Section 1 : General Lab Info

Course name: Microprocessor Systems II and Embedded Systems

Course Number: EECE.5520

Lab title: Sensor Design and Analog Digital Conversion

Instructors names: Yan Luo
Ioannis Smanis

Group number: 7

Student name: Poornima Manjunath

Hand in Date: 09/13/2017

Lab Due Date: 10/02/2017

Section 2 : Contributions

1. Group Member 1 – Poornima Manjunath
   I have contributed to the project by writing codes for the ADC conversion to design the light sensor and for the PWM generation to rotate the servo motor.

2. Group Member 2 – Rinkal Shah
   She has handled the hardware design part of the project.

3. Group Member 3 – Akhila Nair
   She has handled the interfacing part of the project.

Section 3 : Purpose

The controlling of outdoor and street lights, home appliances, etc., are typically operated manually. The manual operation is not only risky but also causes wastage of power due to the negligence of operating personnel and unusual conditions in monitoring these electrical appliances. Hence, by using a light sensor circuit, we can easily operate the loads as it facilitates automatic switching of the loads. In this project, we are designing a light sensor circuit using PIC16F18857. The system will include a data acquisition using a light intensity sensor and two output devices: and LED and a servo motor. This project helps us to understand

1. The design of a sensor circuit
2. ADC operation
3. Operation of PWM signals
4. Controlling a mechanical actuator like servo motor

Section 4 : Introduction

Sensor is a device that is used to detect the changes in quantities or events and appropriately produce the outputs. There are different types of sensors such as light sensors, temperature sensor, humidity sensor, pressure sensor, fire sensor, ultrasonic sensors, IR sensor, touch sensor, and so on. Light sensor is used to sense amount of light. In order to detect the intensity of light or darkness, we use a sensor called an LDR (Light Dependent Resistor). The LDR is a special type of resistor which allows higher voltages to pass through it (low resistance) whenever there is a high intensity of light, and passes a low voltage (high resistance) whenever it is dark. We can take advantage of this LDR property and use it in our light sensor circuit.
The light sensor circuit is a simple electrical circuit, which can be used to control the (switch on and off) electrical load appliances like lights, fans, coolers, air conditioners, street lights, etc., automatically. By using this light sensor circuit, we can eliminate manual switching as the loads can be controlled automatically based on the daylight intensity.

In this project, we are also generating PWM signals using delay to control the Servo motor which turns its blade to specified degree and covers/exposes the LDR from/to the light. Thus in this project we will design an embedded system where its actuator device (Servo motor) affects the sensor device(LDR) and triggers a visible indicator(LED) to turn ON/OFF.

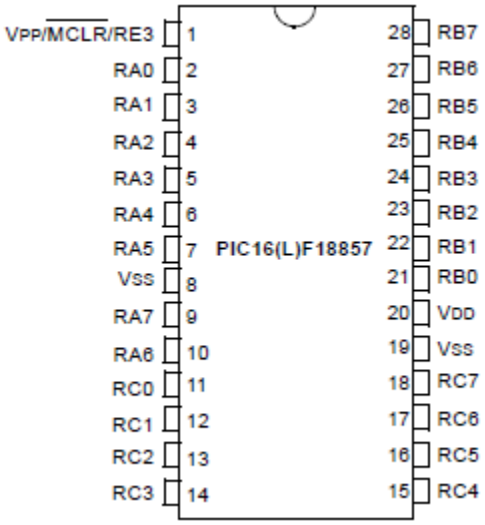
Section 5 : Materials, Devices and Instruments

Materials:

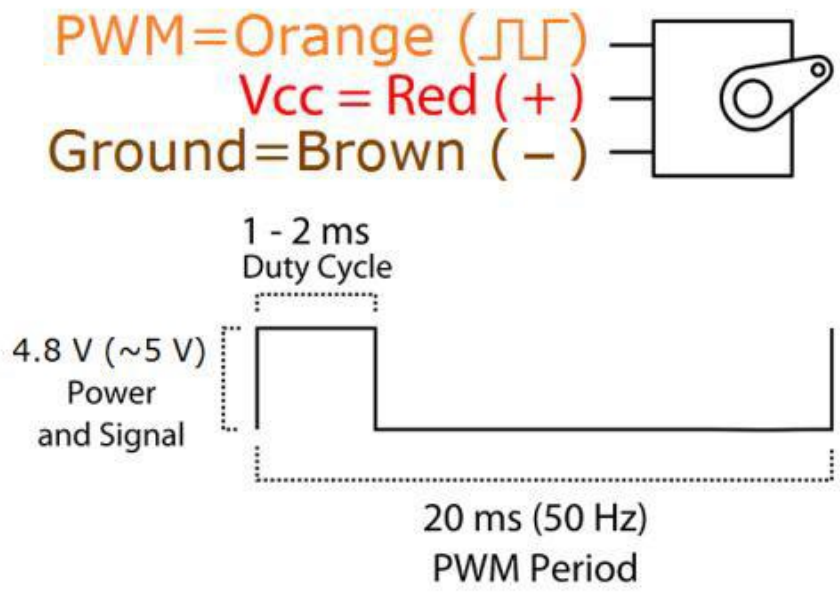
1. Breadboard
2. Resistors - 10K , 4.75K
3. Jumper cables
4. Solid core wires
5. LDR
6. LED

Devices:

