Section 1: General Lab Info */0.5 points*

Microprocessors II and Embedded Systems

EECE 4800

Lab 1: Sensor Design and Analog Digital Conversion

Prof: Yan Luo, TA: Ioannis Smanis

Group # 1

Andrew MacGregor

Hand in: October 2, 2017

Due: October 2, 2017

Section 2: Contributions */1 points*

1. **Group Member 1 - Andrew MacGregor**

Wrote main loop logic
Figured out how to time different events
Helped configure ADC parameters
Debugged in lab

2. **Group Member 2 - Grayson Colwell**

Wrote servo control code
Wrote main loop logic
Helped configure ADC parameters
Debugged in lab

3. **Group Member 3 - Jose Velis**

Helped configure ADC parameters
Debugged in lab
Made circuit schematic

Section 3: Purpose */0.5 points*

The purpose of this project is to understand how to work with embedded microcontrollers when creating a sensor. ADC operation was demonstrated by connecting an analog sensor circuit to an ADC input channel. Also PWM signals and servo motor operation were explored.

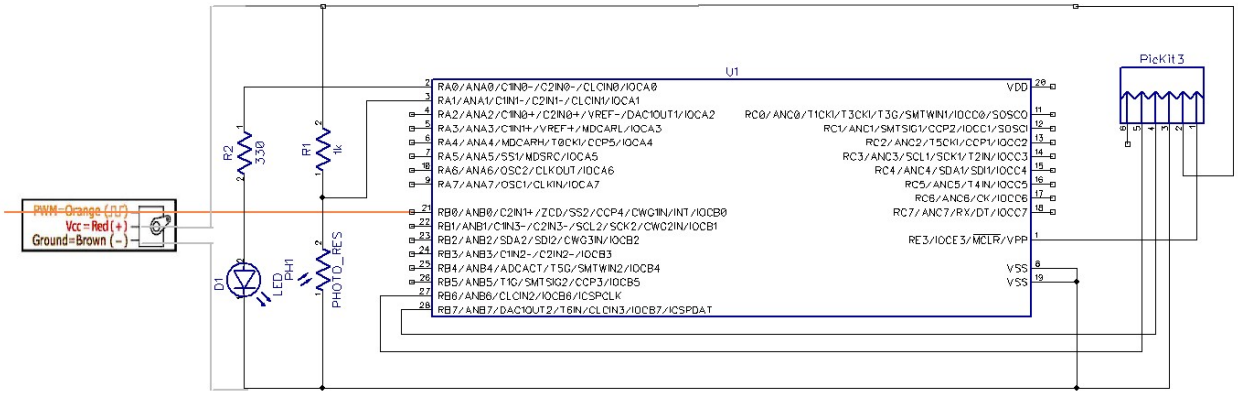
Section 4: Introduction /0.5 points

The final deliverable of the project is a microcontroller that controls a light sensor and a servo motor that can cover/uncover the sensor from light. The light sensor circuit consisted of a photoresistor worked into a voltage divider circuit. High light intensity causes the photoresistor's resistance to decrease, and caused the ADC to read a lower value. The servo motor was placed next to the photoresistor, and a piece of cardboard was taped to the servo fan so that it would cover and uncover the photoresistor as the servo rotated. The microcontroller was programmed to rotate the servo motor approximately every two seconds.

Section 5: Materials, Devices and Instruments /0.5 points

- PIC16F18857 at 3.3V
- Pickit3 using low voltage programming
- LED
- Servo motor
- 330 Ω resistor
- 1k Ω resistor
- Photoresistor
- Breadboard & wiring kit
- Lab bench oscilloscope
- Lab bench digital multimeter
- MPLAB X IDE with XC8 compiler

The following circuit diagram shows the circuit diagram for the sensor connections as well as the programming connections.



PicKit 3 Pin #	Description
1	MCLR'/Vpp
2	VDD Target
3	Vss (ground)
4	Data
5	Clock
6	PGM (null)

Figure 1: Shows all circuit connections made in this project.

