Section 1: General Lab Info /0.5 points

Microprocessors II and Embedded Systems

EECE.4800

Sensor Design and Analog Digital Conversion

Yan Luo

Group 1

Grayson Colwell

Handed in: October 2, 2017

Lab Due: October 2, 2017

Section 2: Contributions /1 points

1. Group Member 1 – Grayson Colwell
   - Developed the Servo Functions, helped work on the ADC Initialization module, and assisted with the debugging of the main module.

2. Group Member 2 – Andrew Macgregor
   - Developed the Main module, Wired the circuit, and worked on the debugging of the main module.

3. Group Member 3 – Jose Velis

   - Developed the ADC conversion module, helped work on the ADC Initialization module, and created the hardware schematic.

Section 3: Purpose /0.5 points

The purpose of this lab was to gain an understanding of how the PIC16F18857 interacts with sensors and other circuit elements. After completing this project, students should understand how to use the Analog to Digital Conversion (ADC) module of the PIC microcontroller, how to use sensors in circuits, and how to control a servo motor using either delay functions or the PWM module of the PIC microcontroller.

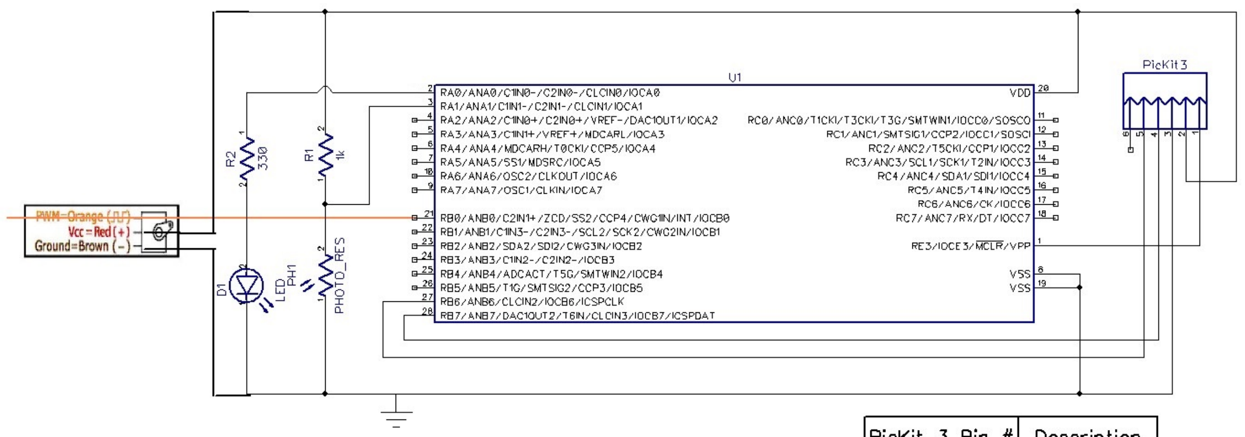
Section 4: Introduction /0.5 points

To be successful with the lab, there were a few key concepts that needed to be understood. This lab was centered around using the PIC16F18857 to interact with a photoresistor and a servo motor. To have the PIC to work correctly, it needed to be programmed using MPLAB X and then can interface with both the servo motor and photoresistor so that when the servo covers the photoresistor, an LED turns on and when the photoresistor is not covered, the light turns off. The duty cycle of the servo motor is an important to this lab because the time period of the servo directly correlates to how the motor will rotate. The ADC component of the PIC was necessary as this reads in a voltage value from the photoresistor, and based off of this value, the LED will turn on or off and the servo will rotate to a certain position.

Section 5: Materials, Devices and Instruments /0.5 points

- **Servo Motor SG90**: Motor which rotates between 0 and 180 degrees depending on the duty cycle.
- **Photoresistor (LDR)**: Resistor with a variable resistance depending on the amount of light that is read by it.
- **PIC16F18857**: Embedded microcontroller which would be the brain of the circuit controlling the servo, ADC module, and LED.
- **Light Emitting Diode (LED)**: Used as a visual indicator of how the photoresistor operates.
- **Analog Discovery, Serial #210244630597**: Used the Discovery to act as a voltage source and as an oscilloscope to measure the voltage of the LDR.
- **1kΩ and 330Ω Resistors:** The 1kΩ resistor was used to form a voltage divider with the LDR and the 330Ω was in series with the LED to ensure that it didn't burn out from being given too much voltage.

