 10.6
- SPI Block User Guide

The 9S12 Serial Peripheral Interface (SPI)

• The 9S12 has a Synchronous Serial Interface

• On the 9S12 it is called the Serial Peripheral Interface (SPI)

• Information on the SPI can be found in the SPI Block User Guide.

• If an 9S12 generates the clock used for the synchronous data transfer it is operating in Master Mode.

• If an 9S12 uses and external clock used for the synchronous data transfer it is operating in Slave Mode.

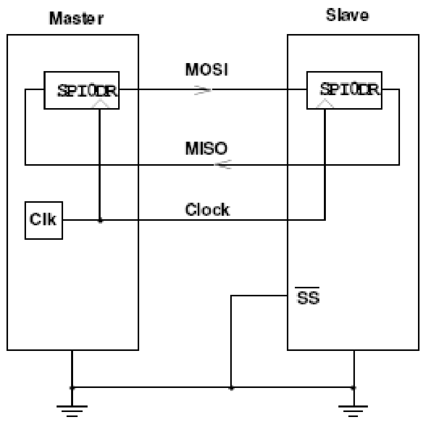
• If two 9S12's talk to each other using their SPI's one must be set up as the Master and the other as the Slave.

• The output of the Master SPI shift register is connected to the input of the Slave SPI shift register over the Master Out Slave In (MOSI) line.

• The input of the Master SPI shift register is connected to the output of the Slave SPI shift register over the Master In Slave Out (MISO) line.

• After 8 clock ticks, the data originally in the Master shift register has been transferred to the slave, and the data in the Slave shift register has been transferred to the Master.

