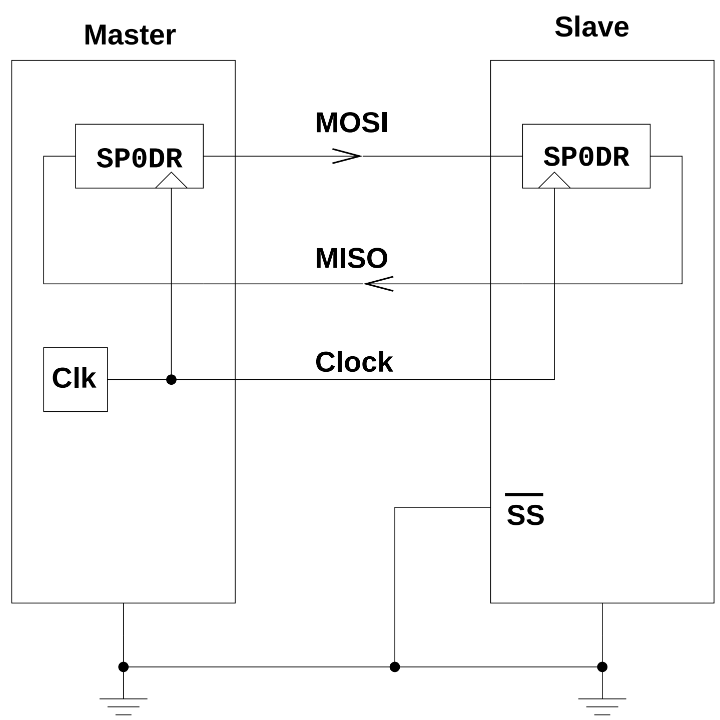
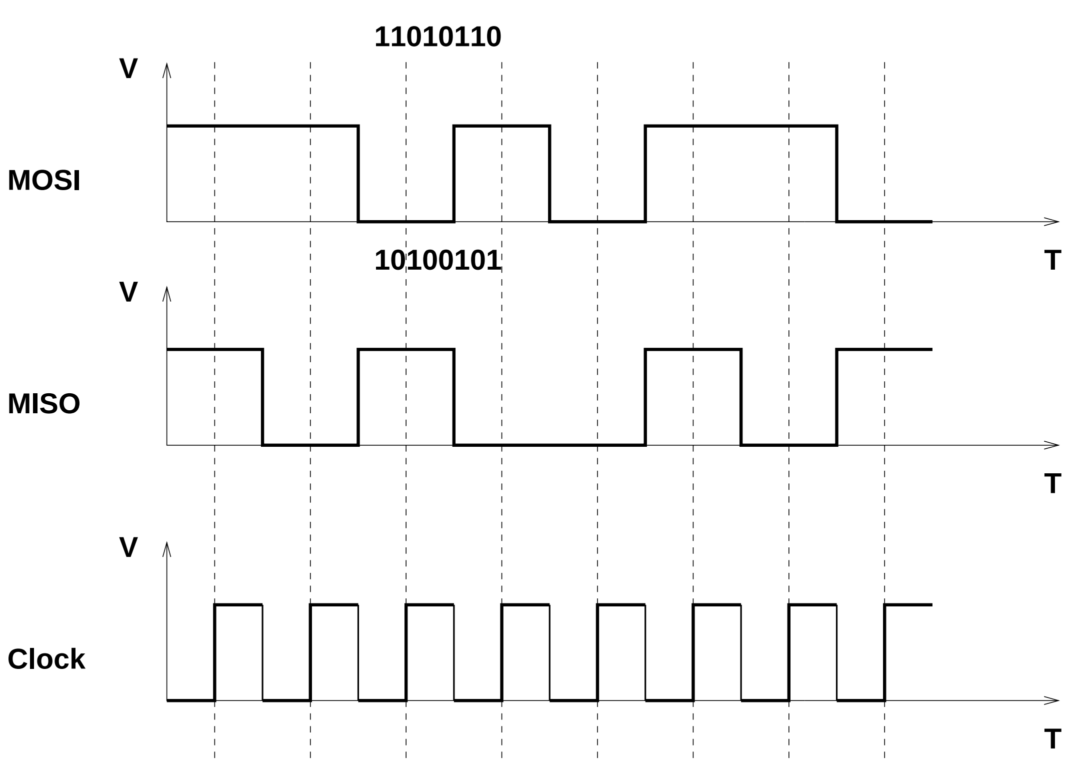
The HC12 Serial Peripheral Interface (SPI)

