ECE 3720

Microcomputer Interfacing Laboratory

Section 7

Parth Patel

Date Performed: April / 25 / 2020

Final Project

ABSTRACT:

A motor was controlled via a rotary encoder. The rotary encoder controlled the PWM signal output from the PIC32 and the PWM signal would control the motor speed.

**INTRODUCTION:**

Once configured, the PIC32 microcontroller can be programmed to perform a variety of tasks. This final project utilized many different concepts taught throughout lab such as PWM signals, PPS remapping, and interrupts. The circuit contained a rotary encoder, whose output was connected to the PIC32. Based on the value of the rotary encoder, the PWM signal would change. This PWM signal was directly outputted to a motor driver. The motor driver used the PWM signal and separate power supply (The 9V battery) to actually drive the motor. The materials used were the PIC32MX150F128D, the analog discovery 2 device, a rotary encoder, a motor driver chip, a motor and basic circuit element (breadboard, wires, capacitors, resistors, etc….)

**EXPERIMENTAL PROCEDURES:**

In order to be able to program the PIC32MX150F128D, the device was firstly setup according to the steps that were provided (email sent by Dillion).

Next the desired circuit was constructed. Pin B1 and B2 of the PIC32 were respectively connected to the A and B outputs of the rotary encoder. These were arbitrary chosen pins (interrupts were not used since there was random triggers due to the large current draw of the motor). The rotary encoder was wired as required by its datasheet: 5V and ground were connected to the respective pins. In addition, pull-up resistors of value 8.2k ohms were connected to Pins A and B. This specific value was used as it was recommended by the datasheet.

The motor driver was wired to the motor as recommended by its datasheet. It was given 5V power and ground. Section 4 on the motor driver was used (for arbitrary reasons). The input to section 4, pin 15 or 4A, was connected to pin A0 of the PIC32. Pin A0 was where the PWM signal was outputted from the PIC32. The output from section 4, pin 14 or 4Y, was connected to the motor. A diode was wired in parallel to the motor as recommended by the motor driver’s datasheet. The power supply for the motor was on pin 8 of the motor driver or VCC2. Due to a strain of supplies, a 9V battery sufficed.

The AD2 was configured. Under supply voltage, A positive 5V was supplied to the board as a part of the general setup for the PIC32 microcontroller.

The code was mostly derived from previous labs. An interrupt was not used. All of the same code for the construction of the PWM signal was used from lab 8. However, there were some modifications since a rotary encoder was used. Since interrupts were not available for usage, a circumnavigated method was needed. Within the main function, the current values from the rotary encoder were read and stored in global variables. These variables were called oldAVal and oldBVal. There was a global array created for each the A and B values from the rotary encoder. The arrays held values in the order that they occur by position. For example, the values in A were {0,1,1,0}. This was derived from the tables below in the datasheet of the rotary encoder.

Table

Description automatically generated

Within the while loop, each array (Array A and Array B) were traversed with each iteration of the while loop. The arrays were traversed until right combo (i.e. a[0] and b[0]) matched the current values read from the rotary encoder. If this was the case, then the next step was to deem whether the rotary encoder was incremented or decremented (if neither, then the code did nothing). This was deemed by comparing the currently read A, B vals with the values previously stored in oldAVal and oldBVal. If the currently read values were “up” (in terms of position of the rotary encoder) from the old stored values, then a counter would be incremented. However, if the currently read values were “down” from the old stored values, then the counter would decrement. In the end, the value of this counter was used to fetch the width of the PWM signal from a PWM array. This is how the new code (rotary encoder stuff) interacted with the copied lab 8 code.

In addition, extra code segments were added in order to loop the rotary encoder position back around. However, this failed.

**RESULTS and DISCUSSION:**

The final behavior of the MC/Circuit was partially as expected. When the rotary encoder was turned, the PWM signal would adjust accordingly and the motor speed would adjust as well. However, the final product failed to loop around (i.e when you turned from position 4 to position 1, the microcontroller failed to increment). This project was made a lot more difficult because interrupts could not be used. In the future, however, the interrupts can be used if the power disturbance from the motor could be isolated from the rest of the circuit. Also, another alternative improvement would be brute force testing and remaking of the code in order to get the final product to loop around. A good application of today’s lab would be in a ceiling fan. When the rotary encoder is turned, the fan speed is adjusted accordingly. The reason being that the microcontroller was small, and its accompanying circuit can be downsized to allow for very useful applications.

**CONCLUSION:**

In conclusion, the experiment did not completely go as expected. The project was a big hassle given the limited and fault parts provided. But ultimately the result that was expected was partially found. We were able to control the motor speed with the rotary encoder.

**FIGURES AND TABLES:** **Diagram, schematic

Description automatically generated**

**Figure 2: Pre-lab for Final Proj**

**A close-up of a circuit board

Description automatically generated with medium confidence**

**Figure 3: The actual wiring for the circuit picture**

**CODE:**

#include<xc.h>

#include<sys/attribs.h>

#define total\_vals 5

//This is here to make timing calculations easier!

int PWM[total\_vals] = {0, 250, 500, 750, 1001};

int i;

int Avals[4] = {0,1,1,0};

int Bvals[4] = {0,0,1,1};

int oldAval = 0;

int oldBval = 0;

int newAval = 0;

int newBval = 0;

int count = 0;

void delay(){

int i, j;

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

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

}

main()

{

i = 0;

OC1RS = PWM[i];

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

CFGCONbits.JTAGEN = 0;// jtag line

ANSELB = 0;

ANSELA = 0;

TRISBbits.TRISB1 = 1; // input A

TRISBbits.TRISB2 = 1; // input B

oldAval = PORTBbits.RB1; // read and store initial values

oldBval = PORTBbits.RB2;

TRISAbits.TRISA0 = 0; // output

RPA0R = 0b101;

// setup timer

T2CONbits.TCS = 0;

T2CONbits.TGATE = 0;

T2CONbits.TCKPS = 0;

PR2 = 1000;

//setup the OC registers

OC1CONbits.OC32 = 0;

OC1CONbits.OCTSEL = 0;

OC1CONbits.OCM = 6;

// enable timer then enable the OC

T2CONbits.ON = 1;

OC1CONbits.ON = 1;

while(1) //Run Continuously

{

if(i==4){

i = 0;

}

newAval = PORTBbits.RB1;

newBval = PORTBbits.RB2;

if (Avals[i] == newAval && Bvals[i] == newBval){

// found the right combo

if(Avals[i+1]==oldAval && Bvals[i+1]==oldBval){

oldAval = newAval; // get ready for next change

oldBval = newBval;

count--;

if(count==-1){

count = 4;

}

}

if(Avals[i-1]==oldAval && Bvals[i-1]==oldBval){

oldAval = newAval; // get ready for next change

oldBval = newBval;

count++;

if(count==5){

count = 0;

}

}

}

OC1RS = PWM[count];

i++;

}

}