UMASS LOWELL

Section 1 : General Lab Info

EECE.4800 Microprocessors II and Embedded System Design

11024

Lab 1: Sensor Design and Analog-Digital Conversion

Professor Yan Luo

Group number: 1

Jose Velis

October 2, 2017

October 2, 2017

Section 2 : Contributions

1. Group Member 1 – Jose Velis

   Responsible for ADC configuration and ADC results, LDR implementation, and some circuit layout on the breadboard.

2. Group Member 2 – Grayson Colwell

   Responsible for servo/PWM implementation as well as for the ADC, main logic implementation, and serial communication.

3. Group Member 3 – Andy MacGregor

   Responsible for main code logic as well as timer configuration and serial communications. Also did the final neat breadboard layout and ADC results.

Section 3: Purpose

The purpose of this lab was to understand the operation of an analog-to-digital converter by interfacing an analog sensor to an embedded microcontroller, also to understand the design of sensor circuitry, understand the operation of Pulse Width Modulation (PWM) signals, and to learn how to control a mechanical actuator (a servo motor for this lab).

Section 4 : Introduction

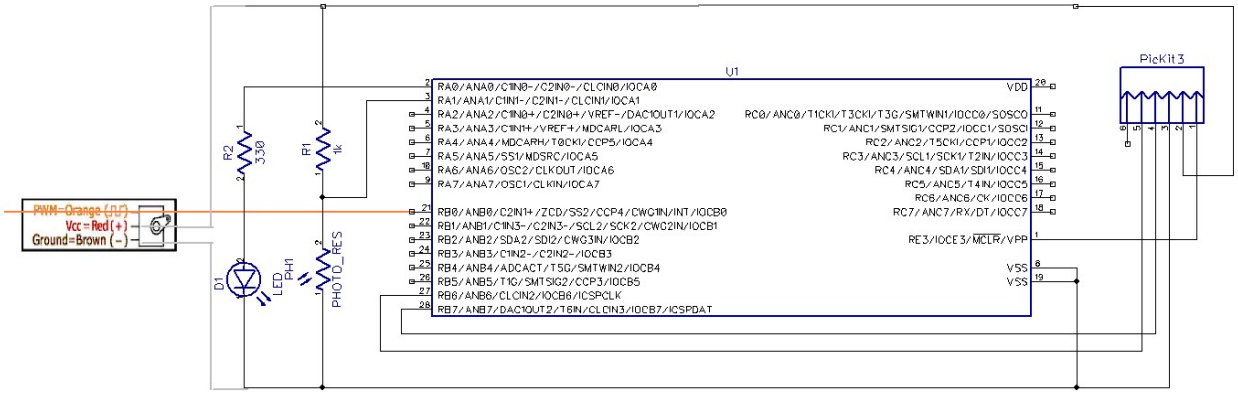
In this lab, a light intensity sensor and a microcontroller were used to control whether an LED would turn on or not. The light intensity sensor was a photoresistor (LDR) that changes its resistance value depending on the amount of light that is placed over it. The change in resistance causes a change in voltage and so this principle was used in order to interface the LDR to the Microchip Technology Inc. microcontroller. The microcontroller was used to measure the voltage change caused by the LDR. The voltage value was sent to the microcontrollers ADC module and the ADC measurement was then compared to a predetermined value that represented the turn on voltage for the LED. The final piece of the lab was controlling a servo motor. The purpose of this servo motor was to function as a cover/uncovering device for the LDR.

Section 5 : Materials, Devices and Instruments

- Microchip PicKit, Version 3, used to program the PIC device, operates at 5V
- Microchip PIC16F18857 microcontroller, 28 dip device, operates at 3.3 volts
- MPLAB X IDE with XC8 compiler
- Servo motor, SG90, operates between 4.8v -5V
- FTDI cable, operates at 5V
- Photoresistor, measures light intensity, operates between 0V-5V
- Resistors, 1kΩ and 330Ω
- LED
- Breadboard & wiring
- Lab oscilloscope and power supply

Section 6: Schematics

The schematic for the circuit and programmer can be seen in figure 1.



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

Figure (1)

Section 7 : Lab Methods and Procedure

Hardware design:

For this lab, we first started by figuring out the pins, programming pins and power requirements of the PIC16. Then we moved on to how the servo motor works and what the best method for controlling it would be. We became acquainted with the servo pinout and power demands. The device has a 4.8V-5V power, ground, and signal pin. The signal pin was connected to pin 21 (RB0) of the PIC16 microcontroller, and the other two pins to their respected connections (breadboard power and ground). The next step was connecting the photoresistor, resistors and LED. A simple voltage divider circuit between the 1kΩ resistor and photoresistor was used to obtain a change in voltage value for when the LDR reacted to different light intensity. The voltage divider network was connected to the 3.3V supply and the changing voltage signal was connected to pin 3 (RA1) of the PIC16. The LED

anode was connected to a 330Ω resistor and then the resistor to pin 2 of the PIC16. LED cathode was grounded. This was all connected on a breadboard.

Software design:

The ADC channel was on Port A pin 1 (RA1). The ADC was first configured according to the recommended settings and then a function for obtaining results was made. With the ADC values coming from the voltage divider network, it was now possible to find a threshold voltage that would correlate to the LED turning on or off. This value was used in the main loop. The program flow chart can be seen in figure 2.