- The HC12 has a Synchronous Serial Interface
- On the HC12 it is called the Serial Peripheral Interface (SPI)
- If an HC12 generates the clock used for the synchronous data transfer it is operating in Master Mode.
- If an HC12 uses and external clock used for the synchronous data transfer it is operating in Slave Mode.
- If two HC12'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 transfered to the slave, and the data in the Slave shift register has been transfered to the Master.

Synchronous Serial Communications

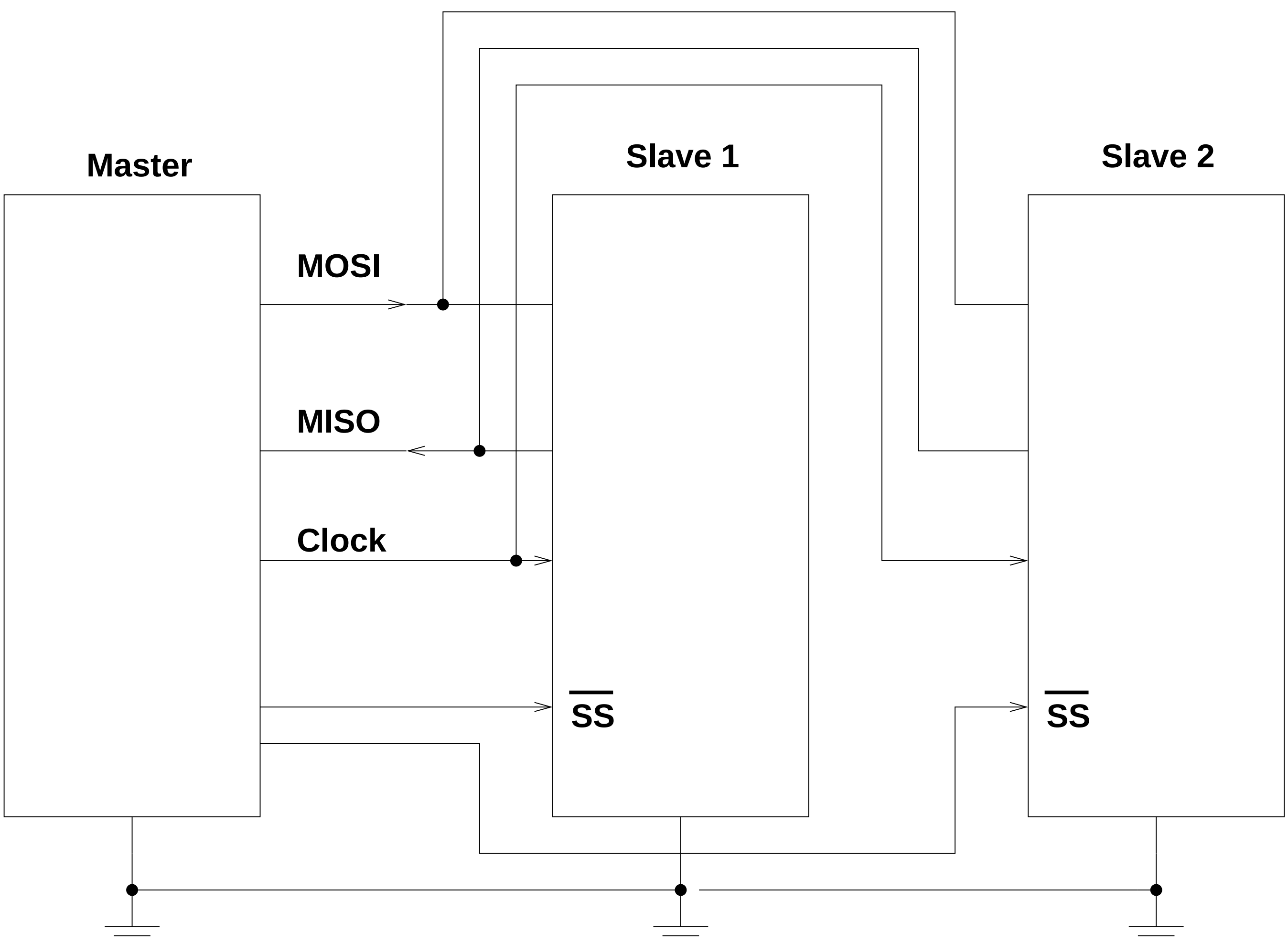




Use of Slave Select with the HC12 SPI

- A master HC12 can talk with more than one slave HC12's
- A slave HC12 uses its Slave Select (SS) line to determine if it is the one the master is talking with
- There can only be one master HC12, because the master HC12 is the device which generates the serial clock signal.

