Lab 4: Multithreaded Programming and Image Processing

16.480/552 Microprocessor Design II and Embedded Systems

Instructor: *Prof. Yan Luo*

Group-I

Due:12/12/16

Submitted: 12/12/16

By,

- ***Naga Ganesh Kurapati***
- *Sayali Vaidya*
- *Zubin Pattnaik*
- *Akriti Sharan*

Table of contents

I. Contributions

Naga Ganesh Kurapati – Worked on configuring the I2C communication between the intel Galileo Gen2 and Gesture sensor APDS-9960 to work on an independent thread to support concurrency. Debugging the codes. Understand client server connectivity using HTTP connection.

Zubin Patnaik – Worked on configuring the camera to capture picture on Galileo using OpenCV. Debugging the codes. Understand client server connectivity using HTTP connection.

Sayali Vaidya –Worked on configuring the communication (using strobe) between microcontroller PIC16F18857 and the intel Galileo Gen2 to work on an independent thread to support concurrency. Understand client server connectivity using HTTP connection.

Akriti Sharan – Worked on configuring the I2C communication between the intel Galileo Gen2 and Temperature sensor TMP102. Debugging the codes. Understand client server connectivity using HTTP connection.

II. Purpose

The main purpose of this lab is to understanding the multithreading programing using PThreads. Synchronization of those threads using Mutex. Understanding usage of curl library, HTTP protocol using a client and server application. Understanding of image processing using OpenCV library.

III. Introduction

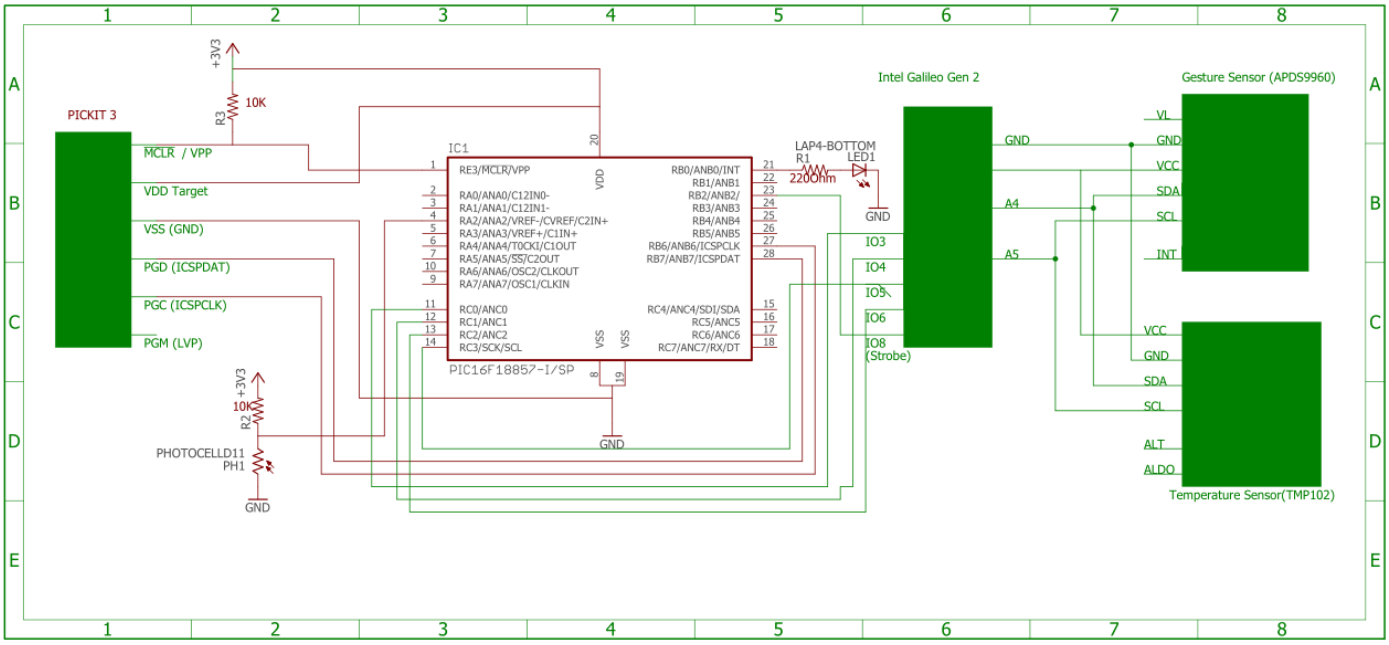
The main objective of this lab is to read the sensor data from a I2C devices Gesture sensor(APDS-9960) and Temperature sensor (TMP102). To read the sensor data (Photo resistor ADC value) from microcontroller PIC16F18857 through strobe communication. Trigger the camera to capture a picture when the required threshold value of the sensor data is reached. Processes the captured image for facial recognition using OpenCV library. And then transfer those images and sensor data to server through HTTP protocol using curl library. Make all this actions concurrent using threads using POSIX thread library.

IV. Materials, Devices and Instruments Used

1. Bread board
2. Wires to connect
3. Temperature sensor TMP102
4. Gesture sensor APDS-9960
5. two 5k Ohm resistors to connect SCL, SDA to VCC
6. Serial to USB connector
7. Multi-meter
8. Voltage supply (3.3V) from Galileo
9. Intel Galileo Gen 2 Board
10. Yocto Linux
11. Putty Software
12. PIC16F18857 microcontroller

13. Pickit3
14. Photo resistor
15. two 10k, one 220 Ohm resistors
16. MPLAB IDE
17. PThread library
18. Curl library

V. Schematics



VI. Lab Methods and Procedure

Hardware Design:

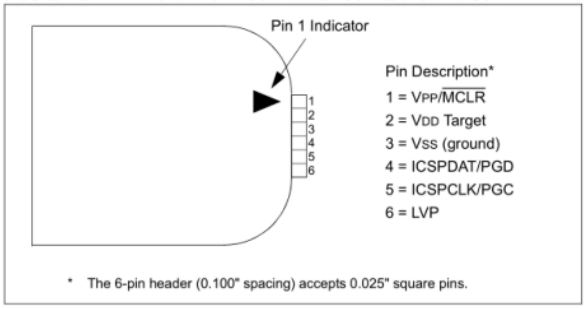
1) **I2C devices and camera:** Galileo is connected to a laptop using serial to USB connector. It is powered from the adaptor cable. I2C bus is designed on the bread board by connecting SCL, SDA pins from the Galileo board and the sensors as shown in the schematic. Those lines are made active high by connected to VCC through 5k Ohm resistors. On Galileo SCL is A5 and SDA is A4. The VCC (3.3) and ground to two sensors is supplied from the Galileo. In this I2C protocol communication Galileo is the master and the two sensors are slaves. The slave address of Gesture sensor APDS-9960 is 0x39 and Temperature sensor TMP102 is 0x48 (by connecting ADD0 to ground selects default address). After the connection, by typing “i2cdetect -r 0” shows all the I2C devices connected to the Galileo as shown in the below picture. Camera is connected to the Galileo board through the USB cable.

```
root@galileo:~#
root@galileo:~#
root@galileo:~# i2cdetect -r 0
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-0 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- UU UU UU -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- 39 -- -- -- -- -- --
40: -- -- -- -- -- -- -- UU 48 -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
root@galileo:~#
```

I2C detect on intel Galileo Gen 2

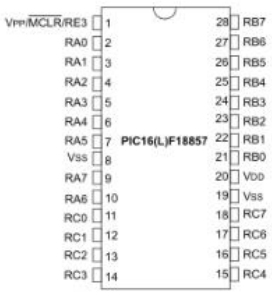
2) **PIC Microcontroller:** Initially Pickit3 is connected to the microcontroller. If you observe the pin diagram of both Pickit 3 on top and PIC. Both MCLR, Vdd, Vss, ICSPDAT/PGD, ICSPCLK/PGC are connected to each other. ICSPDAT is pin 27 and ICSPCLK is pin 28 for the PIC. The MCLR is connected to Vdd 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. Pin RB2 is connected to strobe(GPIO8) of Galileo. RC0, RC1, RC2 & RC3 pins are connected to the GPIO3,4,5,6 pins of Intel Galileo.

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.

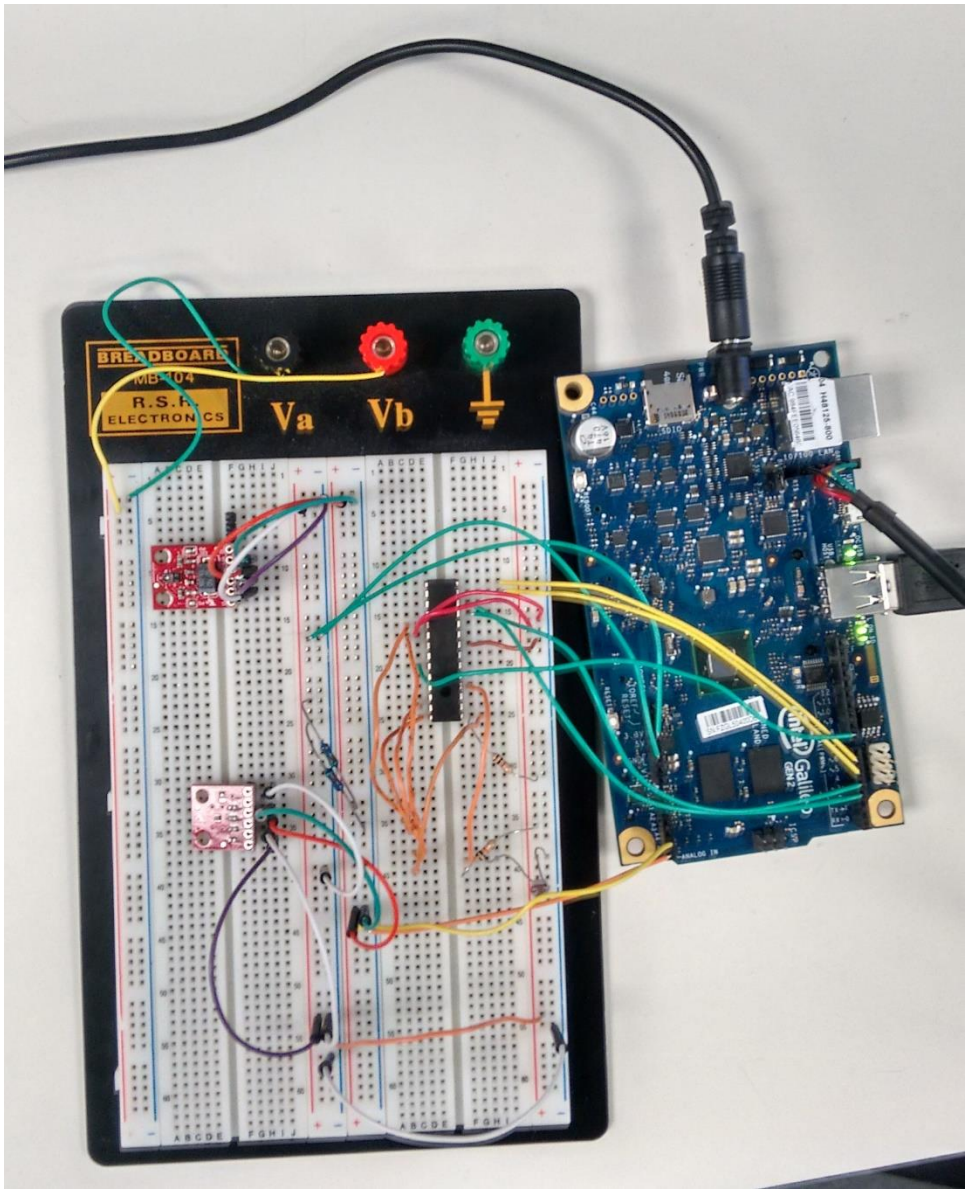
2: All VDD and all VSS pins must be connected at the circuit board level. Allowing one or more VSS or VDD pins to float may result in degraded electrical performance or non-functionality.

3) Wi-Fi connectivity: It is configured using connmanctl software, after plugging-in the Wi-Fi card to intel Galileo. Use commands from Yacto linux *connmanctl scan wifi* to scan the Wi-Fi networks, *connmanctl servies* to view the Wi-Fi networks and *connmanctl connect* $Wi-Fi-id to connect to the selected Wi-Fi network.

COM11 - PuTTY

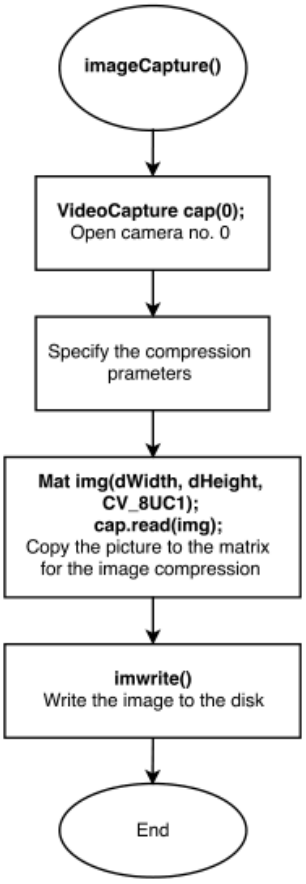
```
root@galileo:~/New folder# connmanctl scan wifi
Scan completed for wifi
root@galileo:~/New folder# connmanctl services
*AO eduroam              wifi_00216a2e1f3e_656475726f616d_managed_ieee8021x
    UMassLowell          wifi_00216a2e1f3e_554d6173734c6f77656c6c_managed_none
root@galileo:~/New folder#
```

Overall circuit view:



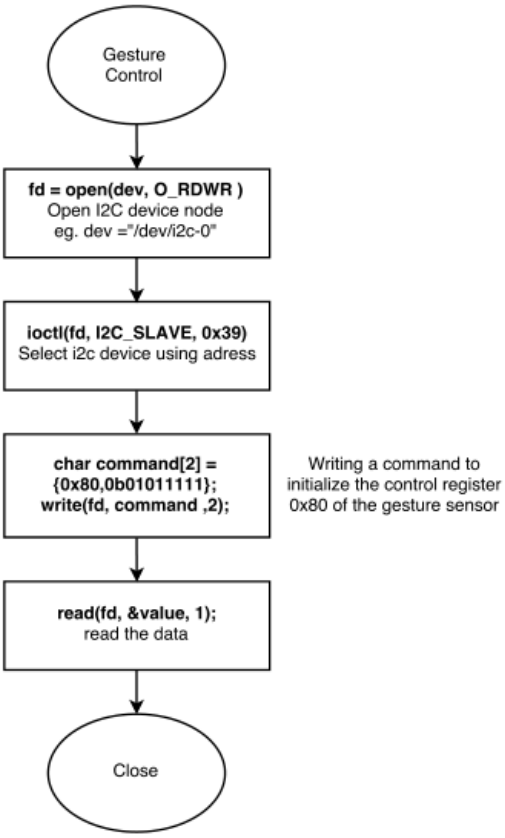
Software Design:

Flow chart for the camera:

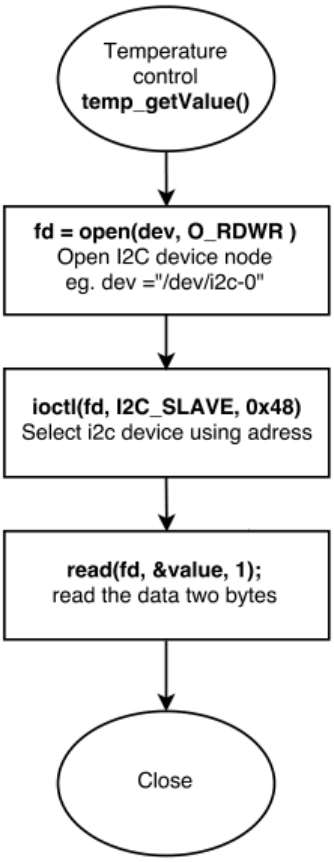


Flow Chart for the gesture:

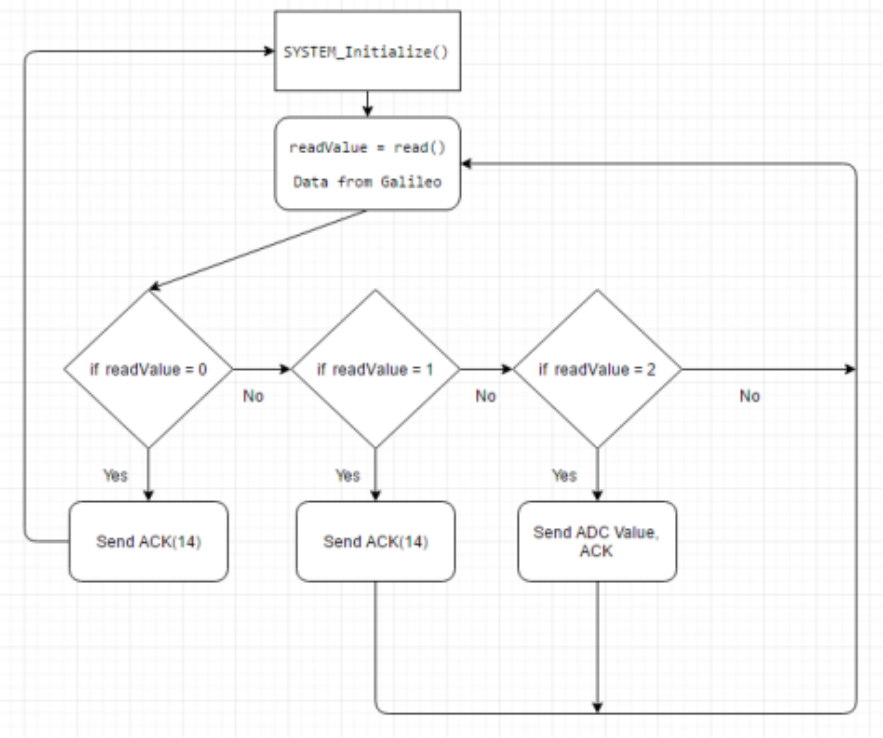
void *gesture_thread(void *arg)



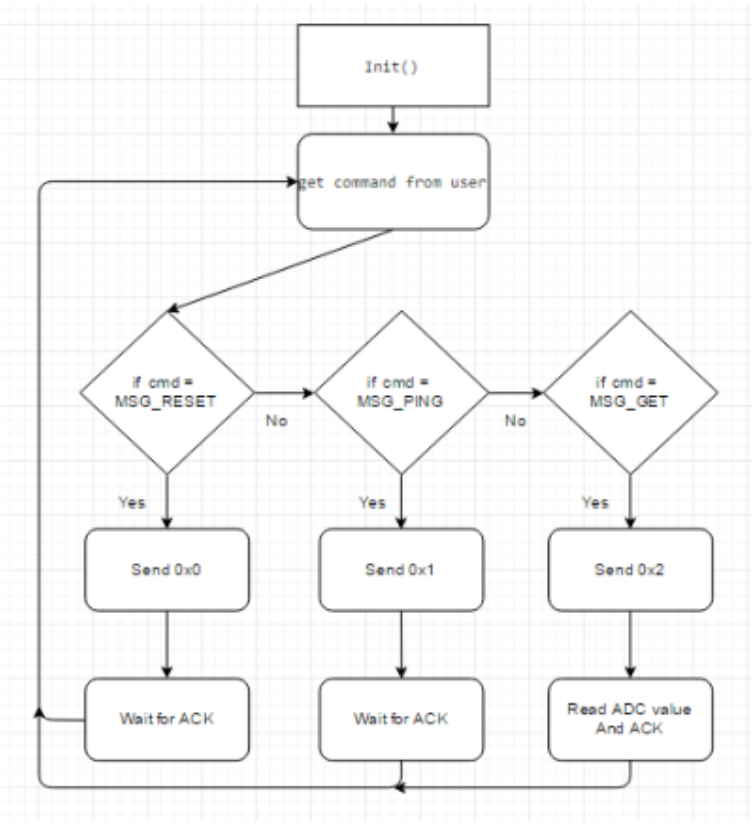
Flow chat for the temp:



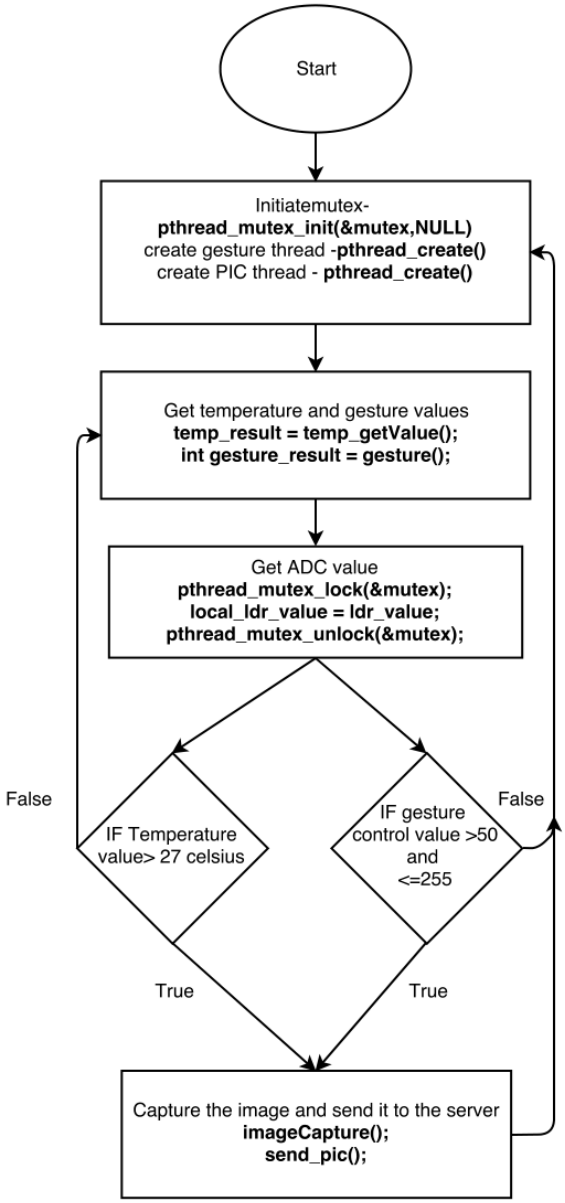
Flowchart for PIC: It is programmed using MPLAB IDE



Flowchart for PIC communication on intel Galileo: **void *ldr(void *id)**



Flowchart of the main program:



To configure camera to capture image OpenCV library is used and the sensors are configured using i2c device library of the intel Galileo. Pthreads library is used to create service threads. Curl library is used for initiating HTTP protocol.

Note: Detailed code explanation can be found in the Appendix. Code contains comments.

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 troubleshot the PORTC- data pins configurations by blinking the LED at every pin used. Third is to configure GPIO ports 3,4,5,6,8 of Intel Galileo by blinking LED at each port used. PORT configuration code can be found in the Appendix.

First make sure that supply is 3.3V at each sensor using multi-meter. Using the command "i2cdetect -r 0", make sure it display all the I2C devices connected to the I2C bus. While running the code, print the temperature on the console and check whether it is varying to the temperature change. Same with gesture sensor, move the hand and see the values varying. Also check camera is triggered at required time by observing the picture captured.

VIII. Results

```
Ack:14
ADC Result:330
Temperature: 24.062500
Gesture = 0
Send value: 2
Ack:14
ADC Result:357
Temperature: 24.062500
Gesture = 255
Send value: 2
Ack:14
ADC Result:378
Frame Size = 640x480
Send value: 2
Image size: 46029B
<html>
        <head>
                <title>Submitted</title>
                <meta http-equiv="refresh" content="5;url=/" />
                <link href="fashion.css" rel="stylesheet" type="text/css">
        </head>
        <body>
                Submitted data. Redirecting to main page.
        </body>
</html>
Image captured
```

As you in the above fig., temperature value is 24 in Celsius. The gesture value is variable from 0 to 255 based on the hand distance from the sensor. More value is read when the hand is too near. The camera is triggered if temperature value is greater than 27 or gesture value is between 50 to 255. You can see that image is captured a point where gesture value is 255. The image captured is shown below. You can also see the ADC value received from the PIC which 378 and Acknowledgement 14. Image is transferred to the server successfully. html response can be seen from the server.

ID	Group Name	Value	Status	Last Update	Image
1	Team Awesome	378	Very good thank you	20141116-09:12:34	img.jpg

Server status, you can see the ADC value same as above.

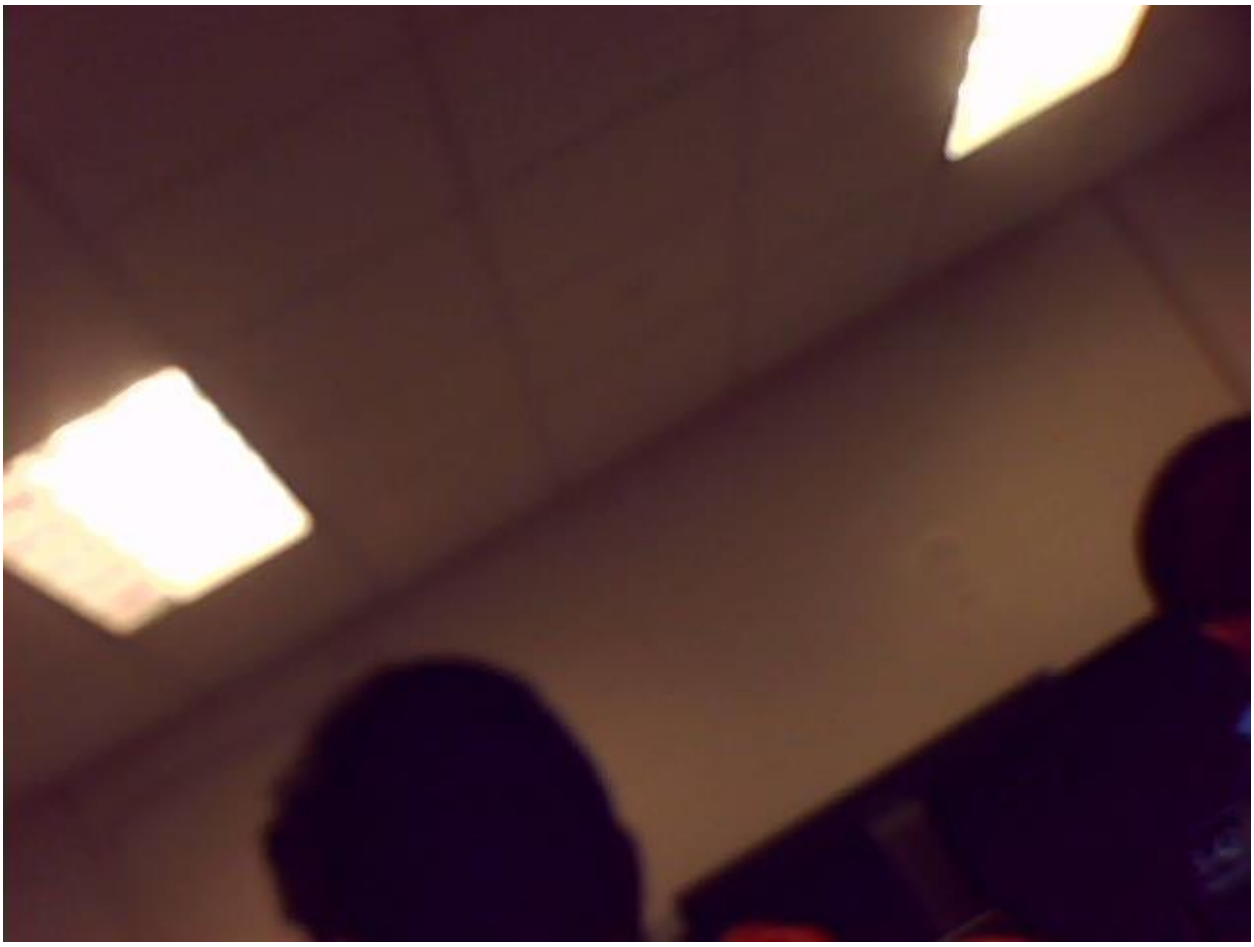


Image captured and transferred to the server

IX. Appendix – Code

1. Makefile: To executables of the program

```
-----------------------------------------
all:lab4

lab4:
     g++ lab4_final.cpp `pkg-config --cflags --libs opencv` -o lab4

clean:
     rm *.o lab3
```

2. Gesture thread: To configure Galileo to communicate with gesture sensor

```
-------------------------------------------------------------------------
void *gesture_thread( void *arg)
{
        int i;
        int r;
        int fd;
        unsigned char value[2] ={0,0} ;
        useconds_t delay = 2000;
        //To specify the i2c bus to connect
        char *dev = "/dev/i2c-0";
        //Address of the gesture sensor
        int addr = 0x39;
        //get the control over i2cbus using mutex
        pthread_mutex_lock(&mutex);
        //To open the i2c device
        fd = open(dev, O_RDWR );
        //To notify error if cannot open device
        if(fd < 0)
        {
              perror("Opening i2c device node\n");
              return 1;
        }
        //to select the slave device 0x39
        r = ioctl(fd, I2C_SLAVE, addr);
        //Notify if error in selecting the device
        if(r < 0)
        {
               perror("Selecting i2c device\n");
        }
       //Command to be send to configure the sensor
       char command[2] = {0x80,0b01011111};
       //write the command to selected device
       r = write(fd, command ,2);

        if(r != 2)    //Notify error
              printf("ERROR: Writing I2C device\n");
       //Give away the control over i2cbus
       pthread_mutex_unlock(&mutex);
```

```
        //Infinite loop
        while(1)
          {
                //get the control over i2cbus using mutex
                pthread_mutex_lock(&mutex);
                //To read the gesture control value
                for(i=0;i<2;i++)
                {
                       value[0] = PDATA;     //Register to be read
                       value[1] = 0x00;      //Value to be read
                       //To read the value
                       r = read(fd, &value[i], 1);

                       //Notify if error in read the value
                       if(r != 1)
                       {
                              perror("reading i2c device\n");
                       }
                       usleep(delay);
                }
                //Open a file to write the sensor value to a text file
                FILE *fp;
                //Open a file
                fp = fopen("gout.txt", "w");
                //write the data to the file
                fprintf(fp,"%d\n", value[1]);
                //Close the file
                fclose(fp);
                //Give away the control over i2cbus
                pthread_mutex_unlock(&mutex);

         }
        //Close i2c device
         close(fd);
         return 0;
}
```

3. temp_getValue() function to read value from temperature sensor

---

```
float temp_getValue()
{
    int i;
    int r;
    int fd;
    float result = 0.0;
    char value[2] ={0} ;
    useconds_t delay = 2000;
    //The i2c bus to be configured
    const char *dev = "/dev/i2c-0";
    //Address of the slave, temperature sensor
    int addr = 0x48;
    //get the control over i2cbus using mutex
    pthread_mutex_lock(&mutex);
```

```
    //To open i2c device
    fd = open(dev, O_RDWR );
    //Notify if error in opening the device
    if(fd < 0)
    {
        perror("Opening i2c device node\n");
        return 1;
    }
    //To open the slave device with address 0x48
    r = ioctl(fd, I2C_SLAVE, addr);

    //Notify if selecting the device
    if(r < 0)
    {
        perror("Selecting i2c device\n");
    }
    //To read msb and lsb of the temperature value(12 bits)
    for(i=0;i<2;i++)
    {
            //To read the values, 1 byte at a time
            r = read(fd, &value[i], 1);
            //Notify if error in reading the value
            if(r != 1)
            {
                perror("reading i2c device\n");
            }
            usleep(delay);
    }
    //To combine the two bytes to interpret the data
    float  tlow =0;
    //Data maunipulation
    tlow = (float)(((value[0] << 8) | value[1]) >> 4);
    //Converting to Celsius
    result = 0.0625*(tlow);
    printf("Temperature: %f\n",result);
    //Close the device and return the value
    close(fd);
    //Give away the control over i2cbus
    pthread_mutex_unlock(&mutex);

    return result;
}
```

4. To capture the image using OpenCV

```
   ------------------------------------------------------------
void imageCapture()
{
    // open the video camera no. 0
    VideoCapture cap(0);
    // if not success, exit program
    if (!cap.isOpened())
    {
        cout << "ERROR: Cannot open the video file" << endl;
```

```
    }
   //get the width of frames of the video
   double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH);
   //get the height of frames of the video
   double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
   cout << "Frame Size = " << dWidth << "x" << dHeight << endl;
   //vector that stores the compression parameters of the image
   vector<int> compression_params;
   //specify the compression technique
   compression_params.push_back(CV_IMWRITE_JPEG_QUALITY);
   //specify the jpeg quality
   compression_params.push_back(95);
   //Matrix to read the image and compress
   Mat img(dWidth, dHeight, CV_8UC1);
   //Read the image from the camera
   cap.read(img);

//write the image to file
bool bSuccess = imwrite("/media/card/img.jpg", img, compression_params);
//Notify if error in writing the image
if ( !bSuccess )
   {
        cout << "ERROR : Failed to save the image" << endl;
   }
}
```

5. Gesture function to read the sensor value

```
int gesture()
{
       int result;
       FILE *fp;
       fp = fopen("gout.txt", "r"); //Open the text file
       fscanf(fp, "%d",&result); // read the result from the file
       fclose(fp);                  // Close the file
       printf("Gesture = %d\n", result);
       return (result);             //Return the value
}
```

6. Main function to integrate the all the device to trigger the camera

```
pthread_mutex_t mutex;
int main(int argc, char* argv[])
{
       //Create mutex
       pthread_mutex_init(&mutex,NULL);
       int pid=0,pid2=1;
       pthread_t g,ld;
       pthread_attr_t attr,attr2;
       //Create attribute variable for the threads
       pthread_attr_init(&attr);
       pthread_attr_init(&attr2);
```

```
        //Create greasure and PIC threads
        pthread_create(&g, NULL, gesture_thread, (void *)pid);
        sleep(1);
        pthread_create(&ld, NULL, ldr, (void *)pid2);
        sleep(1);
        //Infinite loop
        while(1)
        {
                //To get temperature value in Celsius
                float temp_result = temp_getValue();
                sleep(1);
                //To get gesture value
                int gesture_result = gesture();
                sleep(1);
                //To lock the mutex
                pthread_mutex_lock(&mutex);
                //Get the ADC value
                local_ldr_value = ldr_value;
                //Give away the mutex
                pthread_mutex_unlock(&mutex);
                // To trigger camera if temperature is greater than 27
      Celsius and if gesture control value between 50 and 255
                if( temp_result > 27.0 ||
                        (gesture_result >50 && gesture_result <= 255))
                {
                        //To capture the image
                        imageCapture();
                        //Send the image to the server
                        send_pic();
                        sleep(0.5);
                        printf("Image captured\n");
                }
        }
        //Wait for the threads to finish it work
        pthread_join(g, NULL);
        pthread_join(ld, NULL);
        //Destroy the mutex
        pthread_mutex_destroy(&mutex);
        return 0;
}
```

7. ldr thread: To get the ADC value from the PIC microcontroller

---

```
void *ldr(void *id)
{
        char cmd[200];
        int option, ack, nibl1, nibl2, nibl3, data = 0;
        //Get the control over ports
        pthread_mutex_lock(&mutex);
        //Initialize intel ports to communicate with PIC
        init();
        //Give away the control over ports to other threads
        pthread_mutex_unlock(&mutex);
```

```
//Infinite loop
while(1)
{
	//Get the control over ports
	pthread_mutex_lock(&mutex);
	//Using only option 3- MSG_GET for now
	option = 3;
	switch(option)
	{
		case 1: // reset
			write_gpio(0);
			ack = read_gpio();
			if(1)
			{
				printf("Received Value:%d\n", ack);
			}
		option = 0;
		break;

		case 2: // ping
			write_gpio(1);
			ack = read_gpio();
			if(1)
			{
				printf("Received Value:%d\n", ack);
			}
		option =0;
		break;

		case 3: // get
			//Write to the ports
			write_gpio(2);
			//Reading from the ports
			nibl1 = read_gpio();
			nibl2 = read_gpio();
			nibl3 = read_gpio();
			ack = read_gpio();
			//printf("Nibl1:%d\n", nibl1);
			//printf("Nibl2:%d\n", nibl2);
			//printf("Nibl3:%d\n", nibl3);
			printf("Ack:%d\n", ack);
			//Extracting ADC data(12bits) from the 3
			nibbles
			data = 0;
			data = nibl1;
			data = (nibl2 << 4) | data;
			data = ((nibl3 & 0x3)<< 8)  | data;
			//check if you got the ACK
			if(ack == 14)
			{
			  printf("ADC Result:%d\n", data);
			  ldr_value = data;
```

```
                        }
                       else
                       {
                   printf("ADC Result without correct ack:%d\n", data);
                            printf("Received ACk:%d\n", ack);
                       }
                      option = 0;
                      break;

                      default:
                            option = 0;
                      //    printf("ADC Result:%d\n", data);
                      break;
               }
               //Give away the control over ports to other threads
               pthread_mutex_unlock(&mutex);
               sleep(2);
       }

}
```

8. Configuring PIC thread, functions used below

```
-------------------------------------------------------------------------
To wirte to output ports
-------------------------------------------------------
void write_gpio(int data)
{
    // make strobe high
    //printf("make strobe\n");
system("./pin_out_strobe.sh 40 1"); // set GPIO_8 as output - Strobe
    //printf("after making strobe high\n");

    if(data == 0) // reset
    {
    system("./pin_out.sh 16 14 0"); // set GPIO_3 as output - D0
    system("./pin_out.sh 36 6 0"); // set GPIO_4 as output - D1
    system("./pin_out.sh 18 0 0"); // set GPIO_5 as output - D2
    system("./pin_out.sh 20 1 0"); // set GPIO_6 as output - D3
    }
    else if(data == 1) // ping
    {
    system("./pin_out.sh 16 14 1"); // set GPIO_3 as output - D0
    system("./pin_out.sh 36 6 0"); // set GPIO_4 as output - D1
    system("./pin_out.sh 18 0 0"); // set GPIO_5 as output - D2
    system("./pin_out.sh 20 1 0"); // set GPIO_6 as output - D3
    }
    else if(data == 2) // get
    {
    system("./pin_out.sh 16 14 0"); // set GPIO_3 as output - D0
    system("./pin_out.sh 36 6 1"); // set GPIO_4 as output - D1
    system("./pin_out.sh 18 0 0"); // set GPIO_5 as output - D2
    system("./pin_out.sh 20 1 0"); // set GPIO_6 as output - D3
    }
```

```
    printf("Send value: %d\n", data);
    sleep(0.01);

    // make strobe low
system("./pin_out_strobe.sh 40 0"); // set GPIO_8 as output - Strobe

    system("./pin_out.sh 16 14 0"); // set GPIO_3 as output - D0
    system("./pin_out.sh 36 6 0"); // set GPIO_4 as output - D1
    system("./pin_out.sh 18 0 0"); // set GPIO_5 as output - D2
    system("./pin_out.sh 20 1 0"); // set GPIO_6 as output - D3

    sleep(0.01);

}
```

TO read from the ports

```
int read_gpio()
{
    int a;
    FILE *fp;
    // make strobe high
system("./pin_out_strobe.sh 40 1"); // set GPIO_8 as output - Strobe

    system("./pin_in.sh 16 14"); // set GPIO_3 as input - D0
    fp = fopen("out.txt", "r");
    a = convertStrToInt(fgetc(fp));

    system("./pin_in.sh 36 6"); // set GPIO_4 as input - D1
    fp = fopen("out.txt", "r");
    a = a |(convertStrToInt(fgetc(fp)) << 1);

    system("./pin_in.sh 18 0"); // set GPIO_5 as input - D2
    fp = fopen("out.txt", "r");
    a = a | (convertStrToInt(fgetc(fp)) << 2);

    system("./pin_in.sh 20 1"); // set GPIO_6 as input - D3
    fp = fopen("out.txt", "r");
    a = a |(convertStrToInt(fgetc(fp)) << 3);
    sleep(0.01);
    // make strobe low
system("./pin_out_strobe.sh 40 0"); // set GPIO_8 as output - Strobe
    sleep(0.01);
    return a;
}
```

To initiate intel galelio ports to communicate with the PIC

```
void init()
{
    system("./pin_out.sh 16 14 0"); // set GPIO_3 as output - D0
    system("./pin_out.sh 36 6 0"); // set GPIO_4 as output - D1
    system("./pin_out.sh 18 0 0"); // set GPIO_5 as output - D2
```

```
    system("./pin_out.sh 20 1 0"); // set GPIO_6 as output - D3
system("./pin_out_strobe.sh 40 0");//set GPIO_8 as output Strobe

}
```

To configure Intel Galileo GPIO pins as input

```
#!/bin/sh
echo -n "$1" > /sys/class/gpio/export
echo -n "in" > /sys/class/gpio/gpio$1/direction
echo -n "$2" > /sys/class/gpio/export 2>>error
echo -n "in" > /sys/class/gpio/gpio$2/direction
cat /sys/class/gpio/gpio$2/value 1> out.txt
echo -n "$1" > /sys/class/gpio/unexport
echo -n "$2" > /sys/class/gpio/unexport
```

To configure Intel Galileo GPIO pins as output

```
#!/bin/sh
echo -n "$1" > /sys/class/gpio/export
echo -n "out" > /sys/class/gpio/gpio$1/direction
echo -n "$2" > /sys/class/gpio/export 2>>error
echo -n "out" > /sys/class/gpio/gpio$2/direction
echo -n "$3" > /sys/class/gpio/gpio$2/value
echo -n "$2" > /sys/class/gpio/unexport
echo -n "$1" > /sys/class/gpio/unexport
```

Read function in PIC

```
int read(void)
{
int data;
IO_RC0_SetDigitalInput(); // To set RC0 as Input
IO_RC1_SetDigitalInput(); // To set RC1 as Input
IO_RC2_SetDigitalInput(); // To set RC2 as Input
IO_RC3_SetDigitalInput(); // To set RC3 as Input
while(!IO_RB2_GetValue()); // Wait for strobe high
while(IO_RB2_GetValue()) // During strobe high, write the data to
data = PORTC;  ports
return data;
}
```

Write function in PIC

```
void write(int data)
{
IO_RC0_SetDigitalOutput(); // To set RC0 as Output
IO_RC1_SetDigitalOutput(); // To set RC1 as Output
IO_RC2_SetDigitalOutput(); // To set RC2 as Output
IO_RC3_SetDigitalOutput(); // To set RC3 as Output
while(!IO_RB2_GetValue()); // Wait for strobe high
while(IO_RB2_GetValue())// During strobe high, read the data from
PORTC = data;  port
PORTC = 0x0;
}
```

```
   Main function in PIC
   ----------------------------------------
   void main(void)
   {
   SYSTEM_Initialize(); // initialize ports, configure them
   while (1)
   {
   int readValue = read(); // read the value from port
   if(readValue==0x0) // If MSG_RESET,call system initialize&Send ACK
   {
   SYSTEM_Initialize();
   write(0xE); //ACK
   }
   else if(readValue==0x1) //If MSG_PING, send ACK
   {
   write(0xE); //ACK
   }
   else if(readValue==0x2)// If MSG_GET, send ADC value and ACK
   {
   uint16_t adc_result = ADCC_GetSingleConversion(channel_ANA2);
   unsigned int nib1, nib2, nib3; // Extract nibble 1,2,3
   nib1 = adc_result & 0x0f;
   nib2 = (adc_result >> 4) & 0x0f;
   nib2 = (adc_result >> 8) & 0x03;
   write(nib1);
   write(nib2);
   write(nib3);
   write(0xE); //ACK
   }
   }
   }
```

9. To connect to the server

To initialize HTTP POST request

```
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_POST, 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);
       }
}
```

To send the image to the server

```
void send_pic(){
       const char* hostname="10.253.78.222";
       const int   port=8000;
       const int   id=1;
       const char* password="password";
       const char* name="Team+Awesome";
       const int   adcval= local_ldr_value; //ADC value
       const char* status="Very+good+thank+you";
       const char* timestamp="20141116-09:12:34";
       const char* filename="img.jpg";
       char buf[1024*1024];
       snprintf(buf, 1024*1024,
"http://%s:%d/update?id=%d&password=%s&name=%s&data=%d&status=%s&times
tamp=%s&filename=%s",
                     hostname,
                     port,
                     id,
                     password,
                     name,
                     adcval,
                     status,
                     timestamp,
                     filename);
       FILE *fp;

       struct stat num;
       stat(filename, &num);
       int size = num.st_size;
       printf("Image size: %dB\n", size);

       char *buffer = (char*)malloc(size);

       fp = fopen(filename,"rb");
       int n = fread(buffer, 1, size, fp);

       HTTP_POST(buf, buffer, size);
       fclose(fp);
}
```

10. Face recognition

```
   void FaceDetection()
   {
       Mat tmp;
       int width  = img.size().width,
           height = img.size().height;
   Size minScaleSize = Size(_minSizeRatio  * width, _minSizeRatio  *
                            height),maxScaleSize = Size(_maxSizeRatio  *
                                    width, _maxSizeRatio  * height);
```

```
//convert the image to grayscale
    cvtColor(img, tmp, CV_BGR2GRAY);
    equalizeHist(tmp, tmp);

    //clear the vector:
    res.clear();
//detect the face
 _cascade.detectMultiScale(tmp, res, _scaleFactor, _minNeighbors,
                                    0, minScaleSize, maxScaleSize);
}
```