Lab 1: Sensor Design and Analog Digital Conversion

16.480/552 Microprocessor Design II and Embedded Systems

Instructor: *Prof. Yan Luo*

Group-I

Due:10/3/16

Submitted: 10/3/16

By,

- ***Naga Ganesh Kurrapati***
- *Sayali Vaidya*
- *Zubin Pattnaik*

Table of contents

I. Personal contribution

My contribution to the project is to generate the generate ADC and USART module code using MP-Lab code configure. And troubleshooting those components in the circuit.

II. Purpose

The main purpose of this project is to design sensor circuit using embedded microcontrollers and understand the operation of Analog to digital convertor (ADC).

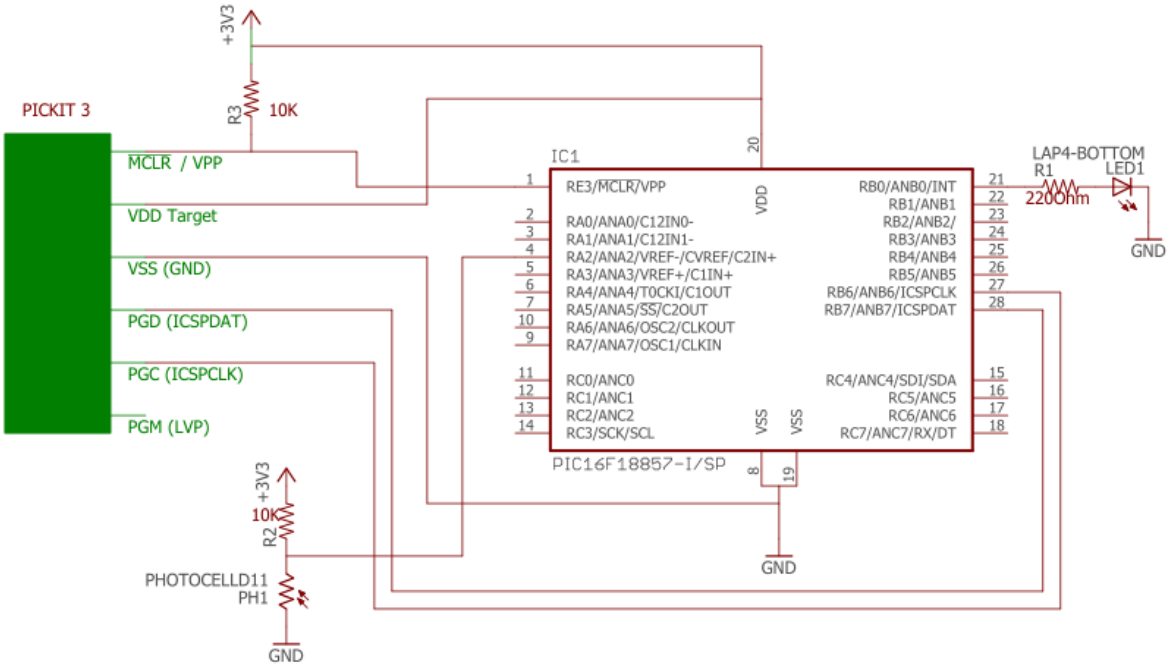
III. Introduction

The main objective of this lab is to read the data from a photo resistor through ADC module of the microcontroller PIC16F18857. Using those data, glow a LED at threshold value of the blocked (no light) sensor. Estimate the threshold value by measuring the voltage across the sensor during the normal light and blocked condition. Compare those data with the ADC data measured using USART module of the PIC.

