ECE 3720

Microcomputer Interfacing Laboratory

Section 8

Timothy Driscoll

Date Performed: November / 7 / 2019

Lab 9

ABSTRACT:

A lab designed to show how the PIC32MX150F128D microcontroller’s SPI peripheral can be set up in master mode to output data to the SN74HC595 shift register.

**INTRODUCTION:**

In this lab, we created a new project that showed how the PIC32MX150F128D microcontroller’s SPI peripheral can be set in master mode to output data to a shift register. The objective of this lab was to set up the SPI peripheral in master mode to output data to a shift register. The external interrupt (INT0) was used to cycle through the data that was being sent the shift register and outputted. SPI is a form of communication that implements a master slave protocol where the master can have multiple slaves, but the slaves are only able to connect to the one master. SPI uses separate clock and data lines as well as a select line to determine which device is being communicated with. By combing the external interrupt, SPI peripheral and shift register we were able to have the DIO display the data and sequentially change as we trigger the interrupt.

**EXPERIMENTAL PROCEDURES:**

In this lab, we started by unpacking the microcontroller and attaching the chip kit in the proper orientation to ensure that the pins wouldn’t have a flipped result. Then we could start wiring the circuit to complete the project. This circuit is represented by figure 1 below. The circuit includes the three necessary external hardware components, the shift register IC, the LEDs and the push button. When the circuit was designed in lab the DIO on the NI-ELVIS board was used to represent the button and LEDs in order to simplify the circuit. When a 1 was written to the given DIO pin it represented a push of the button, and when a 1 was read from the DIO it represented an on LED. The button (DIO) was wired to pin 37 (register B7) specifically because INT0 is the only hard-mapped external interrupt on the microcontroller and it is connected to pin 37. There was then three outputs wired from the microcontroller and into the shift register these outputs represented the SRCLK, RCLK and SER. The SRCLK was wired to pin 16 (register B14) which corresponded to the SPI peripheral SCK pin. The RCLK pin was wired to pin 24 (register C0) which corresponded to the SPI peripheral SS pin. The SER was wired to pin 25 (Register C1) which corresponded to the SPI peripheral MOSI pin. All the outputs from the shift register were then wired to the LEDs which corresponded to us wiring the outputs to the NI-ELVIS’s DIO pins. After the completion of the circuit wiring, we created a new project using the MPLAB X IDE software. After creating the project, we added a new C type source file where we wrote the code in order to have the microcontroller function properly. The code can be viewed below in the figures and tables section. The flow of this code starts by defining two global variables, one that is used as an index and the other that is an array which stores the data that is to be transmitted. Next, we included the delay function which was implemented in past labs and was used in this particular lab was used to provide a pause when the ISR was triggered. After the delay function we implemented the ISR for the external interrupt (INT0). This block of code tells the microcontroller what to do when the INT0 interrupt has occurred. The interrupt service routine was designed to cycle through the array of data that was to be sent to the shift register using the SPI communication. Now looking at the main function INTEnableSystemMultiVectoredInt() had to be included to tell the microcontroller to handle interrupts. TRISxbits was then used to initialize INT0 as an input and the RCLK, SER and SRCLK as outputs. INT0 was then properly initialized and the PPSOutput function was used twice to map the SPI peripherals. The slave synchronization pulse was mapped to register C0 and the SPI data out was mapped to register C1. The SPI 1 peripheral was then properly initialized, with our desired conditions. There was nothing needed in the while loop for this lab, but it was included to have the program continuously run while the microcontroller had power.

**RESULTS and DISCUSSION:**

In this lab, we were successfully able to get the desired output from the project and achieve all the desired goals that were described in the slides. This involved having the microcontroller’s SPI peripheral set up in master mode to output data to a shift register. From the shift register the data was then outputted to the DIO pins where they could be read using the digital reader on the ELVIS. For this lab to work there were three main things that we needed to understand and implement. These three main things included properly writing the shift register, properly implementing the SPI peripheral on the microcontroller and properly setting up the external interrupt. Starting with the wiring of the shift register there a few key points that needed to be accounted for. Specifically, the output enable pin needed to be grounded and the SRCLK was wired into the SRCLK pin on the shift register. Also pin 24 and 25 on the microcontroller were wired to the RCLK and the SER pins on the shift register respectively. After completing the wiring, we could work on completing the necessary C code. We started by correctly implementing the external input. INT0, the external interrupt was triggered on a falling edge. In order to set this up we used the INTCON register and set it to 1. There were three other registers that had to be setup in order to properly initialize the interrupt. These registers included the IEC, IFS and IPC. The IEC register was set to a one in order to enable the interrupt. The IFS register was set to a zero to indicate that the interrupt has not yet occurred. Lastly the IPC was initialized to a one to give this interrupt the highest priority, this was done because we only had one external interrupt. Next, to initialize the SPI peripheral we had to adjust the bits in multiple registers. We started by changing several bits in the SPI1CON register. From this register we started by setting the ON bit to a 0 in order to disable for setup. The ENHBUF bit was set to a zero in order to disable the enhanced buffer mode. Then the MSTEN and MSSEN bits were set to 1 to the set the SPI peripheral in master mode and so the slave select SPI support was enabled respectively. We then set AUDEN, MODE16 and MODE32 bits to be 0 in order to set up the SPI peripheral into an eight-bit communication mode. We then set the CKE bit to a one in order to have the output data transition occur on the transition from and active clock to and idle clock. The CKP bit was set to a 0 to indicate that the idle state of the clock was a low level and the active state was a high level. Then the DISSDO and MCLKSEL bits were set to be zeros. The DISSDO bit indicated that the SDO1 pin was to be controlled by the module, and the MCLKSEL bit was set so the PBCLK was used by the baud rate generator. The other register that we initialized bits in included the SPI1STAT and we set the SPIROV bit to 0 in order to not generate any errors when overflow was received. Lastly, we initialized the SPI1BUF to 0 and the SPI1BRG to 2, which was information provided in the lab. The last step to initialization was to set the ON bit to 1 in order to enable the SPI peripheral. After completing this initialization, the main function of the code was completed. The only other important component from the code involved creating the ISR for the external interrupt which increased an index from 0 to 17 and reset at 17. The index value was then used to assign the data from the array that was to be outputted to the shift register.

