EECE 5141C/6041C
Introduction to Mechatronics
Lab 4: SPI Bus Port Expansion

Name: ___________________________

In this lab, you will design, program, and construct a simple microcontroller system capable of interfacing two serial shift registers on an SPI bus. One shift register will read in eight switch values and the second shift register will display eight binary outputs using an LED array.

Deliverables

1. Completed Worksheet
2. Annotated Source Code for Each Method

System Description

You will interface a serial-input, parallel-output shift register (STPIC6D595) and a parallel-input, serial-output shift register (SN74LS165A) using the SPI bus controller built into the ATMEGA328p. In addition to the normal SPI bus signals (SCK, MOSI, and MISO), you will also need to generate the parallel load and parallel output register signals in order to control the parallel latching of input and output data. These signals will be controlled by software and will be output using general I/O pins on your microcontroller.

Preliminary SPI Bus Configuration

To begin construction of the system, you will first configure and test the microcontroller's SPI bus to operate as the Master and continuously output a constant value (0b01101010). Don't forget to configure the SCK (PB5) and MOSI (PB3) pins as outputs. Also, you must also configure the microcontroller SS pin (PB2) as an output as well in order to prevent the microcontroller from switching to slave mode if that pin is pulled low. We will eventually interface with our two shift registers using the SPI bus. Therefore, we should configure the phase and polarity of the SPI data and clock signals appropriately for the two devices. From the waveform diagrams of each device shown in Figure 1 and Figure 2, we see that serial data is clocked on the rising edge of the CLK signal, and the data is setup on the falling edge. Therefore, we want the SPI bus to operate in Mode 0, which is achieved by setting `CPOL=0` and `CPHA=0`. Since we have no particular transmit speeds to satisfy, we can choose the slower SCK source of $f_{osc}/128$ (`SPI2X=0,SPR1=1,SPR2=2`). Once configured, your program should wait until an SPI transfer is completed by polling the `SPIF` flag within the `SPSR` register. Once the previous transmission is complete, it should transmit the same value (0b01101010) again. All of this can be placed in the main `while(1)` loop. Once implemented, you should measure the SCK and MOSI signals coming from your microcontroller, and answer Design Question 1.

© 2020 Zachariah Fuchs

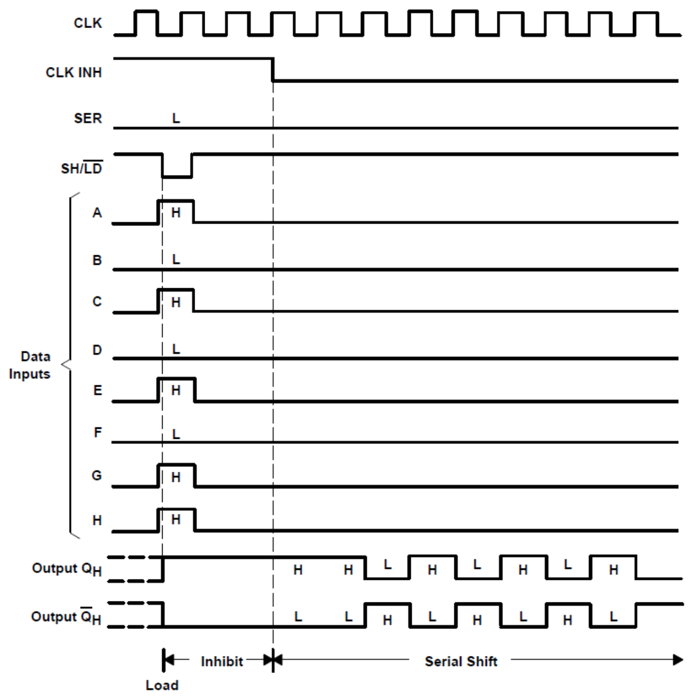

Figure 1:SN74LS165A Waveform Diagram

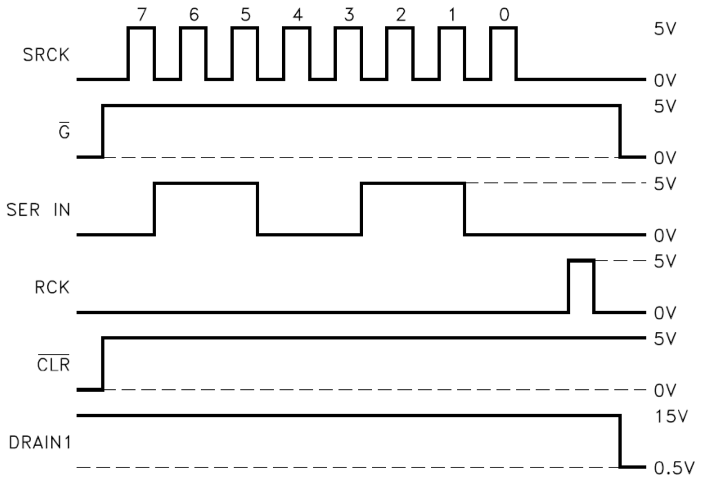

Figure 2:STPIC6D595 Waveform Diagram

Interfacing with Serial-In, Parallel-Out Shift Register (STPIC6D595)

After configuring the SPI bus to continuously transmit a fixed data byte, we will now store and display this byte using a serial-in, parallel-out shift register. The pinout for the STPIC6D595 is provided in Figure 3. We will not use clear (CLR) or the output enable (G) signals for this lab, so you should connect them to 5V and GND respectively. You should connect the SRCK and SER_IN pins to the SCK and MOSI pins of the SPI bus. Lastly you should connect the output register latch signal (RCK) to a general I/O pin of your microcontroller. You should connect each drain output to the cathode side of an LED circuit so that you can display the 8-bit output on the LED array. Once connected, you can now modify your SPI code so that it strobes (pulls high then low) the RCK signal after each SPI transfer. This will latch the newly transmitted SPI data into the parallel output registers of the STPIC6D595 chip. Once completed, measure the MOSI signals and RCK signals using the oscilloscope and answer Design Question 2. Modify the constant transmitted data within your code in order to test the functionality of your circuit.