1. PIC16F18577



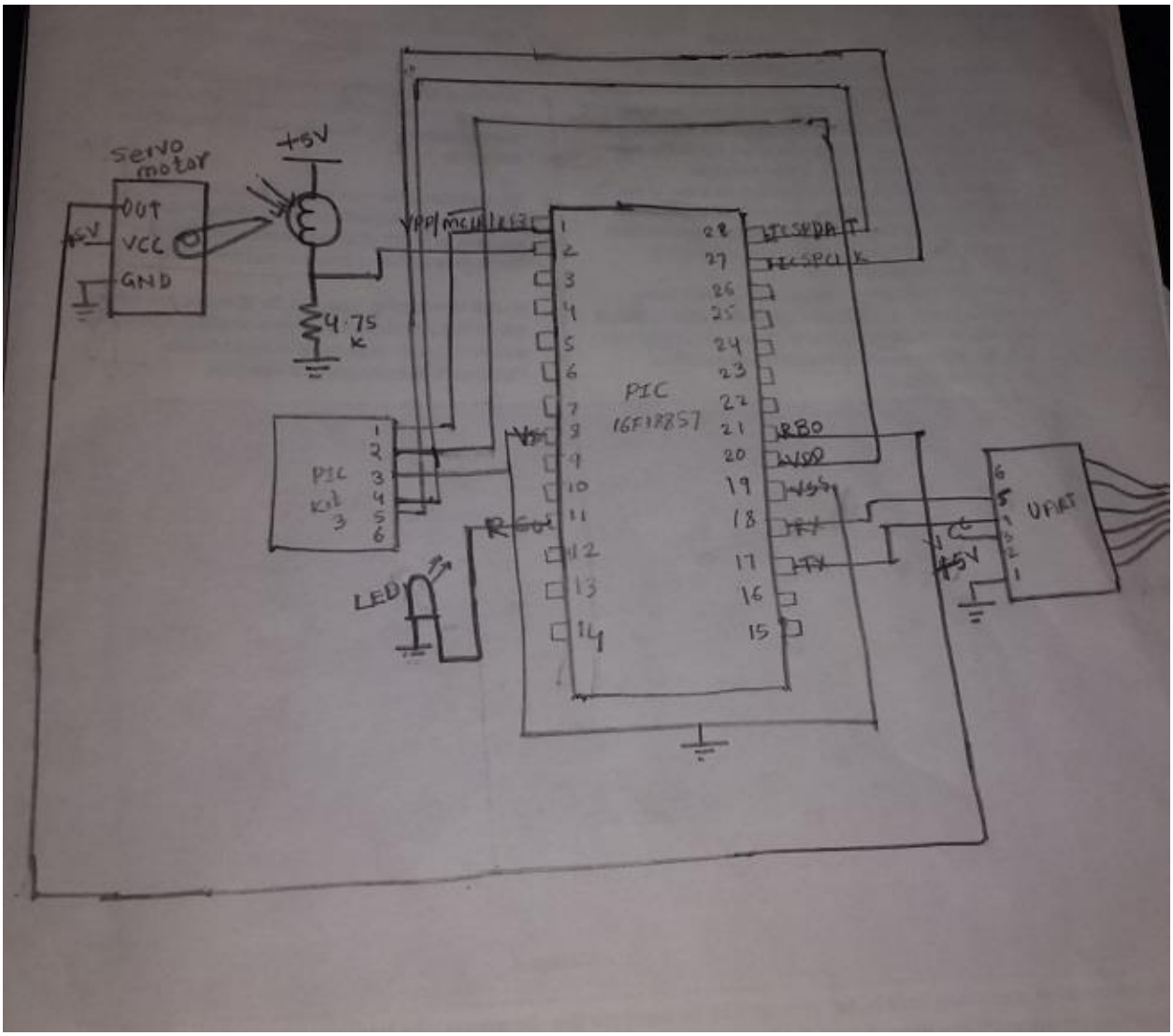
2. FTDI cable
3. Servo Motor



Instruments and Software:

1. PICkit-3
2. Power Supply – 5v
3. Laptop
4. MPLAB X IDE- V4.01
5. PuTTY

Section 6 : Schematics



Section 7 : Lab Methods and Procedure

Hardware design:

Working Principle of the circuit:

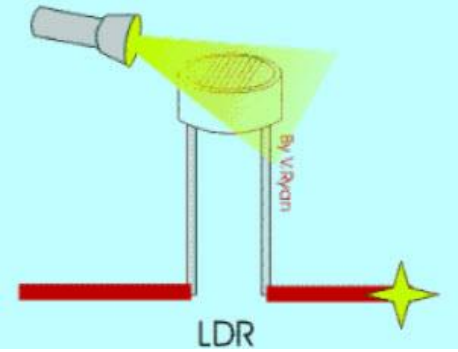
LDR is used to detect intensity of light. PIC161F18857 microcontroller is used interfaced with the light sensor(LDR) to sense amount of light available. Control signal is generated with the help of

PIC microcontroller after analyzing amount of light. Control signal generated by pic microcontroller is used to turn ON and OFF an LED.

The Servo motor is made to rotate by the PWM signals generated by the PIC microcontroller. A blackboard is attached to the shaft of the Servo motor. So as it rotates it covers the LDR from the light.

Working Principle of Light Dependent Resistor:

The working principle of an LDR is photo conductivity, that is nothing but an optical phenomenon. When the light is absorbed by the material then the conductivity of the material reduces. When the light falls on the LDR, then the electrons in the valence band of the material are eager to the conduction band. But, the photons in the incident light must have energy superior than the band gap of the material to make the electrons jump from one band to another band (valance to conduction).



Hence, when light having ample energy, more electrons are excited to the conduction band which grades in a large number of charge carriers. When the effect of this process and the flow of current starts flowing more, the resistance of the device decreases.

The LDR gives out an analog voltage when connected to Vcc (5V), which varies in magnitude in direct proportion to the input light intensity on it. That is, the greater the intensity of light, the greater the corresponding voltage from the LDR will be. Since the LDR gives out an analog voltage, it is connected to the analog input pin(RA0) on the PIC microcontroller. The PIC, with its built-in 10 bit ADC (Analog to Digital Converter), then converts the analog voltage (from 0-5V) into a digital value in the range of (0-1023). When there is sufficient light in its environment or on its surface, the converted digital values read from the LDR through the PIC microcontroller will be in the range of 800-1023.

Furthermore, we then program the PIC to turn ON a LED when the light intensity is low (can be done by covering the surface of the LDR with any object). That is, when the digital values read are in a higher range than usual.

Working Principle of Servo Motor:

Servo motor can be easily be controlled using microcontrollers using Pulse Width Modulated (PWM) signals on the control wire. Here we are using a servo whose angular rotation is limited to 0 – 180°. Frequency/period are specific to controlling a specific servo. A typical servo motor expects to be updated every 20 ms with a pulse between 1 ms and 2 ms, or in other words, between a 5 and 10% duty cycle on a 50 Hz waveform. With a 1.5 ms pulse, the servo motor will be at the natural 90 degree position. With a 1 ms pulse, the servo will be at the 0 degree position, and with a 2.2 ms pulse, the servo will be at 180 degrees.

Duty Cycle

When the signal is high, we call this "on time". To describe the amount of "on time" , we use the concept of duty cycle. Duty cycle is measured in percentage. The percentage duty cycle specifically describes the percentage of time a digital signal is on over an interval or period of time.

duty cycle = TON / TON+TOFF

The servo shaft can be positioned to specific angular positions by via varying the Duty Cycle of a PWM signal.

PIC and LDR Sensor Connections:
First, connect the LDR to the analog input pin (RA0 in our case) on the PIC. Use a voltage divider configuration to do this. One leg of the LDR is connected to Vref (5V) on the PIC and the other to the analog pin 0 on the Arduino. A 4.75K resistor is also connected to the same leg and grounded. Like resistors, LDRs are not polarized. You can insert its pins without regard to polarity.