IV. Materials, Devices and Instruments Used

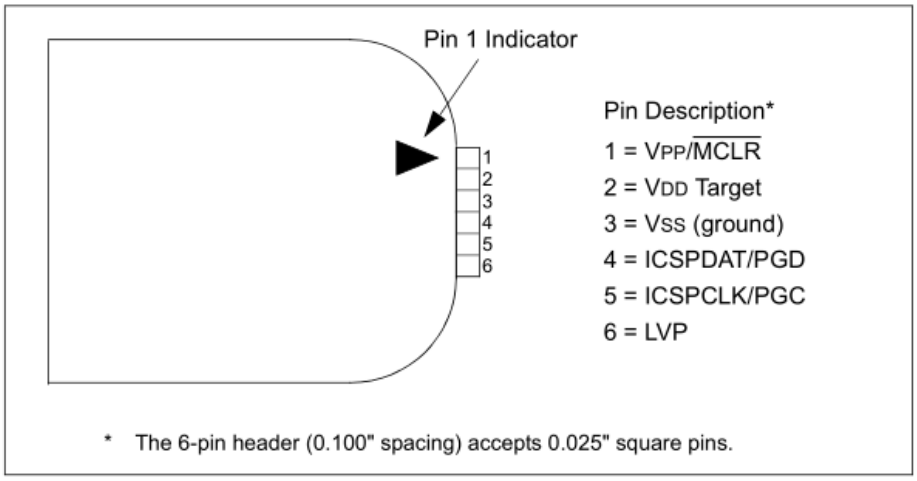
1. Bread board
2. Wires to connect
3. PIC16F18857 microcontroller
4. Pickit3
5. LED
6. Photo resistor
7. two 10k, one 220 Ohm resistors
8. Serial to USB connector
9. Multi-meter
10. Voltage supply (3.3V)

V. Schematics



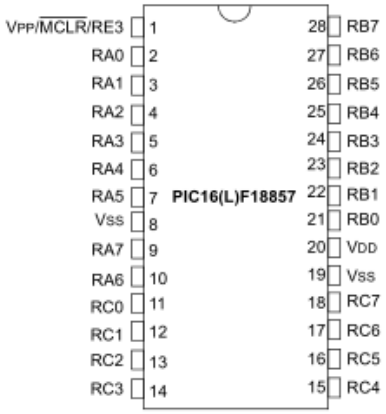
VI. Lab Methods and Procedure

FIGURE 1-2: PICKIT™ 3 PROGRAMMER CONNECTOR PINOUT



PIN DIAGRAMS

28-pin SPDIP, SOIC, SSOP

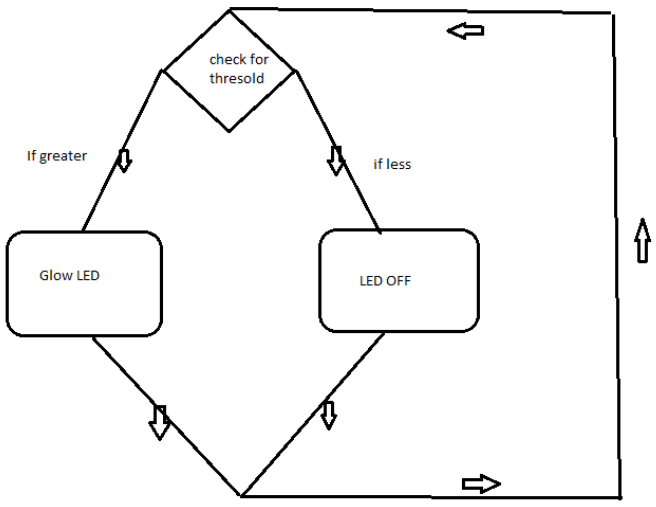


Note 1: See Table 2 for location of all peripheral functions.

2: All VDD and all Vss pins must be connected at the circuit board level. Allowing one or more Vss or VDD pins to float may result in degraded electrical performance or non-functionality.

Hardware Design: Initially Pickit3 is connected to the microcontroller. If you observe the pin diagram of both Pickit 3 on top and PIC. Both MCLR,Vdd,Vss, ICSPDAT/PGD, ICSPCLK/PGC are connected to each other. ICSPDAT is pin 27 and ICSPCLK is pin 28 for the PIC. The MCLR is connected to Vdd through 10K ohm resistor. The sensor is connected through ADC Channel 2(Pin 4). And LED is connected to the pin PB0 (Pin21). A 220-ohm resistor is connected in series to the LED, for protection.