Synchronous Serial Communications

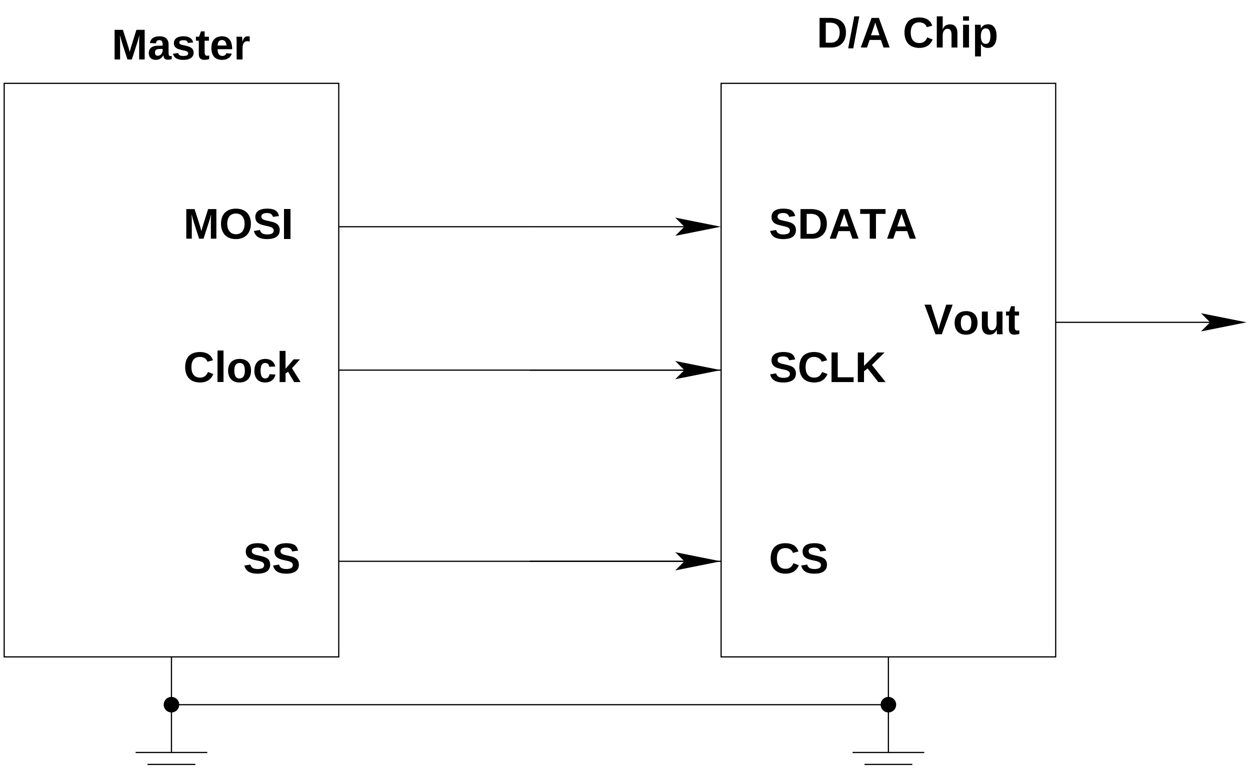


With select lines, one master can communicate with more than one slave

Using the HC12 SPI with other devices

- The HC12 can communicate with many types of devices using its SPI
- For example, consider a D/A (Digital-to-Analog) Converter
- The D/A converter has three digital lines connected to the HC12:
  - Serial Data
  - Serial Clock
  - Chip Select
- The HC12 can send a digital number to the D/A converter. The D/A converter will convert this digital number to a voltage.

