Cloud (ThingSpeak) on Beagle bone Wireless : Final Phase

- Zain Rajani (c0752681)

Group # 8

Outline

- Until Now
- Task Objective
- Task Introduction
- Task Requirement
- Introduction: ThingSpeak
- Interface Programming: Flowcharts
- Coding Connect to Cloud
- Understanding Concepts (WiFi, HTTP, MQTT)
- What did we observe
- Troubleshooting
- Conclusion
- References

Until Now.....

- Until now we have tried to completed all the desired phases of the interfacing which was to get the LED, IR sensor, GSM and LDR module interfaced with the Beagle bone Black Wireless
- For the interfaces we used the user made header files and tried to use the POSIX (Portable Operating System Interface) API function library such as the unistd.h (Unix Standard Header) and using the time module for GSM Module in python programming language.
- Using the header files we tried to control the GPIO pins of the BB-WI and made a complete interface in C programming Language with a small support from python for GSM Module.
- For GSM we used python language to send messages to the user and then we integrated this program in our C program using the system(".....") command avaliable in the stdlib.h library of C/C++ language.
- Lastly, we demonstrated the process of interface live and showed that whenever a letter arrives in the letterbox the sensor reads low (being active high in nature) and showed that even if there is a single letter in the letter box the LED will be set high also how the GSM sends message for each letter that arrives and send the SMS when the letters are removed and thus resets the counter back to zero.

Task Objective

- To connect (interface) **B**eaglebone **B**lack **Wi**reless (BB-WI) to the Cloud service (**T**hing**S**peak)
- To make the coding done in previous parts of the interfacing compatible with the objective mentioned above
- When connected to the cloud service the BB-WI must send the number of letters present in the letter box which also include when the letter is removed it must send a zero* after a fixed duration of time.

**To indicate that no letters are present in the letterbox*

Task Introduction

- In the previous meeting we saw how we broke the interfacing of sensors and peripherals into various tasks and thus we achieved their completion.
- In today's task we try to accomplish the final phase of making the project and taking to it to near completion of the interfacing process i.e. we try to connect ThingSpeak with Beaglebone Black Wireless.
- This task is intended to complete in 2 stages first one being just interfacing the BB-WI with ThingSpeak and in the second stage being trying to accommodate the previously made programs with ThingSpeak.
- For this purpose of task we intend to use ThingSpeak as the Cloud Service. The connection is intended to be completed using the IEEE 802.11 (**WiFi** – Wireless Fidelity) and the data would be transferred using the MQTT (**M**essage **Q**ueuing **T**elemetry **T**ransport)
- When connected to the service we would send the number of letters present in the letter box.

Task Requirements: Hardware

- In order to accomplish this one must have the following hardware present with you:
  - BeagleBone Black Wireless (BB-WI)
  - Connecting Wires
  - Wall Adapters
  - ThingSpeak (Cloud Service)
  - Micro USB Cable*
  - Part 1: Sensor and LED**
  - Part 2: GSM Module **
  - Part 3: LDR Module **

* *Required in case of power not sufficient*

** *Required to demonstrate the Part 1, Part 2 and Part 3 together*

Task Requirements: Software

- As for our task objective since we need to get connected to the cloud via the BeagleBone Black Wireless. Once connected we need to send the desired data to the cloud.
- As mentioned in the proposal and previous meetings we might use Eclipse if the coding cannot be performed on the Beagle bone Wireless itself but since this task can be completed using the nano editor present and the GCC (GNU Compiler Collection) Complier
- To finalize we require the following software tools:
  - Debian OS (Installed on the BB-WI)
  - GCC (GNU [Not Unix] Compiler Collection) to run the code
  - Nano Editor (To write the code)
  - Any required libraries for making interfacing simpler
  - ThingSpeak Access

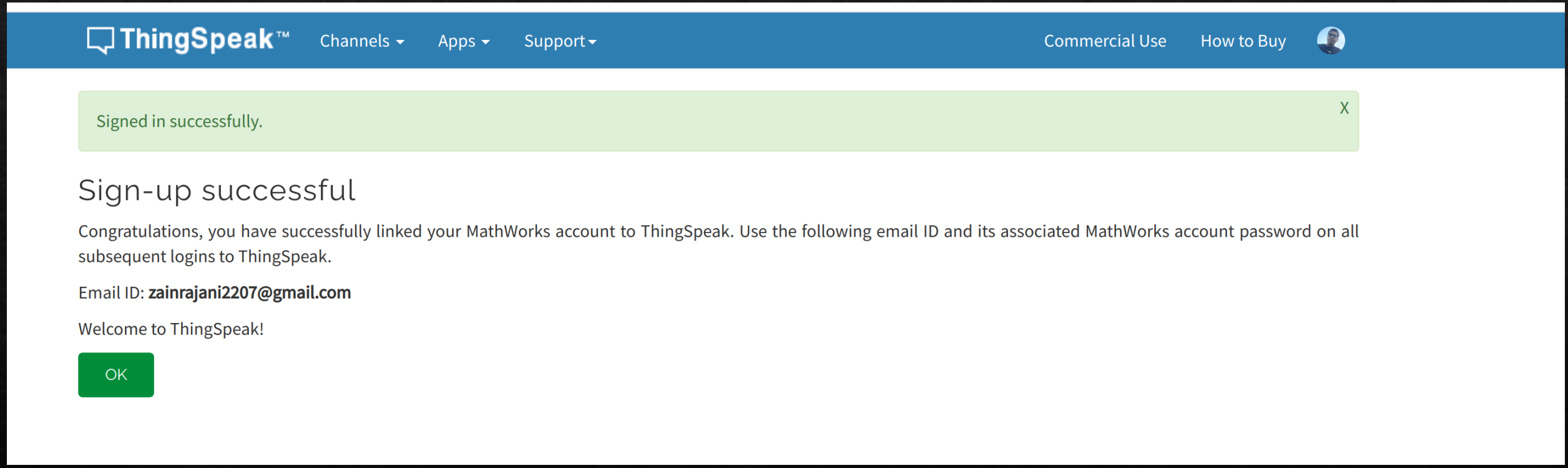
Know ThingSpeak

- Since the user community of the Beaglebone Black Wireless is strong enough thus we have large amount of contribution made each day. Thus, it most common to see often that the user community uses ThingSpeak as their common interface to connect to the cloud service.
- Though some prefer other cloud services like AWS (Amazon Web Servcies), Google Drive, Microsoft Azure, etc. Which depends on their comfort of working or depends on the needs of the project/ product
- Before we get to the ground lets understand ThingSpeak Cloud Service first. ThingSpeak is an open-source Internet of Things (IoT) application and API to store and retrieve data from things using the HTTP (Hyper Text Transfer Protocol) and MQTT protocol over the Internet or via a Local Area Network.
- ThingSpeak enables the creation of sensor logging applications, location tracking applications, and a social network of things with status updates.

