**PERIPHERAL PIN SELECT**

**Aayahna M. Herbert**

**October 23, 2017**

**ECE 3720 Section 007**

**Microcontroller Interfacing Laboratory**

**Abstract:** A circuit was built in order to send two interrupts to the circuit board’s LED lights. Using a Grayhill 61C Optical Encoder, as the knob is turned clockwise, the LED lights should represent the binary numbers incrementing from zero to fifteen and fifteen to zero when turned counter-clockwise.

**INTRODUCTION**

The purpose of this lab was to design a circuit that sends two inputs to the microcontroller from the Grayhill 61C optical encoder. The microcontroller would then output signals to four LEDs that represent binary numbers 0 through 15 when turned on. When done correctly, as the knob is turned, each click should increment or decrement the binary number represented by the LEDs.

**EXPERIMENTAL PROCEDURES**

The equipment used include one PIC32 MC, NI-ELVIS II board, and a Grayhill 61C optical encoder. The first thing to be done, to avoid forgetting it later, is to set ports B0 through B3 as outputs and B7 and C4 as inputs. Pins B0 through B3 are used to enable the board’s LED lights. Because we are dealing with two interrupt signals, the registers INT0 and INT1 should be enabled, set to high direction, be interrupted on a low value, and have a priority of one by using INTCON, IECx, IFSx, and IPCx. In addition to the interrupt, it must also be globally enabled by adding INTEnableSystemMultiVectoredInt() to the main function. With the main function comes the ISR functions for the specific register numbers in use, void \_\_ISR(3) int0(void) and void \_\_ISR(7) int1(void). Inside of these functions goes the commands for what should happen to the LED lights when the interrupt button is pressed. For both functions, we want the count for the LEDs to increment or decrement when clockwise or counter-clockwise. Because INT0 is connected to Output A from the optical encoder, we want the count to increase when it is set to high and the port for Output B is low or when Output A is low and the port for Output B is also low; if these aren’t the case, then the count will decrease. Because INT1 is connected to Output B from the optical encoder, the count will increase when Output B and the port for Output A are both high and when they are both low; if these aren’t the case, then the count will decrease. Because we want the signals to be inversed after the knob has been turned and the count has been changed, both functions should include code that changes the signals to the opposite of what its port is at that moment; make sure the flag is set to low at the end of each function to show that the interrupt is done. In terms of wiring, make sure the power the chip and encoder are receiving is +5V and the encoder’s two resisters are 10kΩ each since it has to be at least 8.8kΩ to be effective.

**RESULTS**

After the circuit was hooked up and ready to be tested, the program detected no compiling errors when ran so the board could then be tested. While running tests, when the encoder’s knob was turned, the lights lit up correctly and went through the binary numbers zero through fifteen and looped through them when the end of the number range was reached both ways.

**DISCUSSION**

Problems arose initially when the LEDs didn’t light up in order each time the knob was turned with each click. The LEDs and interrupt button were tested individually to make sure none of the ones in use were broken. The code and wiring were second-checked by the TA and the wires all came from a new pack recently purchased to avoid the possibility of dead wires being used and affecting the lab results. The source of the error was the missing piece of code that inversed the interrupt from high or low to make sure it is able to react to each turn from the knob of the encoder.

**CONCLUSIONS**

In general, the take-away of this lab was to learn how to use PPS function and to get better understanding of a rotary pulse generator, how to use it and understand its logic on a deeper level.

