Introduction

This document discusses Serial Peripheral Interface (SPI), a commonly used protocol for interfacing with embedded hardware devices, at data rates of hundreds of kilobits, up to a few megabits. SPI uses three wires for full-duplex synchronous data transfer between one master and one or more slaves. SPI is not standardized – instead it is simply a common convention for serial data transfer. Thus, most MCUs feature SPI controllers which can be configured to operate in a variety of different modes.

Contents

1 Overview

SPI is a simple protocol with many advantages: because the master device on the bus controls the clock rate, the protocol itself does not place any upper bound on performance (though most devices will only operate within a specific range of clock rates). Messages may be arbitrarily long, though most devices operate on 8-bit words. Notably, slaves do not require addressing like I^2C, and the bus does not include any special arbitration or hand-off, meaning it cannot deadlock[1].

Unlike UART, SPI uses both clock and data lines. This means that it is not necessary for all devices attached to an SPI bus to know the clock rate in advance. When the data lines are sampled depends on the clock polarity and phase, which are configurable for most MCUs. Slave devices are differentiated using an out-of-band slave-select line, which the master should pull low while transferring data to or from the slave. Said slave-select line is usually driven from GPIO pins on the master, and is held high when the slave is not in use.

2 Terms

SCK Serial Clock, sometimes called **SCLK** or **CLK**.

MISO "Master In Slave Out", the data transmission line used to send data from the slave to the master.

MOSI "Master Out Slave In", the data transmission line used to send data from the master to the slave.

SS "Slave Select", an active-low control line which the master pulls low to signal to a slave that it should become active. The master usually uses a GPIO pin for this purpose. Sometimes called "Chip Select" (CS). Often denoted as $\overline{\text{SS}}$ or $\overline{\text{CS}}$ to indicate that it is active-low.

3 SPI on ATMega328P

Refer to chapter 19 "SPI - Serial Periphreal Interface" of the ATMega328P datasheet for detailed information.

The most important register for configuring SPI is `SPCR` (SPI Control Register). In this course, we want to enable SPI in master mode at 4MHz. We will leave the data order at it's default value (MSB-first), and will use the default clock polarity and clock phase.

Therefore, we only need to modify the SPI Enable bit (SPE), the master/slave select bit (MSTR), and the SPI Clock Rate Select bits (SPR). We can enable SPI as follows:

```
1 //      SPI enable | master mode | clock rate select, SPR1=1 and SPR0=1 => F_CPU/4
2 SPCR =  (1<<SPE)   | (1<<MSTR)   | (1<<SPR1) | (1<<SPR0);
```

Listing 1: Example code to enable SPI.

[1] Protocols such as i2c and CAN-bus involve complex state machines and arbitration schemes to determine which device is in control of the bus at any given time, therefore it is possible for incorrectly written firmware to deadlock these busses, such that both the master and slave devices are pending on one-another. This is not possible with SPI.

It is also necessary to decide on a pin to use as the SS signal. This will be managed using GPIO, and should be set to logic high when the slave device is not in use. This can be done in the same fashion as discussed in previous handouts.

Data can be written to SPI slave device by first holding the chosen SS line low, then writing the desired value to `SPDR`. `loop_until_bit_is_set(SPSR, 7);` can be used to pend until the value in `SPDR` has been fully transmitted (meaning the buss is ready for a new value to be written). Likewise, values returned from the slave can be accessed by reading from `SPDR`.

3.1 Hello World SPI master example

```
// Assumes default SPI pinout on PORTB, with PB2 for SS.
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>

// SPI slave-select macros.
#define SPI_SELECT() PORTB &= ~(1 << PB2);
#define SPI_DESELECT() PORTB |= (1<<PB2);

uint8_t spi_transfer(uint8_t value) {
    // Pull slave's SS low, to make it active.
    SPI_SELECT()

    // Load the value we wish to transfer into SPDR.
    SPDR = value;
    loop_until_bit_is_set(SPSR, 7); // Wait for transfer to complete.
    // Get the value the slave sent back.
    value = SPDR;

    // Let the slave's SS go high to deactivate it.
    SPI_DESELECT()
    return value;
}

void init(void) {
    //      SPI enable | master mode | clock rate select, SPR1=1 and SPR0=1 =>
    F_CPU/4
    SPCR =  (1<<SPE)   | (1<<MSTR)   | (1<<SPR1) | (1<<SPR0);

    // Configure output pins for SPI.
    DDRB  = (1<<DDB2)|(1<<DDB3)|(1<<DDB5); // Set outputs pins.
    PORTB |= (1<<PB2); // PB2 is the default SS pin.
}

int main(void) {
    const char message[12] = {'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd
    ', '\n'};

    init();

    // Print "Hello world" over SPI once per second.
    while (1) {
```



```
45          for (uint8_t i = 0; i < 12; i++) {
46              spi_transfer(message[i]);
47          }
48          _delay_ms(1000);
49      }
50  }
```

Listing 2: SPI master transferring "Hello world" over PORTB