Know ThingSpeak

- ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize and analyze live data streams in the cloud. ThingSpeak provides instant visualizations of data posted by your devices to ThingSpeak.
- ThingSpeak first launched in 2010 by ioBridge was first seen as an integration with the MATLAB software. After then it was enabled for most of the IoT based prototypes.
- ThingSpeak has a close relationship with Mathworks, Inc. In fact, all of the ThingSpeak documentation is incorporated into the Mathworks' Matlab documentation site and even enabling registered Mathworks user accounts as valid login credentials on the ThingSpeak website.
- ThingSpeak is often used for prototyping and proof of concept IoT systems that require analytics. It can act on the data and communicate using third party apps like Twitter and Twilio.

Getting Started: ThingSpeak



New Channel

Name

Description

Field 1 — Field Label 1 — ☑

Field 2 — ☐

Field 3 — ☐

Field 4 — ☐

Field 5 — ☐

Field 6 — ☐

Field 7 — ☐

Field 8 — ☐

Metadata

Tags

(Tags are comma separated)

Link to External Site — http://

Link to GitHub — https://github.com/

Elevation

Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- **Show Channel Location:**
  - **Latitude:** Specify the latitude position in decimal degrees. For example, the latitude of the city of London is 51.5072.
  - **Longitude:** Specify the longitude position in decimal degrees. For example, the longitude of the city of London is -0.1275.
  - **Elevation:** Specify the elevation position meters. For example, the elevation of the city of London is 35.052.
- **Video URL:** If you have a YouTube™ or Vimeo® video that displays your channel information, specify the full path of the video URL.
- **Link to GitHub:** If you store your ThingSpeak code on GitHub®, specify the GitHub repository URL.

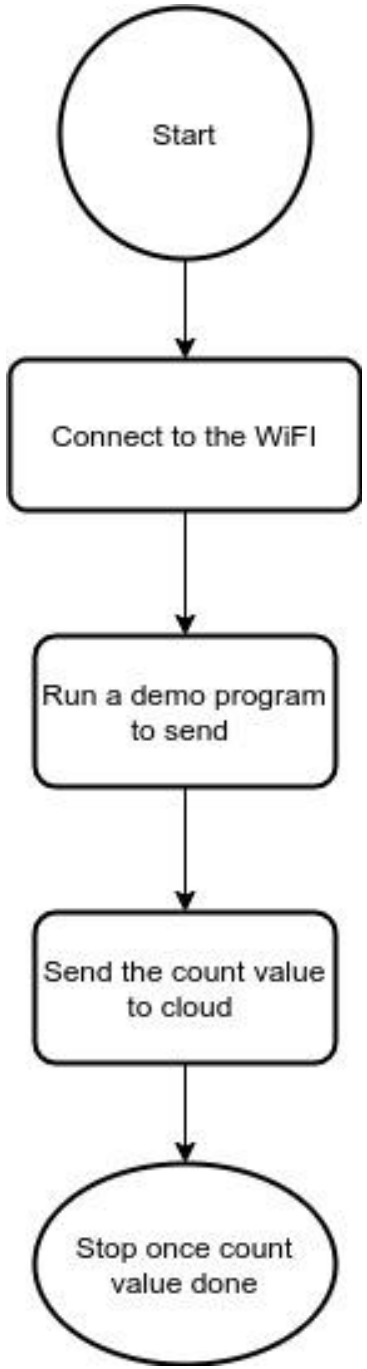
Using the Channel

You can get data into a channel from a device, website, or another ThingsSpeak channel. You can then visualize data and transform it using ThingSpeak Apps.

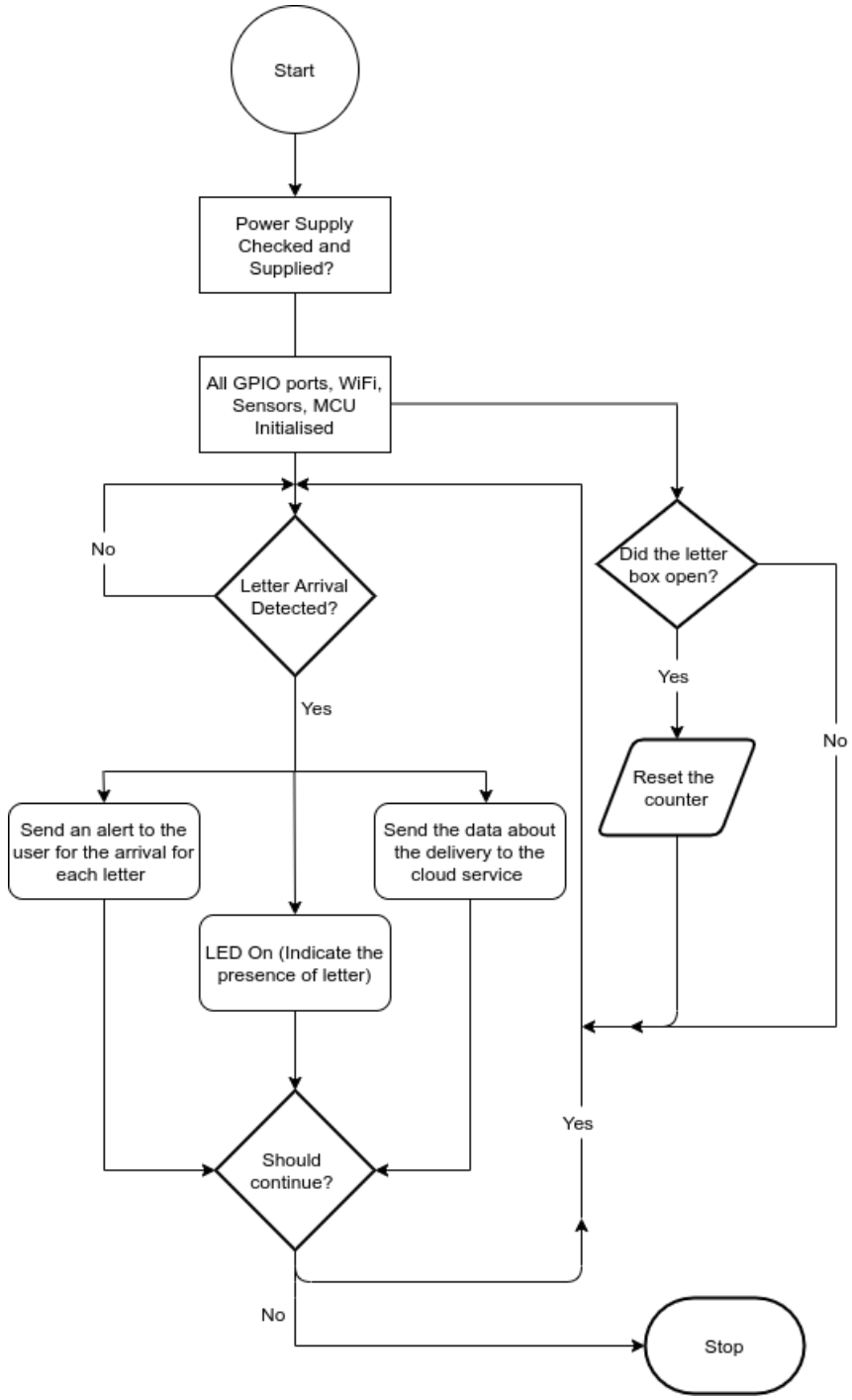
Getting Started: Programming

- Before starting to solve any problem, it is a good idea to have a rough map or approach as to how the problem can be solved. Thus to have a suitable map designed for us which helps us to make ourselves clear we draw a flowchart.
- The flowchart basically depicts our approach to solve the problem logically and helps us understanding better how our data is being processed by the machine (roughly)
- Incase if we fail to demonstrate the task in our first attempt the flowchart will helps go through all the steps we have performed and the same will help finding possibly where we might have gone wrong thus indirectly helping us use the debugging concept without enabling the software debugger.
- But not to ignore the software/hardware debugger will be simpler to use and less headache compared to the flowchart debugging approach. This approach helps us understand the system in detail in terms of functionality and process of data on how it occurs