Section 6: Schematics /0.5 points


U1
RA0/ANA0/C1IN0-/C2IN0-/CLCIN0/IOCA0
RA1/ANA1/C1IN1-/C2IN1-/CLCIN1/IOCA1
RA2/ANA2/C1IN0+/C2IN0+/VREF-/DAC1OUT1/IOCA2
RA3/ANA3/C1IN1+/VREF+/MDCARL/IOCA3
RA4/ANA4/MDCARH/T0CKI/CCP5/IOCA4
RA5/ANA5/SS1/MDSRC/IOCA5
RA6/ANA6/OSC2/CLKOUT/IOCA6
RA7/ANA7/OSC1/CLKIN/IOCA7
RB0/ANB0/C2IN1+/ZCD/SS2/CCP4/CWG1IN/INT/IOCB0
RB1/ANB1/C1IN3-/C2IN3-/SCL2/SCK2/CWG2IN/IOCB1
RB2/ANB2/SDA2/SDI2/CWG3IN/IOCB2
RB3/ANB3/C1IN2-/C2IN2-/IOCB3
RB4/ANB4/ADCACT/T5G/SMTWIN2/IOCB4
RB5/ANB5/T1G/SMTSIG2/CCP3/IOCB5
RB6/ANB6/CLCIN2/IOCB6/ICSPCLK
RB7/ANB7/DAC1OUT2/T6IN/CLCIN3/IOCB7/ICSPDAT
VDD
RC0/ANC0/T1CKI/T3CKI/T3G/SMTWIN1/IOCC0/SOSCO
RC1/ANC1/SMTSIG1/CCP2/IOCC1/SOSCI
RC2/ANC2/T5CKI/CCP1/IOCC2
RC3/ANC3/SCL1/SCK1/T2IN/IOCC3
RC4/ANC4/SDA1/SDI1/IOCC4
RC5/ANC5/T4IN/IOCC5
RC6/ANC6/CK/IOCC6
RC7/ANC7/RX/DT/IOCC7
RE3/IOCE3/MCLR/VPP
VSS
VSS
PicKit3
R2 330
R1 1k
D1 LED
PH1 PHOTO_RES
PWM=Orange (⊓⊔)
Vcc=Red (+)
Ground=Brown (-)

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

Figure 1: Pinout and layout of the circuit used for this lab drawn in DipTrace.

In Figure 1, the schematic shown is how the circuit was designed by our group. One thing that was excluded in our design was a 4.7kΩ-10kΩ resistor placed between the VPP pin and the VDD pin as we just missed the figure showing to use it. This had no adverse effect on our project but this could have been an issue so going forward, it would be best not to leave this component out.

Section 7: Lab Methods and Procedure /2 points

Hardware design:

When looking at our implemented analog circuit, no real advanced concepts were used in our design. The photoresistor was placed in series with a 1kΩ resistor to create a voltage divider where the ADC of the PIC would read in the voltage value of the resistor. This was connected to the PIC at pin 3, ANA1. The ADC would then read the voltage of the LDR and compare it to the threshold value. Depending on this value, the RA0 pin would either output voltage or not output voltage. This signal would power the LED so that if the LDR sensed darkness, the LED would turn on and then the LED would turn off if the LDR detected light. The circuit was also set so that the input voltage was 3.3V to avoid frying the PIC. When reading the voltage values read in by the ADC, it was found that the voltage of the LDR ranged from 2.8V to 3.1V.