Connecting the LED to the PIC:
LED is connected to the digital output pin of the PIC microcontroller (RC0 in our case). An LED is a polarized component. The longer leg indicates the positive terminal. It must be connected to the digital output pin of the PIC and the other leg must be grounded. LED is forward biased.

Connecting PIC kit3 to the PIC16F18857:

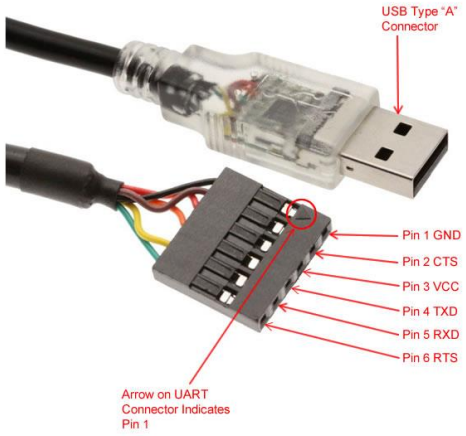
1. Pin 1 of PIC kit 3 i.e MCLR to pin 1 of PIC
2. Pin 2 of PIC kit 3 i.e VDD to pin 20 of PIC
3. Pin 3 of PIC kit 3 i.e VSS to pin 8 and pin 9 of PIC
4. Pin 4 of PIC kit 3 i.e ICSPDAT to pin 28 of PIC
5. Pin 5 of PIC kit 3 i.e ICSPCLK to pin 27 of PIC.

Pin 1 of PIC kit 3 has the arrow pointing to it. A pull up resistor of 10 K is added between MCLR and VDD pins.

We are powering our PIC IC using PIC Kit 3 and the voltage level is set to 3.375v.

Connecting the FTDI cable to the PIC:

FTDI cable has the below pinout



Pin 1 is Grounded and Pin4 TXD is connected to Pin 17 of PIC16F18857 and Pin5 RXD is connected to Pin18 of the PIC IC. The FTDI is connected to the serial port of the PC. Pin 3 of the FTDI cable supplies VCC =+5v which is given to the Vref of LDR and VCC of the Servo motor.

Connecting Servo motor to the PIC:

Speaking about interfacing of Servo motor to the PIC microcontroller, Servo Motors have three wires, two of them (RED and BLACK) are used for VCC(+5v) and Ground respectively and the third one is used to give control signals which is connected to digital output pin(RB0) in our case.

Operating voltages:

* We are powering our PIC IC using PIC Kit 3 and the voltage level is set to 3.375v.

* Pin 3 of the FTDI cable supplies VCC =+5v which is given to the Vref of LDR and VCC of the Servo motor.

Input and Output pins:

RA0 -Analog input pin

RC0- Digital Output pin

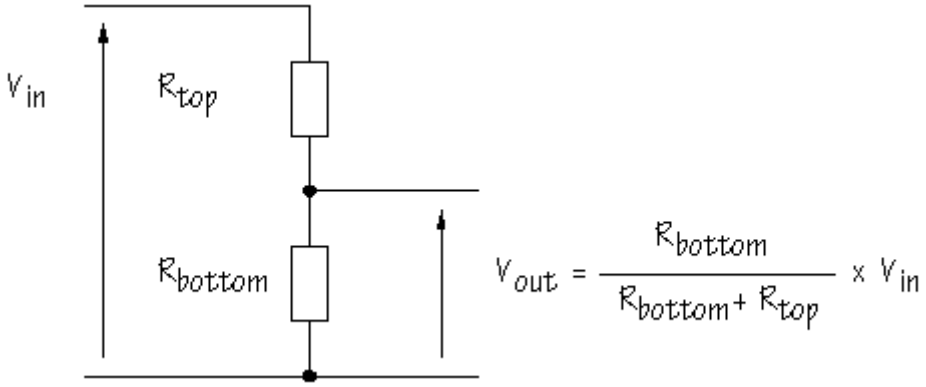
RB0- Digital Output pin

Circuit theory used:

Voltage divider Rule:

A **light sensor** uses an LDR as part of a voltage divider.

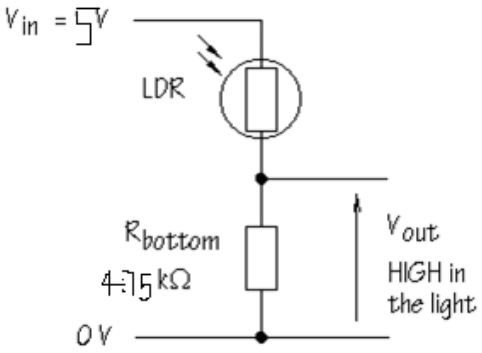
The essential circuit of a voltage divider, also called a **potential divider**, is:



As you can see, two resistors are connected in series with V_{in} , which is often the power supply voltage, connected above R_{top} . The output voltage V_{out} is the voltage across R_{bottom} and is given by:

$$V_{out} = \frac{R_{bottom}}{R_{bottom} + R_{top}} \times V_{in}$$

Here is the voltage divider built with the LDR in place of R_{top}



The resistance of the LDR is less (3K in our case) when measured in light compared to the resistance of LDR in darkness (19.2K). So as per voltage divider rule V_{out} becomes HIGH when the LDR is in the light, and LOW when the LDR is in the shade.

Software design:

ADC configuration:

Most of our computer (or Microcontrollers) are digital in nature. They can only differentiate between HIGH or LOW level on input pins. For example, if input is more than 2.5v it will be read as 1 and if it is below 2.5 then it will be read as 0 (in case of 5v systems). So, we cannot measure voltage directly from MCUs. To solve this problem most modern MCUs have an ADC unit. ADC stands for analog to digital converter. It will convert a voltage to a number so that it can be processed by a digital system like MCU.

PIC16F18857 has 10- bit ADC. If the reference voltage (explained latter) of ADC is 0 to 5v then a 10bit ADC will break it in 1024 divisions so it can measure it accurately up to 5/1024 v= 4.8mV approx.

When configuring and using the ADC the following functions must be considered:

• **Port configuration**

The ADC can be used to convert both analog and digital signals. When converting analog signals, the I/O pin should be configured for analog by setting the associated TRIS and ANSEL bits.

We are configuring RA0 pin of PORTA by setting TRISA = 0x01 and ANSELA= 0x01

• Channel selection

We are selecting RA0 by setting PORTA =0x01

• ADC voltage reference selection

The ADPREF bits of the ADREF register provides control of the positive voltage reference. The positive voltage reference can be:

•VREF+ pin •V DD • FVR 1.024V • FVR 2.048V • FVR 4.096V

The ADNREF bit of the ADREF register provides control of the negative voltage reference. The negative voltage reference can be: •VREF- pin •V SS

We are setting positive voltage reference to VDD and negative voltage reference to VSS by setting ADREF= 0x00.