Interface Programming : Flow Chart



Interface Programming : Flow Chart



Interface Programming: Coding

- This time is something different from all other meetings as we don't have any physical component, we are dealing with networks thus we needed library files which helps us connecting to the websites/ cloud services
- So, after researching and reading the text by Derrek Molley (Exploring Beaglebone) we found that we can use the `SocketClient.h` to get connected to the cloud service i.e. ThingSpeak
- Once all the libraries have been imported, we next try to get the code ready as per the flow diagram we designed in the previous steps.

Getting Beaglebone Connected to the Cloud : Example

debian@beaglebone: ~

File Edit View Search Terminal Help

GNU nano 3.2 dummy.cpp Modified

```
#include <iostream>
#include <sstream>
#include <fstream>
#include "exploringBB/chp11/thingSpeak/network/SocketClient.h"
#include <unistd.h>

using namespace std;
using namespace exploringBB;

int main()
{
   int i=0;
   for(i=0;i<=20;i++){
   ostringstream head,data;
   cout<<"Starting ThingSpeak Example"<<endl;
   SocketClient sc ("api.thingspeak.com",80);
   data << "field1="<<i<<endl;
   cout<<"Sending the data "<<i<<endl;
   sc.connectToServer();
   head<<"POST /update HTTP/1.1\n"<<"Host: api.thingspeak.com\n"
   <<"Connection: close\n"<<"X-THINGSPEAKAPIKEY:VMCNRV25Y3J1XTOZ\n"
   <<"Content-type: application/x-www-form-urlencoded\n"<<"Content-Length:"
   <<string(data.str()).length()<<"\n\n";
   sc.send (string(head.str()));
   sc.send (string(data.str()));
   string rec=sc.receive(1024);
   cout<<"["<<rec<<"]"<<endl;
   cout<<"End of ThingSpeak Example"<<endl;
   sleep(15);}
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell

Glimpse of Final Phase Code

debian@beaglebone: ~

File Edit View Search Terminal Help

GNU nano 3.2 test.cpp

```
/* Program to illustrate the interfacing of BB-WI
   and IR (Infrared Sensor) and LED  */

   // Author: Zain Rajani
   // Date: Nov. 01, 2020

/* Libraries to be included here */

#include <stdlib.h>        // For system(...) command to run some external com$
#include <stdio.h>        // For Standard Input Output Function like printf(),$
#include <unistd.h>      // For usleep function
#include <iobb.h>       // To access the GPIO pins easily library already down$
#include <iostream>
#include <sstream>
#include <fstream>
#include "exploringBB/chp11/thingSpeak/network/SocketClient.h"

using namespace std;
using namespace exploringBB;

int thingspeak (int count)
{
   ostringstream head,data;
   SocketClient sc ("api.thingspeak.com",80);
   data << "field1="<<count<<endl;
   sc.connectToServer();
   head<<"POST /update HTTP/1.1\n"<<"Host: api.thingspeak.com\n"
   <<"Connection: close\n"<<"X-THINGSPEAKAPIKEY:VMCNRV25Y3J1XTOZ\n"
   <<"Content-type: application/x-www-form-urlencoded\n"<<"Content-Length:"
   <<string(data.str()).length()<<"\n\n";
   sc.send (string(head.str()));
   sc.send (string(data.str()));
   sleep(10);
   return 0;
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell

Glimpse of Final Phase Code

```
 GNU nano 3.2                          test.cpp                          Modified

int main()
{
  // Variable Declarations;
  int counter=0;

  //Initialise the GPIO function libraries
  iolib_init();

  //Set the pin function either as input or output pins
  iolib_setdir(9,15,DigitalIn);
  iolib_setdir(8,11,DigitalOut);
  iolib_setdir(9,41,DigitalIn);

  //Setting initially output pin low
  pin_low(8,11);          //P8.11 Low (0)

  // Loops forever
  while(1)
  {
     // Check the input value of the pin
    // Condition: If IR Sensor sensed arrival of the letter and brightness is there then
    if ((is_low(9,15)) && (is_low(9,41)))
    {
      pin_high(8,11);
      system("/usr/bin/python2.7 pysms.py");
      counter++;
      printf("Letter # %d \n",counter);
      thingspeak(counter);
      usleep(100000);
   }
    // If no letter is sensed and there is darkness then
    if ((is_high(9,15)) && (is_high(9,41)))
    {
      // If sensor detects no object do nothing only reset the counter if there is darkness and send$
      counter=0;
      printf("Counter Resetted # %d\n",counter);
      pin_low(8,11);
      system("/usr/bin/python2.7 pysend.py");
      thingspeak(counter);
    }

  }
// Free up the resources like the GPIO pins for other use if required from memory
iolib_free();

return (0);     // If code success then return 0 else -1
}
```

Understanding Cloud Working

- Before we get started, we need to make sure that we are connected to the wireless network (in our case as we don't have a wired connection feature) and understand the various protocols that we use as this would help us understand cloud much better.
- For the cloud connection we use the WiFi communication (IEEE 802.11), to establish the connection we use HTTP and for transferring the data we use MQTT protocols. Let's understand each one of them and then draw a difference between them.

WiFi (IEEE 802.11)

