Christopher Brant
C19816588
Due on 11/16/2017

ECE 3270 Microcontroller Interfacing Lab
Lab 9: Serial Peripheral Interface

Abstract

The purpose of this lab was to learn about the use of Serial Peripheral Interface communication, or SPI. This was done by utilizing a SPI specific IC that output a value in parallel bits that was passed into it the IC serially. The parallel outputs of the SPI specific IC were then wired to 8 LEDs so that the binary value of the output was visible on the LEDs.

Introduction

In this experiment, we learned how to utilize the PIC32MX150F128D's SPI module and we utilized an external interrupt to alert the MCU when the value that the SPI module outputs should change. When the interrupt is pressed, the value that the SPI module is supposed to output will change as the value iteratively goes through a predetermined list. In the case of the index going above the highest index value, the index will wrap around back to the beginning.

Experimental Procedures

- The first thing that must be done is to wire the overall circuit to be just as it is shown in Figure 1.
- The next thing that is to be done is to write the code for the interrupt to increment the index value and push the new value for transmission into the SPI module's buffer.
- Then, in your code you must write the set up of the SPI module in master mode, including the 10 basic steps that are outlined in Section 23.3.3.1, as well as setting up the external interrupt to be used as well.
- Then the rest of the code should be written just as it is seen in Figure 2, as the set up is the majority of this lab, since the SPI module takes care of the majority of the necessary steps for sending information as long as the module has data in its buffer.

Results

There were no tabulated values and no values necessary for the record. Observations of this lab show that the difficulty in ensuring your wires are in the correct rows they should be in on the breadboard becomes more and more difficult for each successive lab. Overall, this lab was quite straightforward and made sense as long as all of the setup bits were set correctly in your code. Otherwise, there is not much to worry about other than wiring.

Christopher Brant
C19816588
Due on 11/16/2017

Discussion

Overall, the final conclusions of this experiment include the statements that the microcontroller can be programmed to serially output an 8 bit value that will be latched into a dedicated SPI IC so that the output can be displayed in parallel, and can virtually seem output in parallel directly with the speed at which the instructions complete. Another conclusion would be that the importance of having every setup bit set correctly cannot be understated, as anything set incorrectly will cause the SPI module to function incorrectly or not at all, and this is especially important as there are so many setup bits to set.

Conclusions

The conclusions of this experiment include that the MCU can be programmed to serially output individual bit values of different sized binary numbers, which is what the SPI module is designed to do of course, as well as another conclusion is that the importance of ensuring your setup bits are correct cannot be understated. Lastly, this experiment propelled us to learn the setup for the configuration necessary to utilize the SPI module in master mode.

Figures and Tables

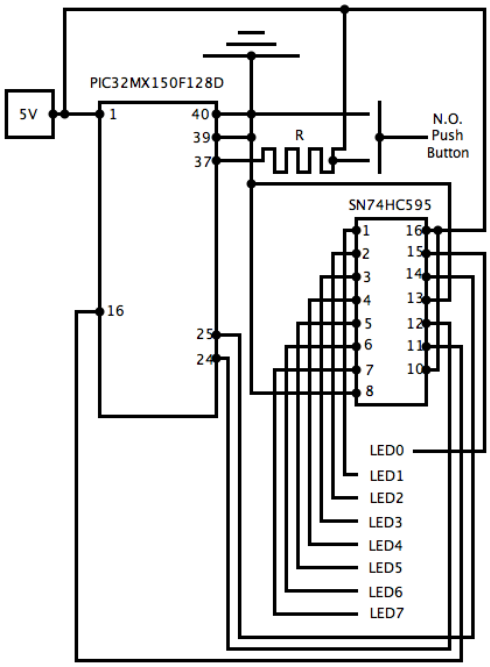

Figure 1: Wiring Diagram of the circuit for Lab 9: Serial Peripheral Interface

Christopher Brant
C19816588
Due on 11/16/2017

```c
/* Christopher Brant
 * C19816588
 * Lab 9: SPI
 * 11/9/2017
 */

#include <plib.h>

char spiChars[18] = {0,1,2,4,8,16,32,64,128,255,
                     254,253,251,247,239,223,191,127};

int clearData, sendIndex = 0;

void __ISR(3) ButtonPress(void)
{
    // Decide if the index needs to wrap around
    if (sendIndex >= 17)
        sendIndex = 0;
    else
        sendIndex++;

    // Send the next value to the buffer
    SPI1BUF = spiChars[sendIndex];

    IFS0bits.INT0IF = 0;
}
int main(void)
{
    // Setting Tristate register bits
    TRISB = 0x0000;
    TRISBbits.TRISB7 = 1;
    TRISC = 0x0000;
    // Setting interrupt function
    INTEnableSystemMultiVectoredInt();
    INTCONbits.INT0EP = 0;
    IFS0bits.INT0IF = 0;
    // Setting the Peripheral Pin Selects for SDO1 and SS1
    PPSOutput(1,RPC0,SS1);
    PPSOutput(3,RPC1,SDO1);
    // The following steps set up Master Mode Operation
    IEC0CLR = 0x03800000;       // disable all interrupts
    SPI1CON = 0;                // stop and reset SPI1
    clearData = SPI1BUF;        // clear the receive buffer
    SPI1CONbits.ENHBUF = 0;     // clear the enhanced buffer bit
    IFS0CLR = 0X03800000;       // clear any existing event flags
    IPC5CLR = 0x1f000000;       // clear the priority
    SPI1CONbits.MCLKSEL = 0;    // Set the PBCLK for the BRG
    SPI1BRG = 1;             // use Fpb/4 clock frequency
    SPI1STATbits.SPIROV = 0;    // Clear the SPIROV bit
    SPI1CONbits.MSSEN = 1;      // Enable Master slave select control
    SPI1CONbits.SSEN = 1;       // Set slave select on
    SPI1CONbits.DISSDO = 0;     // Enable the SDO1 pin
    SPI1CONbits.MODE32 = 0;     // Set the data width to 8 bits
    SPI1CONbits.MODE16 = 0;     // Set the data width to 8 bits
    //SPI1CONbits.SMP = 1;        // Set data to sample at end of output time
    SPI1CONbits.CKE = 1;        // Output changes on change from active to idle
    SPI1CONbits.CKP = 0;        // Idle state is low, active is high
    SPI1CONbits.MSTEN = 1;      // Set to master mode
    SPI1CON2bits.AUDEN = 0;     // Set audio enable off
    // Turn on SPI1 module last
    SPI1CONbits.ON = 1;

    IEC0bits.INT0IE = 1;
    IPC0bits.INT0IP = 0x1;

    while(1);

    return 0;
}
```

Christopher Brant
C19816588
Due on 11/16/2017

Figure 2: Code for Lab 9: Serial Peripheral Interface