ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 02 / 25 / 2021

Lab 6

ABSTRACT:

A lab designed to demonstrate how to program the PIC32 microcontroller to use its peripheral pin select (PPS). Then, we can utilize multiple interrupts at once which is used for our Grayhill 61C Optical Encoder. Similar to the previous labs, this will also enhance skills in reading documentations for new and existing parts.

**INTRODUCTION:**

The goal of lab 6 is to program our microcontroller to understand the input values of A and B from our Grayhill 61C Optical Encoder and represent either an increment or a decrement depending on the direction of rotation. We will be utilizing the INT0 and INT1 interrupts of the PIC32. Since there is only one hardwired interrupt we must assign our pin to INT1 so that it can be used. The pin for INT1 interrupt is the B9 pin; we select this with IPC1bits.INT1IP = 0b0010;. To perform our lab, we required a variety of materials including:

* A breadboard
* Wires
* Resistors (2x 5.1kΩ)
* Grayhill 61C Optical Encoder
* PIC32MX150F128D
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

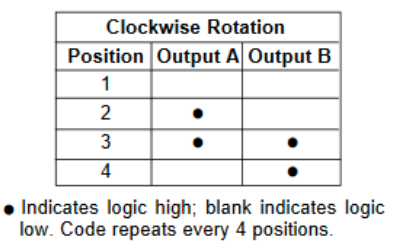
The AD2 is going to be a partial power source and our output LEDs. Instead of using actual LEDs we will use the WaveForms software and simulate the LED output. The partial power source will power the PIC32 chip and be a 5V source for the Grayhill 61C Optical Encoder. However, it will not power the MPLAB Snap Debugger. It will be powered off of a Micro USB cord.

**EXPERIMENTAL PROCEDURES:**

We must assume that the individual reproducing this lab has already setup their breadboard in a manner that their PIC32 chip can be programmed using the MPLAB X IDE software.

To begin the lab, we will observe the circuit diagram figure 1 to show exactly how we need to wire our breadboard. To begin we will start with the simplest part of the circuit, the LEDs. We are using StaticIO in WaveForms to simulate real LEDs. We want to connect our LEDs to A0 through A3. Next, we need to set our waveform generator to 5V square wave so that we can power our Grayhill 61C Optical Encoder. That 5V will connect to pin 6 of our Grayhill 61C Optical Encoder. Next, we want to connect two separate 5.1kΩ resistors connecting from pin 6 of the Grayhill 61C Optical Encoder one going to A (pin 5) and one to B (pin 4). These are pull up resistors for our A and B lines, thus we do not need internal pull up resistors. However, it is worth mentioning that we must use the 5V tolerant lines since we’re applying 5V to our PIC32. Hence why we used pins B9 and B7 for our A and B inputs. Next, we will ignore pins 3 and 2 of our Grayhill 61C Optical Encoder since those are for the push button which we’re not using. Lastly, pin 1 is ground which can be connected by itself. The output of A from the Grayhill 61C Optical Encoder will connect to INT0 which will be B7 or pin 16 on our microcontroller. And similarly, the output of B from the Grayhill 61C Optical Encoder will connect to INT1 which will be B9 or pin 18 on our microcontroller.

The code for this project is mostly provided in our skeleton code for the lab. However, the overall idea is not much different from lab 5. First, we need to include our sys/attribs.h so that we can use our interrupts. Next, we setup a global count variable so that we can keep track of the count across all of our functions. We want to setup the INT0 interrupt first which is Int0Handler. We’re using vector 3 since that maps to INT0 and then we’re going to check if B9 or our B output is high and then determine if A or B7 is also high; in that case we need to decrement our counter and then reset our interrupts. This code is better understood from this diagram below:



When A and B are both high we know that we’re in position 3 and when we switch from both A and B being high to only B being high we know that we’ve rotated in the clockwise direction. The code for the second interrupt (INT1) is a copy paste of INT0 but inverting the check for A and B. In our main function we need to first grab the initialization code from lab 5 which helps setup our interrupts. This is the first three lines of main. Next we will make all of the A pins outputs which will be our LEDs. The pins B7 and B9 will be inputs since those are coming from the Grayhill 61C Optical Encoder. And lastly, we want to set the interrupt’s priority to anything other than 0. Next, we want to determine if our A interrupt has been triggered or if we’re in a position with A being logic high. If so, we want to set interrupt 1’s edge polarity to rising edge otherwise falling edge. Then, we enable external interrupt 0. We assign INT1 to pin B9 and set its interrupt polarity using 0b0010. Similar to before we want to set the edge polarity depending on if we’re disabled on pin B9. Lastly, we enable the INT1 interrupt.