© 2020 Zachariah Fuchs

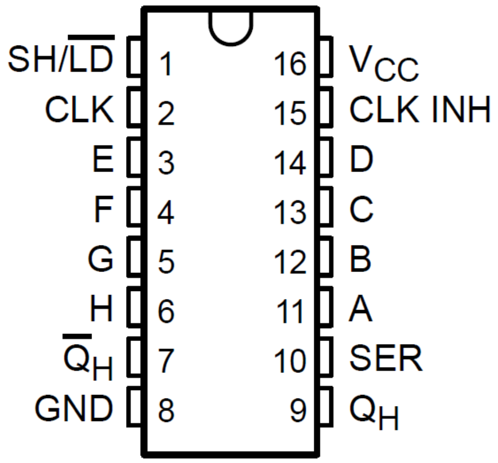

Figure 3: SN74LS165A Pinout Diagram

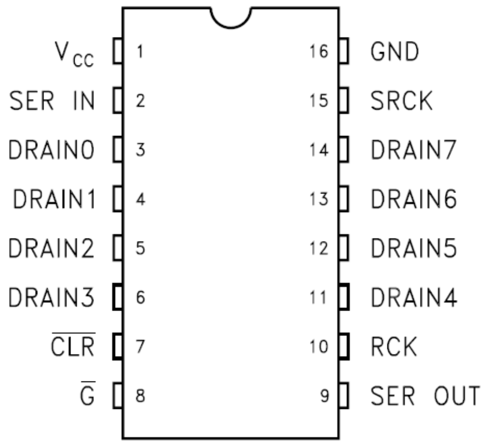

Figure 4: STPIC6D595 Pinout Diagram

Interfacing with Parallel-In, Serial-Out Shift Register (SN74LS165A)

Now that we can continuously transmit data to the output register, we will interface the Parallel-In, Serial-Out shift register so that we can read data in from the SPI bus and then send that data back out to be displayed on the LED array. The pinout for the SN74LS165A chip is provide in Figure 4. Connect the CLK INH signal to GND so that the clock is always enabled. Connect the CLK pin to the SPI's CLK signal and the QH pin to the MISO wire of the SPI bus. Connect the parallel input signals A through H to eight pull-up resistor switch circuits. Lastly, you will connect SH/_LD pin to a general purpose I/O. You will strobe this signal with a falling edge to load the parallel data into the serial shift registers. Once connected, modify your software so that it strobes the SH/_LD signal before sending the previously read SPI data out to the STPIC6D595 ship. While this transfer is being complete, it will be simultaneously reading the freshly latched data in from the SN74LS165A chip. After the SPI transfer is complete, you should store the SPI read data into a variable to be transferred out to the SN74LS165A in the next iteration of the `while(1)` loop. A flowchart of the data transfer processes is shown in Figure 5.

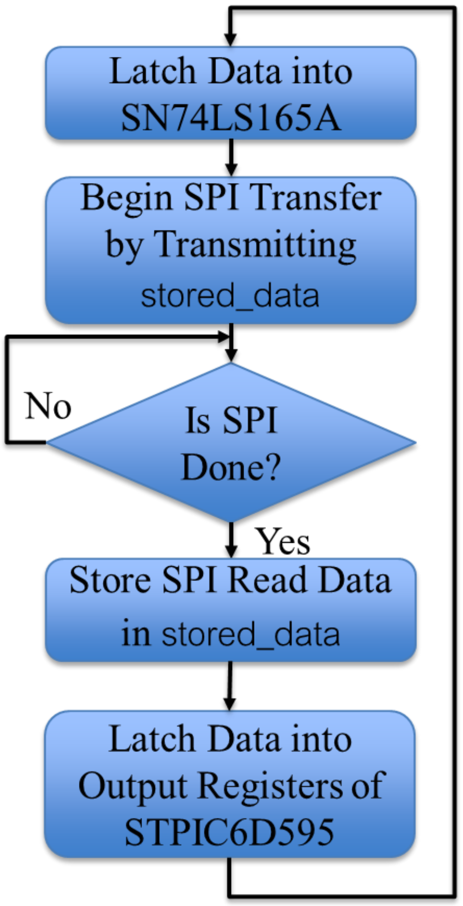
Latch Data into SN74LS165A
Begin SPI Transfer by Transmitting stored_data
Is SPI Done?
No
Yes
Store SPI Read Data in stored_data
Latch Data into Output Registers of STPIC6D595

Figure 5: SPI Data Transfer Flowchart

© 2020 Zachariah Fuchs

Design Questions

1. Sketch the measured MOSI and SCK signals. Pay particular attention to the alignment of the edges of the signals.

2. Sketch the measure MOSI and RCK signals.

3. If you needed to add additional output or input registers how would you modify the hardware and software?

© 2020 Zachariah Fuchs