Serial Peripheral Interface (SPI)
Serial Peripheral Interface (SPI)

- High speed serial communications (> 12 Mbits/sec) interface, originally developed by Motorola
- Synchronous: a separate clock signal is provided
- Commonly used for chip-to-chip communications
  - Eg., MCU to external memory, sensors, control devices, clocks, ...
- Advantages:
  - Saves on pin count (only 4 needed); good for embedded systems
  - Simple to use
- A “de facto” standard
  - A wide variety of SPI-compatible devices are available
  - However, since there isn’t a formal standard, there are some variations among devices, such as
    - word size
    - polarity of clock and select signals
    - LSB first or MSB first
  -> the SPI system has options to select among these variations
SPI principles of operation

- One device is the “master” (typically the MCU), and generates the clock signal.
- There are one or more “slaves” (typically peripheral chips).
- Data is shifted serially from a shift register in the master to a shift register in the slave.

The shift registers are connected in a “ring” configuration:
- When the master shifts its data to the slave, it automatically gets back the data that was in the slave’s shift register.

Figure 10.8 Master/slave transfer block diagram
SPI Signals

• MOSI (Master Out, Slave In)
  – MOSI connects to MOSI
  – Is an output on master, input on slaves

• MISO (Master In, Slave Out)
  – MISO connects to MISO
  – Is an input on master, output on slaves
  – If a slave is not selected, the slave makes its output high impedance (so other slaves can use that line)

• SCK (Serial clock)
  – Generated by master, input to slaves
  – Only runs when master wants to transmit data

• SS (Slave select)
  – Generated by master, used to select (enable) a slave
  – When SS is asserted, the slave will transfer contents of its shift register
Bi-directional synchronous communications

- To send (or receive) an 8-bit word, the master generates 8 pulses of the clock (SCK)
Multiple slaves (method 1)

- In this method, the master uses multiple signals to individually select each slave
- Only the selected slave may transmit on the MISO line

![Diagram of SPI Master and Multiple Slaves](image-url)

Figure 10.9 Single-master and multiple-slave device connection (method 1)
Multiple slaves (method 2)

- In this method, the master and all the slaves are on a giant ring ... data is shuttled through all of them.
- There is no way to individually access a particular slave.
  - If the master wants to send data to slave 1, say, it has to go through slave 0.
• An HCS12 can have multiple SPI channels called SPI0, SPI1, SPI2
• Our chip (the C version) has only one SPI channel (SPI)
• It uses Port M pins (PM2:PM5)
Clock Signal

- There are four possible combinations of clock polarity and phase

- **CPOL**
  - Clock polarity
  - 0: clock pulses are high
  - 1: clock pulses are low

- **CPHA**
  - Clock phase
  - 0: data is valid on 1\(^{st}\) edge
  - 1: data is valid on 2\(^{nd}\) edge
SPI System Registers

• You write to (or read from) the 8-bit data register SPI DR
  – A write to this register allows the byte to be queued and transmitted
  – If your system is the master, the queued byte is transmitted immediately after the previous transmission is complete

• Flags (such as the transmission complete flag) are in the status register SPI SR

• You set the baud rate using the SPI BR register

• There are two control registers SPI CR1, SPI CR2
SPI Control Registers

- **SPICR1**
  - **SPE** – SPI system enable
  - **SPIE** – SPI interrupt enable
  - **MSTR** – set to 1 for master mode
  - **CPOL, CPHA** – clock format
  - **SSOE** – enable SS output for master
  - **LSBF** – set to 1 to send LSB first

<table>
<thead>
<tr>
<th>SPIE</th>
<th>SPE</th>
<th>SPTIE</th>
<th>MSTR</th>
<th>CPOL</th>
<th>CPHA</th>
<th>SSOE</th>
<th>LSBFE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**SPIE**: SPI interrupt enable bit
- 0 = SPI interrupts are disabled.
- 1 = SPI interrupts are enabled.

**SPE**: SPI system enable bit
- 0 = SPI disabled.
- 1 = SPI enabled and pins PS4-PS7 are dedicated to SPI function.

**SPTIE**: SPI transmit interrupt enable
- 0 = SPTEF interrupt disabled.
- 1 = SPTEF interrupt enabled.

**MSTR**: SPI master/slave mode select bit
- 0 = slave mode
- 1 = master mode

**CPOL**: SPI clock polarity bit
- 0 = active high clocks selected; SCK idle low
- 1 = active low clocks selected, SCK idle high

**CPHA**: SPI clock phase bit
- 0 = The first SCK edge is issued one-half cycle into the 8-cycle transfer operation.
- 1 = The SCK edge is issued at the beginning of the 8-cycle transfer operation.