Software Design: It is simple design where LED is glowed when the sensor input is greater than the threshold value. LED is switched off when it is below the threshold. The threshold value calculation is shown in flowing sections. The flow chart can be seen below.



VII. Trouble Shooting

The first is to assure the correctness of the circuit by blinking the LED for few minutes without the sensor data. Then relate the sensor data to ON the led. To check the sensor, you need to measure the voltage across it during blocked and unblocked situations. The measured values are 2.8v during unblocked and 1v during the blocked.

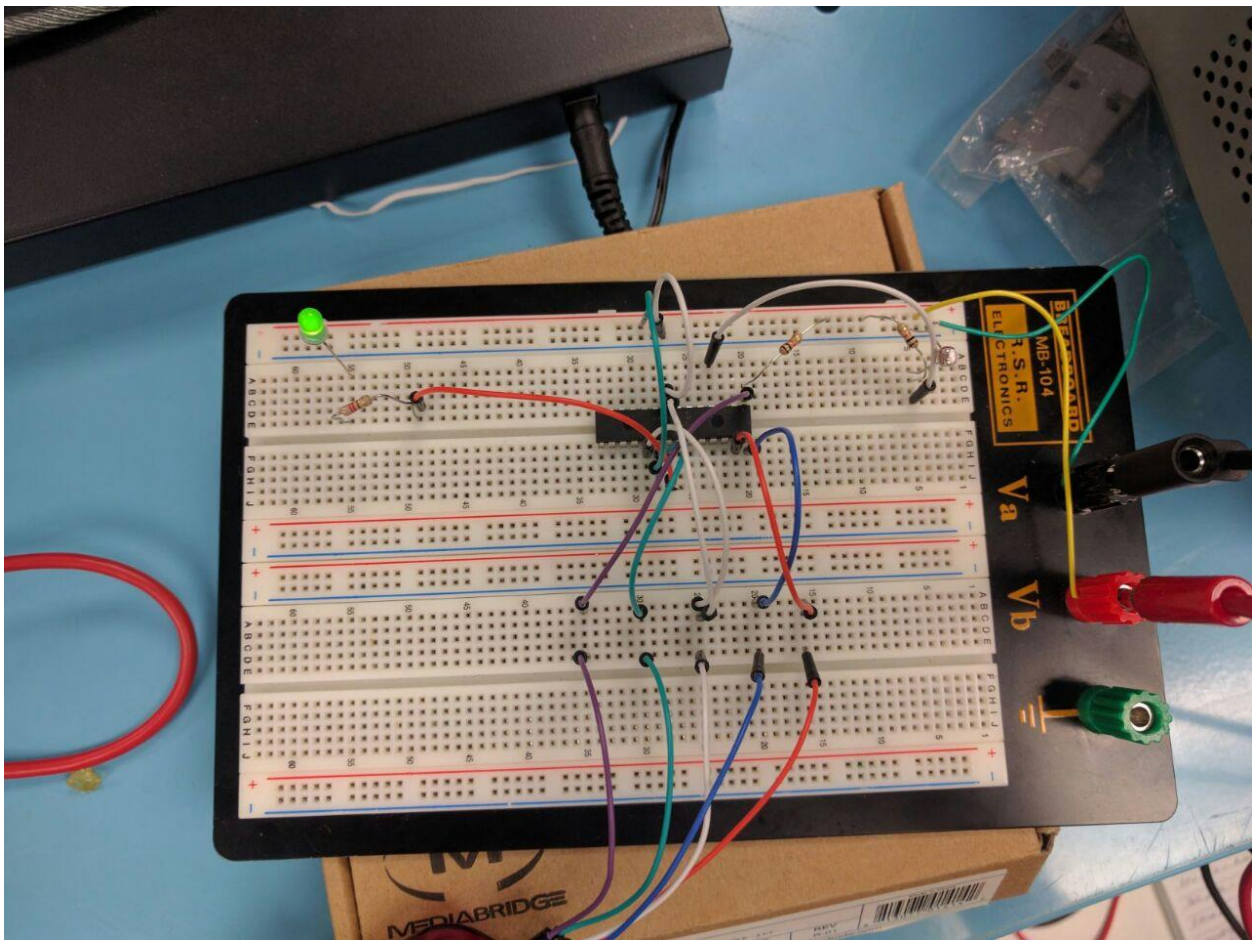
To estimate ADC output using the formula. Signal = (sample/1024) * Reference voltage.