Synchronous Serial Communications



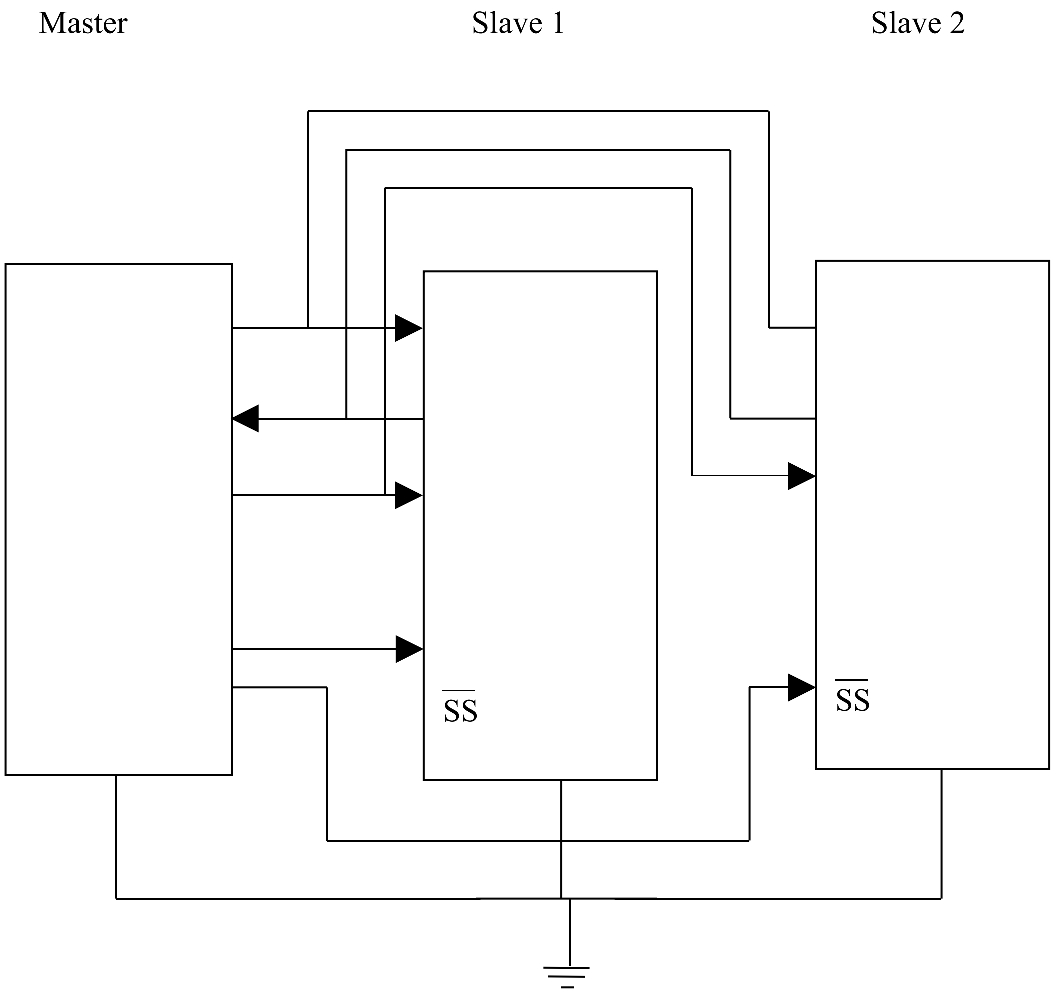
Use of Slave Select with the 9S12 SPI

• A master 9S12 can talk with more than one slave 9S12's

• A slave 9S12 uses its Slave Select (SS) line to determine if it is the one the master is talking with

• There can only be one master 9S12, because the master 9S12 is the device which generates the serial clock signal.

SYNCHRONOUS SERIAL COMMUNICATIONS



With select lines, one master can communicate with more than one slave

Using the 9S12 SPI

• In synchronous serial communications, one device talks to another using a serial data line and a serial clock.

• There are a number of decisions to be made before communication can begin.

• For example
– Is the 9S12 operating in master or slave mode?
– Is the serial data sent out most significant bit (MSB) first, or least significant bit (LSB) first?
– How many bits are sent in a single transfer cycle?
– Is the data valid on the rising edge or the falling edge of the clock?
– Is the data valid on the first edge or the second edge of the clock?
– What is the speed of the data transfer (how many bits per second)?
– Are there two uni-directional data lines or one bi-directional data line?

• The 9S12 SPI is very versatile, and allows you to program all of these parameters.

• The 9S12 SPI has 5 registers to set up and use the SPI system.

SPI0CR1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBF	0x00D8
SPI0CR2	0	0	0	MODFEN	BRIDIROE	0	SPSWAI	SPC0	0x00D9
SPI0BR	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0	0x00DA
SPI0SR	SPIF	0	SPTEF	MODF	0	0	0	0	0x00DB
SPI0DR	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	B IT 2	BIT 1	BIT 0	0x00DD

SPIE: SPI interrupt enable bit
0 = Disabled
1 = Enabled

SPE: SPI system enable bit
0 = Disabled
1 = Enabled

MSTR: SPI master/slave mode select bit
0 = Slave
1 = Master

CPOL: SPI clock polarity bit
0 = Active high clocks; SCK idle low
1 = Active low clocks; SCK idle high

SPHA: SPI clock phase bit
0 = The SCK edge is issued one-half cycle into transfer operation.
1 = The SCK edge is issued at the beginning of transfer operation.

SSOE: Slave select output enable bit

The $\overline{SS}$ output feature is enabled only in master mode by asserting the SSOE bit.

LSBF: SPI least significant bit first enable bit
0 = Most significant bit first
1 = Least significant bit first

BIDIROE: Output enable in the bidirectional mode of operation
0 = Disabled
1 = Enabled

SPC0: Serial pin control bit
With the MSTR , this bit enables bidirectional pin configuration

SPC0	MSTR	MISO	MOSI
0	0	slave out	slave in
0	1	master in	master out
1	0	slave I/O	-
1	1	-	master I/O

SPPR2 – SPPR0: SPI baud rate preselection bits
SPR2 – SPR0: SPI baud rate selection bits
BaudRate = [Bus Clock]/[(SPPR+1)x2$^{(SPR+1)}$]

SPIF: Bit is set after the eight SCK cycles in a data transfer, and is cleared by reading the SPI0SR register.
0 = Transfer not yet complete
1 = new data copied to SPI0DR

