Interfacing with a Sensor Device on an Embedded Computer System

Course: Microprocessors Design II and Embedded Systems

Course #: EECE 4800/5520

Instructor: Yan Luo

Group #: 12

Student name: Zubair Nadaph

Hand in Date: 10/30/17

Lab Due Date: 10/30/17

Section 2: Contributions

1. Group Member 1 – Zubair Nadaph

    - Studied the Galileo board's GPIO driver settings required to establish the communication between the board and PIC.
    - Studied the bus protocols and generated the code for Strobe.
    - Developed the code for PIC controller along with the partner to read and write the instructions from Galileo board.

2. Group Member 2 - Aravind Dhulipalla
    - Developed the ADC code to control the light output using ADC, based on the LDR output
    - Developed the voltage divider circuit along with partner for ADC unit at RA0.
    - Developed the pwm code including setting up the modules, setting up timer for pulse width control and starting the motor operation.
3. Group Member 3 -
    - Worked on connecting the hardware circuit and assisted with the programming.

Section 3: Purpose

It focused on use of GPIO of Galileo board to communicate with PIC16F18857. It involved developing of protocol (for communication) between the master (Galileo) and slave (PIC) to transfer of data with each other.

Section 4: Introduction

This lab is based on communication of two embedded systems. In the first lab working of model of light detector was made and for this lab it was used to communicate with Galileo board.
The Galileo board is the master and PIC controller is the slave, a communication protocol is developed to transfer commands and receive data from slave system. A strobe is used as signal for communication between them. The aim of this lab is to ask the sensor device (PIC16F18857) to read light intensity (ADC) using galileo and light and LED if it's below certain value using MSG_PING, MSG_RESET, MSG_GET and MSG_TURNxxx.

Section 5: Materials, Devices and Instruments

1. PIC16F18857
2. PICKit3 Programmer/Debugger
3. Light Detecting Resistor (LDR)
4. FTDI cable
5. Breadboard and jumper wires
6. 10K and 220 ohm resistors
7. Servo motor – 2.5 Kg-cm, 4.8 – 6 V
8. Intel Galileo Gen2
9. Micro SD card and adaptor
10. LED
11. MPlab X IDE software.
12. XC8 compiler
13. Notepad C++ editor
14. Analog Discovery2- Xilinx

Section 6: Schematics

Firstly, to program pic using PicKit3 the needed pin diagram and their attributes are:

1. PicKit3 connections:

FIGURE 1-2: PICKIT™ 3 PROGRAMMER CONNECTOR PINOUT

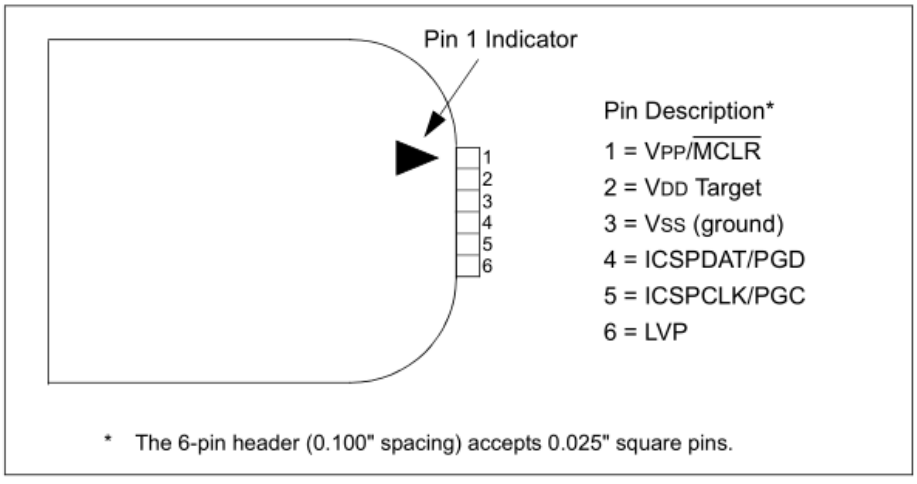
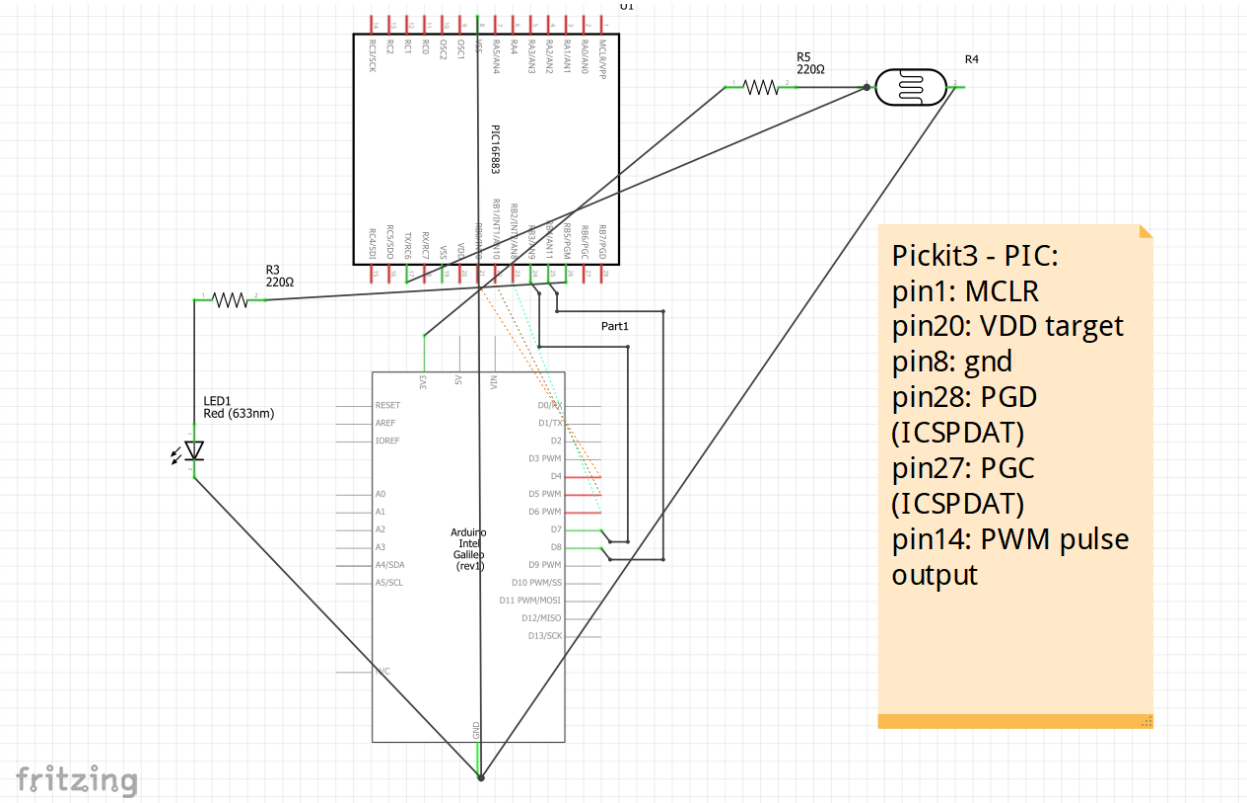


Fig. 1 pickit 3 pin description diagram



Section 7: Lab Methods and Procedure */2 points*

This lab is based on master and slave protocol in which the sensing element (PIC) is slave and Galileo Board is master. The master sends instruction via strobe, on receiving the instruction slave responds message acknowledgment by giving back a strobe and data.

In this lab slave is supposed to convert light intensity into digital value (received from LDR) and rotate servo motor by certain degree (0/60/120) upon request form master. The strobe signal controls the direction of communication between the slave and master. This information obtained is then processed and displayed on the screen.

To use the GPIO for data communication, they need to be configured and the configuration is done using the table show below:

Table1. Galileo GPIO configuration details

Galileo GPIO	PIC GPIO	Galileo GPIO	Linux OS
D4	RB0	4	6
D5	RB1	5	0
D6	RB2	6	1
D7	RB3	7	38
D8	RB4	8	40

From table 1 one can see that RB0-RB3 are the 4 data lines for communication and RB4 is used for strobe signal, the other connections are made as shown in fig.1. For configuring the GPIO pins, following steps were used are:

- Export the GPIO pin to make it accessible for interfacing
- Set the direction of GPIO i.e. input or output
- Set the mode of I/O between pullup, pulldown, strong or hiz.
- Set the value of port (high/low) if the direction is output.