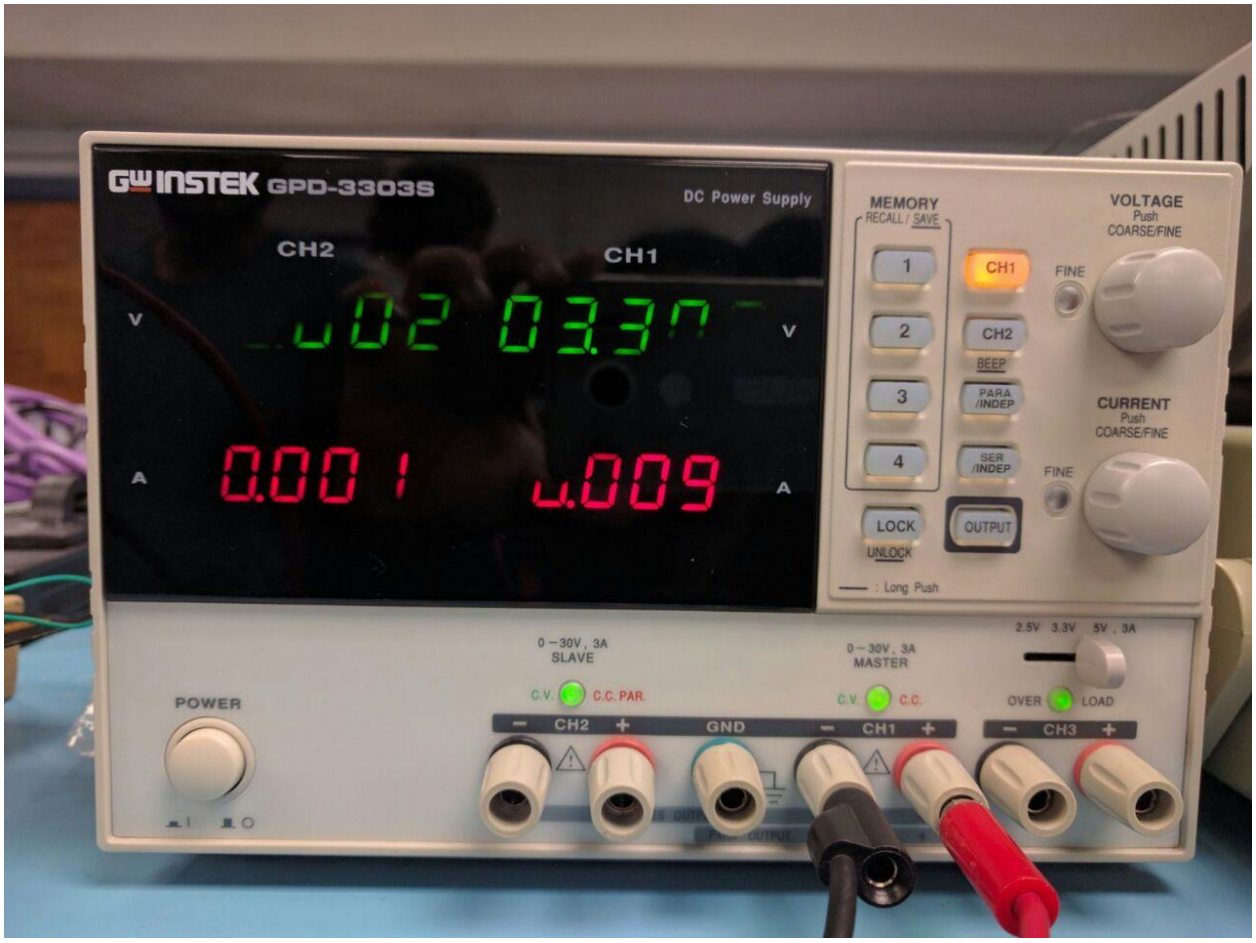
For 2.8v, sample = 868

For 1v, sample = 310 with reference voltage = 3.3 v and (2^10- 10bits of ADC value) 1024 base value.

