## *Section 1: General Lab Info* */0.5 points*


Course name: Microprocessor Systems II and Embedded Systems

Course Number: EECE.5520

Lab title: Multithreaded Programming

Instructors names: Yan Luo
Ioannis Smanis

Group number: 7

Student name: Poornima Manjunath

Hand in Date: 11/16/2017

Lab Due Date: 12/21/2017

## *Section 2: Contributions* */1 points*

1. Group Member 1 – Poornima Manjunath

I have done the coding part of the project. I had to come up with an algorithm to divide the tasks for each thread as per the Lab4.pdf. I read the document and divided the tasks for each thread.

I had to read about the pthread library functions. I learnt using the pthread mutexes. I had to decide which all are the shared variables in the program and implement mutex implementation for protecting them from data race conditions.

In the hardware part, we were not getting proper output when we were using the data pins and the strobe both on the same port of the PIC. I tried different ways to connect them and finally After connecting the strobe and the data pins to different ports we could get the desired output.

1. Group Member 2 – Rinkal Shah
   She handled the hardware design part of the project. She mounted the camera on the servo motor. She also helped in implementing the timestamp function to add into the third thread i.e to display the real time in the information updated to the server. She also helped in debugging the program. She also read the document and classified the functionality of each thread.

2. Group Member 3 – Akhila Nair
   She took part in coding. She implemented the code for rotating the Servo in the scanning mode. She also helped in debugging the program. She helped in interfacing the Galileo with the PIC and the gesture sensor.

## *Section 3: Purpose* */0.5 points*

In this lab, we are designing an embedded system with a variety of sensor devices such as light intensity, proximity and camera. In this lab, we are asked to send these data to a web server and keep the captured image in the Galileo. We are designing a multithreaded client-sensor software program that communicates with connected sensors and a web server. The client-sensor application is a multithreaded C application that handles user inputs, gathers sensors data, triggers a camera and sends data to a remote server.

This lab helps us to

- Learn the designing of multithreaded programs for embedded multicore systems.
- Understand the concepts of synchronization and mutual exclusion
- Understand the POSIX thread libraries.
- Implement and gain knowledge of HTTP protocol and libcurl library.
- Encourages to implement image processing functions.

## *Section 4: Introduction* */0.5 points*

In this lab, we are integrating the functionalities of the PIC based light sensor, Gesture Sensor - APDS-9960 and the USB web cam. The webcam is mounted on the servo motor and it is moving slowly back and forth (scanning mode). When the sensor reaches the predefined (by the user) threshold, the servo freezes and then the camera captures an image. So, this device is like a surveillance camera along with the light intensity measurement. We are building an intelligent client-sensor application that handles user inputs, gathers sensors data, triggers a camera and sends data to a remote server.

This IoT device which can be used for various applications like smart cctv, traffic cams etc. This will allow for combined systems integrating previously disparate devices such as surveillance cameras, proximity detector and light sensor into a common management console providing a ‘single pane of glass’ overview. The result is a huge opportunity for security solutions that are purpose-built to share useful data with other connected devices, all of which can be monitored remotely. This connectivity between devices will provide end users with more complete situational awareness across multiple locations.

Cloud-based computing has touched just about every industry and it will continue to reshape the security and surveillance sector as well. Security can now be offered as a service that is managed remotely, freeing up valuable human and capital resources that no longer need to be on site at every location that requires monitoring. Secure remote access to security systems will increase in use, including by end users who want the convenience and real-time benefits of being able to monitor property and events without having to be physically present.

Cloud storage is another important aspect of how systems are becoming more efficient in this model. Much larger volumes of data can be stored, cost-effectively and securely, at dedicated server facilities, allowing users to archive video and associated data for longer periods of time and improve its accessibility as well.

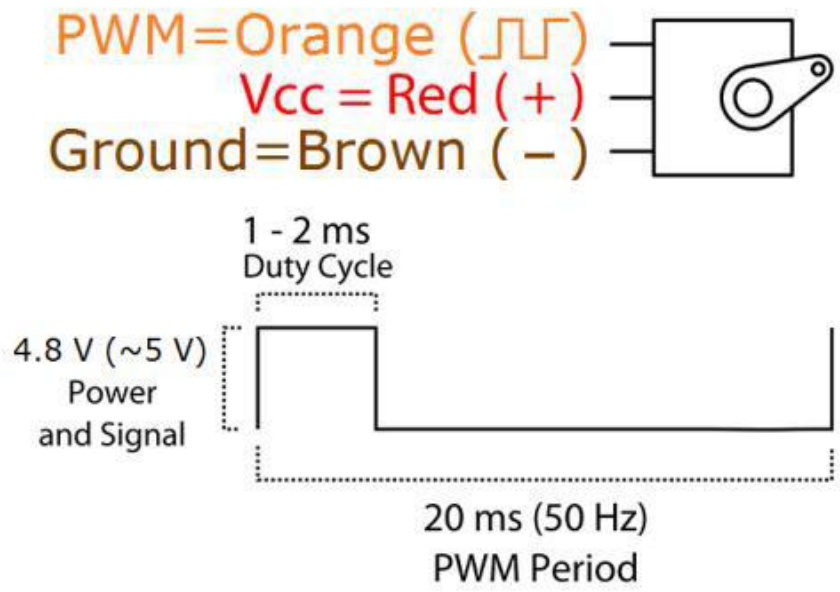
## *Section 5: Materials, Devices and Instruments /0.5 points*

**Materials:**

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

**Devices**:

- PIC16F18577
  - 28 Pin IC
  - Operating voltage Range: - 1.8V to 3.6V (PIC16LF18857/77) - 2.3V to 5.5V (PIC16F18857/77)
  - 10 bit ADC
- Servo Motor




- Intel Galileo Gen2 board
  - Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the I/O pins. This provides support for 5V Uno shields and is the default behavior. By switching the jumper position, the voltage translation can be disabled to provide 3.3V operation at the I/O pins. Digital pins 0 to 13 (and the adjacent AREF and GND pins), Analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1)

- Micro-SD Card

This 8GB card contains the tailored Yocto Linux* image with many libraries and tools to integrate with the IoTDK.

- USB to 6-pin FTDI serial cable

The FTDI cable is a USB to Serial (TTL level) converter which allows for a simple way to connect TTL interface devices to USB. The I/O pins of this FTDI cable are configured to operate at 5V.

- Integrated Wi-Fi* plus Bluetooth* card, including a half to full height PCIe Extender

- Wi-Fi Antennas

- Gesture Sensor:  SparkFun RGB and Gesture Sensor APDS-9960

Power Supply: 3.3V MAX (Please do not exceed this limit) Interface: I2C (3.3V tolerance) Device Address: 0x39

PIN OUT: SDA: I2C Data Line (3.3V tolerance) SCL: I2C Clock Line (3.3V tolerance) VCC: 3.3V GND: Ground INT: Interrupt VL : Optional Power to IR LED (3.3V tolerance)