Linux command to make an accessible IO-pin:

- echo -n “1” > /sys/class/gpio/export
- It exports the value 1 into the file /sys/class/gpio/export to make it an accessible IO pin.
- This pin will now appear as a new directory i.e inside /sys/class/gpio/

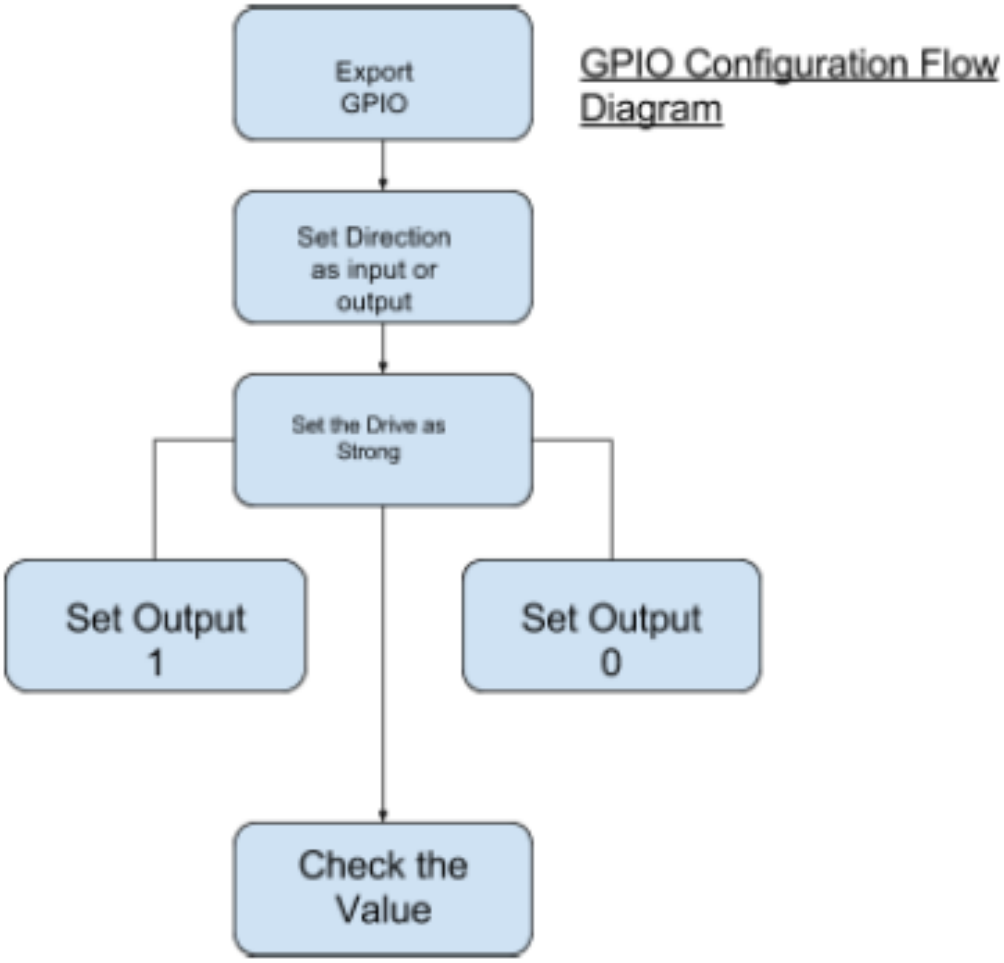
Linux command to make the pin an input or output:

- echo out > /sys/class/gpio/gpio40/direction
- echo in > /sys/class/gpio/gpio40/direction