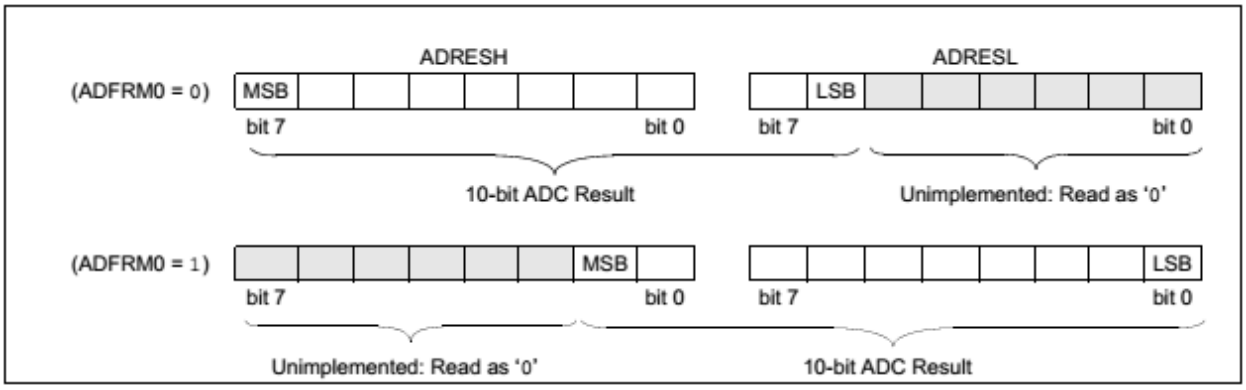
• ADC conversion clock source

The source of the conversion clock is software selectable via the ADCLK register and the ADCS bit of the ADCON0 register. There are two possible clock sources: •FOSC/(2*(n+1)) (where n is from 0 to 63), • FRC (dedicated RC oscillator)

We are setting ADCLK = 0x02 which means clock source will be FOSC/6 and ADCS bit of ADCON0 to 0 which means Clock supplied by FOSC, divided according to ADCLK register.

• Result formatting

The 10-bit ADC conversion result can be supplied in two formats, left justified or right justified. The ADFRM0 bit of the ADCON0 register controls the output format.



We are setting ADFRM0 and hence our result is right justified.

Starting a conversion

To enable the ADC module, the ADON bit of the ADCON0 register must be set to a '1'. A conversion may be started by setting the ADGO bit of ADCON0 to '1'

Completion of conversion

When any individual conversion is complete, the value already in ADRES is written into ADPREV (if ADPSIS=1) and the new conversion results appear in ADRES.

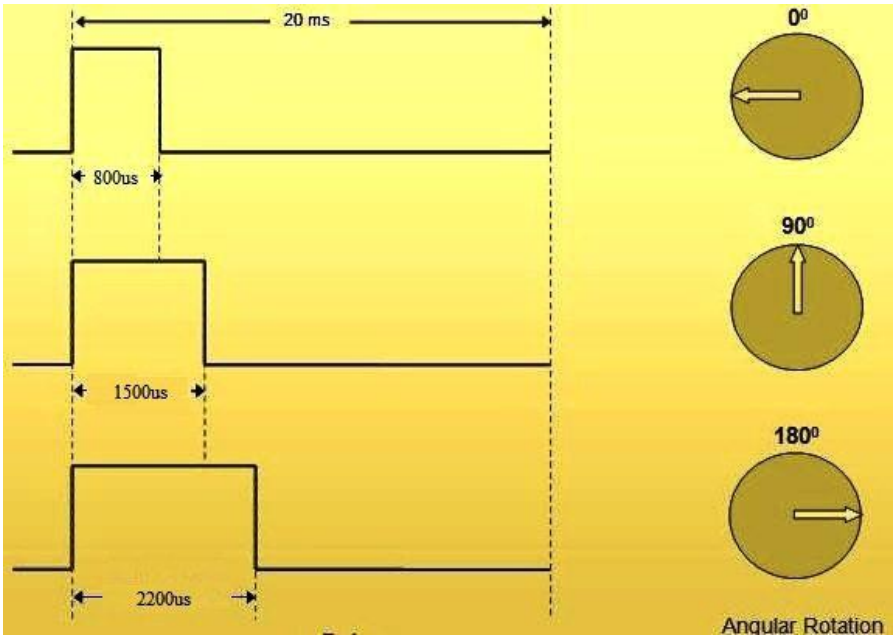
When the conversion completes, the ADC module will clear the ADGO bit (Unless the ADCONT bit of ADCON0 is set)

PIC PWM implementation:

Handling a GPIO pin and delay functions:

The pulse has been created with the help of delays in the program. The delay duration for an angle is equivalent to the length of the pulse required for the motor to rotate up to that corresponding

angle. That is, for 0° angle, the pulse width is approx. 800us, so a delay of 800us is introduced with the PORT pin RB0 set to high. Similarly, a pulse of 1500us is required for a rotation up to 90° and 2200us for a 180° angle.



Delay_us() is used to make delay in microsecond during the program execution.

Explanation of the logic of the main program

Algorithm:

1) Configure the ADC module by setting the required registers as explained in the section ADC configuration. This will be done by the function ADC_init()

2) Set Bit0 of PORTB as output for running Servo motor. This is done by the function Servo_init()

3) Start the ADC conversion by following these steps

* Enable the ADC module by setting ADCON0bits.ADON = 1

* Start the conversion by setting ADCON0bits.ADGO = 1

4) wait for ADC conversion to complete i.e till ADGO bit is cleared. This is done by using a while loop.

while(ADCON0bits.ADGO==1);

5) ADC results will be in the registers ADRESH and ADRESL. Shift left ADRESH by 8 bits and add the ADRESL to the result in order to get the complete 10 bit result value.

```
lightval = ((ADRESH<<8) |ADRESL);
```

6)after getting the result of ADC conversion disable the ADC by setting ADCON0bits.ADON = 0

7) If the ADC conversion result is more than 900.i.e if the light intensity is more then turn the LED OFF by setting RC0 = 0

8) Else, turn on the LED by setting RC0 = 1

9) wait for some time before starting the next conversion using _delay_ms() function.

10) call the ServoRotate180() and ServoRotate90() functions so that the Servo motor rotates and covers the LDR from light or exposes the LDR to the light.

Explanation of ServoRotate0() function:

We need to generate a PWM of close to 50 Hz frequency. So, our time period has to be nearly 20 ms.