**FIGURES AND TABLES**

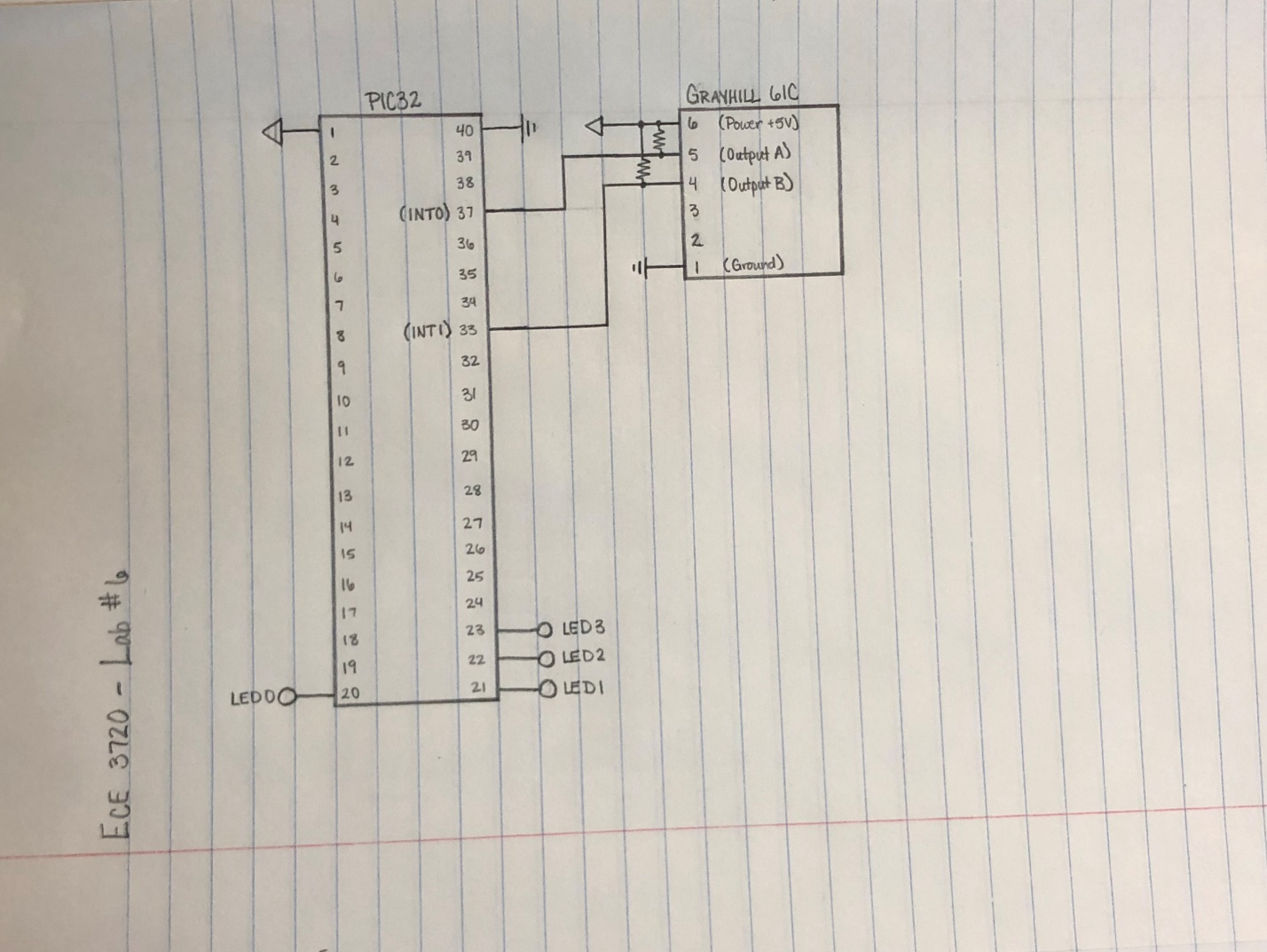


Figure 1: Circuit Schematic

**CODE**

#include<plib.h>

int count = 0;

void \_\_ISR(3) **int0**(void)

{

if(INTCONbits.INT0EP == 1)

{

if(PORTCbits.RC4 == 0)

count++;

else

count--;

}

if(INTCONbits.INT0EP == 0)

{

if(PORTCbits.RC4 == 0)

count--;

else

count++;

}

if(PORTBbits.RB7 == 1)

INTCONbits.INT0EP = 0;

else

INTCONbits.INT0EP = 1;

IFS0bits.INT0IF = 0;

}

void \_\_ISR(7) **int1**(void)

{

if(INTCONbits.INT1EP == 1)

{

if(PORTCbits.RB7 == 0)

count--;

else

count++;

}

if(INTCONbits.INT1EP == 0)

{

if(PORTCbits.RB7 == 0)

count++;

else

count--;

}

if(PORTBbits.RC4 == 1)

INTCONbits.INT0EP = 0;

else

INTCONbits.INT0EP = 1;

IFS0bits.INT1IF = 0;

}

main()

{

INTEnableSystemMultiVectoredInt();

TRISBbits.TRISB0 = 0;

TRISBbits.TRISB1 = 0;

TRISBbits.TRISB2 = 0;

TRISBbits.TRISB3 = 0;

TRISBbits.TRISB7 = 1;

INTCONbits.INT0EP = 1;

IEC0bits.INT0IE = 1;

IFS0bits.INT0IF = 0;

IPC0bits.INT0IP = 1;

TRISBbits.TRISC4 = 1;

INTCONbits.INT1EP = 1;

IEC0bits.INT1IE = 1;

IFS0bits.INT1IF = 0;

IPC0bits.INT1IP = 1;

PPSInput(4,INT1,RPC4);

while(1)

{

LATB = count;

if(count > 15)

count = 0;

if(count < 0)

count = 15;

}

}