The second is to read the ADC value using the USART module. In this case letters are chosen based on the ranges to transmit over the seral port. The example code can be seen in Appendix. The value for this experiment was 218 but 230 was chosen to cancel the noise.

VIII. Results





As shown in the pictures the LED is glows in the dark and stops in the ambient light. The voltage supply is 3.3v.

IX. Contributions to this Lab

I worked on the MP lab software to generate the code and my team members sayali and Zubin worked with threshold value calculation and hardware design.

X. Appendix – Code

Main.c

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

/*
                Main application
 */
void main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    while (1)
    {
        if(ADCC_GetSingleConversion(channel_ANA2)>230)
        {
            IO_RB0_SetHigh();
        }
        else
        {
            IO_RB0_SetLow();
        }
    }
}
```

PinManager.h

```
#ifndef PIN_MANAGER_H
#define PIN_MANAGER_H

#define INPUT   1
#define OUTPUT  0

#define HIGH    1
#define LOW     0

#define ANALOG      1
#define DIGITAL     0

#define PULL_UP_ENABLED      1
#define PULL_UP_DISABLED     0

// get/set IO_RB0 aliases
#define IO_RB0_TRIS             TRISB0
```

```
#define IO_RB0_LAT              LATB0
#define IO_RB0_PORT             RB0
#define IO_RB0_WPU              WPUB0
#define IO_RB0_ANS              ANSB0
#define IO_RB0_SetHigh()    do { LATB0 = 1; } while(0)
#define IO_RB0_SetLow()   do { LATB0 = 0; } while(0)
#define IO_RB0_Toggle()   do { LATB0 = ~LATB0; } while(0)
#define IO_RB0_GetValue()         PORTBbits.RB0
#define IO_RB0_SetDigitalInput()    do { TRISB0 = 1; } while(0)
#define IO_RB0_SetDigitalOutput()   do { TRISB0 = 0; } while(0)

#define IO_RB0_SetPullup()    do { WPUB0 = 1; } while(0)
#define IO_RB0_ResetPullup()   do { WPUB0 = 0; } while(0)
#define IO_RB0_SetAnalogMode()   do { ANSB0 = 1; } while(0)
#define IO_RB0_SetDigitalMode()   do { ANSB0 = 0; } while(0)

/**
 * @Param
   none
 * @Returns
   none
 * @Description
   GPIO and peripheral I/O initialization
 * @Example
   PIN_MANAGER_Initialize();
 */
void PIN_MANAGER_Initialize (void);


#endif // PIN_MANAGER_H
```

PinManager.c

```
#include <xc.h>
#include "pin_manager.h"

void PIN_MANAGER_Initialize(void)
{
   LATB = 0x0;
   LATA = 0x0;
   LATC = 0x0;
   WPUA = 0x0;
   WPUB = 0x0;
   WPUC = 0x0;
```

```
    ANSELA = 0xFF;
    ANSELB = 0xFE;
    ANSELC = 0xFF;
    TRISB = 0xFE;
    TRISC = 0xFF;
    TRISA = 0xFF;

}
```

Mcc.h

```
// CONFIG1
#pragma config FEXTOSC = OFF    // External Oscillator mode selection bits->Oscillator not enabled
#pragma config RSTOSC = HFINT1    // Power-up default value for COSC bits->HFINTOSC (1MHz)
#pragma config CLKOUTEN = OFF    // Clock Out Enable bit->CLKOUT function is disabled; i/o or oscillator function on OSC2
#pragma config CSWEN = ON    // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
#pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable bit->FSCM timer enabled