* To rotate 0 degree, a pulse of 0.8 ms is required. So, set the Bit 0 of PORT B = 1 and keep it high for 800 us which is achieved by

```
PORTBbits.RB0 = 1;
__delay_us(800);
PORTBbits.RB0 = 0;
__delay_us(19200);
```

These statements are in a for loop so that the servomotor stays in that state for some amount of time.

Explanation of ServoRotate90() function:

We need to generate a PWM of close to 50 Hz frequency. So our time period has to be nearly 20 ms.

* In order to rotate 90 degree, a pulse of 1.5 ms is required. So set the Bit 0 of PORT B = 1 and keep it high for 1500 us which is achieved by

```
PORTBbits.RB0 = 1;
__delay_us(1500);
PORTBbits.RB0 = 0;
__delay_us(18500);
```

These statements are in a for loop so that the servomotor stays in that state for some amount of time.

Explanation of ServoRotate180() function:

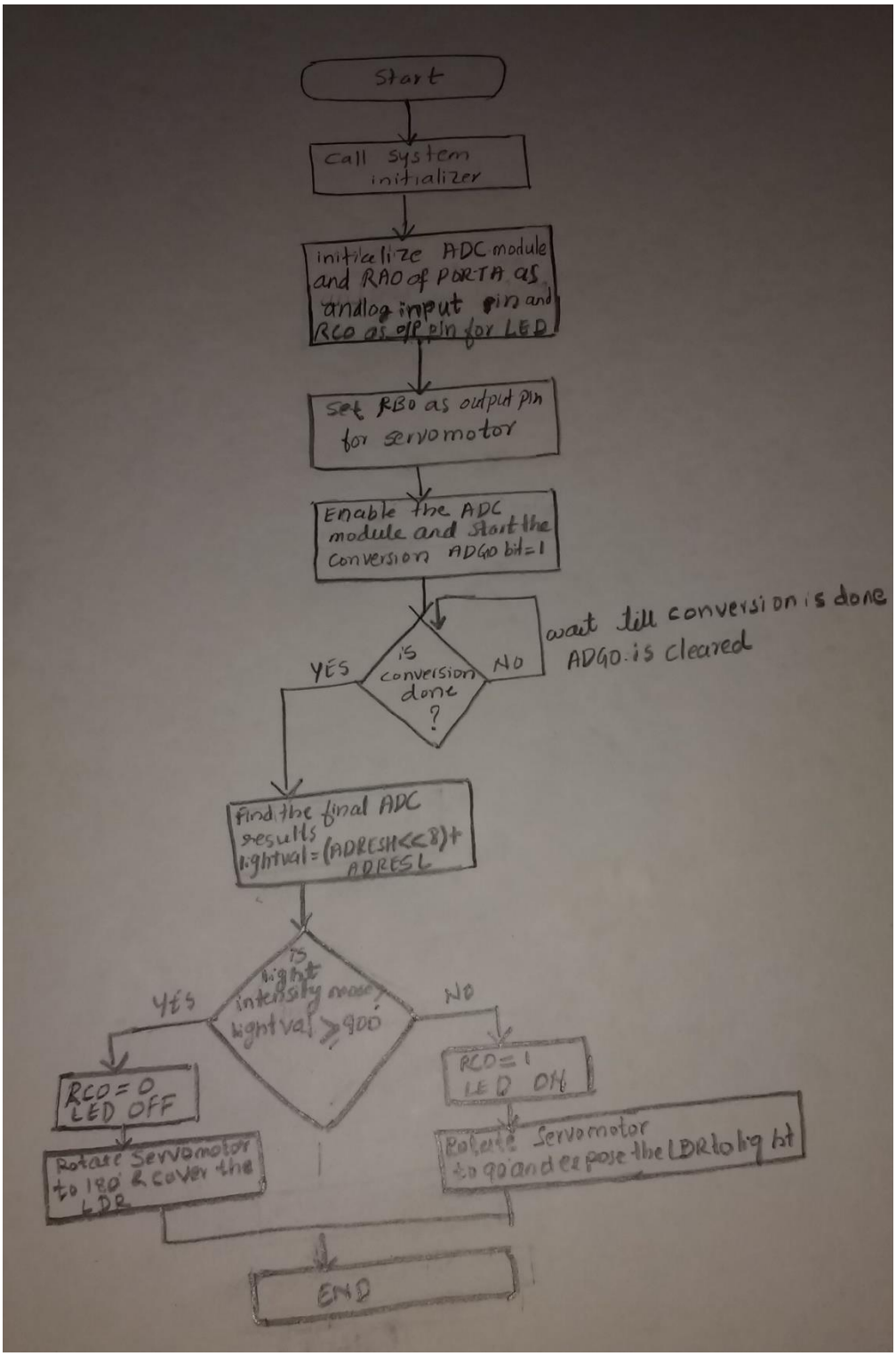
We need to generate a PWM of close to 50 Hz frequency. So, our time period has to be nearly 20 ms.

* In order to rotate 180 degrees, a pulse 2.2 ms is required. So, set the Bit 0 of PORT B = 1 and keep it high for 2200 us which is achieved by

```
PORTBbits.RB0 = 1;
__delay_us(2200);
PORTBbits.RB0 = 0;
__delay_us(17800);
```

These statements are in a for loop so that the servomotor stays in that state for some amount of time.

Flow Chart:


Start
Call system initializer
initialize ADC module and RA0 of PORTA as analog input pin and RC0 as o/p pin for LED
Set RB0 as output pin for servomotor
Enable the ADC module and start the conversion ADGO bit=1
wait till conversion is done ADGO is cleared
is conversion done ?
YES
NO
Find the final ADC results lightval = (ADRESH<<8)+ ADRESL
is light intensity more? lightval >= 900
YES
NO
RC0 = 0 LED OFF
RC0 = 1 LED ON
Rotate Servomotor to 180° & cover the LDR
Rotate Servomotor to 90° and expose the LDR to light
END

Section 8 : Trouble Shooting

ISSUE 1:

At first the Pickit3 was not getting detected by the laptop and hence the basic LED program couldn't be loaded into the Pic microcontroller. Even though all the three LED's of the Pickit3 was working fine, the program was not getting compiled and gave the error "Target device was not found".

We understood that our pickit3 was not working because when we connected it and checked it on the MPLAB IPE it was not able to read the Pickit3.

The problem was solved when we changed the pickit3.

ISSUE 2:

Pickit3 was not providing the sufficient voltage which is required to drive the pic microcontroller, so whenever we tried compiling it gave the error that "Target device requires more power, so please connect adequate power supply and check connections again".