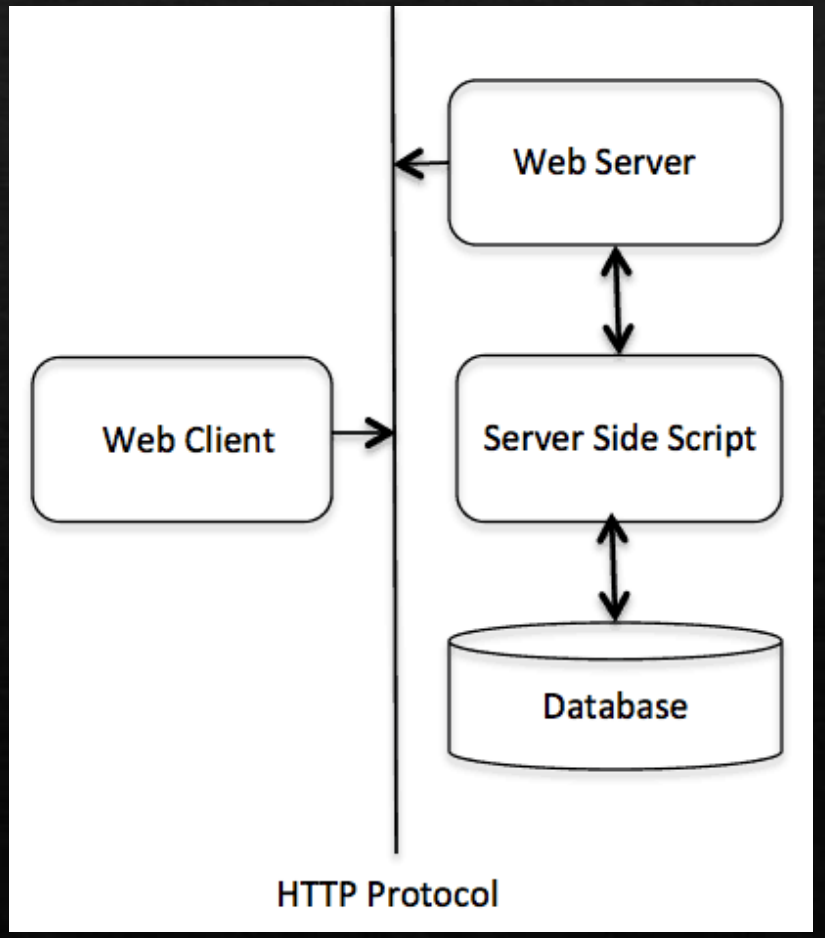
- 802.11 is a standard that was developed by the Institute of Electrical and Electronic Engineers (IEEE). It is the original wireless specification. Extensions of the 802.11 standard were given the same number with a letter suffix.
- The numbers 802.11 come from the Institute of Electrical and Electronics Engineers who established this standard to describe connections in wireless networking.
- WiFi stands for Wireless Fidelity and is the same thing as saying WLAN which stands for "Wireless Local Area Network."
- A wireless network uses radio waves, just like cell phones, televisions and radios do. In fact, communication across a wireless network is a lot like two-way radio communication. Here's what happens:
    1. A computer's wireless adapter translates data into a radio signal and transmits it using an antenna.
    2. A wireless router receives the signal and decodes it. The router sends the information to the Internet using a physical, wired Ethernet connection.

Wifi Setup

- The process also works in reverse, with the router receiving information from the Internet, translating it into a radio signal and sending it to the computer's wireless adapter.
- Each Wifi has a unique SSID (Service Set Identifier) which by default is the manufacturer name and can be changed by the user during the setup of wifi to easily recognise their wifi from their neighbours.
- To connect the Beaglbone to Wifi network we need ti start the connection manager (connmanctl) and require the desired SSID of the network to which one wishes to connect the following commands will help one connect to the network.
- One may face some issues connecting to the network which we shall discuss in the troubleshooting section

HTTP : Introduction

- HTTP stands for Hyper Text Transfer Protocol. The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems that allows users to communicate data on the World Wide Web.
- It was primarily designed alongside for HTML (Hyper Text Markup Language) but today remains the source to get connected to the internet.
- The default port number for connection of HTTP is 80 and HTTPS is 443 which is an extension of HTTP for secure communication over computer networks

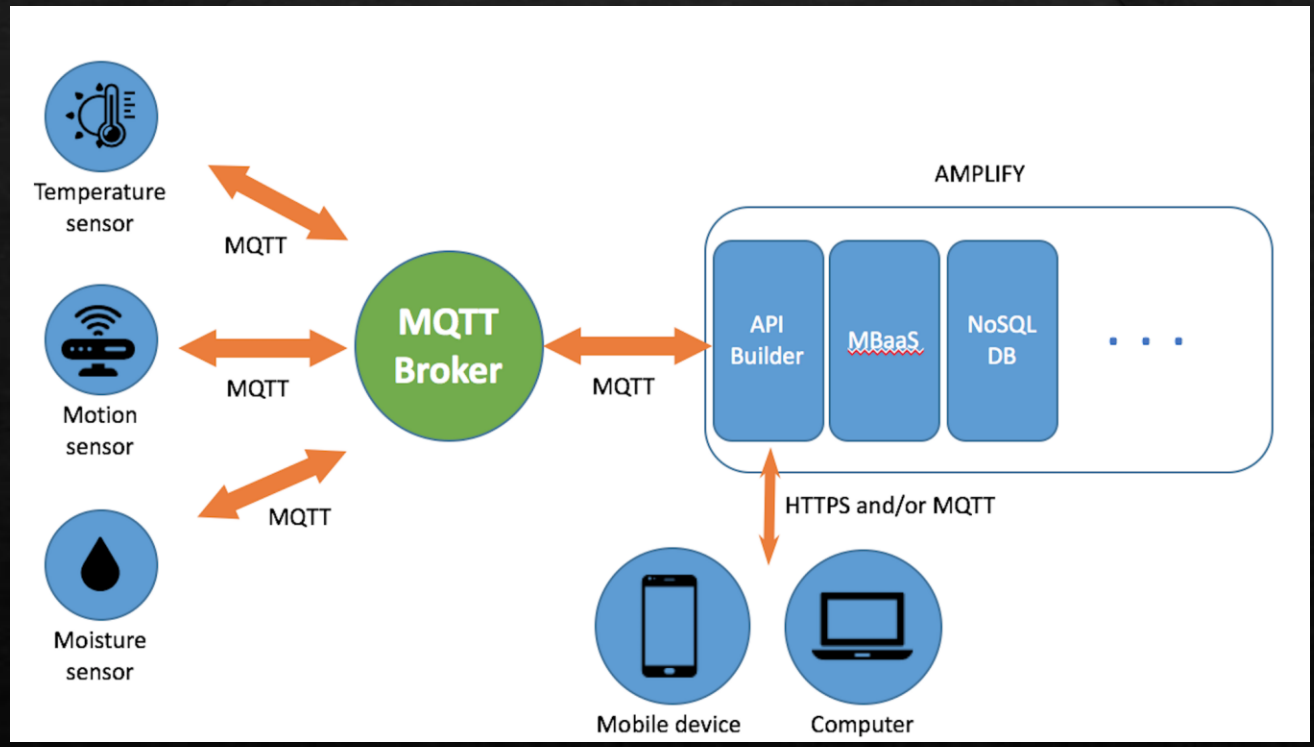


HTTP : Introduction

- As a request-response protocol, HTTP gives users a way to interact with web resources such as HTML files by transmitting hypertext messages between clients and servers. HTTP clients generally use Transmission Control Protocol (TCP) connections to communicate with servers.
- There are various methods by which the HTTP protocol perform some of them include: POST, HEAD, GET, DELETE, OPTIONS, CONNECT, etc.
- Most of the HTTP servers supports GET and HEAD methods but the other methods may not be supported.
- A web browser is an HTTP client, sending requests to server machines. When the browser user enters file requests by either "opening" a web file (typing in a URL [Uniform Resource Locator]) or clicking on a hypertext link, the browser builds an HTTP request and sends it to the Internet Protocol address (IP address) indicated by the URL.

MQTT : Introduction

- It is a lightweight publish and subscribe system where you can publish and receive messages as a client.
- MQTT is a simple messaging protocol, designed for constrained devices with low-bandwidth. So, it's the perfect solution for Internet of Things applications. MQTT allows you to send commands to control outputs, read and publish data from sensor nodes and much more.