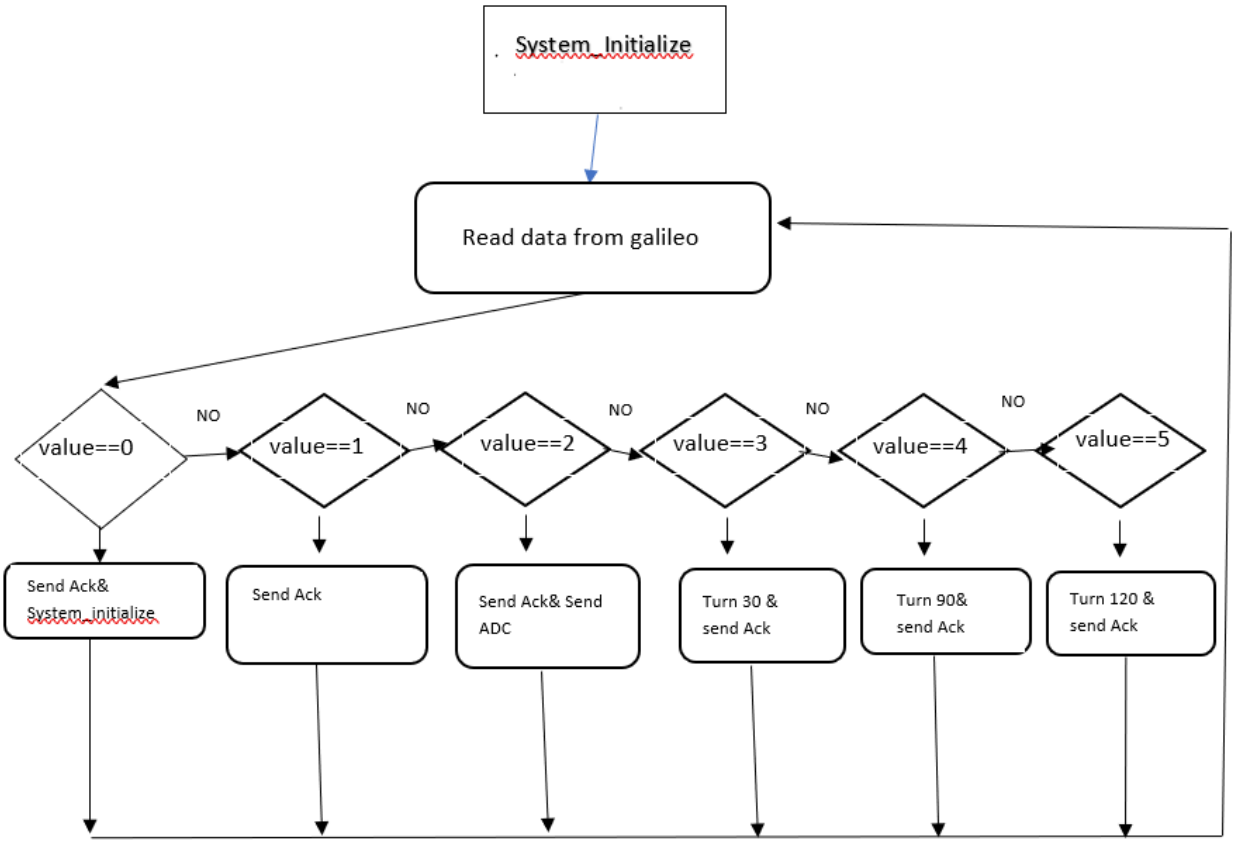
Procedure:

- Using the above-mentioned method, the I/O pins of the Galileo the port were configured first.
- The PIC code was separately written using MPlabs IDE where in we had RC port configured for databus (4 bit data) based on the direction of data. All the instructions form master come in the form of 4 bit data, which is processed (converted to hex) and cases have been set for different instructions.
- The response is then sent back to board using the same bus. The code for galileo board is configured to read the data and display the ADC value.
- It is required to unexport every time once the purpose of GPIO pin is completed and before the next use of same pin.

Flowchart for the configuration of GPIO pins:



Flow chart for PIC code is:



Section 8: Troubleshooting

We first assured the correctness of the circuit by blinking the LED for few minutes without going into ADC and other control. Then with standalone code and circuit for pic we related the sensor data to ON the led. Then after setting up ADC value required we cross checked it by measuring the voltage across it during blocked and unblocked situations. The measured values are 2.8v during unblocked and 1v during the blocked.

Next was to troubleshoot the PORTB- data pins configurations by blinking the LED at every pin used with a standalone code for galileo board.

Third was to check the signal transfer between devices (master and slave) for which ack message was sent back to board using some hex numbers and displayed on home screen of yocto.