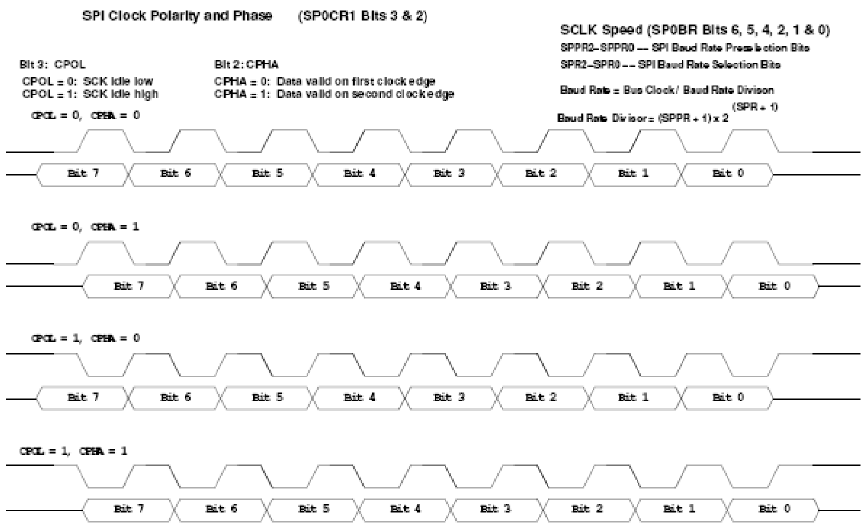
Setting up the 9S12 SPI Clock Mode

• You can program the SPI clock to determine the following things:

Is the data valid on the first or the second edge of the clock (clock phase)?

Is the clock idle high or idle low (clock polarity)?

• This setup is done in the SPI0CR0 register.


SPI Clock Polarity and Phase (SP0CR1 Bits 3 & 2)
SCLK Speed (SP0BR Bits 6, 5, 4, 2, 1 & 0)
SPPR2–SPPR0 — SPI Baud Rate Preselection Bits
SPR2–SPR0 – – SPI Baud Rate Selection Bits
Bit 3: CPOL
CPOL = 0: SCK idle low
CPOL = 1: SCK idle high
Bit 2: CPHA
CPHA = 0: Data valid on first clock edge
CPHA = 1: Data valid on second clock edge
Baud Rate = Bus Clock / Baud Rate Divison
Baud Rate Divisor = (SPPR + 1) x 2^(SPR + 1)
CPOL = 0, CPHA = 0
CPOL = 0, CPHA = 1
CPOL = 1, CPHA = 0
CPOL = 1, CPHA = 1
Bit 7
Bit 6
Bit 5
Bit 4
Bit 3
Bit 2
Bit 1
Bit 0

Setting up the 9S12 SPI Clock Mode

• The speed of the 9S12 clock is set up in the SPI0BR register.

• The clock speed is set only if the 9S12 is being used as a master.

• The possible clock speeds (for a 24 MHz E-clock) are:

SPPR2	SPPR1	SPPR0	SPPR2	SPR1	SPR0	E clock Divisor	Frequency at bus clock=24 MHz
0	0	0	0	0	0	2	12.0 MHz
0	0	0	0	0	1	4	6.0 MHz
0	0	0	0	1	0	8	3.0 MHz
0	0	0	0	1	1	16	1.5 MHz
0	0	0	1	0	0	32	750 kHz
0	0	0	1	0	1	64	375 kHz
0	0	0	1	1	0	128	187.5 kHz
0	0	0	1	1	1	256	93.75 kHz
0	0	1	0	0	0	4	6.0 MHz
0	0	1	0	0	1	8	3.0 MHz
0	0	1	0	1	0	16	1.5 MHz
0	0	1	0	1	1	32	750 kHz
0	0	1	1	0	0	64	375 kHz
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Using the 9S12 Serial Peripheral Interface

Things to set up when using the 9S12 SPI subsystem

• Enable SPI

• Master or Slave?
– Master generates clock for data transfers; slave uses master's clock

• MSB first or LSB first?
– Normally, MSB first

• Clock Polarity
– Clock idle low or clock idle high?