The issue was solved when we added the pull up resistor and changed the power setting of pickit3 to low voltage programming and gave 3.3 volts from the pickit3 itself.

ISSUE 3:

ADC input channel which is at the intersection of pull up resistor and LDR was getting very less voltage, so was not able to drive the ADC and convert the analog value to digital value properly.

This was solved when we increased the pull up resistance from 330 ohm to 4.7 K ohm which gave the ADC input sufficient voltage and was thus able to convert the analog input of LDR.

ISSUE 4:

The servo motor was not working because the input voltage given to it was very less.

When we provided the servo motor 5V from the FTDI Cable, we got proper results.

ISSUE 5:

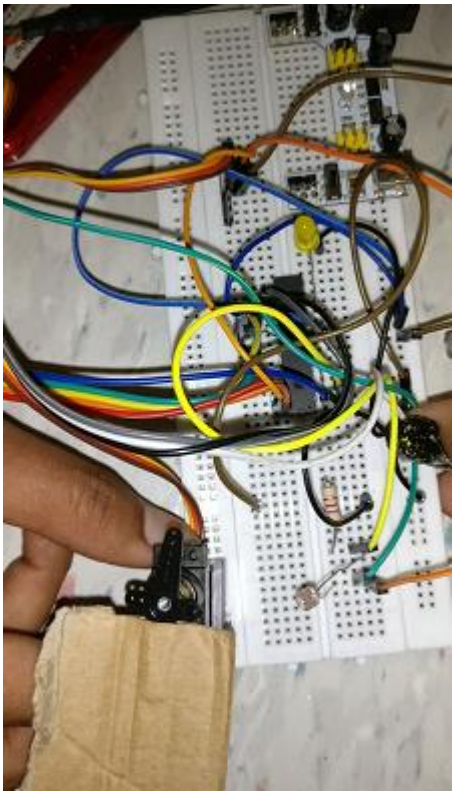
We had some programming and syntax errors also.

We understood that for initializing any port, for example we have to use TRISBbits.RB0=0 and not just RB0 =0.

Section 9 : Results

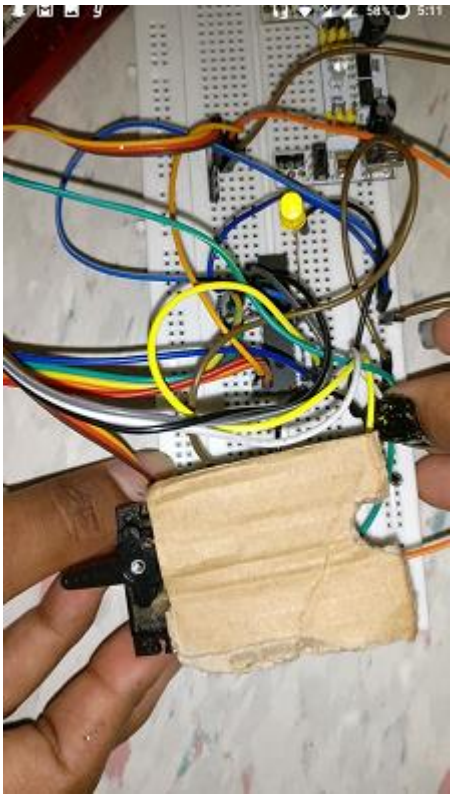
Screenshot 1:

The servo motor rotates and exposes the LDR to the light. The resistance of the LDR decreases and the voltage to the analog input pin increases. The LED is turned OFF.



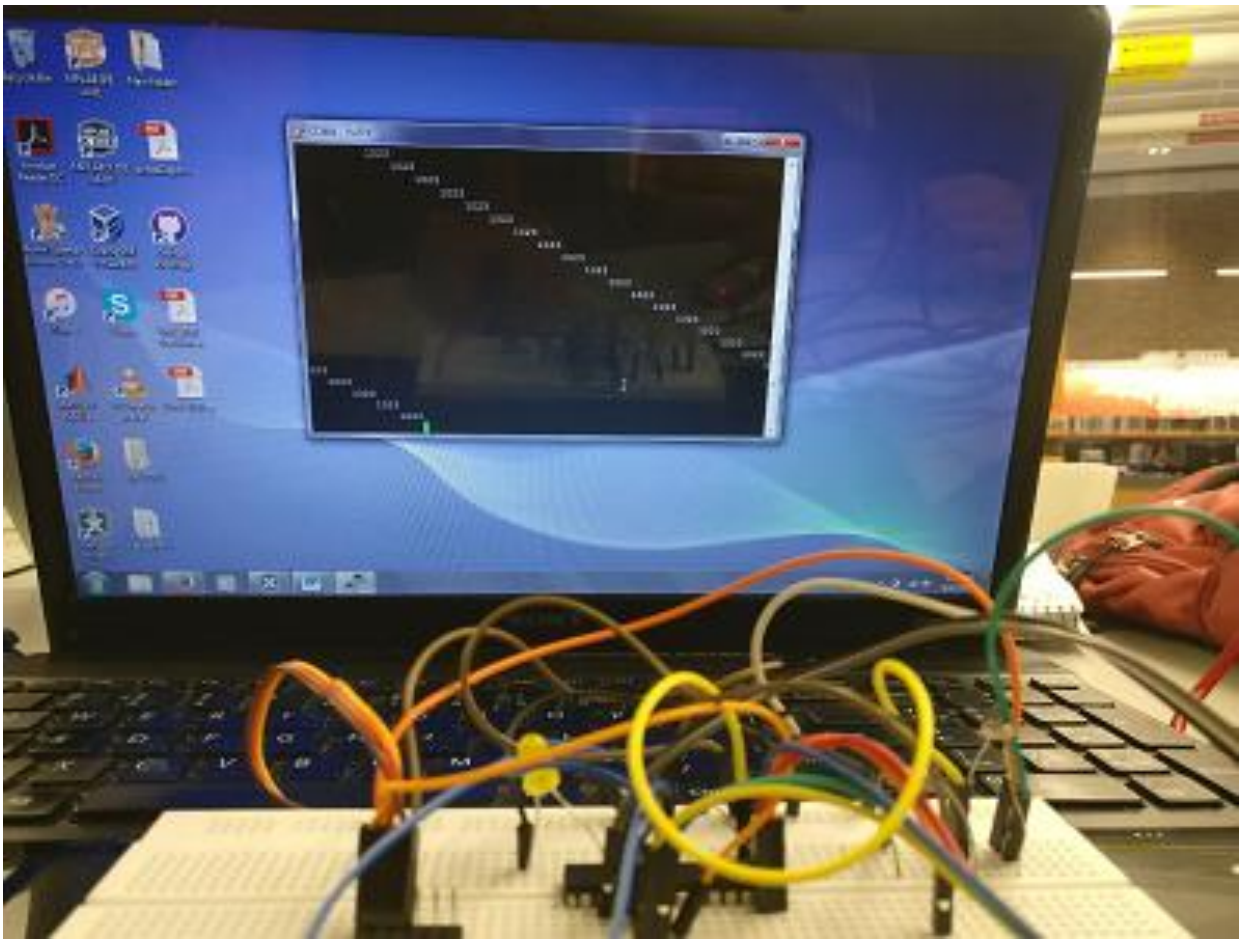
Screenshot2:

The servomotor rotates again 90 degrees and hides the LDR from the light with the blackboard. LED is turned ON in this case.