Section 9: Code

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#define MSG_RESET 0x0
#define MSG_PING 0x1
#define MSG_GET 0x2
#define MSG_TURN30 0x3
#define MSG_TURN90 0x04
#define MSG_TURN120 0x5

void Export()
{
        //export the pin 8 GPIO 40
                system("echo 40 > /sys/class/gpio/export");
                //export the pin 7 GPIO 38
      system("echo 38 > /sys/class/gpio/export");
                //export pin 6 GPIO 1 and SHIFTER GPIO 20
                system("echo 1 > /sys/class/gpio/export");
                system("echo 20 > /sys/class/gpio/export");
                //export pin 5 GPIO 0 and SHIFTER GPIO 18
                system("echo 0 > /sys/class/gpio/export");
                system("echo 18 > /sys/class/gpio/export");
                //export pin 4 GPIO 6 and SHIFTER GPIO 36
                system("echo 6 > /sys/class/gpio/export");
                system("echo 36 > /sys/class/gpio/export");
}
void UnExport()
{
        //export the pin 8 GPIO 40
                system("echo 40 > /sys/class/gpio/unexport");
                //export the pin 7 GPIO 38
      system("echo 38 > /sys/class/gpio/unexport");
                //export pin 6 GPIO 1 and SHIFTER GPIO 20
                system("echo 1 > /sys/class/gpio/unexport");
                system("echo 20 > /sys/class/gpio/unexport");
                //export pin 5 GPIO 0 and SHIFTER GPIO 18
                system("echo 0 > /sys/class/gpio/unexport");
                system("echo 18 > /sys/class/gpio/unexport");
                //export pin 4 GPIO 6 and SHIFTER GPIO 36
                system("echo 6 > /sys/class/gpio/unexport");
```

```
                system("echo 36 > /sys/class/gpio/unexport");
}
void SetGPIO_output()
{
    //setting pin8 as an output
        system("echo out > /sys/class/gpio/gpio40/direction");
        //Setting pin7 as an output
        system("echo out > /sys/class/gpio/gpio38/direction");
        //setting pin6 as an output
        system("echo out > /sys/class/gpio/gpio1/direction");
        system("echo out > /sys/class/gpio/gpio20/direction");
        //setting pin5 as an output
        system("echo out > /sys/class/gpio/gpio0/direction");
        system("echo out > /sys/class/gpio/gpio18/direction");
        //setting pin4 as output
        system("echo out > /sys/class/gpio/gpio6/direction");
        system("echo out > /sys/class/gpio/gpio36/direction");
}

void SetGPIO_Input()
{
        //Setting pin7 as an input
        system("echo in > /sys/class/gpio/gpio38/direction");
        //setting pin6 as an input
        system("echo in > /sys/class/gpio/gpio1/direction");
        system("echo in > /sys/class/gpio/gpio20/direction");
        //setting pin5 as an input
        system("echo in > /sys/class/gpio/gpio0/direction");
        system("echo in > /sys/class/gpio/gpio18/direction");
        //setting pin4 as input
        system("echo in > /sys/class/gpio/gpio6/direction");
        system("echo in > /sys/class/gpio/gpio36/direction");
}

int GetResult()
{

}


int main()
{
        int msg;
        printf("select the number of the command: \n1.MSG-RESET \n2.MSG-MSG-
PING \n3.MSG-GET \n4.MSG-TURN30 \n5.MSG-TURN90 \n6.MSGTURN120\n");
   scanf("%d",&msg);
```

```
        char buffer[50];
switch(msg)
{
          case 1:
          Export();
          SetGPIO_output();
           system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 0 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 0 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                system("echo 0 > /sys/class/gpio/gpio40/value");
           UnExport();
                Export();
           SetGPIO_Input();
                sleep(1);
                sprintf(buffer,"cat /sys/class/gpio/gpio6/value");
                system(buffer);
                sprintf(buffer,"cat /sys/class/gpio/gpio0/value");
                system(buffer);
                sprintf(buffer,"cat /sys/class/gpio/gpio1/value");
                system(buffer);
                sprintf(buffer,"cat /sys/class/gpio/gpio38/value");
                system(buffer);
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                UnExport();
                break;
                case 2:
                Export();
           SetGPIO_output();
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 0 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                break;
                case 3:
                Export();
           SetGPIO_output();
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
```