Software design:

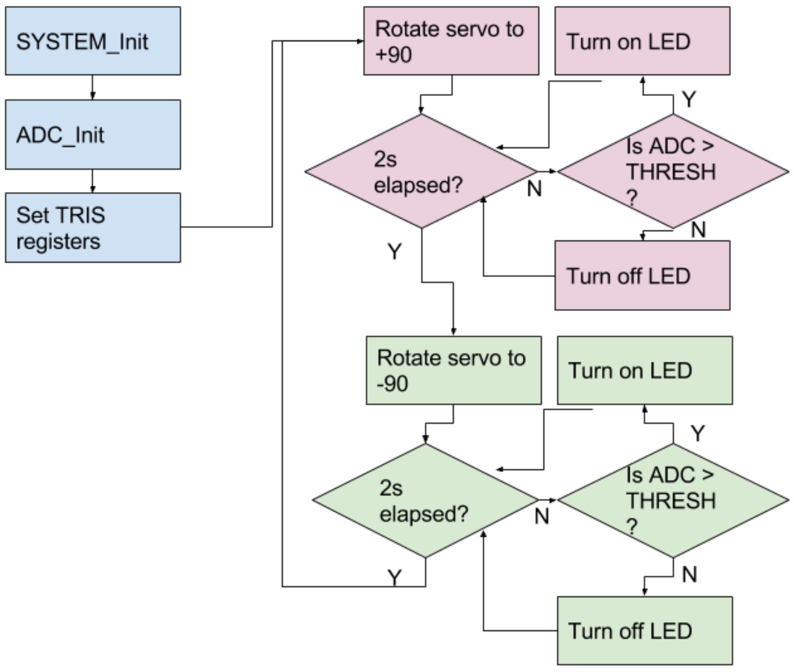
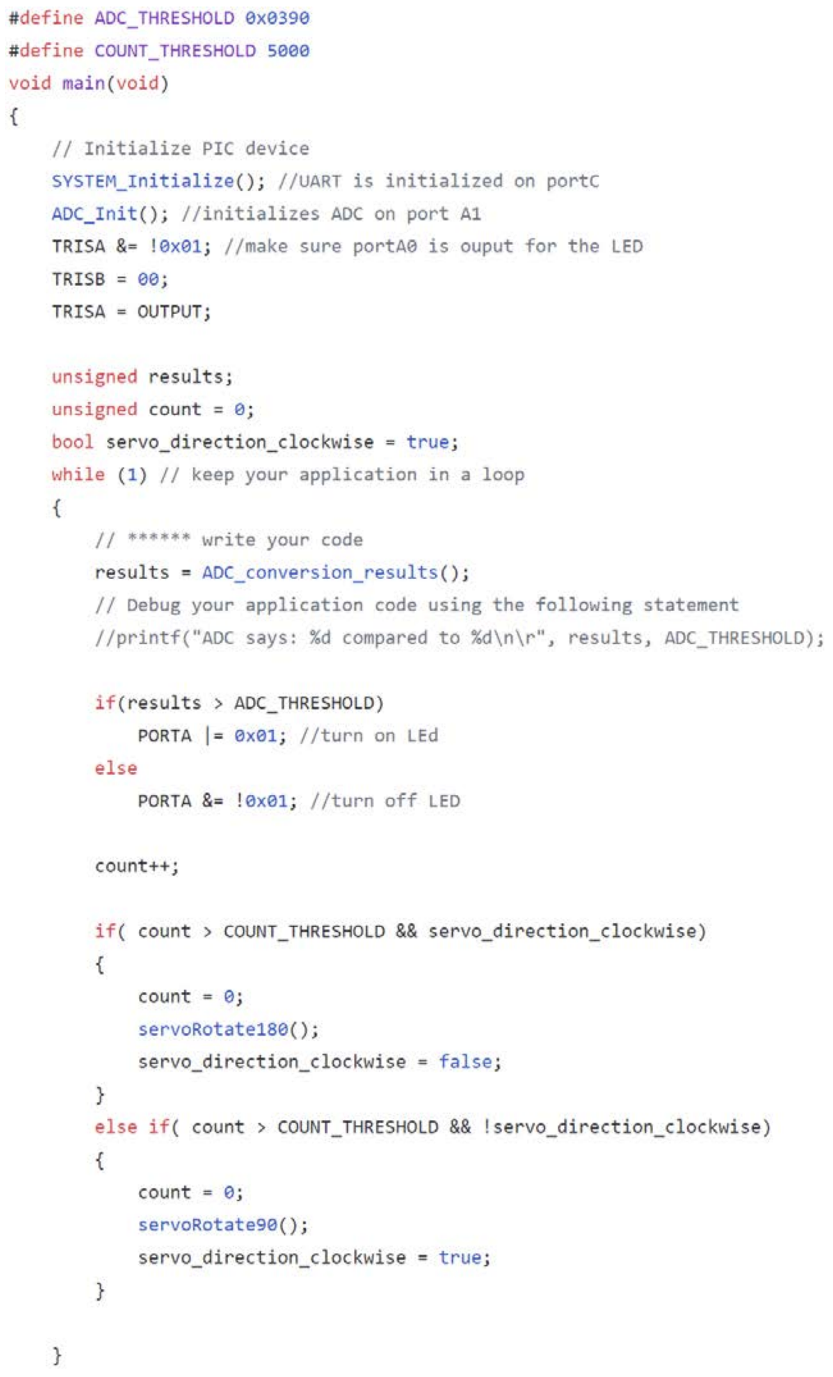