MQTT : Introduction

- MQTT was created with a goal to collect data from various devices and then deliver it to the IT infrastructure. MQTT was first used in the oil and gas industry but today is being used in most of the application like Facebook and even AWS has mentioned that Amazon IoT is based on MQTT.
- It is a publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol.
- Before we start to understand the working of MQTT there are some terms which one must know before getting started like: subscriber, client, publisher, Server/Broker and Topic

MQTT : Terminology

- Subscriber: The one which receives messages that are intended for it.
- Publisher: The one which publishes messages to the outer world.
- Client: A client can be either publisher or subscriber or both. That is a client publish a message and receive another message at the same time.
- Server/Broker: The one receives the messages published by the publisher first, even before the subscriber. Then the server publishes the messages to the subscribers after filtering the messages.
- Topic: An UTF-8 string used by the clients and servers to send and receive messages. Eg: sensors/altimeter/1.

MQTT : Introduction

- Clients connect to the broker, which then mediates communication between the two devices. Each device can subscribe, or register, to a particulaar topics. When another client publishes a message on a subscribed topic, the broker forwards the message to any client that has subscribed.
- MQTT makes it possible to collect, transmit, and analyze more of the data being collected.
- Unlike the usual poll/response model of many protocols, which tend to unnecessarily saturate data connections with unchanging data, MQTT's publish/subscribe model maximizes the available bandwidth.
- After understanding both the MQTT and HTTP one may ask since they perform the same task then how do they differ so let's draw out some differences between those to help understand why we use HTTP for connection and MQTT for data transfer

MQTT v/s HTTP

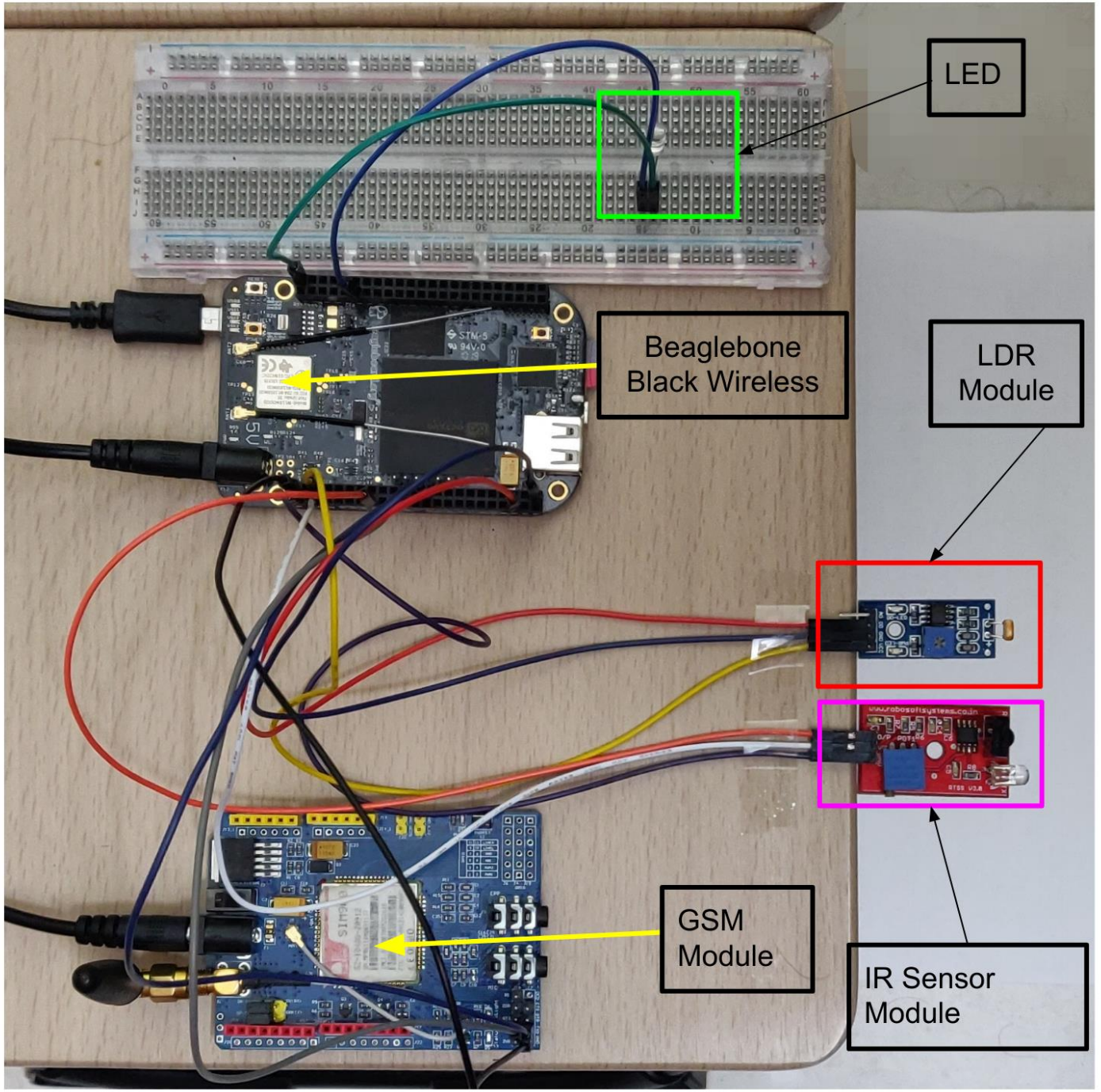
MQTT

- MQTT is a publish/subscribe model having less complexity
- This protocol is designed for data centric based on transmission control protocol
- It has some security features through SSL (Secure Socket Layer) /TLS (Transport Layer Security)
- It works on port number 1883
- MQTT also provides users with options of Last will & Testament and Retained messages. The first means that in case of unexpected disconnection of a client all subscribed clients will get a message from a broker. Retained message means that a newly subscribed client will get an immediate status update.
- It is 93 times faster than HTTP