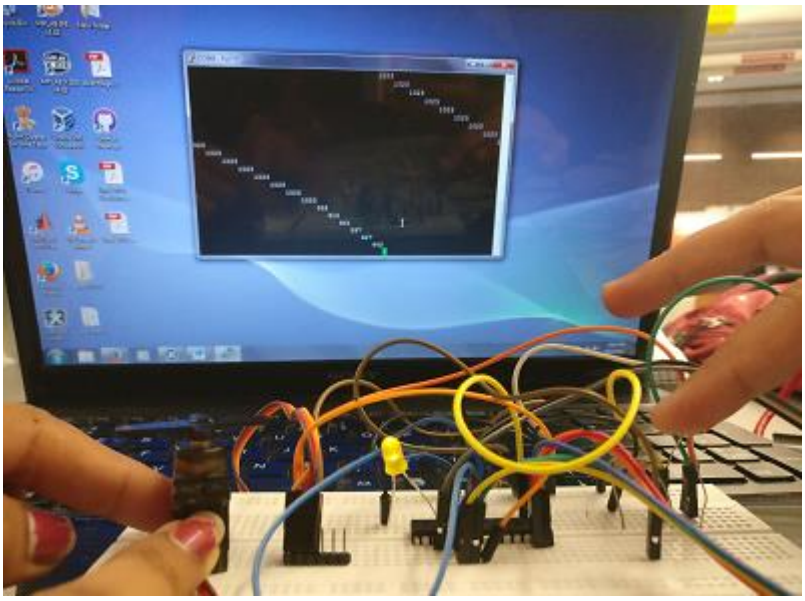
Screenshot 3:

The servo motor rotates and exposes the LDR to the light. The resistance of the LDR decreases and the voltage to the analog input pin increases. As a result, the ADC conversion result increases and is 1023 in our case. The digital output of the ADC is displayed on the Putty terminal.



Screenshot 4:

The servo motor rotates and hides the LDR from the light. The resistance of the LDR increases and the voltage to the analog input pin decreases. As a result, the ADC conversion result decreases. The digital output of the ADC is displayed on the Putty terminal.

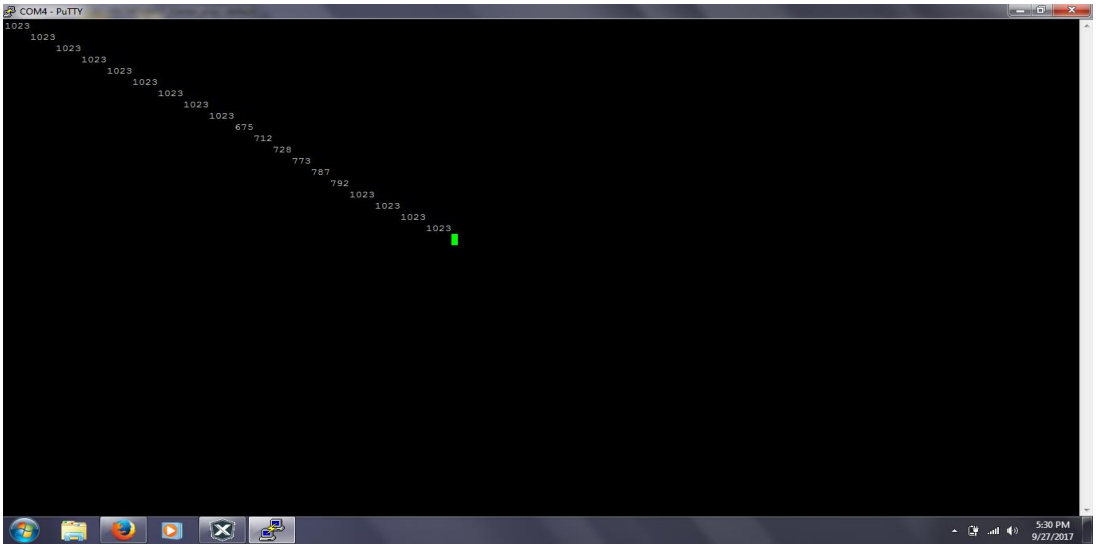


Screenshot 5:

The digital output values of the ADC conversion results depending on the variation of light intensity.

As the light intensity increases, LDR resistance decreases and the ADC output value is increased.

As the light intensity decreases, LDR resistance increases and the ADC output value is decreased.



Section 10: Appendix

Appendix 1

Code for ADC conversion

```
#define _XTAL_FREQ 1000000
#include<xc.h>
#include "mcc.h" //default library
#include<pic16F18857.h>
void ADC_Init(void)
{
/*  Configure ADC module  */
 /*----- Set the Registers below:: */
 /* 1. Set ADC CONTROL REGISTER 1 to 0 */
 ADCON1 = 0x00;
 /* 2. Set ADC CONTROL REGISTER 2 to 0 */
 ADCON2=0x00;
 /* 3. Set ADC THRESHOLD REGISTER to 0 */
 ADCON3 =0x00;
 /* 4. Disable ADC auto conversion trigger control register */ /* 5. Disable ADACT */
 ADACT =0x00;
 /* 6. Clear ADAOV ACC or ADERR not Overflowed  related register */
 ADSTAT =0x00;
 /* 7. Disable ADC Capacitors */
 ADCAP = 0x00;
 /* 8. Set ADC Precharge time control to 0 */
 ADPRE =0x00;
 /* 9. Set ADC Clock */
 ADCLK = 0x02;
 /* 10 Set ADC positive and negative references*/
```

```
 ADREF = 0x00; // VREF+ is connected to VDD and VREF- is connected to analog ground//
 TRISA = 0x01;
 PORTA = 0x01;
 /* 11. ADC chanel - Analog Input */
 ANSELA = 0x01;
 TRISCbits.TRISC0 = 0;
 PORTCbits.RC0=0;
 /* 12. Set ADC result alignment, Enable ADC module, Clock Selection Bit, Disable ADC Continuous Operation, Keep ADC inactive*/
 ADCON0 = 0x84;         //ADON =1,ADCONT =0,ADCS =1, ADFRM0 = 0i.e right justified, ADG0 = 0//
 //ADACQ = 0;
}
void main(void)
{
    //int lightval =0;
    // Initialize PIC device
    SYSTEM_Initialize();
    // Initialize the required modules
    ADC_Init();
    while(1)
    {
  // **** write your code
    int lightval;
    //setting the output port
    ADCON0bits.ADON = 1; //enable the ADC module
    //ADCON0bits.ADCONT=1;//    continuous sampling
    ADCON0bits.ADGO =1; //start the conversion
    while(ADCON0bits.ADGO==1);  //if the conversion has not been completed then wait
```

```
    lightval = ((ADRESH<<8)|ADRESL);
    ADCON0bits.ADON = 0;  //after getting the result of ADC conversion disable the ADC
    printf("%d\n",lightval);
        if(900<=lightval) //if the resistance of the photo resistor is less(light is more) , input voltage to the analog channel  will be more
        //1023= 3.3 v corresponds to 3.3v
        {
            PORTCbits.RC0 = 0;
        }
        else

            PORTCbits.RC0 =1;
        }
    __delay_ms(1000); //wait for some time before starting the next conversion
     //disable the ADC module
    }
}
 End of File
}
```

