A Project Report

on

I-LETTERBOX (Intelligent Letterbox)

For the award of

ONTARIO GRADUATE CERTIFICATE

Submitted by

Zain Rajani (c0752681)

Under the guidance of

Dr. Mike Aleshams

ESE-4009 Embedded System Design Project

(September 2020 – December 2020)

Abstract

Letters have served to communicate information with others since ages. With the modern era emerging up we see that the letters for social communication is on a decline but still one receives letter from government offices which may be regarding the financial issues like taxation or paying the bills as a hard copy of documents serve as a proof of data. We here aim to make the task simple and make the user informed about when the letter arrives in his/her personal letter box. This project basically helps the user know the number of letters that are being received and the user is kept informed when the letter box is opened, and all letters are removed. User may need to have some technical expertise when operating for the first time.

After some research regarding finding about the existing project we came up with this idea to give a push and getting this project would help in benefiting the smart home system. Research about the product also identifies that it would serve as a benefit to the community and this product can be a DIY kits for the existing letter box system.

The product runs on a Linux Kernel and Debian Distribution with a minimum support of third-party software. The product was implemented with careful and appropriate testing techniques with all kinds of possibilities which may occur during actual usage. For first which may or may not have any prior technical expertise we developed a simple easy to understand user manual.

Acknowledgments

"It is not possible to prepare a project report without the assistance and encouragement of others. This one is certainly no exception" In our efforts towards the realization of the project work, we have drawn on the guidance of many people and we would like to express my heartfelt gratitude to all the people who helped in fulfilling the accomplishment of this project.

We would like to express my deep gratitude to Michael Vourakes, Director, Lambton College in Toronto for providing us an opportunity to study and in providing infrastructure and resources required to complete this project.

We also would like to extend our gratitude to Dr. Mike Aleshams for the unconditional support and coordination to make this project a success. We also would like to thank him for sparing time and providing necessary information patiently and providing efficient tips for debugging. His constant motivation and inspiration have led to successful completion of the project.

We also extend Dr. Takis Zourntos, Course Coordinator for Embedded System Design for critical advice and guidance without which the project would not have been possible. We also extend our gratitude to all other Professors of Lambton College in Toronto for their guidance and support in compiling the project and those people who have helped us knowingly and unknowingly for the completion of the project.

Thank you also to Texas Instruments and E2E community forum who provided necessary guidance. We truly appreciate every feedback and tips on all the questions asked.

Lambton College in Toronto is a great place to study and work, largely because of supporting faculty members. Thanks to all my colleagues in college for supporting and encouraging our idea.

Last but not the least we place a deep sense of gratitude to Almighty, our family and friends for their blessings and constant support till date and their constant inspiration from the beginning till the end of this project

~ Zain Rajani

TABLE OF CONTENTS

List of Figures

Chapter I
Introduction

Overview

This project is about the development or improvement to the existing letter box which one can use as a smart letter box for a house. This letter box has come with GSM, Wi-Fi based capabilities and USB connection. The box is aimed to count the number of letters that are received in the letter box while also informing the user. Along with the receiving letters when the letters are removed from the box the same would be informed to the user which will help the user in knowing if letter box was opened without his/her willingness, this feature could act like a security to the letters. The box runs on Linux Kernel installed which can be customised as per user requirements. Cost of the product development can range from $ 400 - $ 500 CAD.

The box can act not only as way to collect letters but also a charity money box collector counting the number of notes collected though the denomination or total amount of money collected may not be shown but could be made customised for that purpose. Currently the box design could be just for a simple house having a personal mail box close in its vicinity.

Problem Statement

Letters are an important part of daily business. People receive letters either from the government offices for either tax payments, or approval of their loans and schemes etc. Thus, people at times forget to check their mailboxes and some days people don't get the mails at all. Thus, a system was needed where when any letter arrived for the user, he/she must be notified or must be made aware of through some system. While the current system uses some of the micro-controller which lack some of the Wi-Fi Capabilities, manual resetting of counters and having lower speed of communication. Communication by means of letters are becoming limited to only official mails either from government or from banks thus a system was needed to make sure that one is aware of letters in the box which he/she may forget due to busy schedule.

Goal and Objectives

- **Goal**
  Develop a smart letterbox with GSM and Cloud based technology running with Linux Kernel by the end of Fall 2020 academic term.

- **Objective**
  - Connecting a host processor Beaglebone Black Wireless (BBB) to one of the available cloud services in market and GSM Module.
  - Interface various sensors which will help us getting to know when the letter arrives and when the letters are removed from the letter box.
  - Configuring Wi-Fi and Bluetooth capabilities to connect cloud services.
  - Inform the user when the letter arrives and is removed from the letter box

Scope of the Project

- **Deliverable**
  - A smart letter box to monitor activity of the letters as a product for the market.
  - Communicate the activity of letters to cloud services.
  - Generate a PCB design for the project.
  - Intimate about the letter activity via SMS services through GSM Modules.
  - Plug and Play the project that is the letters box should function without help of any laptop or any other devices to execute the codes written for completing the deliverables mentioned above
- **Milestones**

Task Name	Start Date	End Date	Person In-charge
Project Proposal	September 18, 2020	October 09, 2020	Zain Rajani
Finalizing Hardware Requirements	October 10, 2020	October 12, 2020	
Getting Hardware	October 13, 2020	October 20, 2020	
Testing of Hardware	October 21, 2020	October 26, 2020	
Circuit Design	October 27, 2020	November 01, 2020	
Interface LED and IR Sensor on BB-WI	November 02, 2020	November 08, 2020	
Interface GSM Module with BB-WI	November 09, 2020	November 17, 2020	
Interface LDR with BB-WI	November 18, 2020	November 22, 2020	
Interface ThingSpeak with BB-WI	November 23, 2020	December 01, 2020	