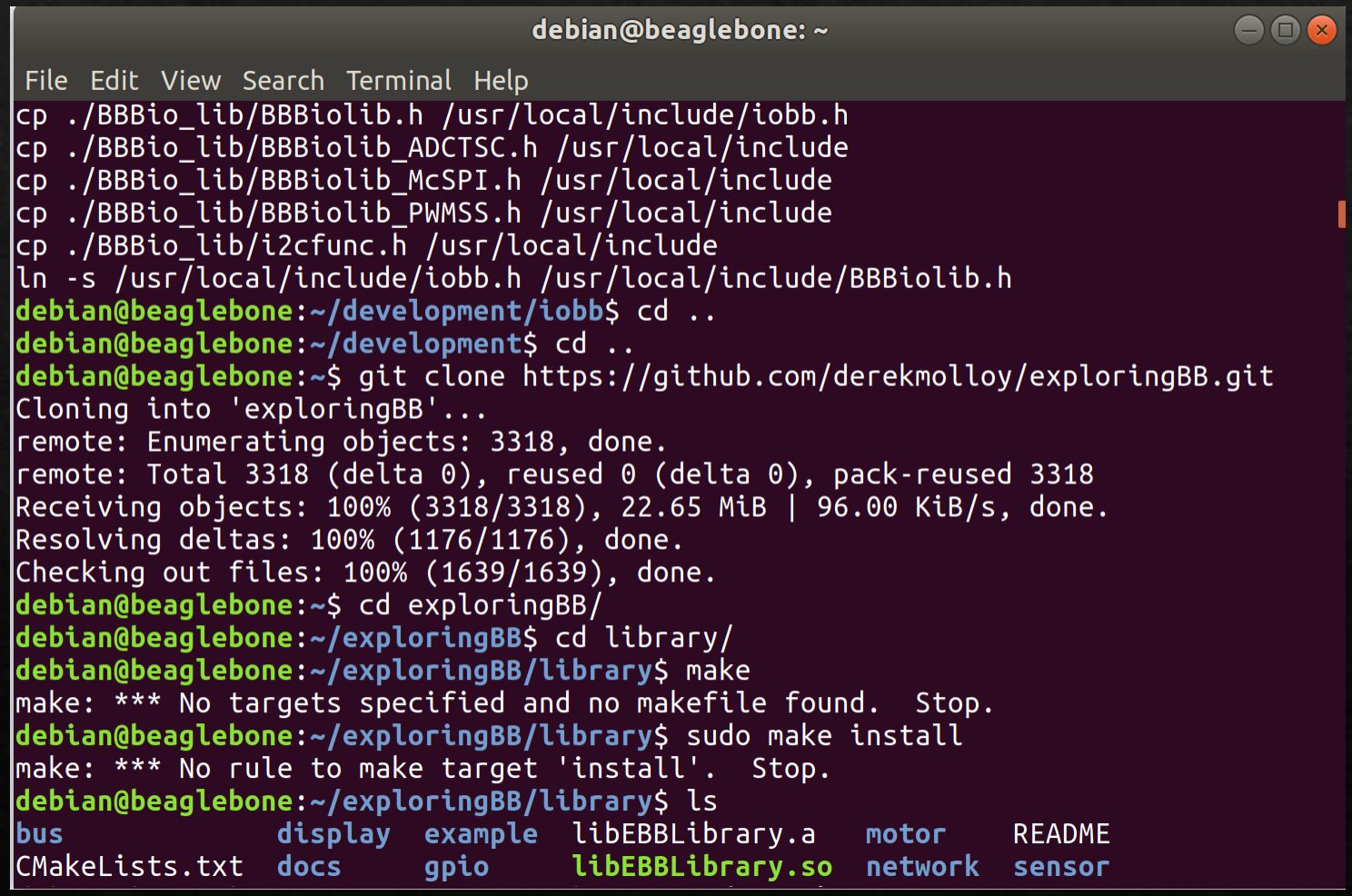
HTTP

- HTTP is request/response model having more complexity
- This protocol is based on document centric based on the user datagram protocol (UDP)
- It does not provide any security feature, but the extension HTTPS does which works on port 43
- It works on port number 80 or 8080
- HTTP has none of these capabilities
- It is slower as compared to MQTT

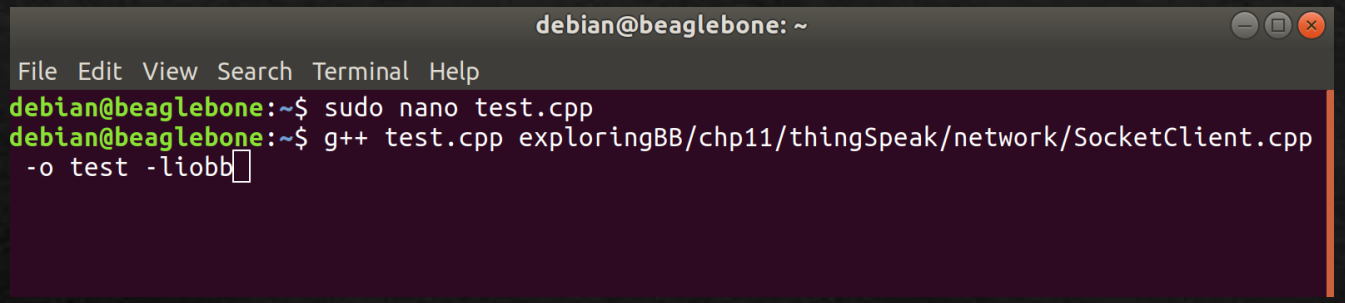
Project Hardware Wiring

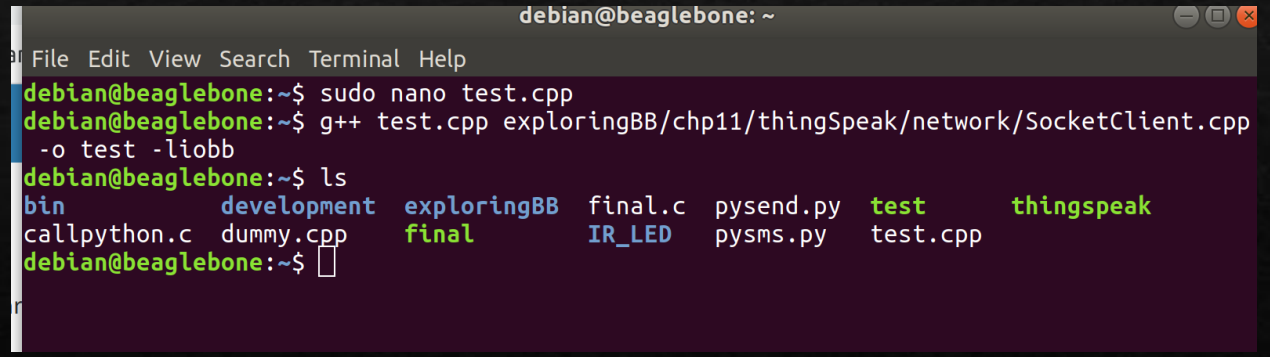


Execution


debian@beaglebone: ~
File Edit View Search Terminal Help
cp ./BBBio_lib/BBBiolib.h /usr/local/include/iobb.h
cp ./BBBio_lib/BBBiolib_ADCTSC.h /usr/local/include
cp ./BBBio_lib/BBBiolib_McSPI.h /usr/local/include
cp ./BBBio_lib/BBBiolib_PWMSS.h /usr/local/include
cp ./BBBio_lib/i2cfunc.h /usr/local/include
ln -s /usr/local/include/iobb.h /usr/local/include/BBBiolib.h
debian@beaglebone:~/development/iobb$ cd ..
debian@beaglebone:~/development$ cd ..
debian@beaglebone:~$ git clone https://github.com/derekmolloy/exploringBB.git
Cloning into 'exploringBB'...
remote: Enumerating objects: 3318, done.
remote: Total 3318 (delta 0), reused 0 (delta 0), pack-reused 3318
Receiving objects: 100% (3318/3318), 22.65 MiB | 96.00 KiB/s, done.
Resolving deltas: 100% (1176/1176), done.
Checking out files: 100% (1639/1639), done.
debian@beaglebone:~$ cd exploringBB/
debian@beaglebone:~/exploringBB$ cd library/
debian@beaglebone:~/exploringBB/library$ make
make: *** No targets specified and no makefile found. Stop.
debian@beaglebone:~/exploringBB/library$ sudo make install
make: *** No rule to make target 'install'. Stop.
debian@beaglebone:~/exploringBB/library$ ls
bus display example libEBBLibrary.a motor README
CMakeLists.txt docs gpio libEBBLibrary.so network sensor

Execution

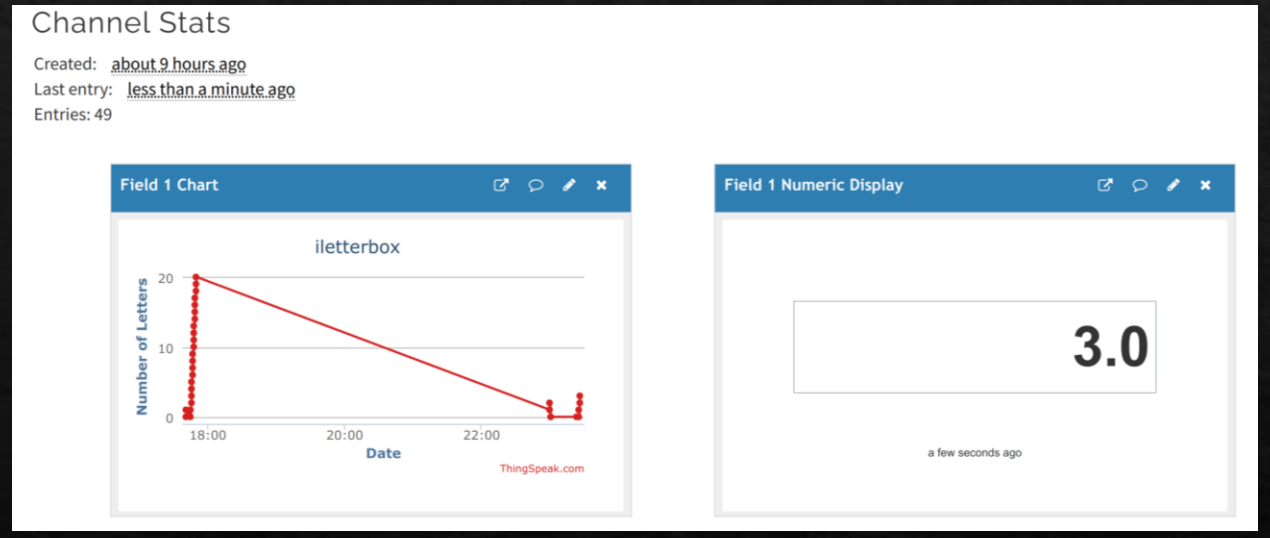

debian@beaglebone: ~
File Edit View Search Terminal Help
debian@beaglebone:~$ sudo nano test.cpp
debian@beaglebone:~$ g++ test.cpp exploringBB/chp11/thingSpeak/network/SocketClient.cpp
-o test -liobb


debian@beaglebone: ~
File Edit View Search Terminal Help
debian@beaglebone:~$ sudo nano test.cpp
debian@beaglebone:~$ g++ test.cpp exploringBB/chp11/thingSpeak/network/SocketClient.cpp
-o test -liobb
debian@beaglebone:~$ ls
bin development exploringBB final.c pysend.py test thingspeak
callpython.c dummy.cpp final IR_LED pysms.py test.cpp
debian@beaglebone:~$

Execution

```
debian@beaglebone: ~
File  Edit  View  Search  Terminal  Help
debian@beaglebone:~$ sudo ./thingspeak
Starting ThingSpeak Example
Sending the data 0
[HTTP/1.1 200 OK
Date: Mon, 23 Nov 2020 04:25:41 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 2
Connection: close
Status: 200 OK
X-Frame-Options: SAMEORIGIN
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH
Access-Control-Allow-Headers: origin, content-type, X-Requested-With
Access-Control-Max-Age: 1800
ETag: W/"25fc0e7096fc653718202dc30b0c580b"
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: b300662e-caec-4c6b-9a71-f87e9fde2ef6
X-Runtime: 0.024772
X-Powered-By: Phusion Passenger 4.0.57
Server: nginx/1.9.3 + Phusion Passenger 4.0.57

8◆◆◆8◆◆◆h
        V]
End of ThingSpeak Example
```



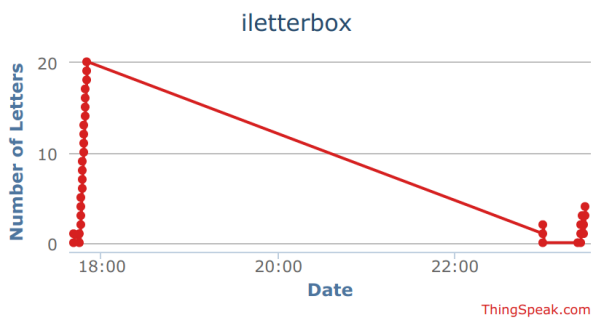
Execution

```
49]
End of ThingSpeak Example
^C
debian@beaglebone:~$ ./test
Segmentation fault
debian@beaglebone:~$ sudo ./test
Letter # 1
Letter # 2
Letter # 3
Letter # 4
```

Channel Stats

Created: about 9 hours ago
Last entry: less than a minute ago
Entries: 53

Field 1 Chart



Field 1 Numeric Display

4.0

a few seconds ago

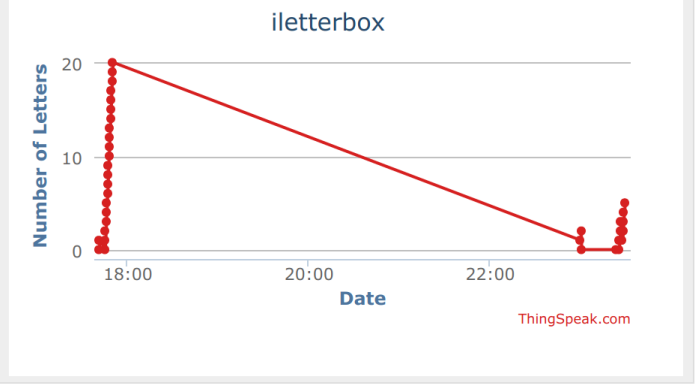
Execution

```
End of ThingSpeak Example
^C
debian@beaglebone:~$ ./test
Segmentation fault
debian@beaglebone:~$ sudo ./test
Letter # 1
Letter # 2
Letter # 3
Letter # 4
Letter # 5
Counter Resetted # 0
```

Channel Stats

Created: about 9 hours ago
Last entry: less than a minute ago
Entries: 54

Field 1 Chart

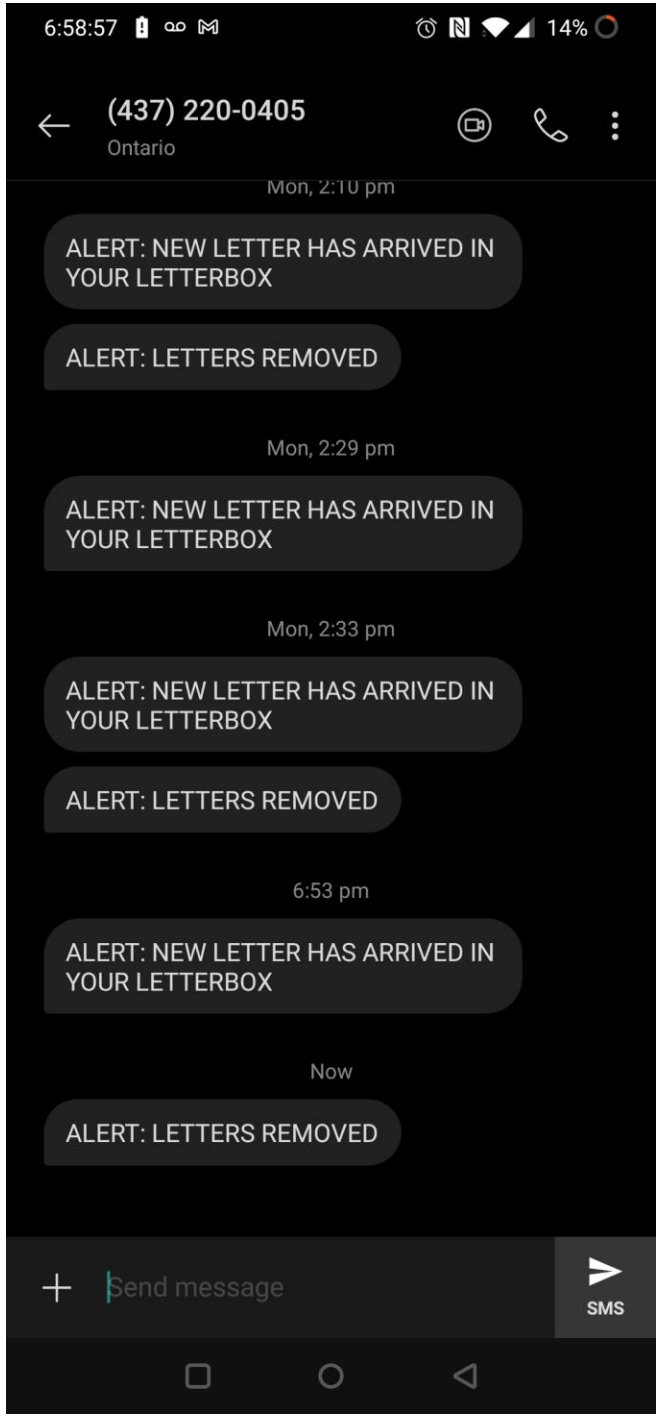


Field 1 Numeric Display

0.0

a few seconds ago

Execution


6:58:57
14%
(437) 220-0405
Ontario
Mon, 2:10 pm
ALERT: NEW LETTER HAS ARRIVED IN YOUR LETTERBOX
ALERT: LETTERS REMOVED
Mon, 2:29 pm
ALERT: NEW LETTER HAS ARRIVED IN YOUR LETTERBOX
Mon, 2:33 pm
ALERT: NEW LETTER HAS ARRIVED IN YOUR LETTERBOX
ALERT: LETTERS REMOVED
6:53 pm
ALERT: NEW LETTER HAS ARRIVED IN YOUR LETTERBOX
Now
ALERT: LETTERS REMOVED
Send message
SMS

Troubleshooting

- Any project that we carry out will have some flaws and one needs to overcome those flaws and try to achieve the obejctive that we set at the start of the project.
- The same scenario occurred with us during the project implementation; we faced some issues and we tried to overcome them using variety of sources avaliable with us.
- The first issue that we dealt with was connecting to wireless network. We couldn't connect to wireless network as we were located far from the modem (in my case since the setup was in the basement it was difficult to connect)
- Thus we had two alternatives either go closer to the network modem or setup/connect to another network. But a strange thing occurred in our case that when we gave some obstacle free path the BB-WI was able to connect with a suitable speed.

Troubleshooting

- The next obstacle was while we were trying to replicate the code from the book by Derrek Molley (Exploring Beaglebone) to connect to the ThingSpeak Cloud service we faced a compilation error saying "`no namespace exploringBB`"
- To resolve this issue up we had to clone the github repository of Derek Molley and while compilation give the path of the "`SocketClient.h`" so that the we could get a successful output file.
- The last thing that we had to deal was that though the program was running successfully but the data was not being written to the cloud service. Many attempts were made to check and verify the code, but we happened to see that everything was working well.
- Thus after properly reading and match the code given by Derrek Molley in his book Exploring Beaglebone and matching each spelling we found we had made a mistake in the word "Field" thus when the data reached the cloud it never knew where should it send the data as there was nothing such as "Feild". Thus correcting the spelling resolved this issue and taught us that somethings need to be specific and to the point and be careful and attentive always.

Conclusion

- With completion of this interface we mark the end of software and hardware interfacing with BB-WI as desired by the project requirements.
- During this task we saw that we were able to connect to the internet and send the data to the cloud service as and when required.
- This system has certain drawbacks which we have already discussed in the last meeting when interfacing the LDR with BB-WI
- The project at the end of this task perfoms the following functions:
  - Detects when a letter arrives in the letterbox and increments the counter and sets the LED high
  - Initimates the user when the any letter arrives and when the letters are removed from the letter box
  - Gets the LED off and counter resetted when the letters are removed from the box
  - Lastly sends the number of data that are being deposited to the cloud and sends a count of zero when the letters are removed.

References

Brain, M., Wilson, T. V., & Johnson, B. (2001, April 30). How WiFi Works. Retrieved November 23, 2020, from https://computer.howstuffworks.com/wireless-network.htm

HTTP - Overview. (n.d.). Retrieved November 23, 2020, from https://www.tutorialspoint.com/http/http_overview.htm

Introduction. (n.d.). Retrieved November 23, 2020, from http://exploringbeaglebone.com/API/index.html

Juneja, M. (2020, July 06). Difference between MQTT and HTTP protocols. Retrieved November 23, 2020, from https://www.geeksforgeeks.org/difference-between-mqtt-and-http-protocols/

Molloy, D. (2019). *Exploring BeagleBone: Tools and techniques for building with embedded Linux*. Indiapolis, Indiana: Wiley.

Serozhenko, M. (2017, March 20). MQTT vs. HTTP: Which one is the best for IoT? Retrieved November 23, 2020, from https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105

ThingSpeak for IoT Projects. (n.d.). Retrieved November 23, 2020, from https://thingspeak.com/

ThingSpeak. (n.d.). Retrieved November 23, 2020, from https://in.mathworks.com/help/thingspeak/

What is HTTP (Hypertext Transfer Protocol)? (2020, July 16). Retrieved November 23, 2020, from https://whatis.techtarget.com/definition/HTTP-Hypertext-Transfer-Protocol

What is MQTT? (n.d.). Retrieved November 23, 2020, from https://inductiveautomation.com/resources/article/what-is-mqtt