// CONFIG2
#pragma config MCLRE = ON    // Master Clear Enable bit->MCLR pin is Master Clear function
#pragma config PWRTE = OFF    // Power-up Timer Enable bit->PWRT disabled
#pragma config LPBOREN = OFF    // Low-Power BOR enable bit->ULPBOR disabled
#pragma config BOREN = ON    // Brown-out reset enable bits->Brown-out Reset Enabled, SBOREN bit is ignored
#pragma config BORV = LO    // Brown-out Reset Voltage Selection->Brown-out Reset Voltage (VBOR) set to 1.9V on LF, and 2.45V on F Devices
#pragma config ZCD = ON    // Zero-cross detect disable->Zero-cross detect circuit is disabled at POR.
#pragma config PPS1WAY = ON    // Peripheral Pin Select one-way control->The PPSLOCK bit can be cleared and set only once in software
#pragma config STVREN = ON    // Stack Overflow/Underflow Reset Enable bit->Stack Overflow or Underflow will cause a reset

// CONFIG3
#pragma config WDTCPS = WDTCPS_31    // WDT Period Select bits->Divider ratio 1:65536; software control of WDTPS
#pragma config WDTE = OFF    // WDT operating mode->WDT Disabled, SWDTEN is ignored
#pragma config WDTCWS = WDTCWS_7    // WDT Window Select bits->window always open (100%); software control; keyed access not required
#pragma config WDTCCS = SC    // WDT input clock selector->Software Control

// CONFIG4
#pragma config WRT = OFF    // UserNVM self-write protection bits->Write protection off
#pragma config SCANE = available    // Scanner Enable bit->Scanner module is available for use
```

```
#pragma config LVP = ON    // Low Voltage Programming Enable bit->Low Voltage programming enabled. MCLR/Vpp pin function is MCLR.

// CONFIG5
#pragma config CP = OFF    // UserNVM Program memory code protection bit->UserNVM code protection disabled
#pragma config CPD = OFF    // DataNVM code protection bit->DataNVM code protection disabled

#include "mcc.h"

void SYSTEM_Initialize(void)
{

   PIN_MANAGER_Initialize();
   OSCILLATOR_Initialize();
   ADCC_Initialize();
}

void OSCILLATOR_Initialize(void)
{
   // NOSC HFINTOSC; NDIV 4;
   OSCCON1 = 0x62;
   // CSWHOLD may proceed; SOSCPWR Low power;
   OSCCON3 = 0x00;
   // MFOEN disabled; LFOEN disabled; ADOEN disabled; SOSCEN disabled; EXTOEN disabled; HFOEN disabled;
   OSCEN = 0x00;
   // HFFRQ 4_MHz;
   OSCFRQ = 0x02;
   // MFOR not ready;
   OSCSTAT = 0x00;
   // HFTUN 0;
   OSCTUNE = 0x00;
   // Set the secondary oscillator

}
```

For USART the Main.c is as below:

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

/*
                Main application
 */
```

```
void main(void)
{
   // initialize the device
   SYSTEM_Initialize();

   while (1)
   {
      uint16_t tmp = ADCC_GetSingleConversion(channel_ANA2);
      //EUSART_Write(tmp);
      //if(tmp>310)
      if(tmp>0 && tmp <210)
      {
         EUSART_Write(65);// A
        //  IO_RB0_SetHigh();
      }
      else if(tmp >= 215 && tmp <= 218)
      {
         EUSART_Write(66); //B
         //IO_RB0_SetLow();
      }
      else if(tmp > 220 && tmp < 230)
      {
         EUSART_Write(67);///C
         //IO_RB0_SetLow();
      }
      else if(tmp >230 && tmp < 240)
      {
         EUSART_Write(68); // D
         //IO_RB0_SetLow();
      }
      else if(tmp > 240 && tmp < 250 )
      {
         EUSART_Write(69);// e
      }
      else if(tmp >= 256 && tmp <= 257)
      {
         EUSART_Write(70); //f
      }
      else
      {
         EUSART_Write(71); // g
      }
      }
  }
```