• Clock Phase
– Data valid on first clock edge or second clock edge?

• Clock Speed (set by Master)

• Generate interrupt after data transfered?

• Bidirectional Mode
Use the following registers: SPI0CR1, SPI0CR2

How to set up the 9S12 Serial Peripheral Interface

1. Enable SPI (SPE bit of SPI0CR1)

2. Clock phase and polarity set to match device communicating with

3. Select clock polarity – CPOL bit of SPI0CR1
• CPOL = 0 for clock idle low
• CPOL = 1 for clock idle high

4. Select clock phase – CPHA bit of SPI0CR1
• CPHA = 0 for data valid on first clock edge
• CPHA = 1 for data valid on second clock edge

5. Select master or slave MSTR bit of SPI0CR1
• Will be master when talking to devices such as D/A, A/D, clock, etc.
• May be slave if talking to another microprocessor

6. If you want to receive interrupt after one byte transferred, enable interrupts with SPIE bit of SPI0CR1
• Normally master will not use interrupts – transfers are fast enough that you could normally wait for transfer to complete
• Will often use interrupts when configured as a slave – you will get interrupt when master sends data

7. Configure LSBF of SPI0CR1 for MSB first (LSBF = 0) or LSB first (LSBF = 1)
• For most devices, use MSB first

8. Configure for uni-directional mode (bit SPC0 = 0) or bi-directional mode (SPC0 = 1) in SPI0CR2
• Bidirectional mode (SPC0 = 1 in SPI0CR2) used for three-wire communication.
• When in bidirectional mode, the BIDIROE bit of SPI0CR2 determines if the data line is input or output.

Master Mode:

1. Set clock rate – SPPR2:0 and SPR2:0 bits of SPI0BR
• Normally select clock at highest rate compatible with slave

2. If using bidirectional mode, MOSI pin is used for data (now called MOMI, or Master Out Master In).

3. MISO automatically configured as input by choosing master mode

4. Configure some way to select slave(s) – probably SS if only one slave; other I/O bits if multiple slaves

5. Start data transfer by writing byte to SPI0DR

6. After transfer complete (8 clock cycles), SPIF bit of SPI0SR set.
• If writing data to slave, can send next byte to SPI0DR
• If reading data from slave, can read data from SPI0DR

7. Set up SSOE of SPI0CR1
• SSOE = 0 if you want to control SS yourself (to be able to send more than one byte with SS low)
• SSOE = 1 and MODFEN = 1 if you want to SS controlled automatically
(SS will be active for one byte at a time)

Slave Mode:

1. No need to set clock speed – slave accepts data at rate sent by master (up to 12 MHz)

2. If using bidirectional mode, MISO pin is used for data (now called SISO, or Slave In Slave Out).

3. No need to Make MOSI, SCLK, and SS inputs – this is done automatically when configuring 9S12 as slave
• If receiving data from master, wait until SPIF flag of SPI0SR set (or until SPI interrupt received), then read data from SPI0DR
• If sending data to master, write data to SPI0DR before master starts transfer