Hardware design:

LED:

The LED was connected in series with a 330Ω resistor in between an output port of the PIC and ground. The 330 Ω resistor was chosen so the LED would shine without burning out. 3.3V was selected as Vdd so none of the components would be damaged at 5V.

Servo:

The servo was connected as follows. The red wire was connected to Vdd, the brown wire was connected to ground, and the orange wire was connected to a PWM signal from an output port from the PIC. The datasheet recommended Vdd = 5V but we tried connecting it to our 3.3V rail, and it had enough power to move the piece of cardboard connected to its blade, so we chose to use that value.

Photoresistor:

The Photoresistor was connected in series with a 1k Ω resistor in between Vdd and ground. The node between the photoresistor and the 1k Ω was connected to an ADC input port. The photoresistor's resistance was measured to vary between about 800 Ω and 3kΩ depending on the intensity of the light. 1k Ω was chosen as the series resistor because it lies somewhere between 800 and 3k Ω. The voltage input to the ADC was calculated using the voltage divider equation below.

$$V_{ADC\ in} = R_{Pr}\frac{V_{dd}}{1k + R_{Pr}}$$

PICkit3:

The PICkit was connected as specified in its datasheet with two exceptions. The reset signal was not tied to Vdd with a pull up resistor and pin 6 was left unconnected. The lack of a pull-up resistor may have caused us problems during programming, but we didn't catch our mistake until Ioannis pointed it out during the demonstration.

Software design:

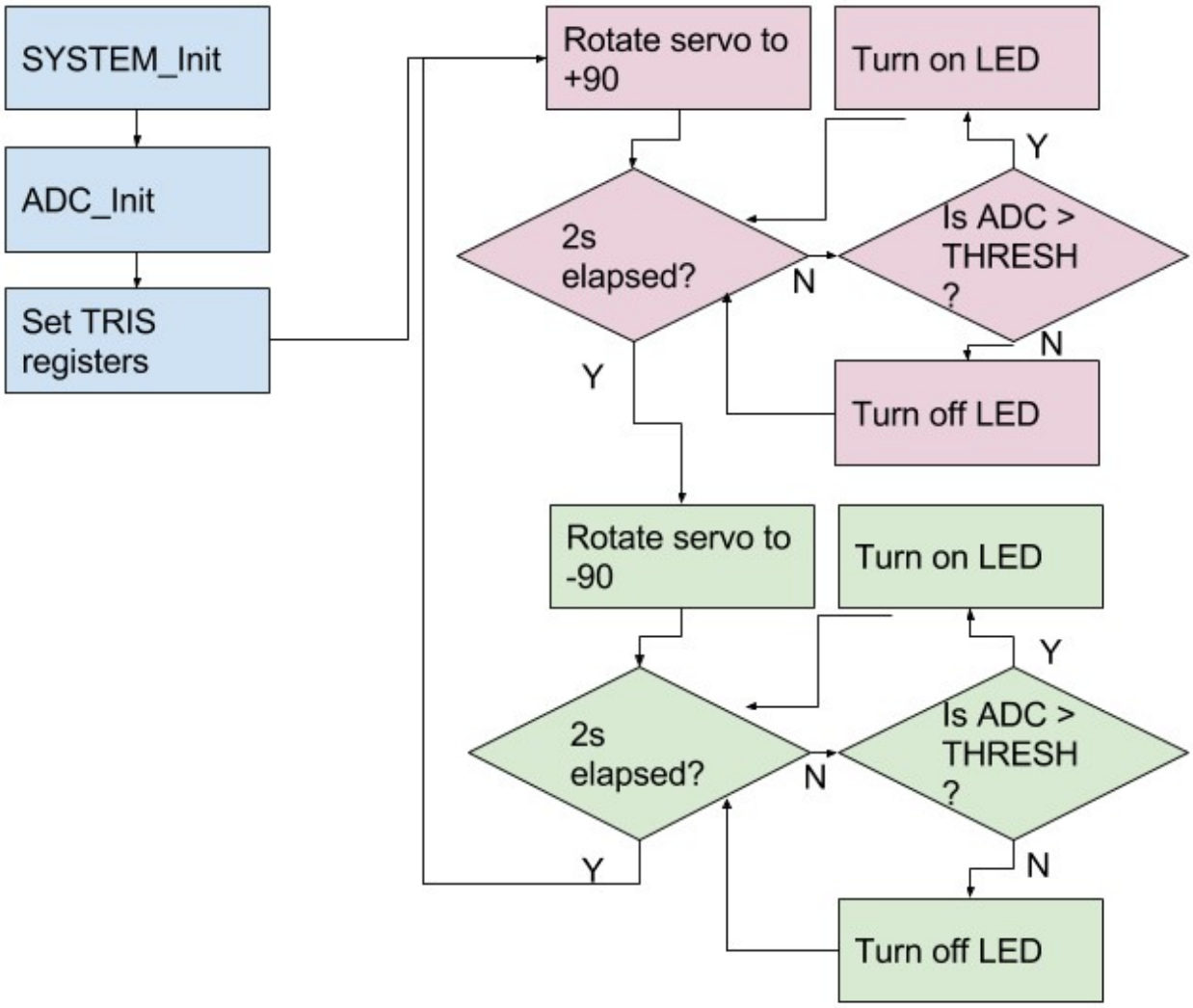
The PWM output pin and the LED output pin were both set to output mode using their TRIS register.