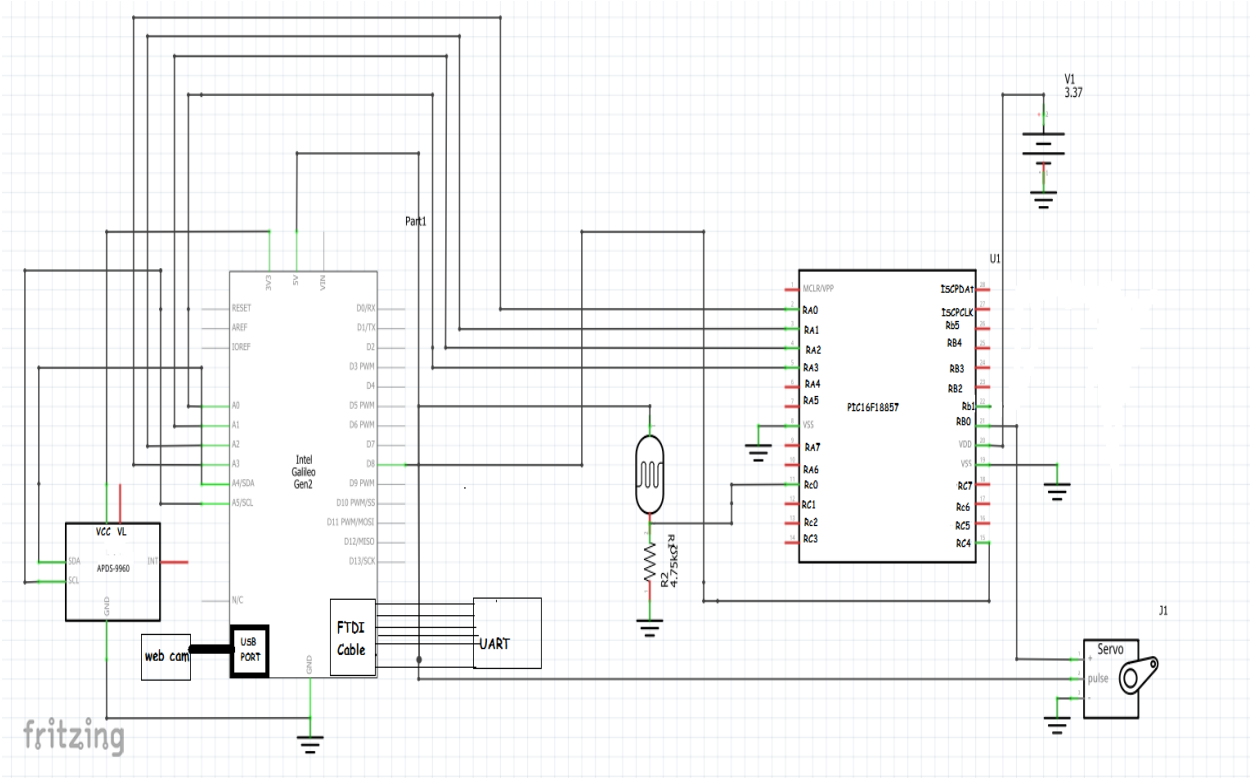
Hooking up Gesture Sensor on Galileo Board: there is no need for pull-up resistors on the I2C lines.Gesture detection range of 4 to 8 inches (10 to 20 cm).

- USB 2.0 Web-Camera - 5.0 MegaPixel

**Instruments and Software**:

1. PICkit-3
2. Power Supply – 5v
3. Laptop
4. MPLAB X IDE- V4.01
5. PuTTY
6. 12V regulated DC security power adapter

## *Section 6: Schematics* */0.5 points*



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

**Hardware design:**
The basic principle of this project is to interface light sensor, gesture/proximity sensor and also the webcam to our embedded system i.e Intel Galileo.

We use the Gesture sensor to detect the proximity and use it to trigger the webcam. We define a threshold and check if the sensor data exceed the threshold. If so, capture images and save them to the file system. There is no LDR threshold set anymore.

**Connecting the PIC to the Galileo**:

The PIC microcontroller on the sensor device is connected to the embedded computer through the GPIO port. Over the GPIO port, five signal lines (D3-D0 and Strobe) are available. The data bus (D3-D0) is 4-bit. The control signal is Strobe driven by the Galileo board to synchronize the information exchange between the sensor device and the embedded system.

We are using five GPIO pins on the Intel Galileo board. Shield pins IO14, IO15, IO16, IO17 i.e (A0-A3) are used as data lines. Shield pin IO8 is used as Strobe i.e the control signal. The strobe pin i.e IO8 from the Galileo is connected to the RC4 of the PIC and the data pins(A0-A3) are connected to the PORTA I/O pins (RA3-RA0) of the PIC. PIC monitors the I/O pin RC4 to see the changes in the strobe and monitors the I/O pins (RA3-RA0) to see the changes in the data lines.

**PIC and LDR Sensor Connections**:

First, connect the LDR to the analog input pin (RC0 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 a 4.75K resistor is also connected to the same leg and other end of the resistor is grounded. Like resistors, LDRs are not polarized. You can insert its pins without regard to polarity.

**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 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.We are supplying VCC(+5v) from the 5v pin of Galileo board.

**Connecting Gesture sensor to the Intel Galileo:**

Gesture sensor APDS9960 has the Pin configuration as below:

SDA: I2C Data Line (3.3V tolerance)

SCL: I2C Clock Line (3.3V tolerance)

VCC: 3.3V

GND: Ground

INT: Interrupt

VL : Optional Power to IR LED (3.3V tolerance)

VCC pin is connected to 3.3 v pin of Intel Galileo board and GND is connected to the Ground pin of the Intel Galileo.

SDA and SCL pins of the sensor should be connected to A4 (SDA) and A5 (SCL) of Galileo's expansion I/O ports. There is no need to wire the pull-up resistors or enable pull-up resistors on Galileo Board for the I2C bus since the sensor breakout boards already have them.

**Connecting webcam to the Intel Galileo**:

Connect the web cam to the USB port of the Intel Galileo board.

**Connecting the FTDI cable to the Intel Galileo:**

FTDI cable has the below pinout:

Green Wire: RTS (3.3V tolerance)
Yellow Wire: RX Data Line (3.3V tolerance)
Orange Wire: TX Data Line (3.3V tolerance)
Red Wire: 5V
Brown Wire: CTS (3.3V tolerance)
Black Wire: Ground

**Connecting wifi to the Intel Galileo:**

Install the wi-fi hardware on the PCI-express socket on the back of the Intel Galileo Gen2 board (Very gently connect the white and the black antenna-wires on the WiFi card). Power the wifi using the command “connmanctl enable wifi”. To connect to a network, first scan the available networks using the command “connmanctl scan wifi”. List the available networks using the command “connmanctl services”.To connect to the “eduroam” network edit the edit “eduroam.config” file applying your credentials (school email and school password) and save it , then store this file in “/var/lib/connman/“ Galileo’s directory. Save the fille and restart connman to connect to the new network.:~# systemctl restart connman

Galileo will connect to the eduroam network. In our case, sometimes Galileo used to get connected to the open network UMass Lowell. In that case, we ran connmanctl in interactive mode.To start interactive mode simply type: $ connmanctl.

Register the agent to handle user requests. The command is: connmanctl> agent on. We can now connect to the eduroam network using the command connmanctl> connect wifi_XXXX(eduroam network)

**Operating voltages**:

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

* Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the I/O pins. This provides support for 5V Uno shields and is the default behavior. By setting the jumper pins, we are providing 3.3V operation at the I/O pins.

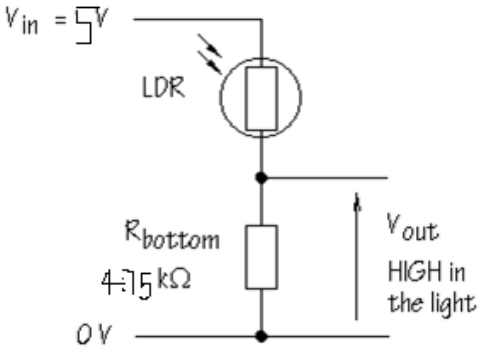
* 5v pin from the Galileo supplies VCC =+5v which is given to the Vref of LDR and VCC of the Servo motor.

* The proximity/gesture sensor is given 3.3 v as VCC.

**Circuit theory used**:

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

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:

### GPIO pin configuration on the Galileo side:

We need to configure the GPIO pins of Galileo Gen2 using the Linux commands. In this project we are using shield pin IO8 as strobe. IO14, IO15, IO16, IO17 are used for sending commands to the PIC and receiving response from the PIC. Hence, we need to configure these pins.

While sending the command the Galileo GPIO pins have to be made as output.

To set a pin as output

```
echo  xx  >   /sys/class/gpio/export                //set as GPIO
echo out  >  /sys/class/gpio/gpioxx/direction   // set as output
echo  1  >  /sys/class/gpio/gpioxx/value     // set as HIGH
echo  0  > /sys/class/gpio/gpioxx/value    //set as low
 where xx is the pin number.
```

While receiving the response the GPIO pins(IO14-IO17) are set as input pins.

To set a pin as input

```
echo  xx  >   /sys/class/gpio/export                //set as GPIO
echo in  >  /sys/class/gpio/gpioxx/direction   // set as output
echo  1  >  /sys/class/gpio/gpioxx/value     // set as HIGH
echo  0  > /sys/class/gpio/gpioxx/value    //set as low
 where xx is the pin number.
```

### Pin configuration on PIC side:

The strobe of the Galileo is connected to RC4 pin of PIC microcontroller. So it is set as an input pin using the piece of code.

ANSELCbits.ANSC4 = 0; TRISCbits.TRISC4 = 1; PORTCbits.RC4 = 0;

IO14 to IO17 data pins from the Galileo board are connected to RA3 – RA0 respectively.

Hence while receiving the command from Galileo these pins(RA3-RA0) on the PIC are set as input pins as follows:

ANSELA = 0x00; TRISA = 0x0F; PORTA = 0x0F;

And while sending the response these pins are set as output pins as follows:
ANSELA = 0x00; TRISA = 0x00; PORTA = 0x0F

**ADC configuration**:
• Port configuration

We are configuring RC0 pin of PORTC by setting TRISC = 0x01 and ANSELC= 0x01 as ADC input.

• Channel selection

We are selecting ANSC0 = 1and selecting channel 0

• ADC voltage reference selection

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

• ADC conversion clock source

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

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.

**Configuration of Gesture sensor APDS-9960:**

We are using the Gesture sensor to detect the proximity. The Proximity detection feature provides distance measurement by photodiode detection of reflected IR energy sourced by the integrated LED. We need to set up the Enable register with the address 0x80.We have to set the ENABLE<PON> (0x80<0>) and ENABLE<PEN> (0x80<2>) bits. We then have to read the proximity data from the PDATA register with the address 0x9C. It will have a higher value if an object is near to the sensor. The value decreases as the object is taken far from the sensor. In our case, we have set 30 as threshold for proximity data. If it is more than 30, web cam will be triggered to take a picture.

**Programming the web cam using OpenCV library functions**:

 Check if the camera is connected to your Galileo board: browse to the "/dev" directory and type "ls" to see the device list of your board. We should see a device called "video0"
 We need to Create pointer and initialize detected camera "0".
CvCapture *capture = cvCaptureFromCAM(CV_CAP_ANY);
CV_CAP_ANY is the constant 0 in the enumerator list defined in OpenCV library file.
 We create pointer to IplImage structure for storing image data in c style interface.
IplImage *iplImg;
 Capture single frame and store in iplImg using camera address.
iplImg = cvQueryFrame( capture );
 Save image data (iplImg) to buffer location on SD drive.
cvSaveImage(buffer, iplImg);
 De-initialize camera before exiting. cvReleaseCapture(&capture);

**Algorithm:**

**Explanation of the Galileo code:**

- We are creating three threads using the pthread_create() function. Each thread has its own set of tasks to perform.
- **Thread1:**

➢ Scans the user input commands using the scanf() function and the Galileo issues the commands to the PIC and the Gesture sensor accordingly by writing the command on the GPIO pins.

➢ It has to display the proximity data from the Gesture sensor. It follows the below steps.
1. Open the i2c device file. 2. Specify with what device address we want to communicate. 3. Write 0x05 to the Enable register with the address 0x80 to make the Power ON and Proximity Enable ON. 4. Write the Device register PDATA address(0x9c) to be accessed to the i2c bus. 5. Read the PDATA register content.

> The PDATA register(proximity data) content will be displayed.

> Any error in the PIC response to the Galileo commands or the error in dealing with Gesture sensor will be passed to the thread2.

> It also scans a new threshold for proximity sensor using scanf() function.

- **Thread2:**

- This thread monitors ambient light intensity change.
  For this we are preserving the previous ADC value in a variable called initial and we are comparing the absolute difference between the initial and the current ADC value with 1023. If it is equal to 1023, we are printing the appropriate message.
- It also checks if gesture sensor passed the threshold.
- If yes, then it will trigger the webcam. Th photo is saved with the incrementing identifier. Say Photoxx.jpg. where xx is the incremented count.
- It also checks for any error variable passed from the first thread. If yes, then it will print the error message.

- **Thread3:**

- This thread posts the ADC value, the image taken and the status of the PIC for every two seconds.
- HTTP protocol and libcurl library are used for sending the information to the remote server.

**Pthread library- APIs and mutexes used:**

**Pthread_create():**The **pthread_create**() function starts a new thread in the calling process.We have used it in the main() function to create three threads.

**Pthread_join():**The pthread_join() function shall suspend execution of the calling thread until the target thread terminates, unless the target thread has already terminated.

**Pthread_exit():** The pthread_exit() function terminates the calling thread and returns a value via *retval* that (if the thread is joinable) is available to another thread in the same process that calls pthread_join().

We are using pthread mutex to protect the shared variables from concurrent modifications, and implementing critical sections. They belong to the type pthread_mutex_t. They are initialized statically, using the constants PTHREAD_MUTEX_INITIALIZER.

**pthread_mutex_lock():** locks the given mutex. If the mutex is currently unlocked, it becomes locked and owned by the calling thread, and **pthread_mutex_lock** returns immediately. If the mutex is already locked by another thread, **pthread_mutex_lock** suspends the calling thread until the mutex is unlocked.

**Pthread_mutex_unlock():**pthread_mutex_unlock() function shall release the mutex object referenced by mutex.

**Explanation of the PIC code**:

- Servo motor will be running slowly from 0 to 90 degree and then 90 to 180 degrees.

- It will meanwhile, receive the command from the Galileo and send the response in return.
- The ADC value is sent in 3 packets of 4 bits each.

## *Section 8: Trouble Shooting /1 points*

ISSUE 1:

In thread3, while posting the information to the remote server, at first, we were unable to run the code. We were getting the error “Bad/Illegal Url format”. We got to know that we have had whitespaces and newline character in the string variable that we had used as part of the character buffer used for HTTP POST function. When we corrected it after removing the whitespace and newline character, it worked fine.

ISSUE 2:

At one point of time while working with the Lab4, the servo motor stopped working. We checked if the VCC given is fine, the delay functions are called properly or not. Everything was just fine. Yet it was not working. Then when we checked the servo motor, we got to know that the servo motor know was stuck, it had become rigid and was not rotating. We made it loosen using another knob, and it started rotating again.

ISSUE 3:

When we tried to run our client – server communication program, we had to face many issues, like we were running it when the server was too busy, or down. We used to think there is some problem in the code. But it got resolved when the server was made available.

Also,we were not knowing how to post the time in EST format. We had to do research about it. And when we added the below piece of code, we got the EST time.

```
// explicitly set the Timezone variable
    setenv("TZ", "EST5EDT", 1);
    tzset();
```

ISSUE 4:

When we tried to get the ADC values from the PIC, initially we were not getting precise values. We noticed that the jumper cables we were using from many days had been corroded. We replaced them. We also made the LDR which was placed at the same place on the breadboard from Lab1, to be placed in another spot. Then we got the ADC values properly.

ISSUE5:

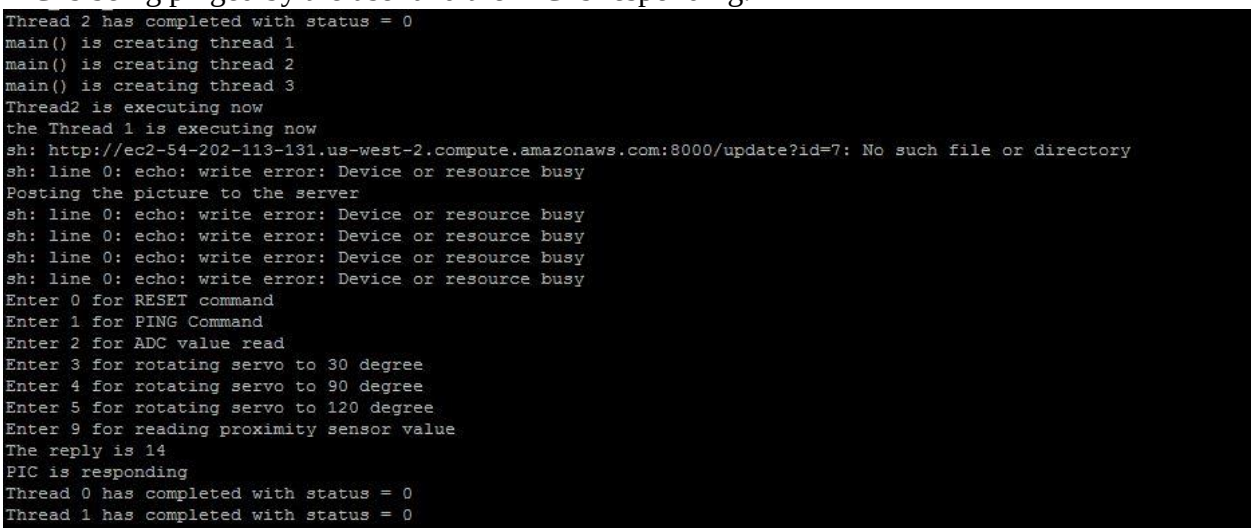
We were initially connecting the strobe and the data pins on the same port on the PIC.We were sometimes getting imprecise responses from the PIC. We then changed the strobe to RC4 and the data pins to Port A. We thus resolved this issue.

## *Section 9: Results* */0.5 points*

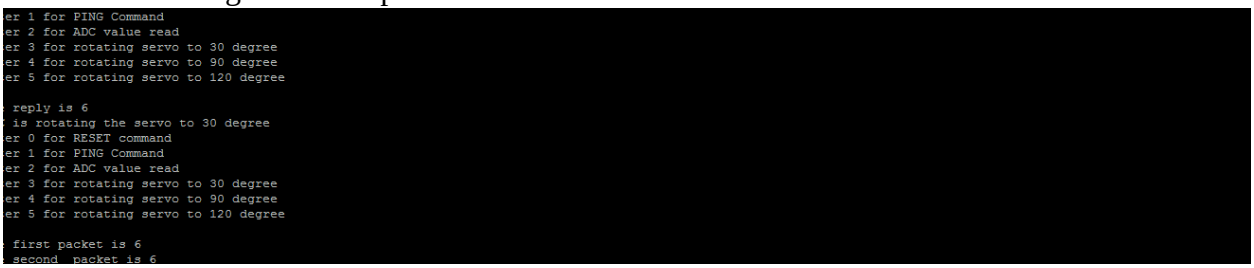
- Compiling and running of the program. The PIC is being reset by the user.

```
root@galileo:~# g++ -I/usr/local/include/opencv -I/usr/local/include/opencv2 -L/
usr/local/lib/  thread3_http.cpp -o lab4.o -lcurl  -lpthread -lopencv_core -lope
ncv_imgproc -lopencv_highgui -lopencv_ml -lopencv_video -lopencv_features2d -lop
encv_calib3d -lopencv_objdetect -lopencv_contrib -lopencv_legacy -lopencv_stitch
ing
root@galileo:~# ./lab4.o
main() is creating thread 1
main() is creating thread 2
main() is creating thread 3
Thread2 is executing now
the Thread 1 is executing now
sh: http://ec2-54-202-113-131.us-west-2.compute.amazonaws.com:8000/update?id=7:
No such file or directory
Posting the picture to the server
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
Enter 9 for reading proximity sensor value
0
The reply is 15
PIC is reset
Thread 0 has completed with status = 0
Thread 1 has completed with status = 0
```

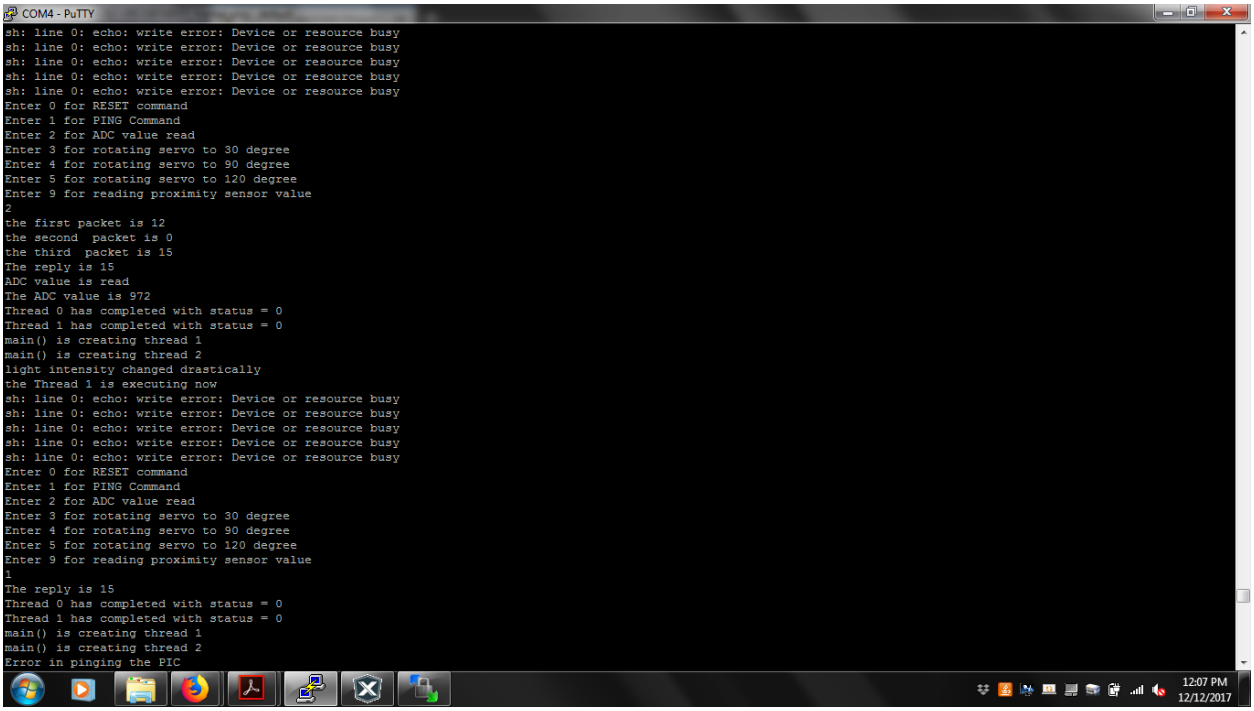
- PIC is being pinged by the user and the PIC is responding.



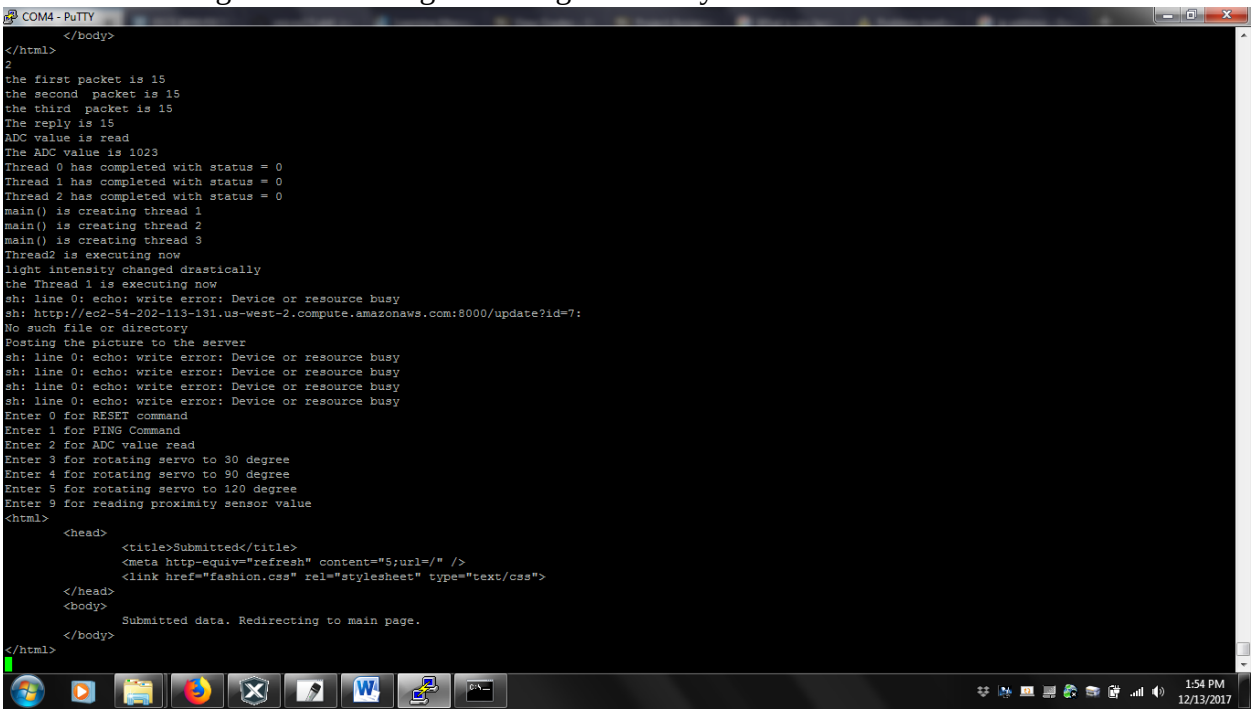

- Servo motor being rotated as per the command.




- Different ADC values being displayed by the Galileo which was received from the PIC based sensor device.




- Galileo observing dramatic change in the light intensity.



- The information being sent to the amazon server.


With the author's name : Poornima, ADC value being 123, Status as alive as there is no error .Time being the time of uploading and the photo with the identifier photo1.jpg.

## *Section 10: Appendix*

Galileo code:

```
/* http post of the image and adc value*/

#include<stdio.h>
#include<stdlib.h>
#include<curl/curl.h>
#include<sys/stat.h>
#include<time.h>

#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <pthread.h>
#include<math.h>
//*** Opencv libraries ***
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv/cv.h>
#include <opencv/highgui.h>

#define MSG_RESET  0x00
#define MSG_PING   0x01
#define MSG_GET    0x02
#define MSG_TURN3  0x03
#define MSG_TURN9  0x04
#define MSG_TURN12 0x05
#define NUM_THREADS 6
int initial = 0;
int light_intensity; //light value
int proximity_value; //proximity value
int prox_threshold = 30; //proximity threshold
char photo_buffer[200]; //photo location in the memory
int trigger;  // for checking the status of the camera
int error_1 ;  //error_in_resetting
int error_2;   //error in_pinging
int error_3; //error in reading ADC
int error_4; //error in rotating servo
int error_5; //error in reading I2C bus
int error_6; //error in capturing picture
int count; //for image capture

pthread_mutex_t mutex_err1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_err2 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_err3 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_err4 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_err5 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_err6 = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_mutex_t mutex_light = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_prox_value = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_prox_threshold = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_image = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t mutex_photobuffer = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_trigger = PTHREAD_MUTEX_INITIALIZER;

void HTTP_POST(const char* url, const char* image, int size){
        CURL *curl;
        CURLcode res;

        curl = curl_easy_init();
        if(curl){
                curl_easy_setopt(curl, CURLOPT_URL, url);
         curl_easy_setopt(curl, CURLOPT_HTTPGET, 1);
                //curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE,(long) size);
         //curl_easy_setopt(curl, CURLOPT_POSTFIELDS, image);
                res = curl_easy_perform(curl);
                if(res != CURLE_OK)
                        fprintf(stderr, "curl_easy_perform() failed: %s\n",
                        curl_easy_strerror(res));
                curl_easy_cleanup(curl);
        }
}

void initialise()
  {
        char buffer[200];
        sprintf(buffer, "echo 40 > /sys/class/gpio/export");
        system(buffer);
        sprintf(buffer, "echo 48 > /sys/class/gpio/export");
        system(buffer);
        sprintf(buffer, "echo 50 > /sys/class/gpio/export");
        system(buffer);
        sprintf(buffer, "echo 52 > /sys/class/gpio/export");
        system(buffer);
        sprintf(buffer, "echo 54 > /sys/class/gpio/export");
        system(buffer);

  }
  void GPIO_reinit()
  {
     char buf[200]={0};
```

```
   sprintf(buf, "echo in > /sys/class/gpio/gpio48/direction");
   system(buf);
   sprintf(buf, "echo in > /sys/class/gpio/gpio50/direction");
   system(buf);

   sprintf(buf, "echo in > /sys/class/gpio/gpio52/direction");
   system(buf);
   sprintf(buf, "echo in > /sys/class/gpio/gpio54/direction");
   system(buf);

}
void GPIO_init()
{
   char buf[200];
   sprintf(buf, "echo out > /sys/class/gpio/gpio48/direction");
   system(buf);

   sprintf(buf, "echo out > /sys/class/gpio/gpio50/direction");
   system(buf);

   sprintf(buf, "echo out > /sys/class/gpio/gpio52/direction");
   system(buf);

   sprintf(buf, "echo out > /sys/class/gpio/gpio54/direction");
   system(buf);

}

void writeGPIO(int command)
{
        int x = command;
        int y = 0;

        char buffer[200]={0};
        //sending first lsb bit of the command to gpio54
        y = x & 0x01;
        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio54/value", y);
   system(buffer);

        //sending second lsb bit of the command to gpio52
        x = x>>1;
        y = x & 0x01;

        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio52/value", y);
```

```
    system(buffer);

    //sending third lsb bit of the command to gpio50
        x = x>>1;
        y = x & 0x01;

        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio50/value", y);
    system(buffer);

    //sending fourth lsb bit (msb) of the command to gpio54
        x = x>>1;

        y = x & 0x01;
        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio48/value", y);
    system(buffer);
    return;

  }

char readGPIO()
  {
        char data = 0;
        char buff1[200]={0};
        char buff2[200]={0};
        char buff3[200]={0};
        char buff4[200]={0};
        int val1 = 0;
        int val2 =0;
        int val3 = 0;
        int val4 = 0;

        int fd1 = -1;
        fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);
        if(fd1!=-1)
        {
                read(fd1,buff1,1);
                close(fd1);
        }
        if(buff1[0]=='0')
        {
                val1 = 0;
        }
        else
        {
```

```
        val1 = 1;
}

int fd2 = -1;
fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);
if(fd2!=-1)
{
        read(fd2,buff2,1);
        close(fd2);
}
if(buff2[0]=='0')
{
        val2 = 0;
}
else
{
        val2 = 1;
}
int fd3 = -1;
fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);
if(fd3!=-1)
{
        read(fd3,buff3,1);
        close(fd3);
}
if(buff3[0]=='0')
{
        val3 = 0;
}
else
{
        val3 = 1;
}

int fd4 = -1;
fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);
if(fd4!=-1)
{
        read(fd4,buff4,1);
        close(fd4);
}
if(buff4[0]=='0')
{
        val4 = 0;
```

```
	}
	else
	{
		val4 = 1;
	}


	data= (pow(2,0)*val4)+(pow(2,1)*val3)+(pow(2,2)*val2)+(pow(2,3)*val1);

	return data;
}
unsigned int readADC1()
{
	unsigned int packet = 0;
	char buff1[200]={0};
	char buff2[200]={0};
	char buff3[200]={0};
	char buff4[200]={0};
	int val1 = 0;
	int val2 = 0;
	int val3 = 0;
	int val4 = 0;

	int fd1 = -1;
	fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);
	if(fd1!=-1)
	{
		read(fd1,buff1,1);
		close(fd1);
	}
	if(buff1[0]=='0')
	{
		val1 = 0;
	}
	else
	{
		val1 = 1;
	}

	int fd2 = -1;
	fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);
	if(fd2!=-1)
	{
		read(fd2,buff2,1);
```

```
        close(fd2);
}
if(buff2[0]=='0')
{
        val2 = 0;
}
else
{
        val2 = 1;
}
int fd3 = -1;
fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);
if(fd3!=-1)
{
        read(fd3,buff3,1);
        close(fd3);
}
if(buff3[0]=='0')
{
        val3 = 0;
}
else
{
        val3 = 1;
}

int fd4 = -1;
fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);
if(fd4!=-1)
{
        read(fd4,buff4,1);
        close(fd4);
}
if(buff4[0]=='0')
{
        val4 = 0;
}
else
{
        val4 = 1;
}

packet= (val4<<3)|(val3<<2)|(val2<<1)|(val1);
return packet;
```

```
   }
void *Thread1(void *t)
{

      printf("the Thread 1 is executing now\n");
      int file;
            int adapter_nr = 0;
            char filename[20];


      char buffer[200]={0};
          int reply =0 ;
          int command;
          unsigned int packet1 = 0;
          unsigned int packet2 = 0;
          unsigned int packet3 = 0;
          unsigned int packet4 =0 ;
          unsigned int ADC_value = 0;
          initialise();
          sprintf(buffer, "echo out > /sys/class/gpio/gpio40/direction");
      system(buffer);

     //while(1)
     //{
          printf("Enter 0 for RESET command\n");
          printf("Enter 1 for PING Command\n");
          printf("Enter 2 for ADC value read\n");
          printf("Enter 3 for rotating servo to 30 degree\n");
          printf("Enter 4 for rotating servo to 90 degree\n");
          printf("Enter 5 for rotating servo to 120 degree\n");
      printf("Enter 9 for reading proximity sensor value\n");

          scanf("%d",&command);

          if(command == 1)
          {

              GPIO_init();   //initialise the gpio pins as output and then send the command
              //1.Strobe high
              sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                                  system(buffer);

              //2.write data
              writeGPIO(MSG_PING);   //ping the pic micro
```

```
        sleep(3);

    GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
    //3.Strobe low
    sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                        system(buffer);
    sleep(0.5);
                        //2.read data
    reply = readGPIO();
    printf("The reply is %d\n",reply);
        if(reply!= 14)
    {
            pthread_mutex_lock( &mutex_err2);
      error_2 = 1;
      pthread_mutex_unlock( &mutex_err2 );

                        }

                        else
                        {

                                printf("PIC is responding\n");
    }
     sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                         system(buffer);
}

else if(command == MSG_RESET)
{


GPIO_init();   //initialise the gpio pins as output and then send the command
    //1.Strobe high
 sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                    system(buffer);
    //2.write data
 writeGPIO(0x00);   //reset the pic micro
 sleep(3);
 GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
 //3.Strobe low
 sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);
 sleep(0.5);
```

```
                    //2.read data
 reply = readGPIO();
 printf("The reply is %d\n",reply);
 if(reply!= 15)
 {
          pthread_mutex_lock( &mutex_err1 );
    error_1 = 1;
    pthread_mutex_unlock( &mutex_err1 );
                    }

                    else
                      {

                              printf("PIC is reset\n");
                              }


                              //3.Strobe low
 sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);

  }

else if(command == 3)
          {

    GPIO_init();   //initialise the gpio pins as output and then send the command
    //1.Strobe high
    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                      system(buffer);

    //2.write data
    writeGPIO(MSG_TURN3);   //rotating  the servo to 30 degree
    sleep(3);
    GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
    //3.Strobe low
    sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                      system(buffer);
    sleep(0.5);

                      //2.read data
    reply = readGPIO();
    printf("The reply is %d\n",reply);
```

```
    if(reply!= 0x06)
    {
            pthread_mutex_lock( &mutex_err4 );
     error_4 = 1;
     pthread_mutex_unlock( &mutex_err4 );

                        }

                        else
                        {

                                printf("PIC is rotating the servo to 30 degree\n");
                                }


                                //3.Strobe low
     sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                         system(buffer);

}
else if(command == 4)
          {

   GPIO_init();   //initialise the gpio pins as output and then send the command
   /* 1.Strobe high*/
    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                        system(buffer);


    /*2.write data*/
    writeGPIO(MSG_TURN9);   //rotating  the servo to 90 degree
    sleep(3);
    GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
    /*3.Strobe low*/
    sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                        system(buffer);
    sleep(0.5);


                        //2.read data
    reply = readGPIO();
    printf("The reply is %d\n",reply);
    if(reply!= 0x07)
    {
```

```
        pthread_mutex_lock( &mutex_err4 );
    error_4 = 1;
    pthread_mutex_unlock( &mutex_err4 );

   }
                    else
                    {
                            printf("PIC is rotating the servo to 90 degree\n");
                            }

                            //3.Strobe low
    sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                     system(buffer);

}
else if(command == 5)
        {

   GPIO_init();   //initialise the gpio pins as output and then send the command
   /*1.Strobe high*/
   sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                    system(buffer);


   /*2.write data*/
   writeGPIO(MSG_TURN12);   //rotating  the servo to 120 degree
   sleep(3);
   GPIO_reinit();  // initialise the gpio pins as input pins and then read the response
  /* 3.Strobe low*/
   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);
   sleep(0.5);

                   /* 2.read data*/
   reply = readGPIO();
   printf("The reply is %d\n",reply);
   if(reply!= 8)
   {
        pthread_mutex_lock( &mutex_err4 );
    error_4 = 1;
    pthread_mutex_unlock( &mutex_err4 );
   }

                    else
```

```
                        {

                                printf("PIC is rotating the servo to 120 degree\n");
                                }


                                /*3.Strobe low*/
    sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                         system(buffer);

}

else if(command == 0x02)
{

   GPIO_init();  //initialise the gpio pins as output and then send the command
    /*1.Strobe high*/
 sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
      system(buffer);

   /*2.write data*/
writeGPIO(MSG_GET);  // reading  the adc


   GPIO_reinit();   //initialise the gpio pins as input pins and then read the response


       sleep(5);
       /*3.Stirobe low*/

   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
       system(buffer);
   sleep(0.5);
                      /* 2.read data*/
   packet1  = readADC1();

   printf("the first packet is %d\n",packet1);

   sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
       system(buffer);
   sleep(1);
   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
       system(buffer);
           sleep(0.5);
```

```
            packet2  = readADC1();

        printf("the second  packet is %d\n",packet2);


        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                system(buffer);
        sleep(1);
        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
            system(buffer);
        sleep(0.5);
                packet3  = readADC1();
        printf("the third  packet is %d\n",packet3);
        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
            system(buffer);

        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
            system(buffer);
        sleep(0.5);

                            /*2.read data*/
        reply = readGPIO();

        printf("The reply is %d\n",reply);


                    printf("ADC value is read\n");


          ADC_value = (packet3<<6)|(packet2<<4)|packet1; //combine the three packets
          pthread_mutex_lock( &mutex_light );
      light_intensity = ADC_value;

      pthread_mutex_unlock( &mutex_light );
            printf("The ADC value is %d\n", ADC_value);

      sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
            system(buffer);


     }
else if(command == 0x09)
{
   snprintf(filename,19, "/dev/i2c-%d",adapter_nr);
```

```
            file = open(filename,O_RDWR);

                        if(file<0)
   {
                    printf("cannot open file\n");

          }
                        int addr = 0x39;
                        if(ioctl(file,I2C_SLAVE,addr)<0)
                {
                        printf("error");

                }

                char buf[10];

// printf("error 1\n");
 sprintf(buffer,"i2cset -y -ff 0 0x39 0x80 0x05");
 system(buffer);
// printf("error 2\n");


      buf[0] = 0x9c;

      if(write(file,buf,1)!=1)
{
           printf("write failed\n");
      }


if(read(file,buf,1)!= 1)
      {
                 pthread_mutex_lock( &mutex_err5 );
  error_5 = 1;
  pthread_mutex_unlock( &mutex_err5 );
      }
else
      {
 pthread_mutex_lock( &mutex_prox_value );
                proximity_value = buf[0];
 printf("the proxmity data reg is %d\n",proximity_value);
 pthread_mutex_unlock( &mutex_prox_value );
}
```

```
        printf("Set the proximity threshold value\n");
        pthread_mutex_lock( &mutex_prox_threshold );
        scanf("%d",&prox_threshold);
       pthread_mutex_unlock( &mutex_prox_threshold );
    }


     //}


   pthread_exit(NULL);
}

void *Thread2(void *t)
{
   int light_buf;

   int diff = 0;
   printf("Thread2 is executing now\n");
   pthread_mutex_lock( &mutex_err1 );
   if(error_1!=0)

   {
      printf("Error in resetting the PIC\n");
           error_1 = 0;

   }
   pthread_mutex_unlock( &mutex_err1 );

   pthread_mutex_lock( &mutex_err2 );
   if(error_2!=0)
   {
      printf("Error in pinging the PIC\n");
           error_2 =0;

   }
   pthread_mutex_unlock( &mutex_err2 );

   pthread_mutex_lock( &mutex_err3 );
   if(error_3!=0)
   {
      printf("Error in reading ADC value from the PIC\n");
           error_3 = 0;
```

```
   }
   pthread_mutex_unlock( &mutex_err3 );

   pthread_mutex_lock( &mutex_err4 );
   if(error_4!=0)
   {
      printf("Error in rotating the servo\n");
           error_4 =0;

   }
   pthread_mutex_unlock( &mutex_err4 );

   pthread_mutex_lock( &mutex_err5 );
   if(error_5!=0)
   {
      printf("Error in reading I2C bus\n");
           error_5 =0;

   }
   pthread_mutex_unlock( &mutex_err5 );

   pthread_mutex_lock( &mutex_light);

      light_buf = light_intensity;
      diff = abs(light_buf- initial);
      initial = light_buf;

      if(diff>900)
      {
         printf("light intensity changed drastically\n");

      }

   pthread_mutex_unlock( &mutex_light);

   pthread_mutex_lock(&mutex_prox_value);

   pthread_mutex_lock(&mutex_prox_threshold);

   if(proximity_value>=prox_threshold)
   {
      printf("Taking a new picture\n");
       sprintf(photo_buffer,"photo%d.jpg",count);//filling buffer
//Create pointer and initialize detected camera "0"
```

```
        CvCapture *capture = cvCaptureFromCAM(CV_CAP_ANY);
        printf("\n!!!!Smile Please!!!!\n");//Tell everyone to Smile
        if (capture == NULL)//Error to detect if unable to open camera
        {
           pthread_mutex_lock(&mutex_trigger);
           trigger = 0;
           pthread_mutex_unlock(&mutex_trigger);
        }
        else{//Indicate if camera was initialized
        pthread_mutex_lock(&mutex_trigger);
        trigger = 1;
        pthread_mutex_unlock(&mutex_trigger);

        pthread_mutex_lock(&mutex_photobuffer);
        printf("Camera is Taking the Picture\n");

//Create pointer to IplImage structure for storing image data in c style interface
         IplImage *iplImg;
         iplImg = cvQueryFrame( capture );//Capture single frame and store in iplImg using camera address

         cvSaveImage(photo_buffer, iplImg);//Save image data (iplImg) to buffer location on SD drive



         cvReleaseCapture(&capture);//De-initialize camera before exiting
          pthread_mutex_unlock(&mutex_photobuffer);
                printf("\nDone!!\n");
                count++;
      }

    }

    pthread_mutex_unlock(&mutex_prox_value);

    pthread_mutex_unlock(&mutex_prox_threshold);

    pthread_exit(NULL);

}

void *Thread3(void *t)
{
     // explicitly set the Timezone variable
     setenv("TZ", "EST5EDT", 1);
```

```
    tzset();

    time_t rawtime;
    struct tm * timeinfo;
    char timestring[120];

    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    strftime(timestring, 120, "%Y_%m_%d_%H:%M:%S",timeinfo);


          const char* hostname="ec2-54-202-113-131.us-west-2.compute.amazonaws.com"; // Server
Hostname or IP address
          const int   port= 8000;              // Server Service Port Number
          const int   id=7;
          const char* password="password";
          const char* name="Poornima";


      int   adcval;
          const char* status="Alive";
      char* timestamp = NULL;

          timestamp = (char*)malloc(120);
   strftime(timestamp, 120, "%Y_%m_%d_%H:%M:%S",timeinfo);

          char *filename= NULL;  // captured picture name + incremented file number
           filename= (char*) calloc(1,200);

          pthread_mutex_lock(&mutex_trigger);
   if(trigger == 1)
          {
                    pthread_mutex_lock(&mutex_photobuffer);
                    strncpy(filename,photo_buffer,strlen(photo_buffer));
                    pthread_mutex_unlock(&mutex_photobuffer);
           }

          {
                    strcpy(filename,"No image captured yet\n");
          }
    pthread_mutex_unlock(&mutex_trigger);

   pthread_mutex_lock( &mutex_light );
   adcval = light_intensity;
```

```
   pthread_mutex_unlock( &mutex_light );
   char http_buf[1024];
   //.............
    // use sprintf() call here to fill out the data "buf":
            // use the provided URL Protocol in the lab description: replace the "server_hostname",
"portnumber", "var_xxxx" with the related format specifiers "%d" or "%s"
    //.....................

            snprintf(http_buf,1024,"http://%s:%d/update?id=%d&password=%s&name=%s&data=%d&stat
us=%s&timestamp=%s&filename=%s",hostname,port,id,password,name,adcval,status,timestamp,filena
me);
   system(http_buf);
   //printf("\n%s is the server",buf);

            // ========== Don't bother the lines below
            FILE *fp;
            struct stat num;
            stat(filename, &num);
            int size = 0;
   size = num.st_size;
            char* buffer=(char*)malloc(size);
   // ============ Don't bother the above lines
   printf("Posting the picture to the server\n");
            HTTP_POST(http_buf, buffer, size);
            //fclose(fp);

            sleep(2);
            pthread_exit(NULL);

}


int main ()
{
   int ret,ret1,ret2,ret3,i;
   pthread_t threads[NUM_THREADS];
   pthread_attr_t attr;
   void *status;


   // Initialize and set thread attributes to "joinable"
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

```
  while(1)
  {
   printf("main() is creating thread %d \n", 1);
   ret1 = pthread_create(&threads[0], NULL, Thread1, NULL );
   if (ret1)
      {
         printf("Unable to create thread %d\n", 1);
         return -1;
      }
    printf("main() is creating thread %d \n", 2);
    ret2 = pthread_create(&threads[1], NULL, Thread2, NULL );
    if (ret2)
      {
         printf("Unable to create thread %d\n",2);
         return -1;
      }

   printf("main() is creating thread %d \n",3);
   ret3 = pthread_create(&threads[2], NULL, Thread3, NULL );
   if (ret3)
    {
         printf("Unable to create thread %d\n",3);
         return -1;
   }


   // deallocate attribute and wait for the other threads
   pthread_attr_destroy(&attr);
   for( i=0; i < 3; i++ )
   {
      ret = pthread_join(threads[i], &status);
      if (ret){
         printf("Error:unable to join thread %d, return = %d\n", i, ret);
         return -1;
      }
      printf("Thread %d has completed with status = %d\n", i, (int)status) ;
    }
  }
   printf("program is exiting.\n");

   pthread_exit(NULL);
}
```

- PIC CODE:

```
#include "mcc_generated_files/mcc.h" //default library
#define MSG_RESET     0x0 /* reset the sensor to initial state */
#define MSG_PING      0x1 /*check if the sensor is working properly */
#define MSG_GET       0x2 /* obtain the most recent ADC result */
#define MSG_TURN3     0x3 /* turn the servo motor blade 30 degrees */
#define MSG_TURN9     0x4 /* turn the servo motor blade 90 degrees */
#define MSG_TURN12    0x5 /* turn the servo motor blade 120 degrees */
static unsigned int light =0 ;
char P0,P1,P2,P3,P4;
void set_and_shift()
{
      P0 = (light) & 1;           //lsb of the packet
   P1 = (light >> 1) & 1;
      P2 = (light >> 2) & 1;
      P3 = (light >> 3) & 1;
   light = light >> 4;
}

 void set_pins()
 {

    TRISAbits.TRISA3 = 0;
    TRISAbits.TRISA2 = 0;
    TRISAbits.TRISA1 = 0;
    TRISAbits.TRISA0 = 0;


     if (P0==1)
  {
     PORTAbits.RA0 = 1;

  }
   else
   {
        PORTAbits.RA0 = 0;

   }
     if (P1==1)
    {
       PORTAbits.RA1 = 1;
```

```
      }
      else
      {
      PORTAbits.RA1 = 0;

       }
              if (P2==1)
       {
               PORTAbits.RA2 = 1;

                 }
                 else
                   {
                              PORTAbits.RA2 = 0;

                           }
      if (P3==1)
      {
        PORTAbits.RA3 = 1;

              }
     else
     {
        PORTAbits.RA3 = 0;

     }
     }


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 = 0x01;

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


 /* 11. ADC chanel - Analog Input */
 ANSELCbits.ANSC0 = 1;
 TRISCbits.TRISC0 = 1;
 PORTCbits.RC0 = 1;

 /* 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//

 ANSELBbits.ANSB0 = 0;
 TRISBbits.TRISB1= 0;
 PORTBbits.RB1 = 0;
}

int ADC_read()
{
   unsigned int lightval =0;
   // Initialize PIC device
   SYSTEM_Initialize();

   // Initialize the required modules
   ADC_Init();
```

```
 // **** write your code

   //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
                {

                        PORTBbits.RB1 = 0;

                         //180 Degree
                }
                else
                {

                        PORTBbits.RB1 =1;

          //90 Degree5
                }
   ADCON0bits.ADON = 0;


   light =  lightval;


}
void reinit_out()
{
   ANSELA = 0x00;
   TRISA = 0x00;
   PORTA = 0x0F;

}
void init_out()
```

```
{
   ANSELA = 0x00;
   TRISA = 0x0F;
   PORTA = 0x0F;

}
unsigned char receive_msg()
{
      unsigned char data =0;
   unsigned char data1 =0;
   unsigned char data2 =0;
   unsigned char data3 =0;
   unsigned char data4 =0;

   /*1.wait strobe high*/
 //   while(PORTAbits.RA4  != 1)
 //   {}
      /*3.read the data */
   data1 = PORTAbits.RA3 << 3;
   data2 = PORTAbits.RA2 << 2;
   data3 = PORTAbits.RA1 << 1;
   data4 = PORTAbits.RA0;
   data = ( data1 | data2 | data3 | data4);

   /*2.wait strobe low */
  //  while(PORTAbits.RA4 != 0)
 //   {}
   return data;




   /*5.return the data*/


}
void send_and_receive()
{

   static unsigned char msg = 0;
   int i =0;
   /*strobe signal*/
   ANSELCbits.ANSC4 = 0;
   TRISCbits.TRISC4 = 1;
```

```
PORTCbits.RC4 = 0;

TRISBbits.TRISB0  = 0;

PORTBbits.RB0 = 1;


   init_out();

  while(PORTCbits.RC4  != 1); //wait till the strobe goes high
   __delay_ms(200);
  msg = receive_msg();
   /*pinging the PIC*/
   if(msg == MSG_PING)    //if the command from the Galileo is PING
   {
      TRISBbits.TRISB3 = 0;
      PORTBbits.RB3 = 1;
      reinit_out();
      while(PORTCbits.RC4  != 0); //wait till the strobe goes low

      PORTAbits.RA3 = 1;   //send the response MSG_ACK
      PORTAbits.RA2 = 1;
      PORTAbits.RA1 = 1;
      PORTAbits.RA0 = 0;
      __delay_ms(1000);
   }


   else if(msg == MSG_RESET)  //if the command from the Galileo is RESET
   {

      reinit_out();
      while(PORTCbits.RC4  != 0); //wait till the strobe goes low

      PORTAbits.RA3 = 1;  //send the response MSG_NTHNG
      PORTAbits.RA2 = 1;
      PORTAbits.RA1 = 1;
      PORTAbits.RA0 = 1;
      __delay_ms(1000);
   }



   else if(msg == MSG_GET) //if the command from the Galileo is MSG_GET
```

```
{
   TRISBbits.TRISB2 =0;
   PORTBbits.RB2 =1;   //to check if the command is received or not

   reinit_out();


    ADC_read();

   while(PORTCbits.RC4 != 0);  //send the  first packet of last 4 bits when the strobe goes low
   set_and_shift();
   set_pins();


  while(PORTCbits.RC4 != 1);  //wait till strobe goes high
  if(PORTCbits.RC4 == 1)
  {}

   while(PORTCbits.RC4 != 0); //send the next 4 bits when the strobe goes low
   set_and_shift();
   set_pins();
   __delay_ms(1000);
  while(PORTCbits.RC4 != 1);  //wait till strobe goes high
  if(PORTCbits.RC4 == 1)
  {}
   __delay_ms(1000);
   while(PORTCbits.RC4 != 0); //send the next 4 bits when the strobe goes low

   set_and_shift();
   set_pins();
   __delay_ms(1000);

  while(PORTCbits.RC4 != 1);  //wait till the strobe goes high
  if(PORTCbits.RC4 == 1)
  {}

   while(PORTCbits.RC4 != 0); //wait till strobe goes low

   PORTAbits.RA3 = 1;  //send the response 0x0E
   PORTAbits.RA2 = 1;
   PORTAbits.RA1 = 1;
   PORTAbits.RA0 = 0;

   __delay_ms(1000);
```

```
    }

    else if(msg == MSG_TURN3)   //if the command from the Galileo is TURN_3 to roate servo to 30
degree
    {

       reinit_out();
       while(PORTCbits.RC4  != 0); //wait till the strobe goes low

       PORTAbits.RA3 = 0;  //send the response 0x06  to Galileo
       PORTAbits.RA2 = 1;
       PORTAbits.RA1 = 1;
       PORTAbits.RA0 = 0;

        for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree
      {
        PORTBbits.RB0 = 1;
        __delay_us(1000);
        PORTBbits.RB0 = 0;
        __delay_us(19000);
       }

    }



    else if(msg == MSG_TURN9)  //if the command from the Galileo is turn servo to 90 degree
    {

       reinit_out();
       while(PORTCbits.RC4  != 0); //wait till the strobe goes low




       PORTAbits.RA3 = 0;  //send the response 0x07
       PORTAbits.RA2 = 1;
       PORTAbits.RA1 = 1;
       PORTAbits.RA0 = 1;
```

```
          for(i=0;i<200;i++)   //rotating the servo for 4s to 90 degree
        {
          PORTBbits.RB0 = 1;
          __delay_us(1500);
          PORTBbits.RB0 = 0;
          __delay_us(18500);
        }

    }

    else if(msg == MSG_TURN12)  //rotating the servo 120 degree
    {

       reinit_out();
       while(PORTCbits.RC4  != 0); //wait till the strobe goes low

       PORTAbits.RA3 = 1;  //send the response 0x08
       PORTAbits.RA2 = 0;
       PORTAbits.RA1 = 0;
       PORTAbits.RA0 = 0;

        for(i=0;i<200;i++)   //rotating the servo for 4s to 90 degree
       {
         PORTBbits.RB0 = 1;
         __delay_us(1800);
         PORTBbits.RB0 = 0;
         __delay_us(18200);
       }

    }


    else if(msg == 0x09)
    {
       PORTCbits.RC5 = 0;
    }

  }
void Servo_PWM(int dutycycle)
{
  int i;
  if(dutycycle==0)
  {
      for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree
```

```
        {
          PORTBbits.RB0 = 1;
          __delay_us(800);
          PORTBbits.RB0 = 0;
          __delay_us(19200);
        }
    }
    if(dutycycle==90)
    {
       for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree
        {
          PORTBbits.RB0 = 1;
          __delay_us(1500);
          PORTBbits.RB0 = 0;
          __delay_us(18500);
        }
    }
    if(dutycycle==180)
    {
       for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree
        {
          PORTBbits.RB0 = 1;
          __delay_us(2200);
          PORTBbits.RB0 = 0;
          __delay_us(17800);
        }
    }

}

int main()
{

    int i=0;
    ANSELBbits.ANSB0 = 0;
    TRISBbits.TRISB0  = 0;
    PORTBbits.RB0 = 1;

    ANSELCbits.ANSC5 = 0;
    TRISCbits.TRISC5 = 0;
    PORTCbits.RC5 = 1;

    while(1)
    {
```

```
  for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree
 {
   PORTBbits.RB0 = 1;
   __delay_us(1500);
   PORTBbits.RB0 = 0;
  __delay_us(18500);
   }

   for(i=0;i<50;i++)   //rotating the servo for 4s to 30 degree
  {
   PORTBbits.RB0 = 1;
   __delay_us(2500);
   PORTBbits.RB0 = 0;
   __delay_us(17500);
   }

//while(1)
//{

 for(i=0;i<50;i++)   //rotating the servo for 4s to 30 degree
 {
        PORTBbits.RB0 = 1;
        __delay_us(1500);
        PORTBbits.RB0 = 0;
        __delay_us(18500);
 }

   send_and_receive();

  for(i=0;i<50;i++)   //rotating the servo for 4s to 30 degree
 {
        PORTBbits.RB0 = 1;
        __delay_us(1500);
        PORTBbits.RB0 = 0;
        __delay_us(18500);
 }


}

}
```