ECE 3720

Microcomputer Interfacing Laboratory

Section 6

Timothy Driscoll

Date Performed: October / 3 / 2019

Lab 5

ABSTRACT:

A lab designed to show the application of the PIC32MX150F128D microcontroller’s interrupt functionality by implementing an ISR using INT0 triggered by a switch.

**INTRODUCTION:**

In this lab, we created a new project that showed the implementation of interrupts on a microcontroller. By creating an interrupt service routine and using INT0 triggered by a button we created a functional lab. The microcontroller was programmed to display a count of 0-15 on four separate LEDs. The ISR was combined with this simple program in order to have the microcontroller react to a push of a button, which was the interrupt condition. When the button was pressed the count would be paused and 15 would be displayed on the LEDs. After a couple of seconds, the LEDs would return to display the value of the count before the interrupt occurred.

**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 four LEDs, button, and NAND gate IC (74LS00). The LEDs were specifically wired to pins 20-23 which correspond to registers B0-B4. This was done to allow us to implement the code used in lab 1 which was the continuous counter from 0 to 15. The switch was then wired into the 74LS00 in order to create a debouncer and get an accurate response from the switch. One of the two individual NAND gate outputs was then wired to pin 37 which corresponds to register B7. This register was specifically used because INT0 is the only hard-mapped external interrupt on the microcontroller and it is connected to pin 37. There are other external interrupts, but they are only accessible through peripheral pin select. 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 includes the delay function which was written in lab one to act a way to briefly delay the flow of the code when it is called. The next piece of code that was included was the interrupt service routine for the INT0 external interrupt. This block of code tells the microcontroller what to do when an interrupt has occurred. Lastly looking at the main function the input and output pins were first defined, then INTEnableSystemMultiVectoredInt() had to be included to tell the microcontroller to handle interrupts. Within in the infinite while loop the code including setting the LEDs equal to the count, setting the interrupt parameters, and lastly restricting the count to 0-15. The program could then be loaded to the microcontroller and the circuit could be tested using the NI-ELVIS Board and software.

**RESULTS and DISCUSSION:**

In this lab, we were successfully able to get the desired output from the project and achieve all the desired goals described in the slides. This involved having the LEDs continuously loop through displaying the numbers 0-15 which utilized the code from lab 1. The second part to the lab we completed was to have the LED count stop when a button was pressed and display the value 15 on the LEDs for a few moments before resuming the count from the stopping point. For this task to work the two main things we needed to implement included a debouncer and an interrupt. The decouncer was used to combat an issue that used switches have. When a switch is first toggled the tend to rapidly bounce between 1 and 0. This rapid bounce could result in upwards of a hundred different toggles when it should only be one. In order to fix this issue, the debouncer was implemented in order to filter out any of the rapid changes in the switch signal. Then next piece of the lab that had to be implemented was the external interrupt. In this lab we only needed one interrupt, so we were able to use INT0 which is the only hard-mapped external interrupt on the microcontroller. The interrupt was triggered on a rising edge because we wanted it to correspond to the push of the button as suppose to the release. In order to set this up we used the INTCON register and set it to 0. 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 1 to give this interrupt the highest priority, this was done because we only had one interrupt. After the interrupt was triggered it sets a flag which tells the microcontroller that an interrupt has occurred. The microcontroller then saves the main state and looks up to the ISR corresponding to the specific interrupt. The ISR is then executed and is where we had the LEDs display 15 and delay briefly. Lastly, we cleared the flag by setting the IFS register back to 0 allowing he microcontroller to restore to its main state an continue the count from the correct number. The only error that I encountered in this lab was a careless human error. This error was wiring the 74LS00 completely backwards because I mixed up the chip orientation.

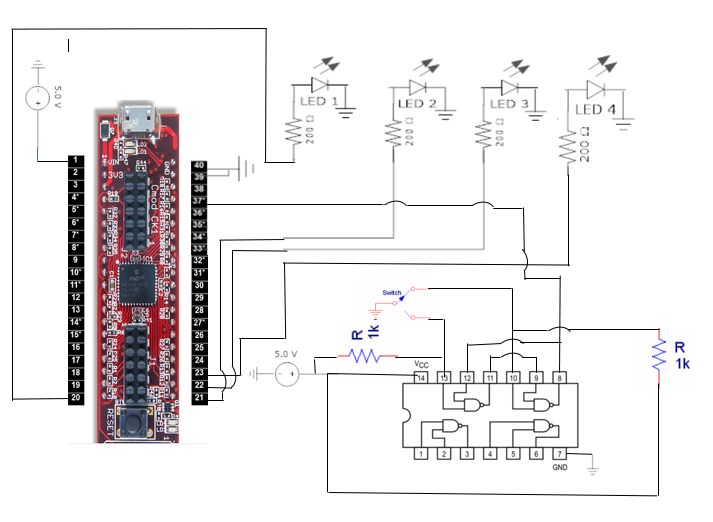
**CONCLUSSION:**

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 goals. To start we needed to first understand the effects that a toggling a switch has on its output signal. After learning that it rapidly bounces between 1 and 0, we learned how to implement a debouncer to solve this issue. The next key point that we had to understand was how interrupts functioned and how the microcontroller handled them. I described the interrupt functionality in the section above. After learning the functionality of the interrupt, we could successfully initialize and write an ISR to have the button trigger our desired results. In order to put the lab all together It was critical that we learned how to determine the proper registers that corresponded to the needed pins as described in the Lab 5 PowerPoint. Lastly, we needed to rely on the PIC32 Datasheet in order to determine how to setup the interrupt with the correct code to achieve the desired outputs. 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 5 PowerPoint.

**FIGURES AND TABLES:**

****

**Figure 1: Wiring for lab 5 (Pin connections described in experimental procedures)**

**CODE:**

#include <plib.h>

delay(){

int i, j;

for(i = 0; i < 500; i++)

for(j = 0; j < 500; j++);

}

void \_\_ISR(3) lab5\_func(void){

LATB = 15;

IFS0bits.INT0IF = 0;

delay();

delay();

delay();

}

main(){

int count = 0;

TRISB = 0x00; //Set all pins to output.

//Set INT0 as input pin

TRISBbits.TRISB7 = 1;

INTEnableSystemMultiVectoredInt();

while(1){

LATB = count; //Output count to B

count++;

INTCONbits.INT0EP = 0;

IEC0bits.INT0IE = 1;

IFS0bits.INT0IF = 0;

IPC0bits.INT0IP = 1;

if(count > 15)//Restrict count to 0-15, needing only 4 bits

count = 0;

delay();

}

}