```
        system("echo 0 > /sys/class/gpio/gpio6/value");
        system("echo 1 > /sys/class/gpio/gpio0/value");
        system("echo 0 > /sys/class/gpio/gpio1/value");
        system("echo 0 > /sys/class/gpio/gpio38/value");
        system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
        Export();
SetGPIO_Input();
        sleep(1);
        system("echo 0 > /sys/class/gpio/gpio40/value");
        system("echo 1 > /sys/class/gpio/gpio40/value");
        sprintf(buffer,"cat /sys/class/gpio/gpio6/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio0/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio1/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio38/value");
        system(buffer);
        system("echo 0 > /sys/class/gpio/gpio40/value");
         system("echo 0 > /sys/class/gpio/gpio40/value");
        system("echo 1 > /sys/class/gpio/gpio40/value");
        sprintf(buffer,"cat /sys/class/gpio/gpio6/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio0/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio1/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio38/value");
        system(buffer);
        system("echo 0 > /sys/class/gpio/gpio40/value");
         system("echo 0 > /sys/class/gpio/gpio40/value");
        system("echo 1 > /sys/class/gpio/gpio40/value");
        sprintf(buffer,"cat /sys/class/gpio/gpio6/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio0/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio1/value");
        system(buffer);
        sprintf(buffer,"cat /sys/class/gpio/gpio38/value");
        system(buffer);
        system("echo 0 > /sys/class/gpio/gpio40/value");
        UnExport();
        break;
        case 4:
                        Export();
```

```
            SetGPIO_output();
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 1 > /sys/class/gpio/gpio0/value");
                system("echo 0 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                break;
                case 5:
                                Export();
            SetGPIO_output();
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 0 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 1 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                break;
                case 6:
                                Export();
            SetGPIO_output();
                system("echo 0 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio40/value");
                system("echo 1 > /sys/class/gpio/gpio6/value");
                system("echo 0 > /sys/class/gpio/gpio0/value");
                system("echo 1 > /sys/class/gpio/gpio1/value");
                system("echo 0 > /sys/class/gpio/gpio38/value");
                system("echo 0 > /sys/class/gpio/gpio40/value");
                UnExport();
                break;
  }
}

//-------------------------------------PIC code:---------------------------------------



#include <pic16f18857.h>
#include "mcc_generated_files/mcc.h" //default library
#define value 0x0
#define MSG_ACK 0xE
#define MSG_NOTHING 0xF
#define ADC_Temp 0b0000000000

/* Circuit Connections
```

```
   Signal STROBE   RC6
   Signal D0       RC2
   Signal D1       RC3
   Signal D2       RC4
   Signal D3       RC5
 */
void Timer2_Init(void)
{
// CCPTMRS0 = 0x01; //SELECTED TIMER 2 FOR PWM
T2CON = 0x80; //CONFIGURED TIMER 2
T2CLKCONbits.CS = 0x01; //clk in relation with osc frequency
T2HLT = 0x00; //TIMER MODE
T2RST = 0x00; //Reset Source
PR2 = 0xFF; //lOAD THE PR2 VALUE
TMR2 = 0x00; //PRESCALE VALUE IS 0
PIR4bits.TMR2IF = 0; // CLEAR THE INTERRUPT FLAG
//T2CONbits.ON = 1; // START THE TIMER
}
void PWM_Init(void)
{
CCP1CONbits.EN = 1; // ENABLING THE
CCP1CONbits.FMT = 0; //RIGHT ALLIGNED FORMAT
CCP1CONbits.MODE = 0xF; // SETTING THE MODE TO PWM
CCPR1H = 0x00; // RH TO 0
CCPR1L = 0X00; //RL TO 0
CCPTMRS0 = 0X01; // SELECTS TIMER2
}
void PWM_signal_out(unsigned int duty)
{
    T2CONbits.ON = 1; // START THE TIMER
PMD3bits.PWM6MD = 0; //PWM 6 is enabled
CCPR1H = duty >>2; // 2 MSB'S IN CCPR1H
CCPR1L = (duty & 0x0003)<<6; //8 LSB'S IN CCPR1L
}
 int ADC_conversion_results()
{
    TRISAbits.TRISA0 = 1; // SETTING PORTA PIN0 TRISTATE REGISTER TO INPUT
    ANSELAbits.ANSA0 = 1; // SETTING PORTA PIN0 AS A ANALOG INPUT
    ADCON0bits.ADON = 1;  // ACTIVATING THE ADC MODULE
    ADCON0bits.GO = 1;    // START CONVERTING
    while(ADCON0bits.ADGO)// WAIT UNTIL THE CONVERSION
    {
    }
    int b = (ADRESH<<8)+(ADRESL); // MAKE THE ADC RESULT IN 10BITS
      ADCON0bits.GO = 0; // STOP CONVERTING
    return(b); // RETURN THE RESULT VALUE
}

void ADC_Init(void)
{
    ADCON1 = 0x00; // setting control register 1 to 0
    ADCON2 = 0x00; // setting control register 2 to 0
    ADCON3 = 0x00; // setting control register 3 to 0
    ADSTAT = 0x00; // setting threshold register and not overflowed  to 0
    ADCAP  = 0x00; // disabling ADC capacitors
    ADACT  = 0x00; // disabling Auto conversion trigger control register
    ADPRE  = 0x00; // setting precharge time control to 0
    ADCLK  = 0x00; // setting ADC clk
    ADREF  = 0x00; // setting ADC positive and negative reference voltages
    ANSELAbits.ANSA0 = 1; // setting ADC analog channel input to 1
    ADCON0 = 0x84; // setting ADCON0 to the required mode.

