Designing a Client-Sensor Software for Embedded based Surveillance System

Aim:

The objective of this project is to build a Surveillance system using multithreading programing – Pthread library; synchronization of those threads using mutex; understanding usage of curl library, HTTP protocol using a client and server application and understanding of image processing using OpenCV library.

Introduction:

The idea of this project is to read the sensor data from a I2C devices Gesture sensor (APDS-9960), Temperature sensor (TMP102) and communicating with PIC16F18857 microcontroller using custom bus protocol to get the LDR value and rotate the camera using servo motor. The camera is triggered to capture a picture when the required threshold values of the sensor data is reached and processes the captured image for facial recognition using OpenCV library. The images and sensor data are then transferred to server through HTTP protocol using curl library. All these actions are made concurrent using POSIX thread library.

Devices/Software's Used:

1. Bread board
2. Wires to connect
3. Temperature sensor TMP102
4. Gesture sensor APDS-9960
5. Serial to USB connector (FTDI)
6. Multi-meter
7. Voltage supply (3.3V) from Galileo and 5V for servo motor
8. Intel Galileo Gen 2 Board
9. Yocto Linux
10. Putty
11. PIC16F18857 microcontroller
12. Resistors - 3 (10K ohms) & 1 (220 ohms)
13. Servo Motor
14. LDR
15. Oscilloscope
16. MPLAB

Schematic:

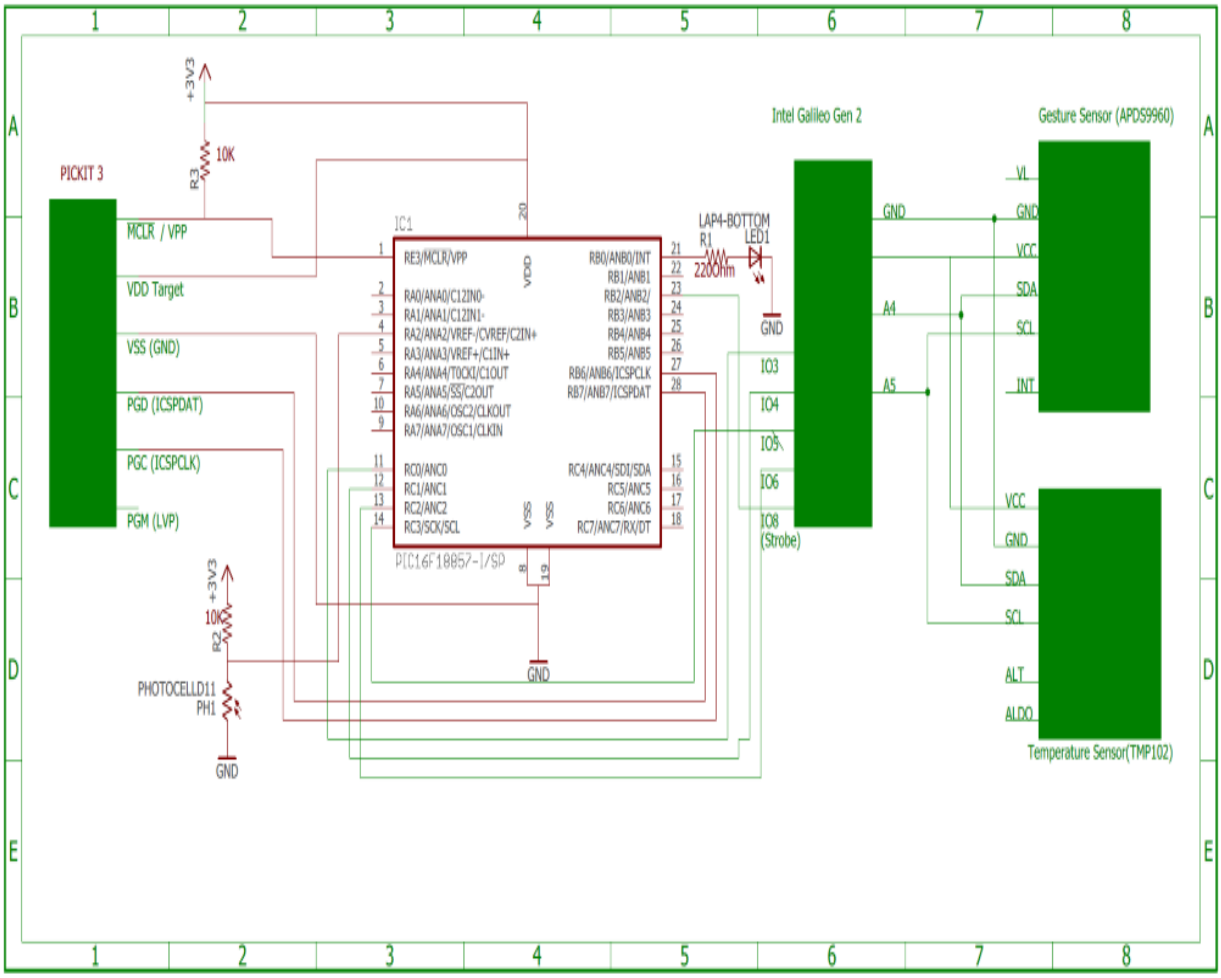


Fig. 1 Circuit connection for the project

Hardware design:

1. **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 RB0 (Pin 21). 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.

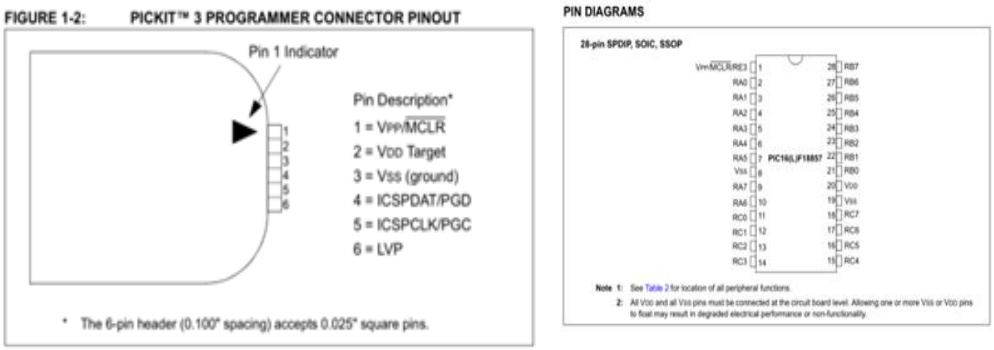


Fig. 2 Pickit and PIC pin connections

2. **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 breadboard 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.

3. **I2C protocol :** I2C is a simple serial bus communication protocol. It uses two wires Serial Data Clock(SCL) and Serial Data(SDA) to communicate between other devices. It is a half duplex communication in which one or more devices can be connected. The device that generates the SCL is called as a Master and the device that receives the clock is called a Slave. There can be more than one slave or master on the bus.

   Procedure for communication :

   1) Wait until I2C bus is free: both SDA and SCL are high.

2) Put a “START” message on bus to claim bus (all other ICs will then
3) listen)
4) Put a clock signal on SCL line for other ICs as reference time (the data
5) on SDA wire must be valid when SCL switching from low to high)
6) Put binary address in series to identify target IC
7) Put one bit to identify direction (SEND or RECEIVE)
8) Ask other IC to acknowledge the address and readiness to transfer
9) Transfer (many) 8-bit data after receiving ack
10) The master send “STOP” message to free up the bus

A typical I2C bus communication is shown below. A start condition is detected when the data goes high to low while the clock is high. The data is updated at each rising edge of the clock and stop condition is detected when data goes from low to high while clock is high.

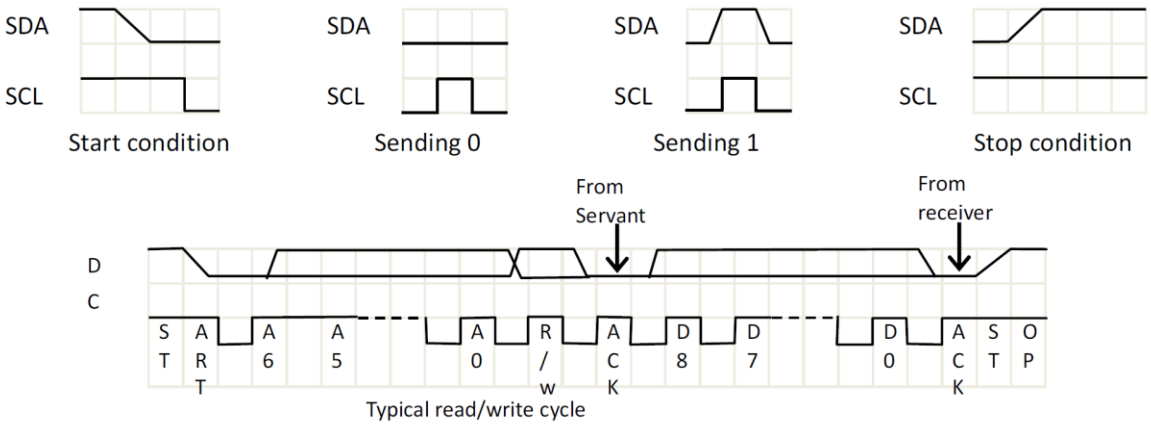


Fig 3. Typical I2C communication

There are two types of addressing: 7 bit and 10 bit depending the available devices for i2c communication with master. For 7 bit addressing, 7 bit slave address is followed by read/write bit, acknowledgement bit, data and so on. Where as for 10 bit addressing the address is divided into two bytes, higher byte comprises of 1111 0XX with XX as higher two bits of address. This address is forwarded to bus with read write and after receiving acknowledgement bit the remaining 8 bit address is sent just like 7 bit addressing. Details of addressing can be viewed in figure 4.

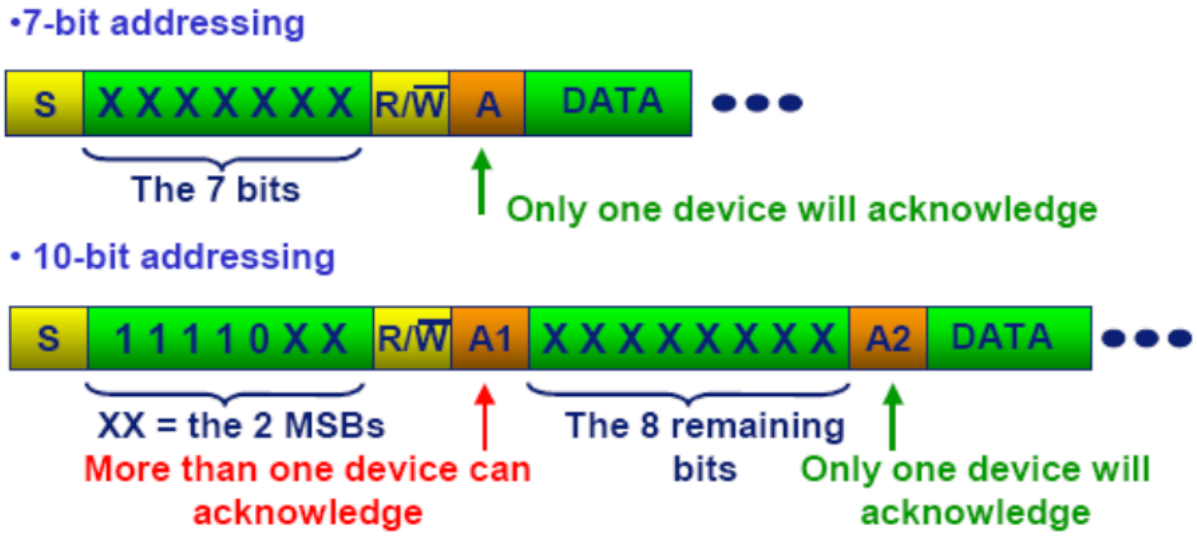


Fig. 4 7-bit & 10-bit addressing

4. **Custom bus protocol:** A simple protocol was implemented to communicate between PIC and Galileo. A strobe signal and four data signals are used in the bus. It is command/ response type protocol in which the Galileo acts as a master and PIC act as a slave. A user application on Yocto Linux issues commands to the PIC device.

   The idea for bus communication is to send/read data to/from slave when strobe signal goes from low to high, which is done using extra gpio pin on master. On the slave side once the slave recognizes low to high signal, it starts to read/send the data on the bus and continues to read/send while the strobe is high.

   Write Operation: The steps (in Figure 5) involved in the write operation are as follows.

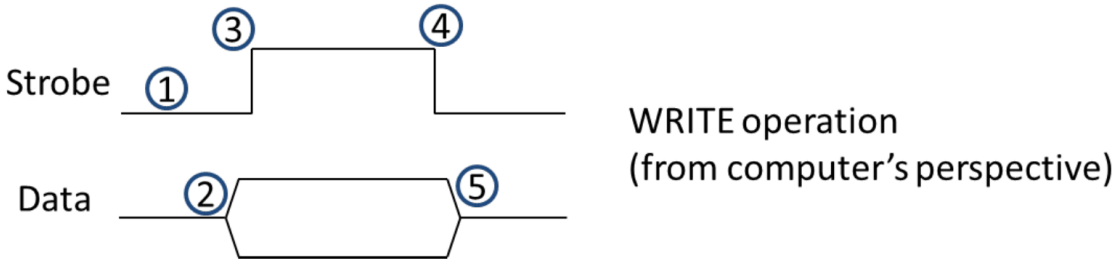


Fig. 5 Write Operation using strobe signal.

(1) The Galileo pulls the Strobe signal low. The PIC microcontroller gets ready to read a 4-bit **command** message.
(2) The Galileo outputs a command on the data bus
(3) The Galileo raises the Strobe signal to indicate the command is ready on

the bus. The PIC microcontroller starts reading the value (i.e. a **command**) from the bus. The Galileo maintains the value for at least 10ms.
(4) The Galileo ends the write operation by pulling the Strobe signal to low again. By this time, the PIC should have already finished reading the value.
(5) The Galileo stops putting the command on the bus. The write operation concludes.

The GPIO port of the Galileo should be in low impedance mode when outputting the command, while the PIC should put the data pins in high impedance mode since it is receiving the value.

Read Operation:

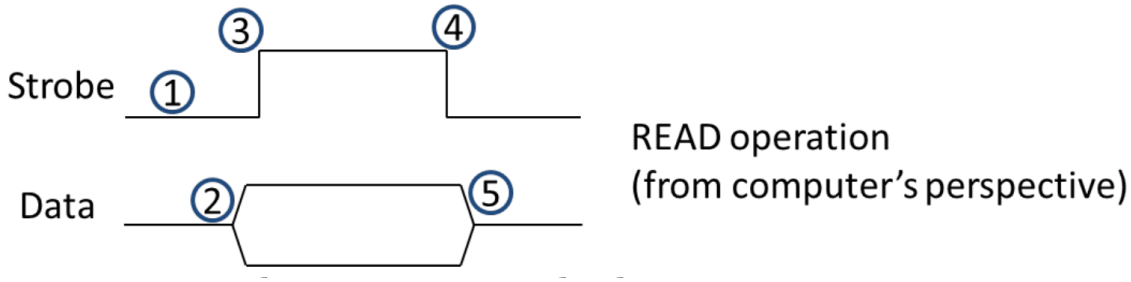


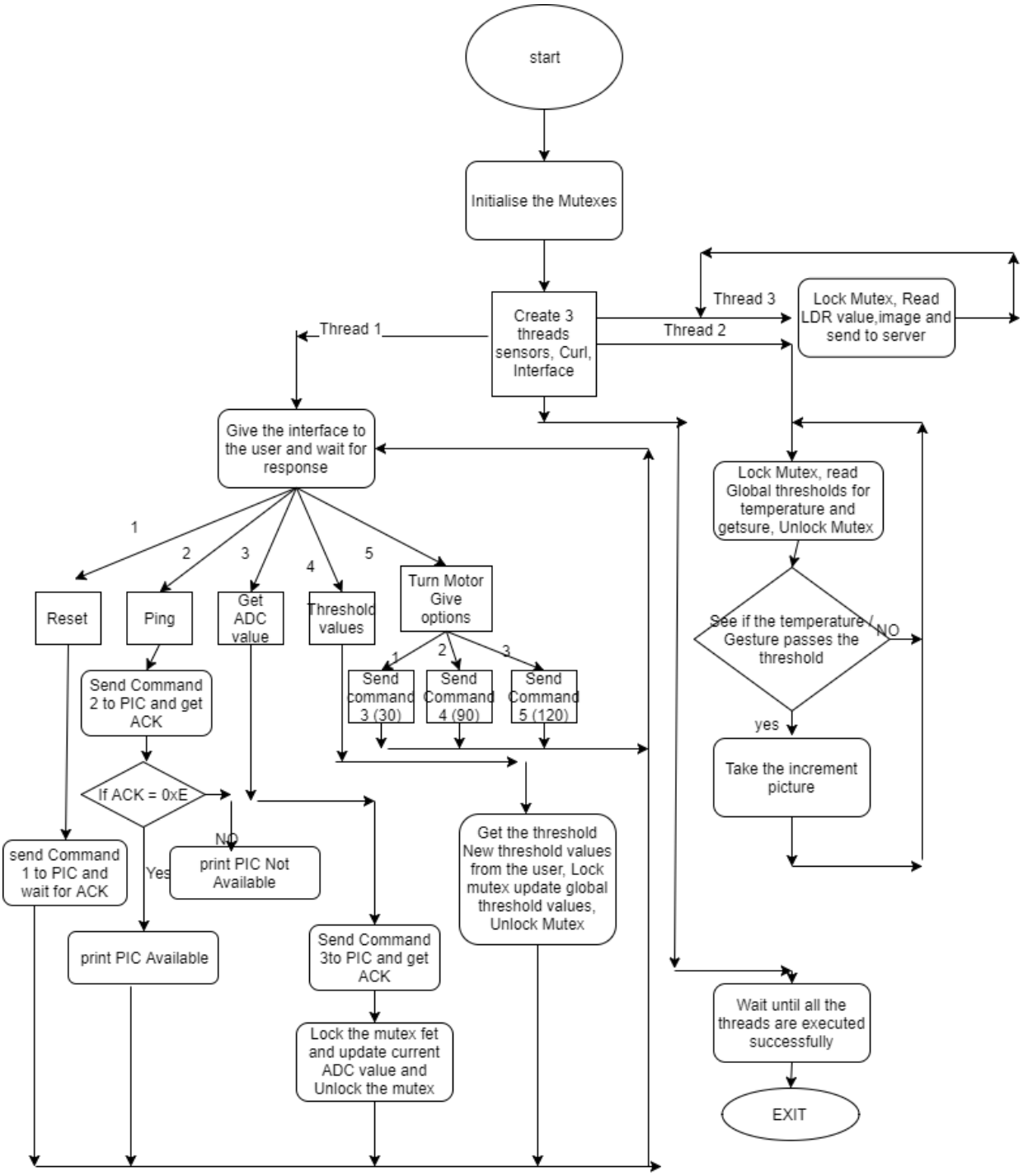
Fig. 6 Read Operation using strobe signal.

Figure 6 illustrates the five steps involved in a read operation from the Galileo (although the timing diagram appears to have the same pattern as Figure 3, there are differences in the steps):
(1) The Galileo pulls the Strobe signal low to get ready for read operation
(2) The PIC microcontroller on the sensor device outputs a 4-bit value (either **MSG_ACK** or a portion of the ADC value) when it sees a low on the strobe line.
(3) The Galileo raises the Strobe signal and start reading the value from the data bus. The PIC checks the Strobe signal and learns that the Galileo has started reading the value.
(4) The Galileo pulls the Strobe signal low again to indicate that the value has been read.
(5) The PIC microcontroller sees the Strobe pulled low and stops outputting the 4-bit value.

the read operation can be repeated for multiple times (e.g. three times for obtaining a 10-bit ADC value).

Software Design :

1. **Flow Chart :**



Threads involved and their objectives:

Threads are created using pthread library on Intel Galileo Gen2 board running Yocto Linux system. The details of threads and their functions are:

Thread1: `Interface`()

It performs the following tasks:

1. A set of options to configure the sensors are created for user, it takes the value from user.
2. Performs the actions mentioned in the options
3. Sets flags like update and capture if user selects the option for gesture.

Thread2: `Sensors()`

It performs the following tasks:

1. Sets up the apds 9960 geture sensor and programs the bits of enable register.
2. Reading the registers form sensor to get relevant data
3. Checks for gesture and takes picture if beyond threshold.

Thread3: `Client()`

It performs the following tasks:

1. If the capture flag is set high then enters into it
2. Takes the username/id, ip address, status, adc values, time-date and image and uploads on server.

Mutexes are used whenever there is sharing of data between two threads. The global variables are protected so that no two threads are accessing this variable at the same time. By this the other threads will be able to see if there is any update that is done by any of the threads.

2. **Functions used in PIC module:** ADC, PWM, Timer 2 are configured accordingly with reference to the data sheet.
    1. `ADC_Init()` - Initializes the ADC module on PIC. It sets the control registers to appropriate values. The ADC channel is set to 1 and the ADC value is right justified.

        Parameters - None

        Return Value - None
    2. `PWM_Init()` - Initializes the PWM module on PIC. Enables the PWM module and sets the mode to PWM. Timer2 is selected for PWM generation.

        Parameters - None

        Return Value - None

3. `Timer2_Init`() - Initializes the timer 2 module on PIC in free running mode. Sets the prescale value and clears the interrupt flag.
4. `PWM_signal_out()` - Starts the Timer and generate the PWM signal with appropriate duty cycle.
   Parameters - duty : PWM duty cycle to be set.
   Return Value : None
5. `ADC_conversion_results`() - Starts the ADC conversion and converts the ADC result from high and low registers to a 10 bit value.
   Parameters - None
   Return Value - ADC value.

6. `Set_Receive()` - Sets the appropriate GPIO's of PIC as inputs.
   Parameters - None
   Return Value - None
7. `Set_Send()` - Set the appropriate GPIO's of PIC to outputs
   Parameters - None
   Return Value - None

8. `Receive_Msg()` - Waits for the strobe signal and receives the command from galileo.
   Parameters - None
   Return Value - The message received from galileo
9. `Strobe()` - Waits for the strobe signal and sends a message to the galileo
   Parameters - Message to be sent
   Return Value - None

10. `SendADC()` - Sends the ADC value to the galileo
    Parameters - ADC value to be sent
    Return Value - None

11. `main()` - Based the command sent required activity is performed on PIC
    Parameters - None
    Return Value - None

3. **Modules used in Galileo :** There are - modules used on galileo. They are GPIO, Client, Interface, Lab4. The Lab4.cpp contains the main module which creates three threads

Client thread, Interface thread and Sensors thread. Each of theses threads are running concurrently need to be synchronized.

Gpio module: The GPIO module is written to provide required functions to manipulate GPIO ports on Galileo.

Functions used :

1. `Export()` - Exports the required GPIO pins' drivers to set the pins to output/input mode and setting the value to them. To export the GPIO pins the linux GPIO Sysfs interface is used in /sys/class/gpio

   Parameters - None

   Return value - None
2. `UnExport()` - Unexports the required GPIO pins' driversSet. To unexport the GPIO pins the linux GPIO Sysfs interface is used in /sys/class/gpio

   Parameters - None

   Return value - None
3. `SetGpio_output`() - Sets the GPIO pins to out mode.

   Parameters - None

   Return value - None
4. `SetGpio_Input()` - Sets the GPIO pins to input mode.

   Parameters - None

   Return value - None
5. `StrtoInt()` - Convert the characters '1' and '0' to binary form

   Parameters - Data of type char

   Return value - returns integer 0 or 1
6. `read_gpio()` - Reads the data from GPIO pins using a script file gpio_in.sh which takes the GPIO number as input, reads the value from the GPIO and writes the value to file.

   Parameters - None

   Return value - Returns the integer value stored in file

Client Module : The client module consists of an Client thread which will use the Curl API to post the data to the Server. The image details like size, time are taken and the image name is sent to the server. The client thread will only execute when the capture variable is 1 which indicates that a new image is captured or available.

Functions Used :

1. `Client()` – This is a thread function to the update all the sensor values, image details to server

   Parameters - Thread id

Return value - None

2. `HTTP_POST()` – Implements the curl functions to be used by Client to post the message stored in a buffer.

   Parameter - url : The address of the URL where it is stored

   Image - The address of the image file

   Size - Size of the whole packet

   Return Value - None

3. `time_stamp()`- To get the current time from the linux kernel using the inbuilt time module.

   Parameter - None

   Return value - Address of the timestamp value

Lab4 Module: The Lab4 consists of the main function which creates the threads and wait for the threads to finish executing and return successfully. The i2C linux device driver is used to interface with the sensors. The interfacing is done by calling i2c node using system call to the kernel device driver. Once the connection is setup, its status is stored in a variable accessible only to the functions of this module. Care should be taken to access it only by one function at a time to avoid corruption of data. The Sensor thread is included in this module which only executes when there is an update from the user. The module also consists the functions required to initialize these sensors. To communicate with gesture/temperature sensor the slave (gesture/temp sensor) is first assigned to an address using ioctl(), which takes node variable, request (slave/master) and address, and then using APDS9960_write() the registers in the slave device are accessed as explained below. The module also contains the Interface thread which will work on creating a user interface. The Interface thread gets the data from the user. The program is designed so that when a “Enter” is pressed the system starts executing the other threads. The Sensor and Client thread will be in sleep state until an update is received from the user. The user can give commands in the form of number from 1-7 in the following sequence:

0x1: Reset PIC
0x2: Ping to PIC to receive ACK
0x3: Read LDR value from PIC
0x4: Turn servo motor by 30 degree
0x5: Turn servo motor by 90 degree
0x6: Turn servo motor by 120 degree
0x7: read temperature from temperature sensor

The commands are send to PIC through custom bus protocol mentioned in the hardware section.

Functions Used:

1. `APDS9960_write()` - Sends the appropriate command on to the I2C bus. This function is only used to send the address followed by a command. If this is not needed use the write function provided by linux i2C device driver.

   Parameter - address : The i2C sensors address

   Command : The data which needs to sent.

   Return value : True if successful or False

2. `Imagecapture()`- Uses the OpenCV library to capture an Image and writes it to a file and stores it in the SD card.

   Parameters - None

   Return value - None

3. `read_gesture()`- Reads the gesture value from the gesture sensor and determines if it is a up, down, left or right gesture or if there is no gesture. It calculates the gesture by reading registers pertaining to each type of gesture.

   Parameters - None

   Return value - None

4. `gesture_enable()`- Configures the appropriate registers for the gesture sensor

   Parameters - None

   Return value - None

5. `Temperature()`- Configures the temperature sensor and reads the temperature from the sensor

   Parameter - None

   Return Value - returns the temperature value.

6. `Sensors()`- Used as a interface to the Sensor thread

   Parameter – Thread id

   Return value - None

7. `Interface()` - A thread function that reads command sent by the user on terminal window. Sends the command to PIC or temperature sensor using strobed communication. Set the update variable ‘0’ for other waiting threads (sensor) to continue with their task.

Parameters - Thread id
Return Value - None

Conclusion

Successfully implemented the user space application for the embedded based surveillance system. Efficient multithreaded program was developed which involves principles of synchronization and mutual exclusion, i2c communication, curl library, POSIX thread library and image processing using opencv library.