Christopher Brant
C19816588
Due on 12/13/2017

ECE 3270 Microcontroller Interfacing Lab
Final Project: Easy Open Safe Cabinet

Abstract

The purpose of this final project is to synthesize all of the things that we have learned this semester and create a unique and interesting system that utilizes the PIC32MX150F128D microcontroller to control all aspects of the system. In my project, this was done by utilizing concepts from Lab 5: Interrupts, Lab 6: Peripheral Pin Select, Lab 8: Pulse Width Modulators, and Lab 10: Analog to Digital Conversion. All together these concepts were used to control a cabinet door by having the pressure registered by a force sensitive resistor dictate the speed and direction that a motor will turn to either open or close a cabinet door. As well as this, a Greyhill 61C Encoder was used to set the cabinet to be locked or unlocked. This culminated in what I am calling the Easy Open Safe Cabinet.

Introduction

Over the course of this project I learned how to combine the output from the PIC32MX150F128D's ADC module with some conditional logic to control the speed and direction of a basic motor using variable amounts of pressure to an FSR, which is the input that was supplied to the ADC module. When the FSR is not being pressed, the motor does not turn, when the FSR is "moderately" pressed, the motor will turn to close the cabinet door, and when the FSR is "forcibly" pressed, the motor will turn to open the cabinet door. The Greyhill 61C Encoder is used to determine the locked state of the cabinet. When initially turned on, the cabinet will be locked, but when the encoder is turned in the counter-clockwise direction, the cabinet will unlock, and when the encoder is turned in the clockwise direction, the cabinet will lock. The encoder input can be set to match an encoded combination if desired, but for the purpose of this project, the locking and unlocking mechanism was simplified for demonstration purposes.

Experimental Procedures

- The first thing that must be done is to wire the overall circuit to be just as it is shown in Figure 1.
- The next thing that is to be done is to write the code for the preprocessor definitions, includes, and global variables.
- Then in the order that the file is formatted, the code will need to be written for what will happen when the ADC interrupt is created, meaning that this interrupt code determines what happens every time a value from the ADC has finished converting and is placed in the buffer.
- Following the writing of the ADC interrupt code, the code for the when Output A interrupts from the encoder must be written. Then the interrupt code must be written for when Output B interrupts from the encoder.
- Next the main function will be written, which will include the basic setup of tristate registers and analog select registers, as well as the basic setup for the two external interrupts.

- After the main function is written, the code for the adcSetup function must be written so that the ADC setup bits will be set correctly for use by the cabinet.
- Lastly, the code for the pwmSetup function must be written so that the PWM will also function as expected for use by the cabinet.

Results

There were no tabulated values and no values necessary for the record. Observations of this project would include the difficulty in making many systems that rely on interrupts to work together. As originally, it was planned to implement a buzzer to make a sound when the cabinet locked or unlocked, however this was unable to be implemented due to time constraints as the base operations took priority. Another observation would be that it was a difficult task to get the locking and unlocking mechanism working, as for some reason the mechanism did not want to re-lock after it was unlocked. Lastly, it was definitely seen how I was unable to notice that the ADC interrupt flag was never getting cleared, therefore not allowing any other code to ever run.

Discussion

Overall, the final conclusions that I could make of this project would include the statement that the main design of this project was not incredibly difficult, yet debugging it was painful due to my inability to recognize that the ADC interrupt flag bit was never getting cleared at the end of its interrupt code. Another conclusion that could likely be made is that the point at which the PWM timer is turned on and off is important, as originally I was turning the PWM timer on and off along with the ADC, however it was actually necessary to do this inside of the ADC interrupt so that the motor would only turn while the ADC was giving a value greater than 8. Lastly, it could also be said that during this project, some extra aspects and features had to be cut, yet in the end, a working motorized cabinet that is run by a force sensitive resistor or “pressure pad” is unique and useful without extra music playing when the locking mechanism changes state.

Conclusions

The conclusions of this experiment include that my ability to debug embedded code for the PIC32MX150F128D is rather lacking, as I was unable to recognize the missing line of code that clears the ADC interrupt flag inside the ADC interrupt code. Just as well, the instances where Timer 2 is turned on and off for the use of the PWM was important, as before correcting it, the motor would sometimes not turn off as it was intended to. Finally, this unique system is now and will be useful for myself in the future as well. Specifically, as I plan to build this exact capability into the side panel door of a retro arcade cabinet to unveil the retro controllers that are stored inside the panel. Overall this project was equally enjoyable and frustrating, yet still satisfying to see work as intended in the end.