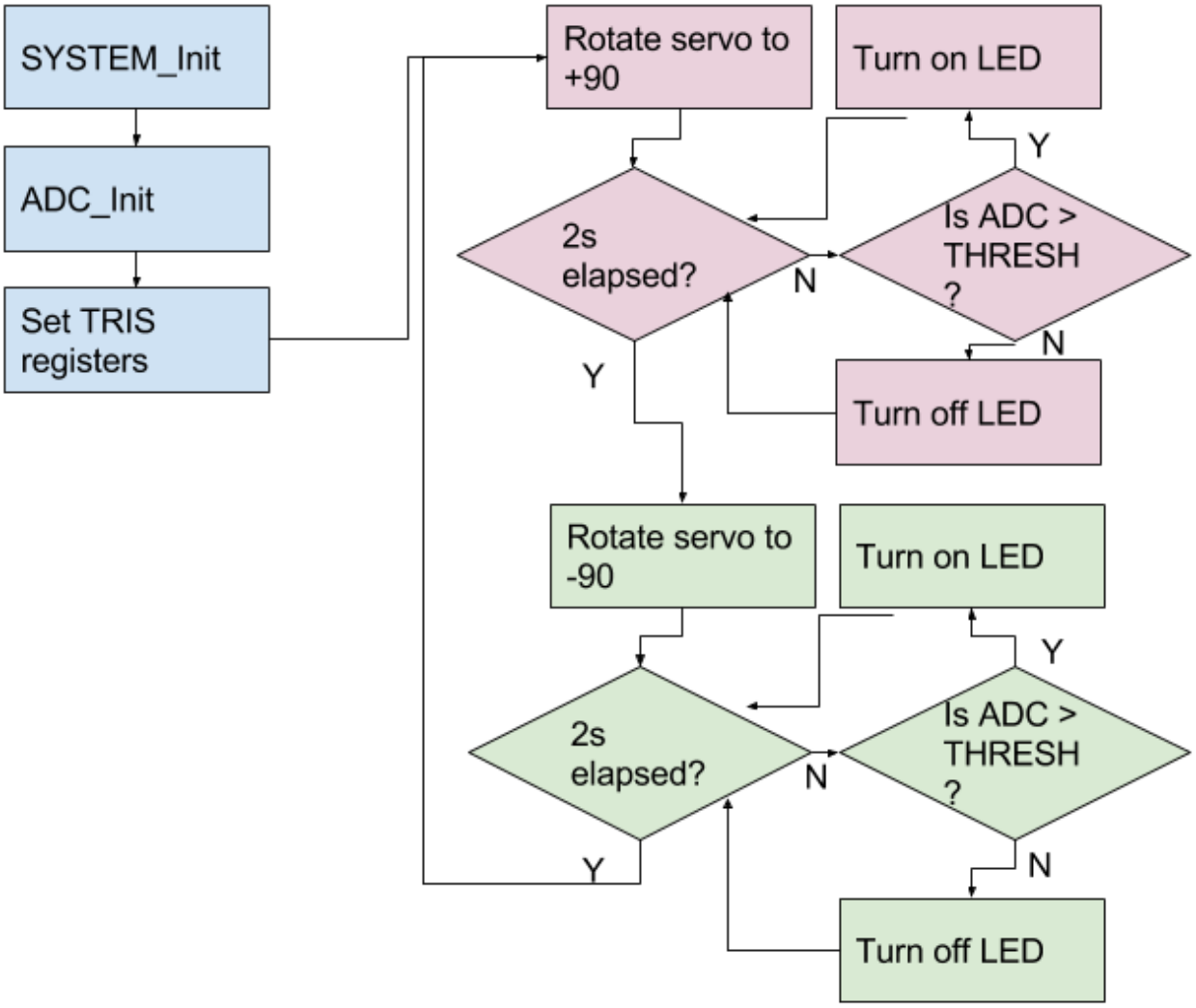


Figure (2)

The basic working of the program are as follows:

- Initialize the chip along with the modules that will be used. In this case it was just the ADC module.
- The next step was to start the servo rotation using delay functions. The servo rotates from -90 to 0 to +90. This will keep running the entire time. The rotation functions were made and called in the main logic loop
- Next was obtaining the data from the ADC, and making it available to the main logic loop.
- In the main logic loop, the ADC conversion results function was called and the value was compared to an LED threshold value. If the ADC value was greater than the threshold, the LED is turned on via the PORTA registers. If not, the LED is kept off.
- The location of the servo was also tracked and used in the main logic to control the direction of rotation.
- The program just keeps doing this forever.

Code snips of the ADC initialization and main logic loop are shown below.

ADC initialization:

```
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;

}
```

Main logic loop:

```
#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;
        }

    }


}
```

The TIMER2 module was not used, since we found out that delays could be used to control the servo.

Section 8 : Trouble Shooting

Issue 1: We had issues getting the timer0 and timer2 to work with the servo and I general. The problem was solved by using delay functions to control and track the rotation of the servo.

Issue 2: We also experienced issues with the serial communications. The output console would rarely work and when it did it just kept showing gibberish. We didn't find out what was causing the problem but debugging became possible by using hardware breakpoints and watch variables in the IDE.

Issue 3: During the first hours with the PicKit3 we had difficulties with power errors that the IDE kept giving us. The issue was that the PicKit3 has power output settings where the user can set it to power the device that it is programming. We found this setting and set it to our needs then the circuit worked with no power errors.

Section 9: Results

Charts/Measurement Tables: No measurements or charts were gathered from the experiment.

Terminal Screenshots: These weren't possible since we didn't get the serial port to work reliably.

We tested the servo output with an oscilloscope probe on the output pin. The ADC and LED were tested by covering the photoresistor and looking at how the LED reacted. Both reacted as expected.

Section 10 : Appendix

A1. Below is the main.c 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
*/
```