**SSOE**: slave select output enable bit
- The SS output feature is enabled only in master mode by asserting the SSOE bit and the MODFEN bit of the SPIxCR2 register.

**LSBF**: SPI least significant bit first enable bit
- 0 = data is transferred most significant bit first.
- 1 = data is transferred least significant bit first.

*Figure 10.1 SPI control register 1 (SPIxCR1, x = 0, 1, or 2)*
SPI Control Registers

- **SPICR2**
  - These bits set up a "bidirectional" mode
  - In this mode, only one serial line is used (so only 3 pins)
  - We won’t cover this mode here

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MODFEN</td>
<td>BIDIROE</td>
<td>0</td>
<td>SPSWAI</td>
<td>SPC0</td>
</tr>
</tbody>
</table>

- **MODFEN**: mode fault enable bit
  - 0 = disable the MODF error.
  - 1 = enable setting the MODF error.
- **BIDIROE**: output enable in the bidirectional mode of operation
  - 0 = output buffer disabled
  - 1 = output buffer enabled
- **SPSWAI**: SPI stop in wait mode
  - 0 = SPI clock operates normally in stop mode.
  - 1 = stop SPI clock generation in Wait mode.
- **SPC0**: serial pin control bit 0
  - With the MSTR bit in the SPIxCR1 register, this bit enables bidirectional pin configuration, as shown in Table 10.2.

Figure 10.2 SPI control register 2 (SPIxCR2, x = 0, 1, or 2)
SPI Baud Rate Register

• SPIBR

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SPPR2</td>
<td>SPPR1</td>
<td>SPPR0</td>
<td>0</td>
<td>SPR2</td>
<td>SPR1</td>
<td>SPR0</td>
</tr>
</tbody>
</table>

SPPR2~SPPR0: SPI baud rate preselection bits
SPR2~SPR0: SPI baud rate selection bits

BaudRateDivisor = (SPPR + 1) × 2(SPR + 1)
Baud Rate = Bus Clock ÷ BaudRateDivisor

Figure 10.3 SPI baud rate register (SPIxBR, x = 0, 1, or 2)

• Example
  – Set baud rate to 2 MHz (assuming a 24 MHz E clock)
  – We need a divisor of 12 (because 24 MHz/12 = 2 MHz)
  – This can be done using: 12 = (2+1)×2(1+1) = 3×2² = 12
  – So SPPR2:SPPR0 = 010, and SPR2:SPR0 = 001
SPI Status Register

- **SPISR**
  - **SPIF** – flag set when receiver register is full
  - **SPTEF** – flag set when transmitter register is empty

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SPIF: SPI interrupt request bit</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>SPTEF: SPI data register empty interrupt flag</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>MODF: mode error interrupt status flag</td>
<td>0</td>
</tr>
</tbody>
</table>

SPIF: SPI is set after the eight SCK cycles 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.

0 = transfer not yet complete
1 = new data copied to SPIxDR

SPTEF: SPI data register not empty
0 = SPI data register not empty
1 = SPI data register empty

MODF: mode fault has not occurred
0 = mode fault has not occurred
1 = mode fault has occurred

**Figure 10.4 SPI status register (SPIxSR)**

- **Clearing flags**
  - **SPIF**: clear by reading SPISR, then reading from SPIIDR
  - **SPTEF**: clear by reading SPISR, then writing to SPIIDR
## Summary of SPI Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPICR1</td>
<td>SPIE</td>
<td>SPE</td>
<td>SPTIE</td>
<td>MSTR</td>
<td>CPOL</td>
<td>CPHA</td>
<td>SSOE</td>
<td>LSBFE</td>
</tr>
<tr>
<td>SPICR2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>MODFEN</td>
<td>BIDIROE</td>
<td>0</td>
<td>SPISWAI</td>
<td>SPC0</td>
</tr>
<tr>
<td>SPIBR</td>
<td>0</td>
<td>SPPR2</td>
<td>SPPR1</td>
<td>SPPR0</td>
<td>0</td>
<td>SPR2</td>
<td>SPR1</td>
<td>SPR0</td>
</tr>
<tr>
<td>SPISR</td>
<td>SPIF</td>
<td>0</td>
<td>SPTEF</td>
<td>MODF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Master writes to SPIDR to start transmission
- SPIE – SPI system enable
- SPIF – flag set when receiver register is full
- SPE – SPI system enable
- SPTEF – flag set when transmitter register is empty
- MSTR – set to 1 for master mode
- Baud rate divisor = (SPPR+1) * 2^(SPR+1)
- CPOL, CPHA – clock format
- SSOE – enable SS output for master
- LSBF – set to 1 to send LSB first
- SSOE – SPI interrupt enable
Example

• Configure SPI to operate with the following settings, assuming that E clock is 24 MHz:
  – 6 MHz baud rate
  – enable SPI to master mode
  – SCK pin idle low with data shifted on the rising edge of SCK
  – transfer data most significant bit first and disable interrupt
  – disable SS function
  – (leave SPICR2 with default values)
Example (continued)

• The baud rate divisor is ...