Figure 2: Software Design Flow-Chart.

In Figure 2, our software design flow chart is pictured. The first thing the code does is initialize its system and ADC. Once the TRIS registers are set, the code then moves to the main module where it will call the first servo function to rotate it. The code then checks at what position the servo is in and determines whether the LED should turn on based off the ADC value of the photoresistor. In our code, the threshold value that worked best for us was 0x0390H. If the ADC value was higher than the threshold, the LED would turn on and off if the ADC value was below the threshold. The code would then wait approximately 2 seconds and after this time would rotate the servo in the opposite direction and continue the same process infinitely. This main module is shown in Figure 3. One thing that we left out in this lab was the inclusion of the PWM module as we decided to use delay functions to implement the duty cycle of the servo motor.



```
#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;
        }

    }
```

Figure 3: Screenshot of our main module in the program.

Section 8: Trouble Shooting /1 points

Issue 1: Interfacing with the UART Cable

When we were beginning to get towards the end of the assignment, we had an issue where we were unable to read the value that was being read in by the ADC. To combat this, we thought that we would have been able to use the UART cable to display the value read in by the ADC in the putty terminal. Unfortunately, this did not go as planned because the UART cable would endlessly output garbage readings that made no sense and it was not printing out anything related to what we needed it to output. To combat this, we ended up not using the UART cable and instead set up a local variable to watch the value read in by the ADC and we single-stepped our code when the value was input and from this, we were able to get a good threshold voltage and our circuit worked.

Issue 2: Using the PWM module

When I first began working on the servo module, I was going to use the PWM module to interface with the servo motor. As I was researching how to initialize the PWM module and timer of the PIC, I found an example where people would use delay functions to control where the servo would rotate to. After considering it more, I decided that it was much easier to just use the delay functions to control the servo.

Section 9: Results /0.5 points

In this lab, there really were not many results that were relevant to being posted. The ADC Threshold was set to 0x0390H because we found this to be the optimal value to prevent false positives when turning on and off the LED. Had this value been set higher, the LED would never have turned on and if was set too low, the LED would always be on. We defined the Count Threshold as being 5000 as this value would give us a period very close to 2 seconds. We had a local variable count which would reset back to 0 after it was greater than the threshold value. This variable would determine how the servo would rotate so it wouldn't endlessly try to rotate in one direction.

Section 10: Appendix

A1.

Servo Motor Functions:

```
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);
              }
```

A2.

ADC Initialization and ADC Conversion Results Functions:

```
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;
    }
```

A3.

Main Module:

```
#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;
            }

        }


    }
```

A4.

Web Reference used to further understand Duty Cycle of the servo motor:

https://learn.sparkfun.com/tutorials/pulse-width-modulation