A C program to use the 9S12 in master mode

```
#include "hcs12.h"
main()
{
        char tmp;

        /* Use Bit 0 of Port A for Slave Select */
        DDRA = 0xff;                        /* Port A output */

        /***************************************************************
        * SPI Setup
        ****************************************************************/
        SPI0CR1 = 0x50; /* 0 1 0 1 0 0 0 0
                           | | | | | | | |
                           | | | | | | | \____ MSB first
                           | | | | | | \______ Do not use SS to automatically
                           | | | | | |         select slave
                           | | | | | \________ 0 phase (data on 1st clock edge)
                           | | | | \_________ 0 polarity (clock idle low)
                           | | | \___________ Master mode
                           | | \_____________ No interrupts on transmit
                           | \_______________ Enable SPI
                           \_________________ No interrupts
                            */

        SPI0CR2 = 0x00;                      /* Normal (not bi-directional) mode */
        SPI0BR = 0x00;                      /* 12 MHz SPI clock */

        /***************************************************************
        * End of SPI Setup
        ****************************************************************/

        PORTA = PORTA & ~0x01;              /* Select slave */
        while ((SPI0SR & 0x20) == 0) ;      /* Wait for transmit register empty */
        SPI0DR = 'h';                       /* Send 'h' */
        while ((SPI0SR & 0x80) == 0) ;      /* Wait for transmission to complete */
        tmp = SPI0DR;                       /* Clear SPIF flag */
        PORTA = PORTA | 0x01;               /* Deselect slave */

        PORTA = PORTA & ~0x01;
        while ((SPI0SR & 0x20) == 0) ;      /* Wait for transmit register empty */
        SPI0DR = 'e';                       /* Send 'e' */
        while ((SPI0SR & 0x80) == 0) ;      /* Wait for transmission to complete */
        tmp = SPI0DR;                       /* Clear SPIF flag */
        PORTA = PORTA | 0x01;
        PORTA = PORTA & ~0x01;
        while ((SPI0SR & 0x20) == 0) ;       /* Wait for transmit register empty */
        SPI0DR = ’l’;                         /* Send ’l’ */
        while ((SPI0SR & 0x80) == 0) ;       /* Wait for transmission to complete */
        tmp = SPI0DR;                         /* Clear SPIF flag */
        PORTA = PORTA | 0x01;

        PORTA = PORTA & ~0x01;
        while ((SPI0SR & 0x20) == 0) ;       /* Wait for transmit register empty */
        SPI0DR = ’l’;                         /* Send ’l’ */
        while ((SPI0SR & 0x80) == 0) ;       /* Wait for transmission to complete */
        tmp = SPI0DR;                         /* Clear SPIF flag */
        PORTA = PORTA | 0x01;

        PORTA = PORTA & ~0x01;
        while ((SPI0SR & 0x20) == 0) ;       /* Wait for transmit register empty */
        SPI0DR = ’o’;                         /* Send ’o’ */
        while ((SPI0SR & 0x80) == 0) ;       /* Wait for transfer to finish */
        tmp = SPI0DR;                         /* Clear SPIF flag */
        PORTA = PORTA | 0x01;
}
```

Here is a version using a loop to transfer data:

```
#include "hcs12.h"
main()
{
        const char data[] = "hello";
        int i;
        char tmp;

        /* Use Bit 0 of Port A for Slave Select */
        DDRA = 0xff;                    /* Port A output */

        /************************************************************
        * SPI Setup
        ************************************************************/
        SPI0CR1 = 0x50; /* 0 1 0 1 0 0 0 0
                           | |  | | | | | |
                           | |  | | | | | \____ MSB first
                           | |  | | | | \______ Do not use SS to automatically
                           | |  | | | |          select slave
                           | |  | | | \________ 0 phase (data on 1st clock edge)
                           | |  | | \__________ 0 polarity (clock idle low)
                           | |  | \___________ Master mode
                           | |  \____________ No interrupts on transmit
                           | \______________ Enable SPI
                           \________________ No interrupts
                    */
        SPI0CR2 = 0x00;             /* Normal (not bi-directional) mode */
        SPI0BR = 0x00;              /* 12 MHz SPI clock */
        /************************************************************
        * End of SPI Setup
        ************************************************************/

        for (i=0; ; i++)
        {
                if (data[i] == '\0') break;             /* Exit loop at end of string */
                PORTA = PORTA & ~0x01;                  /* Select slave */
                while ((SPI0SR & 0x20) == 0) ;          /* Wait for transmit register empty */
                SPI0DR = data[i];                       /* Send data */
                while ((SPI0SR & 0x80) == 0) ; /* Wait for transmission to complete */
                tmp = SPI0DR;                           /* Read SPI0DR to clear SPIF */
                PORTA = PORTA | 0x01;                   /* Deselect slave */
        }
}
```