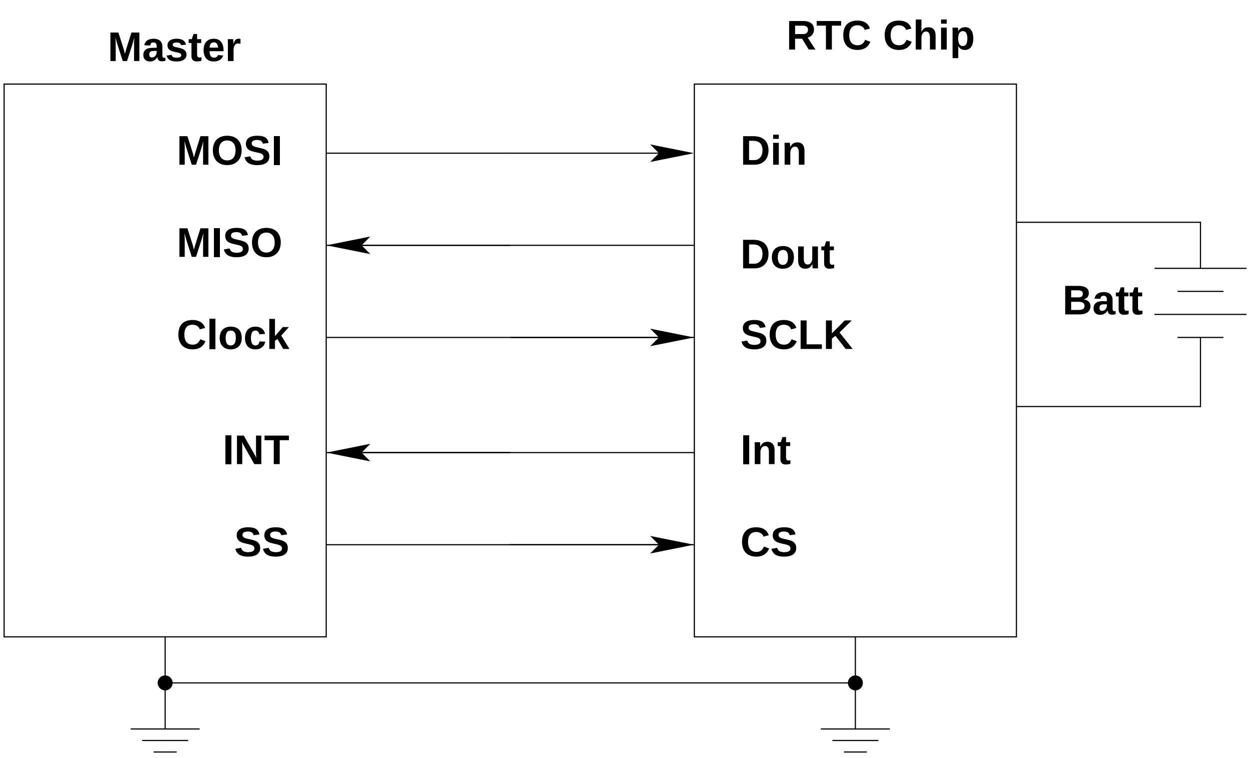
SPI Communication with a D/A Converter



Using the HC12 SPI with other devices

- Another type of device the HC12 can talk to is a Real Time Clock (RTC)
- An RTC keeps track of the time (year, month, day, hour, minute, second)
- An RTC can be programmed to generate an alarm (interrupt) at a particular time (07:00), or can generate a periodic interrupt at a regular interval (once a second, once an hour, etc.)
- The HC12 initially tells the RTC what the correct time is.
- The RTC keeps track of time from then on.

SPI Communication with a Real Time Clock



- In a system, an HC12 can communicate with many different devices over its SPI interface.

Using the HC12 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 HC12 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)?