Christopher Brant
C19816588
Due on 12/13/2017

Figures and Tables

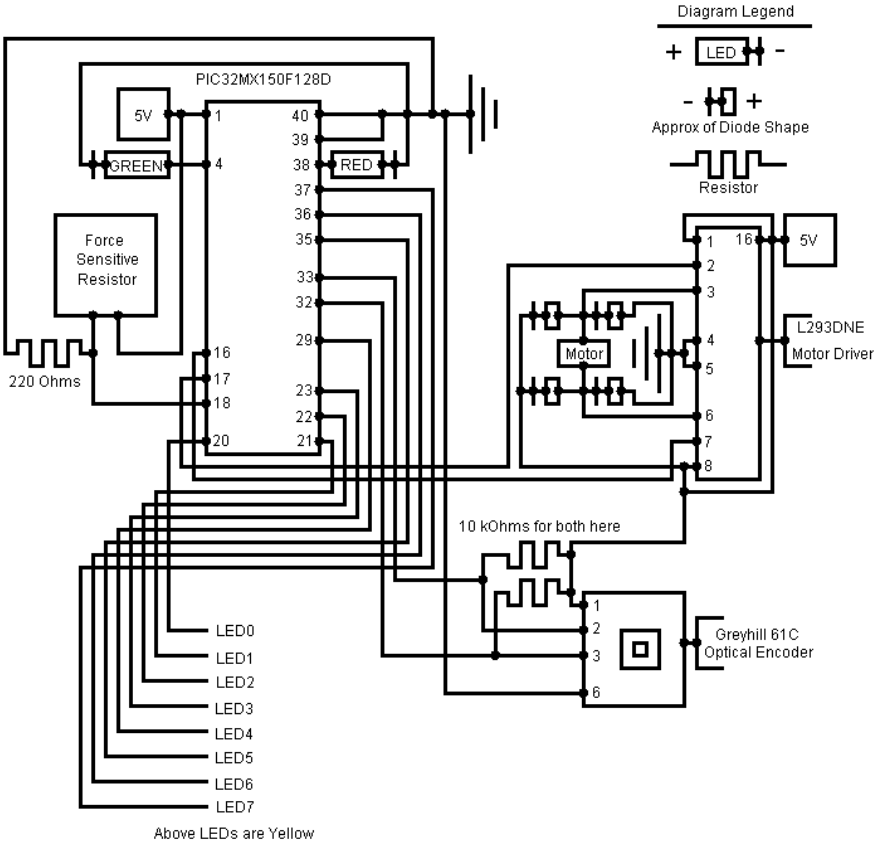

Figure 1: Wiring Diagram of the circuit for Final Project: Easy Open Safe Cabinet

```
/* Christopher Brant
 * C19816588
 * Final Project: Easy Open Safe Cabinet
 * Due on 12/15/2017
 */

#include <plib.h>

// Define motor duty cycle *probably not necessary*
#define fullDutyCycle 10
#define dCycleInc 0.25

void adcSetup(void);
void pwmSetup(void);

// Declare global variables
int i, j, k;
int delayInc;
int dCycleMult = 0;
int dCycleDirect = 1;
int ADCsum, ADCval;
int lockOn = 1;
int stop = 1;
```