The only module that needed to be configured for this project was the ADC module. The module was enabled using the ADON bit of ADCON0. Also results were set to be right-justified. Then the ADC was configured to take input from ANA1 using ADPCH register. This port (Port A1) was configured as an input port using its TRIS register.

The rest of the ADC configuration options were left default (all 0).

The ADC threshold to turn on the LED was set to 0x390. Initially, we chose 0x1FF, because we thought that a value directly in the middle of the 10 bit ADC range would be easy to trigger on. However, when we were debugging, we found that that value of the ADC results hovered around 0x390 which is why we changed it to 0x390.

The application's main logic flow can be explained by the following flow chart.



The SYSTEM_Init function was provided by the compiler. ADC_Init configures the ADC control registers. The ADC results were obtained by setting the ADGO bit of ADCON0 then waiting for it to clear. The 10bit results were obtained by leftshifting the ADRESH register by 8 and adding ADRESL.

The following is a function showing how we moved the servo motor.

```
void servoRotate180() //-90 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(0.5);
    PORTB = 0;
    __delay_ms(19.5);
  }
}
```

Section 8: Trouble Shooting /1 points

Issue 1: **We were unable to get Timer2 to work with the PWM output to output a PWM signal to the servo motor.** We followed all of the instructions in the datasheet but didn't see an output signal.

We solved this issue by using delay functions to generate a PWM signal instead.

Issue 2: **We were unable to get Timer0 to count down to 2 seconds in between servo motor rotations.** This was found with the same process as the Timer2.

We solved this issue by using a for loop that counted up to a high number ADC samples to cause a 2s delay.

Issue 3: **We were unable to use the EUART port and FTDI cable to debug.**

We found that the serial port would work as expected maybe one time in fifteen. The rest of the times, the terminal just displayed gibberish symbols.

We solved this problem by relying on hardware breakpoints and watch variables to debug through the IDE. We didn't end up needing the serial port to debug this program.

Section 9: Results /0.5 points

Charts/Measurements:

No charts or measurements were obtained from this experiment.

Terminal Screenshots:

Because we could not get the serial port to work, there were no terminal screenshots.

How did you test your output signal:

The PWM output signal was tested with an oscilloscope at first when we tried to use Timer2 to use the PIC's PWM module. No signal was obtained.