…… …… ……

Appendix:2

Code for Servo motor

```
    #define _XTAL_FREQ 1000000
    #include <xc.h>
    #include<pic16f18857.h>
    // BEGIN CONFIG
    //END CONFIG
    void servoRotate0() //0 Degree
    {
      unsigned int i;
      for(i=0;i<50;i++)
      {
```

```
    PORTBbits.RB0 = 1;
    __delay_us(800);
    PORTBbits.RB0 = 0;
    __delay_us(19200);
  }
}
void servoRotate90() //90 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTBbits.RB0 = 1;
    __delay_us(1500);
    PORTBbits.RB0 = 0;
    __delay_us(18500);
}
}
void servoRotate180() //180 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
{
    PORTBbits.RB0 = 1;
    __delay_us(2200);
    PORTBbits.RB0 = 0;
    __delay_us(17800);
}
}
void main()
{
  TRISBbits.TRISB0 = 0; // PORTB as Ouput Port
  do
{
    servoRotate0(); //0 Degree
    __delay_ms(2000);
    servoRotate90(); //90 Degree
    __delay_ms(2000);
    servoRotate180(); //180 Degree
  }while(1);
}
```

…… …… ……

Appendix:3

Main program

```
#include "mcc_generated_files/mcc.h" //default library
/*
 */
#define _XTAL_FREQ 1000000
#include<xc.h>
//#include "mcc.h" //default library
#include<pic16f18857.h>


void ADC_Init(void)
{
/*  Configure ADC module  */



 /*----- Set the Registers below:: */
 /* 1. Set ADC CONTROL REGISTER 1 to 0 */

 ADCON1 = 0x00;
 /* 2. Set ADC CONTROL REGISTER 2 to 0 */
 ADCON2=0x00;
 /* 3. Set ADC THRESHOLD REGISTER to 0 */
 ADCON3 =0x00;
 /* 4. Disable ADC auto conversion trigger control register */ /* 5. Disable ADACT */
 ADACT =0x00;
```

```
    /* 6. Clear ADAOV ACC or ADERR not Overflowed  related register */
    ADSTAT =0x00;

    /* 7. Disable ADC Capacitors */
    ADCAP = 0x00;

    /* 8. Set ADC Precharge time control to 0 */
    ADPRE =0x00;

    /* 9. Set ADC Clock */
    ADCLK = 0x02;

    /* 10 Set ADC positive and negative references*/
    ADREF = 0x00; // VREF+ is connected to VDD and VREF- is connected to analog ground//

    TRISA = 0x01;
    PORTA = 0x01;
    /* 11. ADC chanel - Analog Input */
    ANSELA = 0x01;
    TRISCbits.TRISC0 = 0;
    PORTCbits.RC0=0;
    /* 12. Set ADC result alignment, Enable ADC module, Clock Selection Bit, Disable ADC Continuous Operation, Keep ADC inactive*/

    ADCON0 = 0x84;        //ADON =1,ADCONT =0,ADCS =1, ADFRM0 = 0i.e right justified, ADG0 = 0//

    //ADACQ = 0;
```

```
}
void servo_init(void)
{
    TRISBbits.TRISB0 = 0;
    PORTBbits.RB0 = 0;
}
void servoRotate0() //0 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTBbits.RB0 = 1;
    __delay_us(800);
    PORTBbits.RB0 = 0;
    __delay_us(19200);
  }
}

void servoRotate90() //90 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTBbits.RB0 = 1;
    __delay_us(1500);
    PORTBbits.RB0 = 0;
    __delay_us(18500);
  }
```

```
}

void servoRotate180() //180 Degree
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTBbits.RB0 = 1;
    __delay_us(2200);
    PORTBbits.RB0 = 0;
    __delay_us(17800);
  }
}

void main(void)
{
    //int lightval =0;
    // Initialize PIC device
    SYSTEM_Initialize();

    // Initialize the required modules
    ADC_Init();
    servo_init();
    while(1)
    {

  // **** write your code
    int lightval;
```

```
//setting the output port

ADCON0bits.ADON = 1; //enable the ADC module
//ADCON0bits.ADCONT=1;//    continuous sampling
ADCON0bits.ADGO =1; //start the conversion
/*-------------------------------------------------------------------------------------*/
while(ADCON0bits.ADGO==1);  //if the conversion has not been completed then wait

lightval = ((ADRESH<<8)|ADRESL);
ADCON0bits.ADON = 0;  //after getting the result of ADC conversion disable the ADC
printf("%d\n",lightval);
   if(900<=lightval) //if the resistance of the photo resistor is less(light is more) , input voltage to the analog channel  will be more
   //1023= 3.3 v corresponds to 3.3v
   {

      PORTCbits.RC0 = 0;
      servoRotate180();
       //180 Degree
   }
   else
   {

      PORTCbits.RC0 =1;
      servoRotate90();
       //90 Degree5
   }
__delay_ms(1000);
```

```
  //servoRotate0(); //0 Degree
 //servoRotate180(); //180 Degree
 //__delay_ms(2000);
 //servoRotate90(); //90 Degree5
// __delay_ms(2000);

  //disable the ADC module

   }
}
```

Appendix:4

1. http://www.microchip.com/
2. https://github.com/