- The HC12 SPI is very versatile, and allows you to program all of these parameters.
- The HC12 SPI has 6 registers to set up and use the SPI system.

SP0CR1	**SPIE**	**SPE**	SWOM	**MSTR**	**CPOL**	**CPHA**	**SSOE**	**LSBF**	**0x00D0**
SP0CR2	0	0	0	0	PUPS	RDS	0	**SPC0**	**0x00D1**
SP0BR	0	0	0	0	0	**SPR2**	**SPR1**	**SPR0**	**0x00D2**
SP0SR	**SPIF**	WCOL	0	MODF	0	0	0	0	**0x00D3**
SP0DR	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**0x00D5**
DDRS	**DDS 7**	**DDS6**	**DDS5**	**DDS4**	**DDS3**	**DDS2**	**DDS1**	**DDS0**	**0x00D7**

Setting up the HC12 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 `SP0CR0` register.

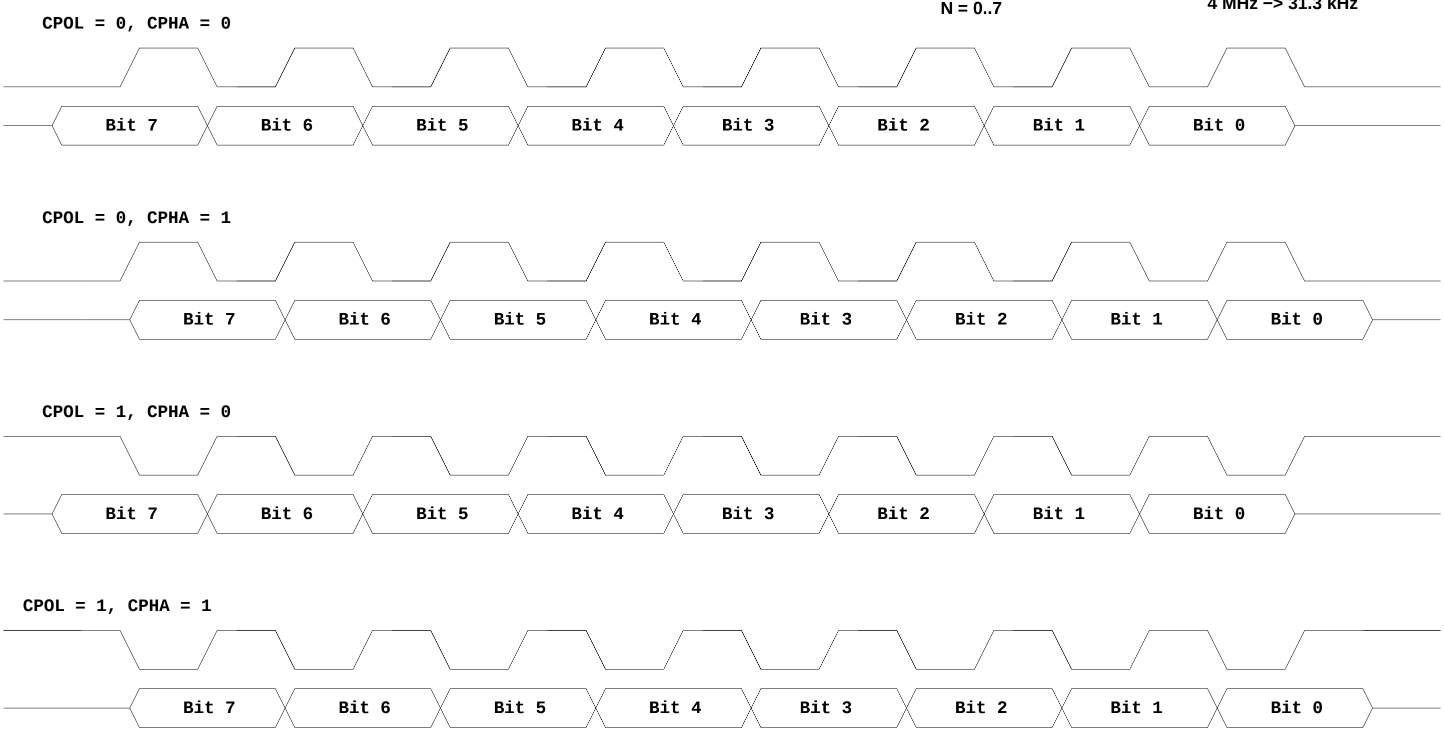
SPI Clock Polarity and Phase (SP0CR1 Bits 3 & 2)