When we used the delay functions to control the PWM module, we connected the motor directly to the pwm signal and incrementally adjusted the duty cycle until the motor rotated a full 180 degrees.

The photoresistor was tested by covering/uncovering it and watching the LED's reaction.

The circuit behaved as expected on a whole.

A1: main.c code file:

```
#include "mcc_generated_files/mcc.h" //default library


// ++++++++++++ Helpful Notes +++++++++++++++


/*
 include or set any library or definition you think you will need
 */

void servoRotate0() //0 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(1.4);
    PORTB = 0;
    __delay_ms(18.6);
  }
}

void servoRotate90() //90 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(4);
    PORTB = 0;
    __delay_ms(16);
  }
}

void servoRotate180() //-90 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(0.5);
    PORTB = 0;
    __delay_ms(19.5);
  }
```

```
}
void ADC_Init(void)  {
 //  Configure ADC module
 //---- Set the Registers below::
 // 1. Set ADC CONTROL REGISTER 1 to 0
 // 2. Set ADC CONTROL REGISTER 2 to 0
 // 3. Set ADC THRESHOLD REGISTER to 0
 // 4. Disable ADC auto conversion trigger control register
 // 5. Disable ADACT
 // 6. Clear ADAOV ACC or ADERR not Overflowed  related register
 // 7. Disable ADC Capacitors
 // 8. Set ADC Precharge time control to 0
 // 9. Set ADC Clock
 // 10 Set ADC positive and negative references
 // 11. ADC channel - Analog Input
 // 12. Set ADC result alignment, Enable ADC module, Clock Selection
Bit, Disable ADC Continuous Operation, Keep ADC inactive


    TRISA = 0b11111110;   //set PORTA to input except for pin0
    TRISAbits.TRISA1 = 1;   //set pin A1 to input
    ANSELAbits.ANSA1 = 1;   //set as analog input
    ADCON1 = 0;
    ADCON2 = 0;
    ADCON3 = 0;
    ADACT = 0;
    ADSTAT = 0;
    ADCAP = 0;
    ADPRE = 0;
    ADCON0 = 0b10000100;
    ADREF = 0;
    ADPCH = 0b00000001;
    // transmit status and control register (UART CABLE)
    TX1STA = 0b00100000;
    RC1STA = 0b10000000;

}

unsigned int ADC_conversion_results() {
    ADPCH = 1;

    ADCON0 |= 1;           //Initializes A/D conversion

    while(ADCON0 & 1);             //Waiting for conversion to
complete
    unsigned result = (unsigned)((ADRESH << 8) + ADRESL);
```

```
    return result;
}

/*
Develop your Application logic below
*/
#define ADC_THRESHOLD 0x0390
#define COUNT_THRESHOLD 5000
void main(void)
{
    // Initialize PIC device
    SYSTEM_Initialize(); //UART is initialized on portC
    ADC_Init(); //initializes ADC on port A1
    TRISA &= !0x01; //make sure portA0 is ouput for the LED
    TRISB = 00;
    TRISA = OUTPUT;

    unsigned results;
    unsigned count = 0;
    bool servo_direction_clockwise = true;
    while (1) // keep your application in a loop
    {
        // ****** write your code
        results = ADC_conversion_results();
        // Debug your application code using the following statement
        //printf("ADC says: %d compared to %d\n\r", results,
ADC_THRESHOLD);

        if(results > ADC_THRESHOLD)
            PORTA |= 0x01; //turn on LEd
        else
            PORTA &= !0x01; //turn off LED

        count++;

        if( count > COUNT_THRESHOLD && servo_direction_clockwise)
        {
            count = 0;
            servoRotate180();
            servo_direction_clockwise = false;
        }
        else if( count > COUNT_THRESHOLD &&
!servo_direction_clockwise)
        {
            count = 0;
            servoRotate90();
```

```
            servo_direction_clockwise = true;
        }

    }


}

/**
 End of File
*/
```