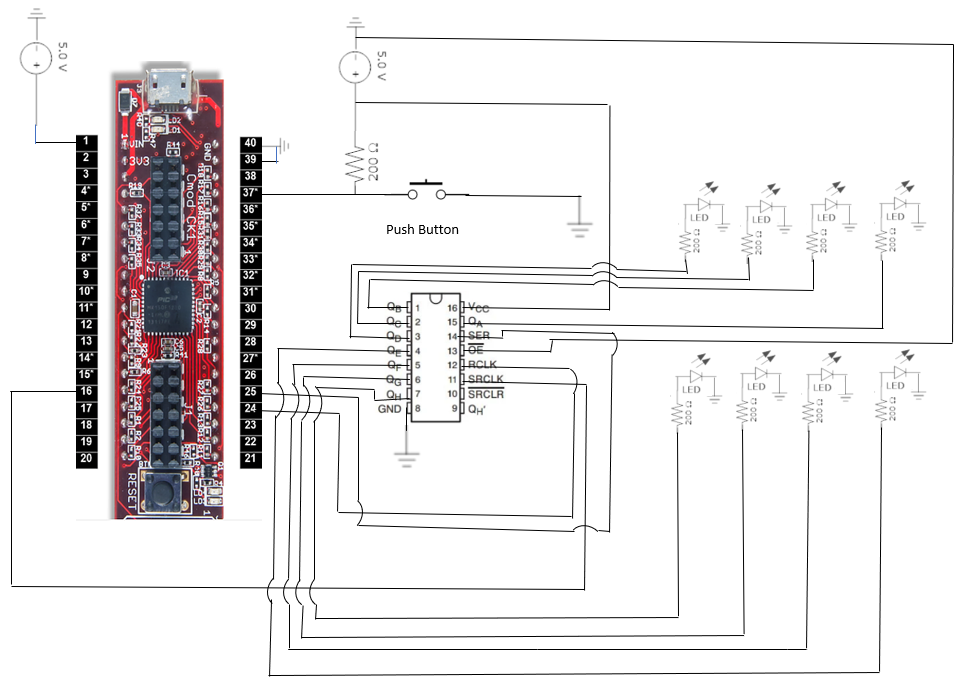
**CONCLUSSION:**

We have come to the conclusion that in this lab there were some key procedural points that needed to be learned in order to achieve all the presented goals. These key procedural points included implementing ideas that were learned in past labs like, setting up the external interrupt, creating an ISR and using the PPSOutput() function. These ideas were then combined with the new ideas learned in the section above which include properly setting up the SPI peripheral in master mode and correctly wiring the shift register. By working through the lab, I was able to determine all the necessary procedural points to understand and achieve the desired outputs for the lab.

**REFERENCES:**

Clemson University’s ECE 372 Lab 8 PowerPoint.

**FIGURES AND TABLES:**

****

**Figure 1: Wiring for lab 9 (Pin connections described in experimental procedures)**

**CODE:**

#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 index = 0;  
  
  
delay(){  
    int i, j;  
    for(i = 0; i < 500; i++)  
        for(j = 0; j < 500; j++);  
}  
  
void \_\_ISR(3) Lab9Func(void)  
{  
   SPI1BUF = spiChars[index++];//Cycle through the array  
     
    if(index>17)  
        index = 0; //make sure to reset the index  
   
  
    delay();  
    IFS0bits.INT0IF=0;  
}  
  
main()  
{  
    INTEnableSystemMultiVectoredInt();  
     
    //Initialize all the input and outputs  
    TRISBbits.TRISB14 = 0; //No further setup for SRCLK  
    TRISCbits.TRISC0 = 0; //RCLK  
    TRISCbits.TRISC1 = 0; //SER  
    TRISBbits.TRISB7=1; //INT0  
     
     
    //Setup the external interrupt  
    INTCONbits.INT0EP=1;  
    IPC0bits.INT0IP=1;  
    IEC0bits.INT0IE=1;  
    IFS0bits.INT0IF=0;  
     
    //Map the external peripherals  
    PPSOutput(1,RPC0,SS1);  
    PPSOutput(3,RPC1,SDO1);  
     
    IEC1bits.SPI1RXIE = 0;  
    IEC1bits.SPI1TXIE = 0;  
     
    SPI1CONbits.ON = 0;//Disable for setup  
    SPI1BUF = 0;  
    SPI1CONbits.ENHBUF = 0;  
    SPI1BRG = 2;  
    SPI1STATbits.SPIROV = 0;  
    SPI1CONbits.MSTEN = 1;  
    SPI1CONbits.MSSEN = 1;  
    SPI1CONbits.MODE16 = 0;  
    SPI1CONbits.MODE32 = 0;  
    SPI1CON2bits.AUDEN = 0;  
    SPI1CONbits.CKE = 1;  
    SPI1CONbits.CKP = 0;  
    SPI1CONbits.DISSDO = 0;  
    SPI1CONbits.MCLKSEL = 0;  
    SPI1CONbits.ON = 1;  
     
     
  
     
    while(1);  
     
}