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

SCLK Speed (SP0BR Bits 0, 1 & 2)

SP0BR 2:0	Divide E by	Speed w/8MHz E
N	$2^{(N+1)}$	8 MHz / $2^{(N+1)}$
N = 0..7		4 MHz –> 31.3 kHz



Setting up the HC12 SPI Clock Mode

- The speed of the HC12 clock is set up in the `SP0BR` register.
- The clock speed is set only if the HC12 is being used as a master.
- The possible clock speeds (for an 8 MHz E-clock) are:

SPR2	SPR1	SPR0	E Clock Divisor	Frequency at E clock = 8 MHz
0	0	0	2	4.0 MHz
0	0	1	4	2.0 MHz
0	1	0	8	1.0 MHz
0	1	1	16	500.0 kHz
1	0	0	32	250.0 kHz
1	0	1	64	125.0 kHz
1	1	0	128	62.5 kHz
1	1	1	256	31.25 kHz

Programming the DDRS Register when using the HC12 SPI

- The HC12 uses bits 7, 6, 5 and 4 of Port S for its SPI data lines
- If a pin of Port S needs to be set up as output in order to use the HC12 SPI, you need to write a 1 to that bit of DDRS.
  - In Master Mode, you need to write a 1 to bits 7 (Slave Select), 6 (serial clock) and 5 (MOSI).
  - In Slave Mode, you need to write a 1 to bit 4 (MISO).
- If a pin of Port S is used as an input on the HC12 SPI, that setup is done automatically for you by the HC12.
  - If you write a 1 to a bit of DDRS which corresponds to an input bit of the SPI, the HC12 will ignore this and use the bit as an input.
  - For example, in Master Mode, bit 4 (MISO) of the SPI will be an input, even if you write a 1 to bit 4 of DDRS.
- You should set up DDRS before you set up the SPI control registers.

PORTS	**$\overline{SS}$**	**SCK**	**MOSI**	**MISO**	PS3	PS2	TsD0	RxD0	**0x00D6**
DDRS	**DDS 7**	**DDS6**	**DDS5**	**DDS4**	**DDS3**	**DDS2**	**DDS1**	**DDS0**	**0x00D7**

Using the HC12 Serial Peripheral Interface

Things to set up when using the HC12 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 (we will not use)
- Wired-OR Mode (we will not use)

Use the following registers:

SP0CR1, SP0CR2, SP0BR, SP0SR, SP0DR, DDRS

1. Enable SPI (`SPE` bit of `SP0CR1`)
2. Clock phase and polarity set to match device communicating with
3. Select clock polarity – `CPOL` bit of `SP0CR1`
   - `CPOL` = 0 for clock idle low
   - `CPOL` = 1 for clock idle high
4. Select clock phase – `CPHA` bit of `SP0CR1`
   - `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 `SP0CR1`
   - 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 transfered, enable interrupts with `SPIE` bit of `SP0CR1`
   - Normally master will not use interrupts – transfers are fast enough that you will normally wait for transfer to complete
   - Will often use interrupts when configured as a slave – you will get interrupt when master sends you data
7. Configure `LSBF` of `SP0CR1` for MSB first (`LSBF = 0`) or LSB first (`LSBF = 1`)
   - For most devices, use MSB first
8. Configure for normal mode by clearing bit `SPC0` of `SP0CR2`
   - Bidirectional mode (`SPC1 = 1` in `SP0CR2`) used for three-wire communication – need some protocol for selecting who is sender and who is receiver

Master Mode:

1. Set clock rate – `SPR2:0` bits of `SP0BR`
   - Normally select clock at highest rate compatible with slave
2. Make MOSI, SCLK, and SS output – bits `DDS5, DDS6, DDS7` of `DDRS`
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 `SP0DR`
6. After transfer complete (8 clock cycles), `SPIF` bit of `SP0SR` set.
   - If writing data to slave, can send next byte to `SP0DR`
   - If reading data from slave, can read data from `SP0DR`
7. Set up `SSOE` of `SP0CR1`
   - `SSOE = 0` if you want to control SS yourself (to be able to send more than one byte with SS low)
   - `SSOE = 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 4 MHz)
2. Need to make MISO output – bit `DDS4` of `DDRS`
3. No need to Make MOSI, SCLK, and SS inputs – this is done automatically when configuring HC12 as slave
   - If receiving data from master, wait until `SPIF` flag of `SP0SR` set (or until SPI interrupt received), then read data from `SP0DR`
   - If sending data to master, write data to `SP0DR` **before** master starts transfer

A C program to use the HC12 in master mode

```
#include <hc12b32.h>

main()
{
    /****************************************************************
     * SPI Setup
    ****************************************************************/
    DDRS = DDRS | 0xE0;  /* SS, SCLK, MOSI outputs */

    PORTS = PORTS | 0x80;  /* Bring SS high to deselect slave */

    SP0CR1 = 0x50;  /* 0 1 0 1 0 0 0 0
                       | | | | | | | |
                       | | | | | | | \____ MSB first
                       | | | | | | \______ multiple bytes with SS asserted
                       | | | | | \________ 0 phase (data on 1st clock edge)
                       | | | | \__________ 0 polarity (clock idle low)
                       | | | \____________ Master mode
                       | | \______________ not open drain
                       | \________________ Enable SPI
                       \__________________ No interrupts
                    */

    SP0CR2 = 0;     /* Normal (not bi-directional) mode */
    SP0BR = 0x00;   /* 4 MHz SPI clock */
    /****************************************************************
     * End of SPI Setup
    ***************************************************************/

    PORTS = PORTS & ~0x80;          /* Bring SS low to select slave */

    SP0DR = 'h';                    /* Send 'h' */
    while ((SP0SR & 0x80) == 0) ;  /* Wait for transfer to finish */

    SP0DR = 'e';                    /* Send 'e' */
    while ((SP0SR & 0x80) == 0) ;  /* Wait for transfer to finish */

    SP0DR = 'l';                    /* Send 'l' */
    while ((SP0SR & 0x80) == 0) ;  /* Wait for transfer to finish */

    SP0DR = 'l';                    /* Send 'l' */
    while ((SP0SR & 0x80) == 0) ;  /* Wait for transfer to finish */

    SP0DR = 'o';                    /* Send 'o' */
```

```
    while ((SP0SR & 0x80) == 0) ;  /* Wait for transfer to finish */

    PORTS = PORTS | 0x80;          /* Bring SS high to deselect slave */
}
```