• The following instruction sequence will configure the SPI as desired:

\[
\begin{align*}
\text{movb } & 10, \text{SPIBR} \quad ; \text{set baud rate to 6 MHz} \\
\text{movb } & 50, \text{SPICR}1 \quad ; \text{disable interrupt, enable SPI, SCK idle low,} \\
& \quad ; \text{data latched on rising edge,} \\
& \quad ; \text{data transferred msb first}
\end{align*}
\]
Sending and receiving bytes and strings

- The following operations are common in many applications and can be made into library functions to be called by many SPI applications:
  - Send a character to SPI: `putcspi`
  - Send a string to SPI: `putsspi`
  - Read a character from SPI: `getcspi`
  - Read a string from SPI: `getsspi`

- Function `putcspi` – send a character to SPI:

```c
void putcspi (char cx)
{
    char   temp;
    while(!(SPISR & SPTEF));  // wait until write is permissible
    SPIDR = cx;               // output the byte to the SPI
    while(!(SPISR & SPIF));   // wait until write operation is complete
    temp = SPIDR;             // clear the SPIF flag
}
```
Sending and receiving functions (continued)

- **Function putsspi** – send a string to SPI:

```c
void putsspi(char *ptr)
{
    while(*ptr) { /* continue until all characters have been output */
        putcspi(*ptr);
        ptr++;
    }
}
```

- **Function getcspi** – get a character from SPI:

```c
char getcspi(void)
{
    while(!(SPISR & SPTEF)); // wait until write is permissible
    SPIDR = 0x00;           // trigger 8 SCK pulses to shift in data
    while(!(SPISR & SPIF)); // wait until a byte has been shifted in
    return SPIDR;          // return the character
}
```
Sending and receiving functions (continued)

- Function getsspi – get a string from SPI:

```c
void getsspi(char *ptr, char count)
{
    while(count) {  // continue while byte count is nonzero
        *ptr++ = getcspi();  // get a byte and save it in buffer
        count--;
    }
    *ptr = 0;  // terminate the string with a NULL
}
```
Some SPI-compatible chips

- 74HC595 8-bit latch
  - shift in serial data using SPI, store in a latch
  - provides 8 bit parallel output (with 3-state capability)
- TC72 digital thermometer
  - 10 bit resolution
  - Least significant bit (LSB) corresponds to 0.25 degrees C
- MCP4922 D/A converter
  - 12-bit serial data in
  - analog voltage out
- MAX6952 LED display driver
  - for 5x7 matrix displays
    - Other matrix sizes are possible; also multiple colors
    - See final project by David Schweitzer and Chris Graff, fall 2011
- 25AA256 EEPROM
  - 256 Kbit
Example – TC72 Digital Thermometer

- Ten-bit resolution and SPI interface
- Capable of reading temperature from -55 degrees C to 125 degrees C
- Can be used in continuous temperature conversion or one-shot conversion mode

Figure 10.13 TC72 pin assignment and functional block diagram
TC72 Interface

- The CE input to the TC72 must be asserted (high) to enable SPI transfer
- Data can be shifted on the rising edge or the falling edge depending on the idle polarity of the SCK source

Figure 10.16 Circuit connection between the TC72 and the HCS12
Example - D/A Converter MCP4922
- The MCP4922 is a 12-bit voltage output DAC with SPI interface.
- The MCP4922 has an output settling time of 4.5 ms.
- A D/A conversion is started by writing a 16-bit serial string that contains 4 control bits and 12 data bits to the MCP4922.
- MCP4922 can operate from 2.7V to 5.5V.

Figure 10.17 The MCP4922 DAC pins and block diagram
Summary / Questions

• The serial peripheral interface (SPI) is a synchronous serial communications interface. It is a de facto communication standard (meaning that a lot of devices use it, but there are slight differences in terminology and functionality).

• It is faster than SCI (although the maximum distance is shorter). It is commonly used in embedded systems to interface to peripheral chips.

• Why would you want to use SPI versus a parallel interface; for example to talk to a LED matrix display?