**RESULTS and DISCUSSION:**

Once the microcontroller was programmed it was apparent that things were working as expected when we started to rotate the Grayhill 61C Optical Encoder and we watched the LEDs display zero to fifteen in binary. Also, as expected, when the Grayhill 61C Optical Encoder is turned in the counterclockwise direction we watch the binary count go in reverse, decrementing one at a time.

One problem during this experiment was the LEDs not displaying the correct output, and not displaying anything on LED 4. I was unsure as to why my LEDs were lighting up in the wrong order even though my code was displaying the correct outputs. Eventually I determined that I was connecting my LEDs to A0, A1, A3, and A4 and then writing to A using LATA which was doing what I wanted but I had it wired incorrectly. It should have been A0-A3 and not skipping A2. Once I switched the wires everything began to work correctly.

This week we demonstrated how to utilize the idea of multiple interrupts using peripheral pin select (PPS). Being able to read the datasheet and determine what pin everything needs to be connected to and how to select your interrupt is crucial in the real world. Utilizing more than one interrupt will be very common in microcontroller programming careers.

**CONCLUSION:**

To conclude, lab 6 taught us many useful techniques on utilizing many interrupts on the PIC32 microcontroller. Also, as with many other labs, this one teaches us how to debug our circuit to check if everything is working. Assuming we are unsure if the Grayhill 61C Optical Encoder is working properly we can put two scope pins on the A and B line and watch the waveform to see what position we are in. These techniques are very informative and will most definitely help one day in the field. This lab also taught us more complex ways to analyze problems and how to solve them.

Diagram, schematic

Description automatically generated**FIGURES AND TABLES:**

**Figure 1: Wiring for lab 6 (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

#include <sys/attribs.h>

int count = 0;

void \_\_ISR(3)Int0Handler(void){

if (PORTBbits.RB9 == 1){ // B Output From Grayhill61C Optical Encoder

if (PORTBbits.RB7 == 1) // A Output From Grayhill61C Optical Encoder

count--;

else

count++;

}

else {

if(PORTBbits.RB7 == 0)

count--;

else

count++;

}

INTCONbits.INT0EP = ~INTCONbits.INT0EP; // Invert Current Polarity of INT0

IFS0bits.INT0IF = 0; // Clear the INT0 interrupt flag

}

void \_\_ISR(7)Int1Handler(void){

if(PORTBbits.RB7 == 1){

if(PORTBbits.RB9 == 1)

count++;

else

count--;

}

else {

if(PORTBbits.RB9 == 0)

count++;

else

count--;

}

INTCONbits.INT1EP = ~INTCONbits.INT1EP; // Invert Current Polarity of INT1

IFS0bits.INT1IF = 0; // Clear the INT1 interrupt flag

}

int main(void) {

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

CFGCONbits.JTAGEN = 0;

TRISA = 0x0; // LED Outputs

TRISBbits.TRISB7 = 1; // A input

TRISBbits.TRISB9 = 1; // B input

IPC0bits.INT0IP = 0x1; // Interrupt 0 Priority

if (PORTBbits.RB7 == 0)

INTCONbits.INT0EP = 1; // Set our edge polarity ( 1 rising, 0 falling edge )

else

INTCONbits.INT0EP = 0;

IEC0bits.INT0IE = 1; // Enable INT0 Interrupt

INT1R = 0b0100; // Assign Interrupt 1 to pin B9

IPC1bits.INT1IP = 0b0010; // Interrupt Polarity

if (PORTBbits.RB9 = 0)

INTCONbits.INT1EP = 1;

else

INTCONbits.INT1EP = 0;

IEC0bits.INT1IE = 1; // Enable INT1 Interrupt

while(1) {

LATA = count;

if(count > 15)

count = 0;

if(count < 0)

count = 15;

}

}