Figure 2: Header Comments, Preprocessor Definitions, and Global Variable Declarations from the code for Final Project: Easy Open Safe Cabinet. *The motor duty cycle *was* necessary*

```
void __ISR(23) adcDone(void)
{
    ADCsum = 0;

    // Sum the values of the 8 most significant bits in each buffer
    ADCsum += (ADC1BUF0 >> 2);
    ADCsum += (ADC1BUF1 >> 2);
    ADCsum += (ADC1BUF2 >> 2);
    ADCsum += (ADC1BUF3 >> 2);
    ADCsum += (ADC1BUF4 >> 2);
    ADCsum += (ADC1BUF5 >> 2);
    ADCsum += (ADC1BUF6 >> 2);
    ADCsum += (ADC1BUF7 >> 2);
    ADCsum += (ADC1BUF8 >> 2);
    ADCsum += (ADC1BUF9 >> 2);
    ADCsum += (ADC1BUFA >> 2);
    ADCsum += (ADC1BUFB >> 2);
    ADCsum += (ADC1BUFC >> 2);
    ADCsum += (ADC1BUFD >> 2);
    ADCsum += (ADC1BUFE >> 2);

    // Find the mean of those values
    ADCval = ADCsum / 15;

    // Usage of LEDs
    LATBbits.LATB0 = (ADCval & 0x01) >> 0;
    LATBbits.LATB1 = (ADCval & 0x02) >> 1;
    LATBbits.LATB2 = (ADCval & 0x04) >> 2;
    LATBbits.LATB3 = (ADCval & 0x08) >> 3;
    LATBbits.LATB4 = (ADCval & 0x10) >> 4;
    LATBbits.LATB5 = (ADCval & 0x20) >> 5;
    LATBbits.LATB6 = (ADCval & 0x40) >> 6;
    LATBbits.LATB7 = (ADCval & 0x80) >> 7;

  if (ADCval < 8)
  {
      stop = 1;
  }
  else if (ADCval > 8 && ADCval < 64)
  {
      dCycleDirect = -1;
      stop = 0;
  }
  else
  {
      dCycleDirect = 1;
      stop = 0;
  }

  if (stop == 0)
  {
      // Turn PWM timer on
      T2CONbits.ON = 1;

      if (dCycleDirect == 1)
      {
          if (dCycleMult == 4)
          {
              dCycleMult = 0;
              dCycleDirect = -1;
              OC1RS = fullDutyCycle * (1 - (dCycleInc * dCycleMult));
              OC3RS = fullDutyCycle;
          }
          else
          {
              dCycleMult++;
              OC1RS = fullDutyCycle;
              OC3RS = fullDutyCycle * (1 - (dCycleInc * dCycleMult));
          }
      }
      else if (dCycleDirect == -1)
      {
          if (dCycleMult == 4)
          {
              dCycleMult = 0;
              dCycleDirect = 1;
              OC3RS = fullDutyCycle * (1 - (dCycleInc * dCycleMult));
              OC1RS = fullDutyCycle;
          }
          else
          {
              dCycleMult++;
              OC3RS = fullDutyCycle;
              OC1RS = fullDutyCycle * (1 - (dCycleInc * dCycleMult));
          }
      }

      // Turn PWM timer off
      T2CONbits.ON = 0;
  }

   else
   {
       T2CONbits.ON = 1;

       OC1RS = fullDutyCycle;
       OC3RS = fullDutyCycle;

       // Turn PWM timer off
       T2CONbits.ON = 0;
   }

   IFS0bits.AD1IF = 0;
}
```

Figure 3: ADC Interrupt code for Final Project: Easy Open Safe Cabinet

```
void __ISR(7) OutputAEncoder(void)
{
    // Declare cw variable
    int cw = 0;

    // Check to see what direction encoder was turned
    // Checks for cw options, otherwise returns 0 for ccw
    if (INTCONbits.INT1EP == 1)
    {
        if (PORTCbits.RC4 == 1 && PORTCbits.RC3 == 0)
            cw = 1;
    }
    else if (INTCONbits.INT1EP == 0)
    {
        if (PORTCbits.RC4 == 0 && PORTCbits.RC3 == 1)
            cw = 1;
    }

    // Set lock on or off
    if (cw == 1)
        lockOn = 1;
    else
        lockOn = 0;

    // Flip the edge trigger bit for Output A
    INTCONbits.INT1EP ^= 1;

    // Turn off interrupt flag
    IFS0bits.INT1IF = 0;
}
```

Figure 4: Output A Interrupt function for the Greyhill 61C Encoder from the code for Final Project: Easy Open Safe Cabinet

```
void __ISR(11) OutputBEncoder(void)
{
    // Declare cw variable
    int cw = 0;

    // Check to see what direction encoder was turned
    // Checks for cw options, otherwise returns 0 for ccw
    if (INTCONbits.INT2EP == 1)
    {
        if (PORTCbits.RC4 == 1 && PORTCbits.RC3 == 1)
            cw = 1;
    }
    else if (INTCONbits.INT2EP == 0)
    {
        if (PORTCbits.RC4 == 0 && PORTCbits.RC3 == 0)
            cw = 1;
    }

    // Set lock on or off
    if (cw == 1)
        lockOn = 1;
    else
        lockOn = 0;

    // Flip the edge trigger bit for Output A
    INTCONbits.INT2EP ^= 1;

    // Turn off interrupt flag
    IFS0bits.INT2IF = 0;
}
```