}
```

```
void set_receive()
{
   //1.set RC6 as digital input
   //2.set RC2, RC3, RC4 and RC5 as digital inputs
   //TRISC = 0xFF;
    ANSELB = 0x00;
   TRISBbits.TRISB0=1;
   TRISBbits.TRISB1=1;
   TRISBbits.TRISB2=1;
   TRISBbits.TRISB3=1;
   TRISBbits.TRISB4=1;
}

void set_send()
{
    ANSELB = 0x00;
    TRISBbits.TRISB0= 0;
    TRISBbits.TRISB1= 0;
    TRISBbits.TRISB2= 0;
    TRISBbits.TRISB3= 0;
}

unsigned char receive_msg()
{
    /* 1.wait strobe high
    2.wait strobe low
    3.read the data
    4.wait strobe high
    5.return the data*/
    set_receive();
    while(PORTBbits.RB4 == 0);
    unsigned char message = 0x00;
     message = ((PORTBbits.RB0)| (PORTBbits.RB1<<1) | (PORTBbits.RB2<<2) |
(PORTBbits.RB3<<3));
    while(PORTBbits.RB4 == 1);
    return message;
}


void Strobe(char message)
{
      ANSELB = 0x00;
    TRISBbits.TRISB0=0;
    TRISBbits.TRISB1=0;
    TRISBbits.TRISB2=0;
    TRISBbits.TRISB3=0;
    TRISBbits.TRISB4 = 1;
    /*PORTBbits.RB0 = 1;
    PORTBbits.RB1 = 1;
    PORTBbits.RB2 = 1;
    PORTBbits.RB3 = 1;
      */
    while(PORTBbits.RB4==1);
    LATBbits.LATB0 = message & 0x01;
    LATBbits.LATB1 = (message>>1)&0x01;
    LATBbits.LATB2 = (message>>2)&0x01;
    LATBbits.LATB3 = (message>>3)&0x01;
    while(PORTBbits.RB4==0);
}


void SendADC(int ADCValue)
{
```

```
    set_send();
    char a = (ADCValue & 0x0F);
    char b = (ADCValue & 0xF0)>>4;
    char c = (ADCValue & 0x300)>>8;
    Strobe(a);
    Strobe(b);
    Strobe(c);
   }
// Main program
void main (void)
{

    int ADC;
    SYSTEM_Initialize();
    ADC_Init();
    Timer2_Init();
    PWM_Init();
    TRISCbits.TRISC2=0;
    unsigned char msg;
//    ANSELC =0;
    while(1)
    {
    msg=receive_msg();
    switch(msg)
    {
        //Reset
        case 0x00:
          Strobe(MSG_ACK);
          __delay_ms(1000);
          SYSTEM_Initialize();
          break;
          //PING
        case 0x01:
            Strobe(MSG_ACK);
            break;
         //Get
        case 0x02:
            Strobe(MSG_ACK);
            ADC = ADC_conversion_results();
            SendADC(ADC);
            break;
        //TURN 30
        case 0x03:
            PWM_signal_out(100);
            for(int i=0;i<=35;i++)
            {
                PORTCbits.RC2 = 1;
                __delay_ms(3.500);
                PORTCbits.RC2 = 0;
                __delay_ms(16.500);
            }
            break;
        //TURN 90
        case 0x04:
            for(int i=0;i<=35;i++)
            {
                PORTCbits.RC2 = 1;
                __delay_ms(3.500);
                PORTCbits.RC2 = 0;
                __delay_ms(16.500);
            }
            break;
        //TURN 120
        case 0x05:
```

```
            for(int i=0;i<=35;i++)
            {
                PORTCbits.RC2 = 1;
                __delay_ms(3.500);
                PORTCbits.RC2 = 0;
                __delay_ms(16.500);
            }
            break;
    }
    }
}
```

Results: