

# **CS795 IoT Lab #3: WIRELESS CONNECTIVITY**

Name: Hrishikesh Gadkari  
UIN: 01085143

## **Abstract**

To design a system with Launchpad Tiva C Series TM4C123GH6PM around an ARM Cortex-M processor core and CC3100 SimpleLink Wi-Fi Boostpack to connect to openweathermap.org server and request for weather details of Norfolk city and then with the Tiva Sensor Hub, implement a communication system between two Tiva boards via an IEEE 802.11 – WiFi module where one Launchpad is used to collect the sensor data and the second to display the data with the Tiva Sensor Hub on PuTTY.

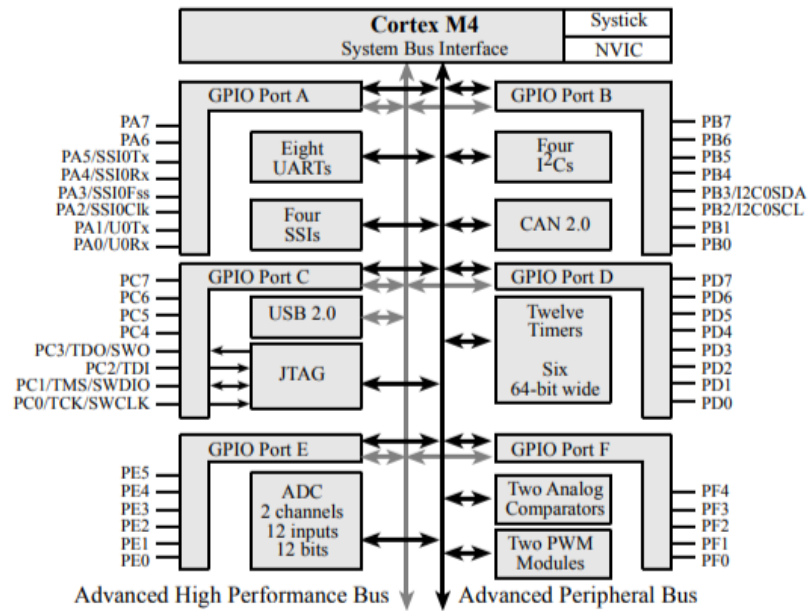
## **I. Introduction**

The system implementation problem comprised of two tasks where task 1 was an individual task whereas the Task 2 was to be implemented in a team of two members.

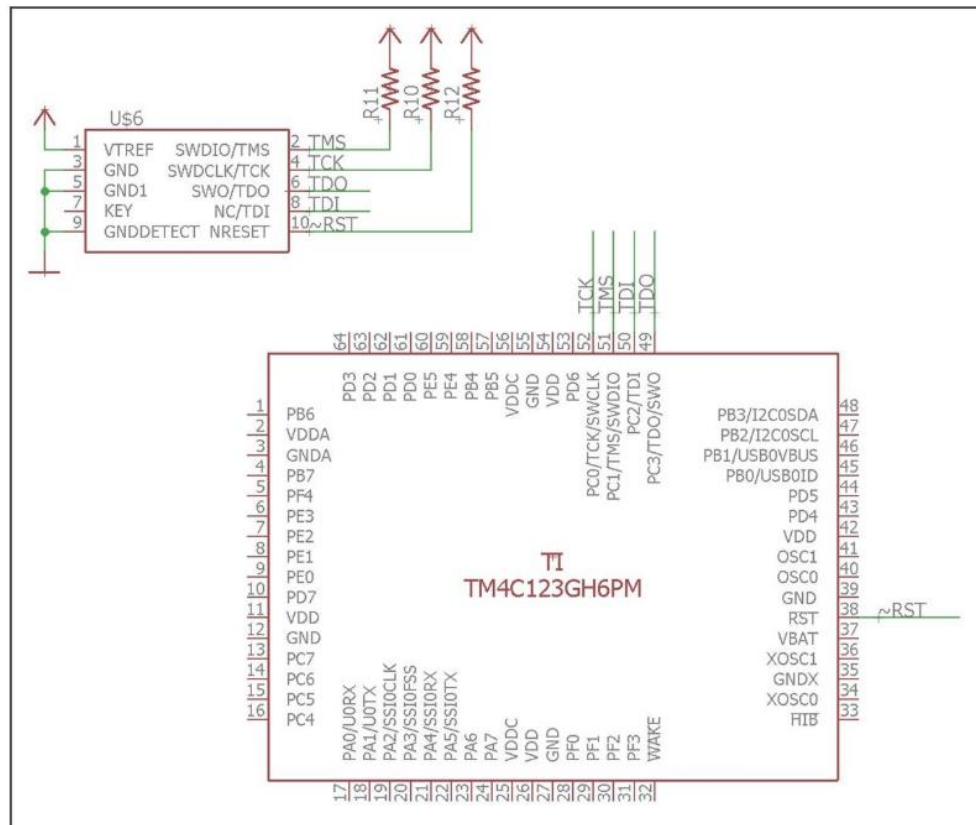
Task 1 was a demonstration of how to connect to openweathermap.org server and request for weather details of Norfolk city. The application opens a TCP socket with the server and sends a HTTP Get request to get the weather details. The received data is processed and displayed on the console. The main.c initializes the device, connects to an AP, opens a TCP socket, requests and displays the weather information. This application requires an access-point with internet connectivity and use UART serial connection in order to display information on the PC with PuTTY.

Task 2 was implemented along with Aniket Chandak as we had to perform the task in a group of two members where my role was to be a server whereas he was the client. The system uses one Launchpad Tiva C Series TM4C123GH6PM to collect the light sensor data and the second to display the data. Data are transmitted across a wireless network using UDP packets. The client measures the sensing data with Sensor Hub from ADC and sends UDP packets to an AP. The server receives UDP packets from the AP and displays the data on PuTTY via UART Virtual COM PORT and plots the data using Python. The CC3100 SimpleLink Wi-Fi Boostpack implements the Internet stack with a combination of Hardware and Software components. Software in the Launchpad performs system calls to the CC3100 to affect wireless communication. We used two Launchpads to develop a solution that transmits UDP packets from one smart object to another. UDP is simpler than TCP and appropriate for applications requiring simplicity and speed. Furthermore, to use UDP the application must tolerate lost or out of order packets. UDP provides a best-effort datagram delivery service.

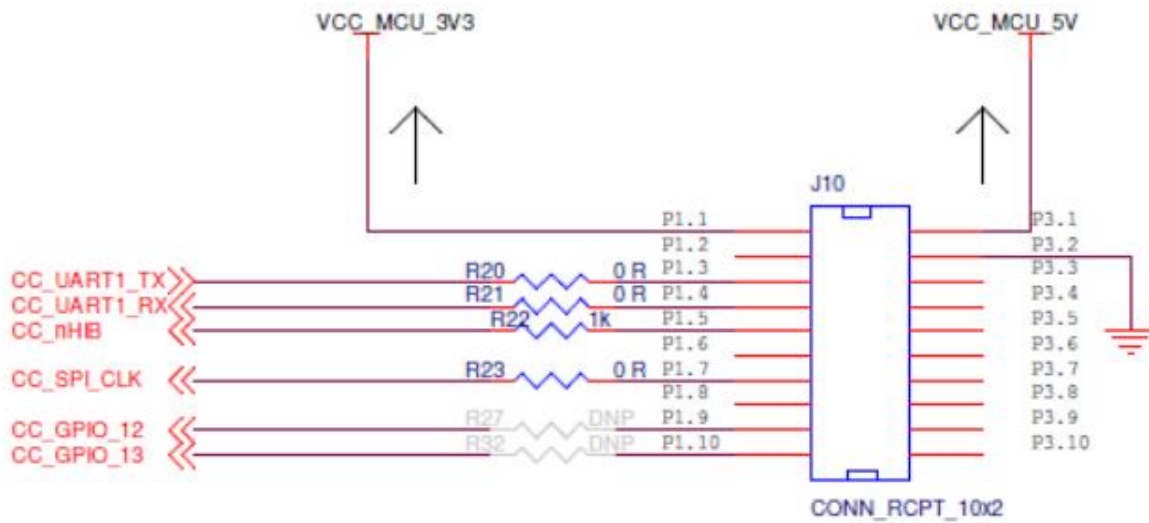
## **II. Interface Diagram**



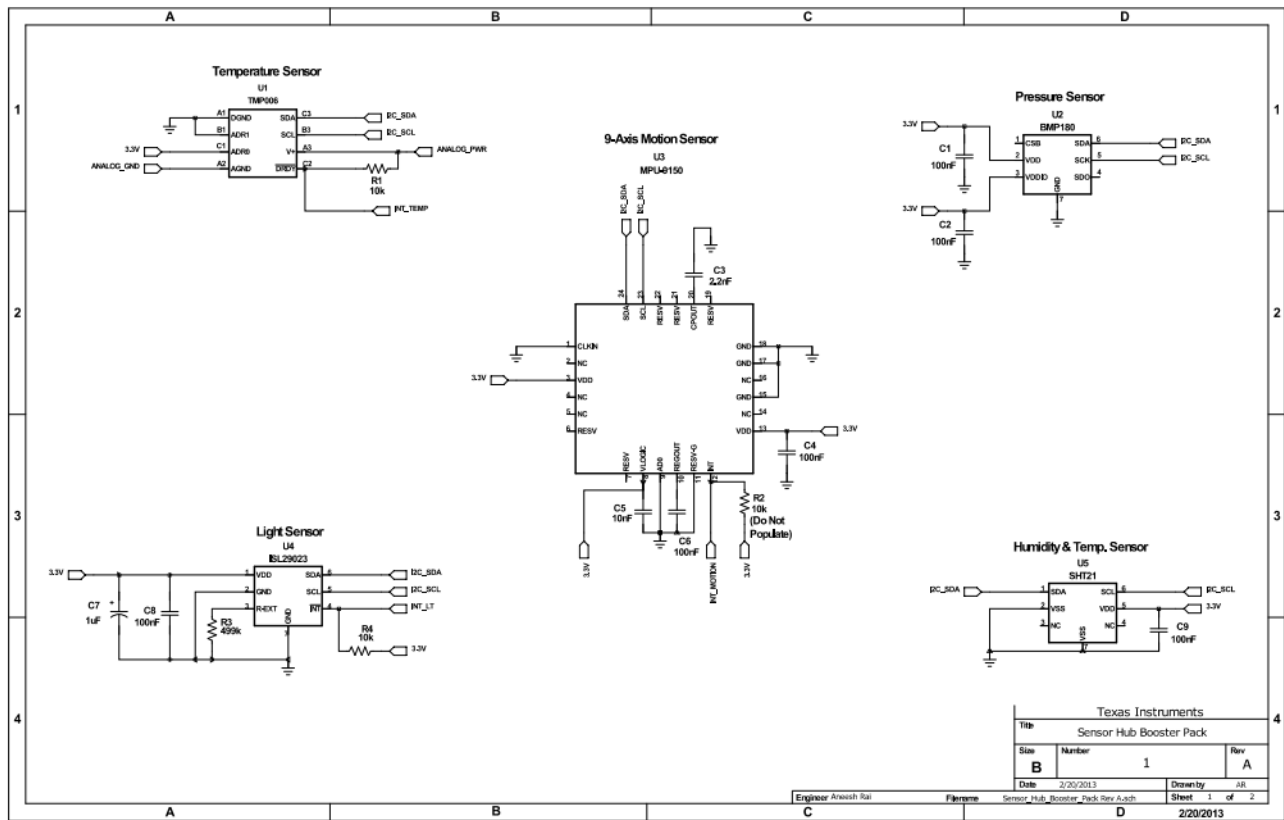
Circuit schematics of ARM Cortex M4 GPIO port connections



Tiva C Series TM4C123GH6PM Launchpad



CC3100 Interface



Sensor Hub Booster Pack

### III. SW Development

#### Task A: Run Get Weather Application

The first step was to create an API Key from openweathermap.org and modify the 'API Key' value in **main.c** file.

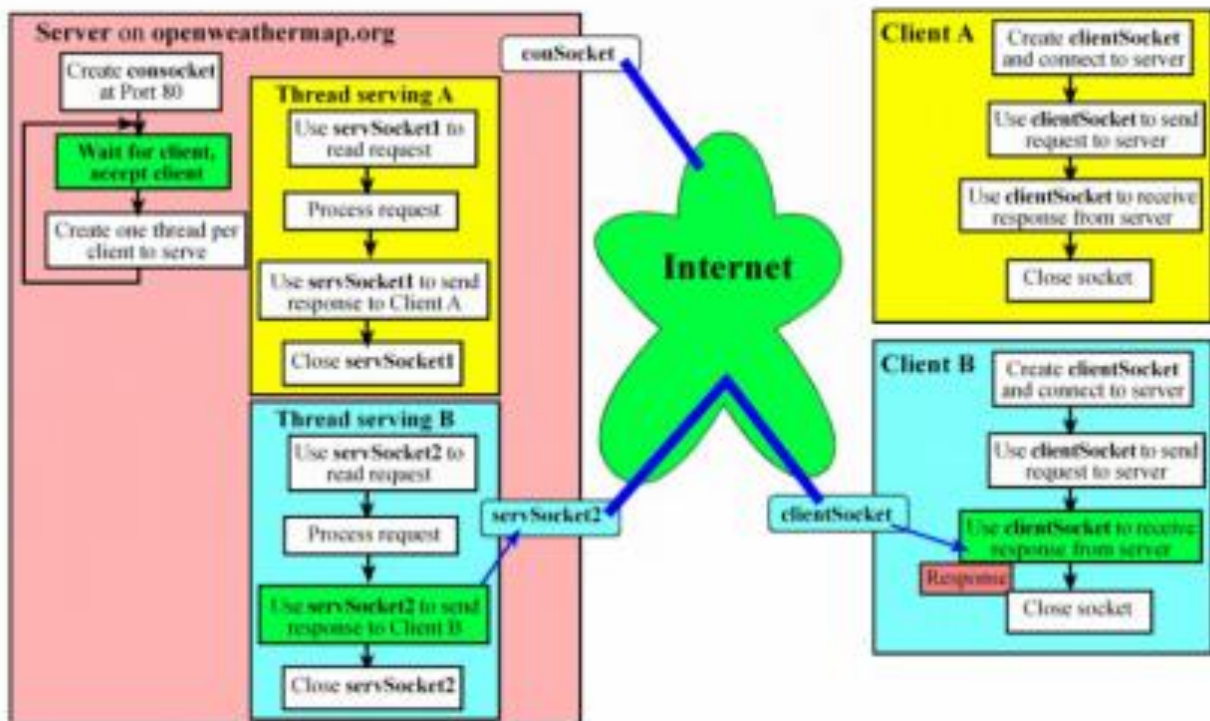
```
#define POST_BUFFER      "&mode=xml&units=imperial&APPID=<API KEY> HTTP/1.1\r\nHOST:api.openweathermap.org\r\nAccept:*/\""
```

Then modify the values of below macros for the device to connect to the Access Point.

```
#define SSID_NAME        "<ap_name>"  
#define SEC_TYPE         SL_SEC_TYPE_OPEN  
#define PASSKEY          ""
```

And finally defining the id for getting the weather information of Norfolk from the server and setting units to imperial to get the data in imperial format

```
#define REQUEST "GET /data/2.5/weather?id=4776222&APPID=6da2fbb39c293af0601b84830b43b1a6&units=imperial"
```



Client-Server (openweathermap.org) Communication

The system is developed using socket connections where the data is fetched in RecvBuff in xml format.

```
713 if(GetHostIP() == 0){
714     if( (appData.SockID = CreateConnection()) < 0 ) return -1;
715
716     /* HTTP GET string. */
717     strcpy(appData.SendBuff,REQUEST);
718     // 1) change city
719     // 2) you can change metric to imperial if you want temperature in F
720     /* Send the HTTP GET string to the open TCP/IP socket. */
721     sl_Send(appData.SockID, appData.SendBuff, strlen(appData.SendBuff), 0);
722
723     /* Receive response */
724     sl_Recv(appData.SockID, &appData.Recvbuff[0], MAX_RECV_BUFF_SIZE, 0);
725     appData.Recvbuff[strlen(appData.Recvbuff)] = '\0';
```

Then the RecvBuff is parsed to get the city name, current temperature and weather conditions.

```
727 /* find ticker name in response */
728 pt = strstr(appData.Recvbuff, "\"name\"");
729 i = 0;
730 if( NULL != pt ){
731     pt = pt + 8; // skip over "name:"
732     while((i<MAXLEN)&&(*pt)&&(*pt!='\"')){
733         City[i] = *pt; // copy into City string
734         pt++; i++;
735     }
736 }
737 City[i] = 0;
738
739 /* find Temperature Value in response */
740 pt = strstr(appData.Recvbuff, "\"temp\"");
741 i = 0;
742 if( NULL != pt ){
743     pt = pt + 7; // skip over "temp:"
744     while((i<MAXLEN)&&(*pt)&&(*pt!='\"')){
745         Temperature[i] = *pt; // copy into Temperature string
746         pt++; i++;
747     }
748 }
749 Temperature[i++] = 0;
750
```

```
751 /* find weather in response */
752 pt = strstr(appData.Recvbuff, "\"description\"");
753 i = 0;
754 if( NULL != pt ){
755     pt = pt + 15; // skip over "description:"
756     while((i<MAXLEN)&&(*pt)&&(*pt!='\"')){
757         Weather[i] = *pt; // copy into weather string
758         pt++; i++;
759     }
760 }
761 Weather[i] = 0;
762 sl_Close(appData.SockID);
763 }
764
```

The challenges faced were linking all the dependent source and header files first and then debugging the files for results. Several files had to be modified to include paths and also make compatible with one another for defining the variables and identifiers, Also there were problems in parsing the buffer data from the openweathermap server to display the results in proper format which were later overcome by my coding and debugging skills along with the instructors help.

### **Task B: Implement a communication system between two IoT devices (e.g., Tiva boards) via an IEEE 802.11 – WiFi module**

The Task was divided between me and Aniket Chandak with me being the server to collect the sensor data and displaying on Putty via UART virtual COM port whereas Aniket's role was to collect the sensor data through Sensor hub and transmitting it to the server.

First, we configured a wireless access point, using our personal smart phone as the access point by setting the name in **starter.c** file

```
#define SSID_NAME "xxxxx"      // Open AP name to connect to.
```

Then defining the server's IP address, port number as well for establishing socket communication.

```
49 #define IP_ADDR      0xc0a82bfb  // 192=0xC0, 168=0xa8, 0x00=0, 0x65 = 101
50 #define PORT_NUM     5001        // Port number to be used
```

Setting up the client:

The client smart object includes a LaunchPad, a CC3100 booster, and sensor interface that creates an analog input on PD0. Then defining Sensor Node to be true to send the UDP packets as client using sockets.

```
52 #define SENSORNODE 1      // true if this node is sending UDP packets, using client
53 #define DISPLAYNODE 0    // true if this node is receiving UDP packets, using server
54 #define CRTNODE 0        // true if this node is runs an interpreter on UART0
```

Setting up the server:

The server smart object includes a LaunchPad, a CC3100 booster. Then defining the Display Node to be true to receive packets as server using sockets.

```
52 #define SENSORNODE 0      // true if this node is sending UDP packets, using client
53 #define DISPLAYNODE 1    // true if this node is receiving UDP packets, using server
54 #define CRTNODE 0        // true if this node is runs an interpreter on UART0
```

First we tested the system connection by defining RAMP to be true and by sending only the UDP packets instead of the sensor data.

```
55 #define EKG 0              // client simulates ekg instead of measuring ADC
56 #define RAMP 1            // client sends ramp data instead of measuring ADC
57 #define ADC 0             // client gets data from ADC, Ain7 = PD0
```

```
COM7 - PuTTY
585.1287
585.3118
585.3118
585.3576
585.2661
585.2355
584.9761
584.9609
585.1135
584.5031
584.6557
585.3118
585.1898
585.1287
585.2508
585.0982
585.1287
585.2966
585.4034
585.2508
585.2661
S9SSection 11.4 IoT example, Volume 2 Real-time interfacing
This node is configured to receive UDP packets
This node should be at IP: 192.168.43.251 Port: 5001

Wlan Connected
SL NETAPP_IPV4 ACQUIRED
This node is at IP: 192.168.43.43
Receiving a UDP packet ...
```

Server Side

```
COM5 - PuTTY
Sending a UDP packet ...
ADC Data: 81 a= 81 ok
Sending a UDP packet ...
ADC Data: 80 a= 80 ok
Sending a UDP packet ...
ADC Data: 79 a= 79 ok
Sending a UDP packet ...
ADC Data: 81 a= 81 ok
Sending a UDP packet ...
ADC Data: 80 a= 80 ok
Sending a UDP packet ...
ADC Data: 82 a= 82 ok
Sending a UDP packet ...
ADC Data: 83 a= 83 ok
Sending a UDP packet ...
ADC Data: 78 a= 78 ok
Sending a UDP packet ...
ADC Data: 79 a= 79 ok
Sending a UDP packet ...
ADC Data: 80 a= 80 ok
Sending a UDP packet ...
ADC Data: 81 a= 81 ok
Sending a UDP packet ...
ADC Data: 81 a= 81 ok
```

Client Side



Then for the client side to collect the sensor data through sensor hub we integrated the light sensor code implemented in lab 2 to make it work. First we configured the UART by initializing ConfigureUART() and to display the floating point values returned by the ISL29023DataLightVisibleGetFloat(&g\_sISL29023Inst, &fAmbient) where fAmbient contained the floating lux values.

```
void
ConfigureUART(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    //
    // Enable UART0
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}
```

And then defined the SYSTICKS\_PER\_SECOND and SYSTICK\_PERIOD\_MS, the SysTickIntHandler(). Also configured and enabled SysTick Timer in main function.

```
    //Configure and enable SysTick Timer
    //
    ROM_SysTickPeriodSet(ROM_SysCtlClockGet() / SYSTICKS_PER_SECOND);
    ROM_SysTickIntEnable();
    ROM_SysTickEnable();

void
SysTickIntHandler(void)
{
    //
    // Go get the latest data from the sensor.
    //
    ISL29023DataRead(&g_sISL29023Inst, ISL29023AppCallback, &g_sISL29023Inst);
}
```



Also defined ADC parameter to be true on PD0.

```
55 #define EKG 0           // client simulates ekg instead of measuring ADC
56 #define RAMP 0          // client sends ramp data instead of measuring ADC
57 #define ADC 1           // client gets data from ADC, Ain7 = PD0
```

The program then reads the values, stores in the buffer and sends the data to the using sockets.

```
uBuf[0] = ATYPE;           // analog data type
uBuf[1] = '=';            //
data = ADC0_InSeq3(); // 0 to 4095, Ain7 is on PD0
Int2Str(data, (char*)&uBuf[2]); // 6 digit number
//UARTprintf((char*)&uBuf[2]);
sl_SendTo(SockID, uBuf, BUF_SIZE, 0,           //
          (SlSockAddr_t *)&Addr, AddrSize); //
ROM_SysCtlDelay(ROM_SysCtlClockGet() / 25); // 40ms
```

Then for the server to receive the sensor data, we configured the socket and buffer code for receiving the data without any errors.

```
SockID = sl_Socket(SL_AF_INET, SL SOCK_DGRAM, 0);
Status = sl_Bind(SockID, (SlSockAddr_t *)&LocalAddr,
                 AddrSize);
Status = sl_RecvFrom(SockID, uBuf, BUF_SIZE, 0,
                    (SlSockAddr_t *)&Addr, (SlSocklen_t *)&AddrSize )
if((uBuf[0]==ATYPE) && (uBuf[1]== '=')) {
    int i,bOk; uint32_t place;
    data = 0; bOk = 1;
    i=4; // ignore possible negative sign
    for(place = 1000; place; place = place/10){
        if((uBuf[i]&0xF0)==0x30){ // ignore spaces
            data += place*(uBuf[i]-0x30);
        }else{
            if((uBuf[i]&0xF0) != ' '){
                bOk = 0;
            }
        }
        i++;
    }
}
```

To plot real-time sensor reading we used the python PySerial API to display the serial data from the board in floating point.

```
1 import serial
2 import re
3 ser = serial.Serial('COM7', 115200)
4
5 while True:
6     signal = ser.readline()
7
8     print(signal.decode("utf-8"))
```

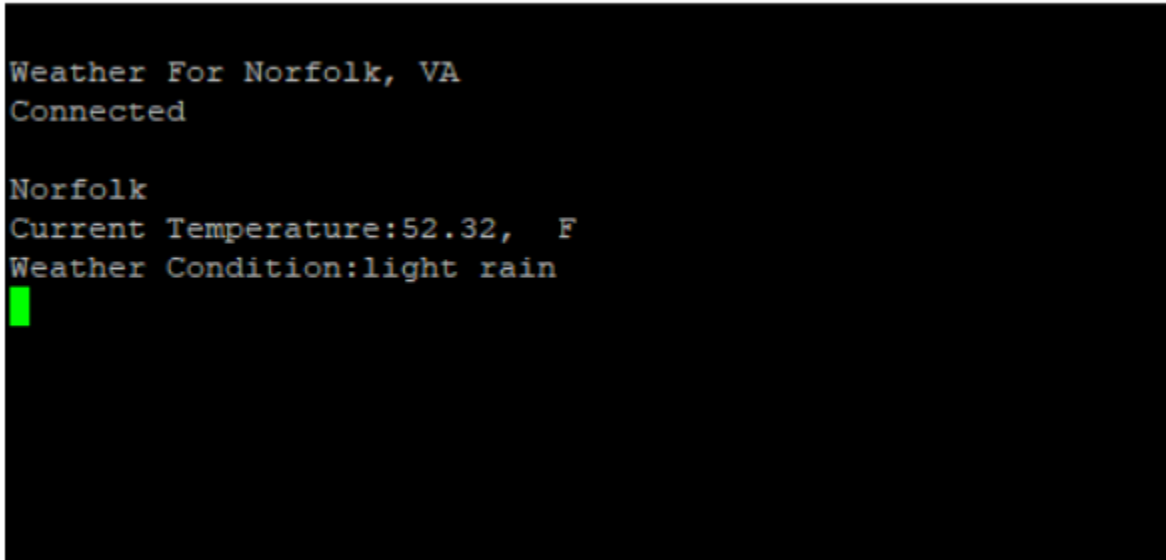
We used the matplotlib.pyplot python library. The readings were more sensitive when the time.sleep() was reduced to 0.01s.

The challenges involved in the implementation were setting the right paths to several header files. Also binding the client socket to server socket with the right IP address and port number. Further there were difficulties in implementing and configuring the system to read the light sensor data as there several errors to debug in the process like undefined symbols, missing handlers in startup\_rvmdk.s as well as ADC() function errors. The errors were overcome by several hours of coding and debugging skills along with correct setting of the analog data reading pin and clearing those which were overlapping with it.

#### **IV. System verification and demonstration**

For Task A, the system gave the correct temperature in fahrenheit and weather condition information as per the openweathermap.org organization.

 COM7 - PuTTY



```
Weather For Norfolk, VA
Connected

Norfolk
Current Temperature:52.32, F
Weather Condition:light rain
```

For Task B the system successfully established a client server communication along with real time sensor transmission and real time plotting of the light sensor data.

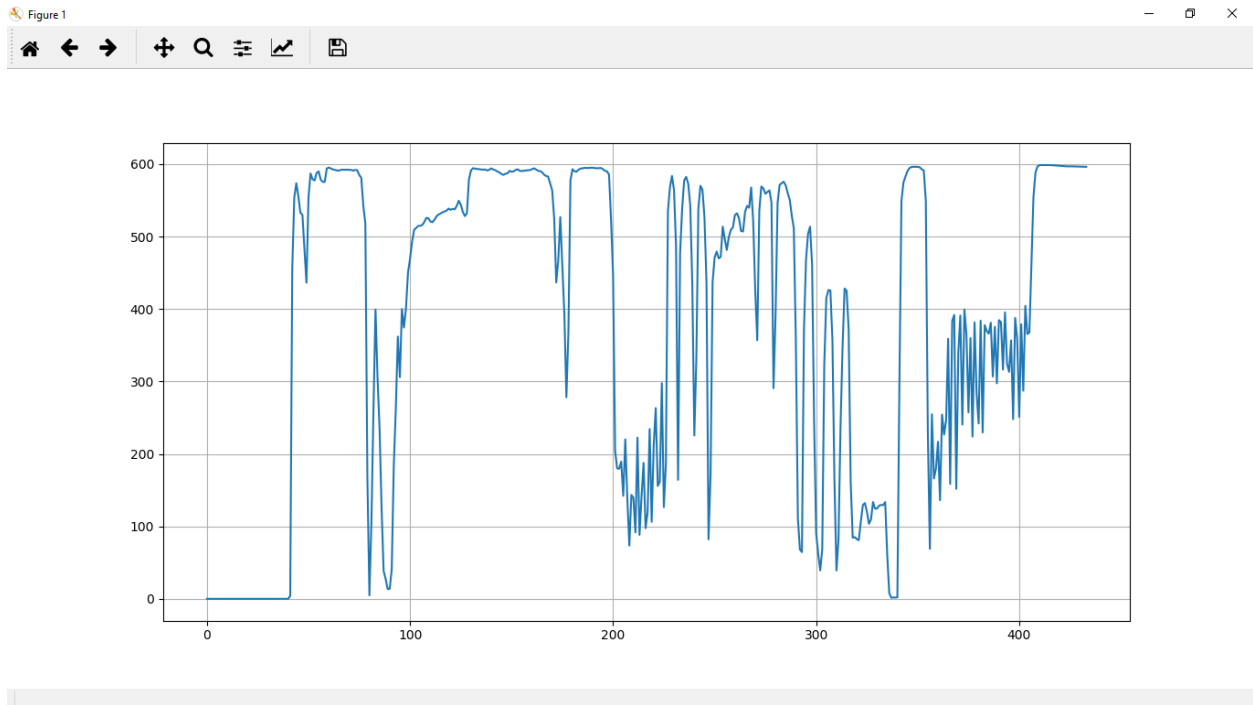
```
COM7 - PuTTY
585.1287
585.3118
585.3118
585.3576
585.2661
585.2355
584.9761
584.9609
585.1135
584.5031
584.6557
585.3118
585.1898
585.1287
585.2508
585.0982
585.1287
585.2966
585.4034
585.2508
585.2661
S8SSection 11.4 IoT example, Volume 2 Real-time interfacing
This node is configured to receive UDP packets
This node should be at IP: 192.168.43.251 Port: 5001

Wlan Connected
SL_NETAPP_IPV4_ACQUIRED
This node is at IP: 192.168.43.43
Receiving a UDP packet ...
```

Server ready to receive sensor data on IP address 192.168.43.251 and Port 5001 via COM7

```
590.0268
590.0878
590.1489
590.3015
589.5843
587.6922
588.5009
586.5631
586.7767
586.5783
588.1347
587.5701
586.4868
586.0748
```

Client sending light receive sensor data on IP address 192.168.43.251 and Port 5001 via COM5



Real-time plot of the sensor data

## V. Conclusion

Overall the lab was successfully demonstrated and helped to implement concepts like making calls to the socket API, making a connection, transmit/receive application data and close the connection as well as collecting sensor data with the combination of Tiva C Series TM4C123GH6PM, CC3100 SimpleLink Wi-Fi Boostpack, Tiva Sensor Hub. The lab really tested some of my coding and debugging skills which gives me confidence to further work in respective industrial organizations.