Figure 5: Output B Interrupt function for the Greyhill 61C Encoder from the code for Final Project: Easy Open Safe Cabinet

```
int main(void)
{
    // Setting all basic registers to 0
    // Necessary bits to be 1 will be set when and where they are necessary
    TRISA = 0x0000;
    TRISB = 0x0000;
    TRISC = 0x0000;
    ANSELA = 0x0000;
    ANSELB = 0x0000;
    ANSELC = 0x0000;

    // Declare multi vector interrupts enabled
    INTEnableSystemMultiVectoredInt();
    // Set peripheral pin macro and tristate for INT1 and INT2
    TRISCbits.TRISC4 = 1;
    TRISCbits.TRISC3 = 1;
    PPSInput(4, INT1, RPC4);
    PPSInput(3, INT2, RPC3);

    // Setting interrupt control register bits
    INTCONbits.INT1EP = 1 ^ PORTCbits.RC4;
    INTCONbits.INT2EP = 1 ^ PORTCbits.RC3;

    // Setting the interrupt enable register bits
    IEC0bits.INT1IE = 1;
    IEC0bits.INT2IE = 1;

    // Setting the interrupt priority control bits
    IPC1bits.INT1IP = 1;
    IPC2bits.INT2IP = 1;

    // Setting the interrupt flag status register bits
    IFS0bits.INT1IF = 0;
    IFS0bits.INT2IF = 0;

    // Set up ADC for preparing digital values for the FSR and Servo
    adcSetup();

    // Set up PWM to run the motor
    pwmSetup();

    // Be an electronically locking and motorized cabinet
    while(1)
    {

        if (lockOn == 1)
        {
            AD1CON1bits.ON = 0;     // Turn off ADC when locked
            LATBbits.LATB8 = 1;     // Turn red 'locked' LED on
            LATBbits.LATB9 = 0;     // Turn green 'unlocked' LED off
        }
        else if (lockOn == 0)
        {
            AD1CON1bits.ON = 1;     // Turn on ADC when unlocked
            LATBbits.LATB8 = 0;     // Turn red 'locked' LED off
            LATBbits.LATB9 = 1;     // Turn green 'unlocked' LED on
        }
    }

    return 0;
}
```

Figure 6: Main function from the code for Final Project: Easy Open Safe Cabinet

```
// This function handles all setup for the FSR and ADC
void adcSetup(void)
{
    // Setting tristate registers
    TRISAbits.TRISA0 = 1;   // Pin A0 is input

    // Setting analog inputs
    ANSELAbits.ANSA0 = 1;  // Pin A0 is analog

    // Setting up ADC interrupts for use
    IFS0bits.AD1IF = 0;     // Clears ADC interrupt flag
    IPC5bits.AD1IP = 7;     // Sets ADC interrupt priority to 3
    IEC0bits.AD1IE = 1;     // Enables ADC interrupt

    // Setting up the data output format
    AD1CON1bits.FORM = 0;   // Sets to 16-bit integer
    // Setting up the sampling and conversion triggers
    AD1CON1bits.SSRC = 7;   // Sets auto-conversion on
    AD1CON1bits.ASAM = 1;   // Sets auto-sampling on
    // Setting voltage references
    AD1CON2bits.VCFG = 0;   // Sets the voltage references to auto-values
    // Setting channel scan and samples/converts per interrupt
    AD1CON2bits.CSCNA = 0;  // Turns scanning off
    AD1CON2bits.SMPI = 14;  // Interrupts after the 15th sample/convert
    // Setting buffer mode select and input sample select
    AD1CON2bits.BUFM = 0;   // Sets buffer to a single 16-bit word buffer
    AD1CON2bits.ALTS = 0;   // Sets MUX A as the only input
    // Setting clock reference and auto-sample time bits
    AD1CON3bits.ADRC = 0;   // Sets clock to the PBCLK
    AD1CON3bits.SAMC = 12;  // Sets auto-sample time to 12 TAD
    // Setting the prescaler for the ADC conversion clock
    AD1CON3bits.ADCS = 0;   // Sets the prescaler to 2*TPB
    // Setting analog channel for reading
    AD1CHSbits.CH0SA = 0;   // Sets AN0 as the input select bit
}
```

Figure 7: ADC Setup function code for Final Project: Easy Open Safe Cabinet

```
// This function handles all setup for PWMs to drive the stepper motor
void pwmSetup(void)
{
    // Setting the Output Peripheral Pin Selects
    PPSOutput(1,RPB15,OC1);
    PPSOutput(4,RPB14,OC3);
    // Setting Timer 2 Prescaler bits
    T2CONbits.TCKPS = 0x0;
    // Setting Timer 2 clock source select bit
    T2CONbits.TCS = 0;
    // Setting Timer 2 gated time accumulation enable bit
    T2CONbits.TGATE = 0;
    // Setting Timer 2 Period and OCxRS registers
    PR2 = fullDutyCycle;
    OC1RS = fullDutyCycle * dCycleInc * dCycleMult;
    OC3RS = fullDutyCycle * dCycleInc * dCycleMult;
    // Setting Output Compare enable and select bits
    OC1CONbits.ON = 1;
    OC1CONbits.OCM = 7;
    OC3CONbits.ON = 1;
    OC3CONbits.OCM = 7;
    // Turn timer on last
}
```

Figure 8: PWM Setup function code for Final Project: Easy Open Safe Cabinet