PCB Designing	December 02, 2020	December 09, 2020	
Final Report Presentation	December 10, 2020	December 14, 2020	
Final Presentation	December 15, 2020	December 17, 2020	

- **Limitations**
  - The user needs to know the use of the terminal to connect Wi-Fi networks.
  - Processing power is limited to 1 GHz for BBB so it will be bit lagging.
  - There would be some amount of design constraints that is the project can be used only with specific design like cannot be used as a community mail box.
  - When there is more than one letter in the box and only is removed then the user will get a message that letters are removed but in reality, there are letters in the box. Thus, the user needs to remove all the letters take the one he/she requires and deposit the remaining one by one.
  - When two letters if deposited together then there would mismatch of the number of letters and the number of letters present in the box in reality.

Outcomes and Benefits

- The system is powered up using a power adapter thus no need to worry about power unless there is no electricity in the entire house or from where the connection is made.
- Even if the user has no phone and acess to the cloud service but has the key to open the lock of the letter box the LED attached on the box will guide and indicate if there is any letter in box.
- Could be an integration to the already present smart home system or could be added as a part of system
- Since the letters are becoming slowly vanishing thus this system will help to user know that they have some physical mail as now a days people have started moving to digital mail and forget that they might get some physical mails as well as the transformation of complete is digitalisation of mail is not there.
- The system helps the user know how many letters are deposited via the cloud service and also keeps them informed regarding the box activities via GSM through SMS service

Facilities and Resources

- **Laboratory**
  - Embedded Systems Lab which we created as the classes were online thus a miniature lab was created at our place.

- Oscilloscope, multimeter, logic analyzers and Linux installed PC's present in our miniature home made lab.

- **Intellectual Resources**
  - Data Sheet of various sensors, GSM module, Beaglebone Black Wireless
  - AT commands Catlogue
  - Forums of Beaglebone and Element14 Communities
  - Exploring Beaglebone Tools and Techniques for Building with Embedded Linux (Second Edition) from Derek Molloy.

Procedure and Methodology

i) Getting the hardware requirements ready and ordered for the project sideways testing the components and understanding their working as they keep on arriving.
ii) Install and update the devices such as the Beaglebone with the latest image OS in order to get to use some of the latest features available in terms of device trees and drivers.
iii) Upon the getting the testing results positive and approved move and design the circuit schematic using software like EasyEDA, Fritzing, KiCad or Eagle, etc.
iv) Installation of various libraries which will make our task of controlling the pins of Beaglebone easy and communication between peripherals and sensor simpler.
v) Keeping the schematic in mind get the connection ready for the interfacing in part as this will help us to know in case we go wrong or if the system is not working at some points. We start first with interfacing simple LED and IR Sensor.
vi) Next, we start to interface LDR with Beaglebone along with the previously connected sensors and LED.
vii) Getting the user information intimation devices interfaced with the host processor i.e., Beaglebone.
viii) To make the data available on the cloud service we need to establish a wireless connection so that Beaglebone can get connected to the cloud and send the appropriate data to the cloud and make it visible to the user.
ix) At the end of all the interfacing we will integrate them and check if we get the desired functionality as mentioned in proposal.
x) In order to minimize the amount of wiring we must generate the PCB design so as make our project look like a market product ready. The PCB could be designed with the same software that we may have used in making the schematic on the software in step iii).
xi) Solder the devices on the PCB and test for the final demonstration of the project.
xii) Make the project run in such a way that when powered on the program should auto-run without the help of any sort of connection with laptop or any external devices.

Chapter II
Literature Review

In a article "***Smart Letter Box***" by Darp Raithatha [21] he explains how he uses to make a smart letter box system with the help of Arduino as microcontroller. He uses an obstacle detector in his project and senses the number of letters that arrive in the letter box and if a single letter is sensed it makes the LED go high thus it will indicate the presence of letter in box. Here the system will be resetted back to its normal working that the LED will go off once the letters have been removed using a television based remote. The remote will reset the LED and it would mean that there are no letter in the box. The setup for the same is shown in Figure 2.1

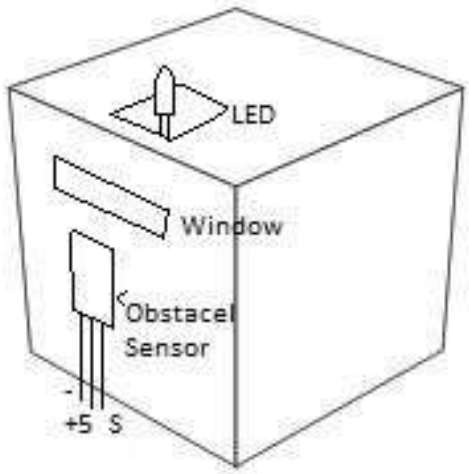


Figure 2.1 *Smart letter box controlled by remote*

Further during research we found another article title "***Intelligent Letter Using Arduino and GSM"*** by Salman Khan [12] wherein the same principle was used using sensor circuit to detect the arrival of letters in the box and the same was intimated to the user using GSM module and in the message the date and time of arrival was sent using RTC Circuit which is part of the Arduino based GSM module. Also attached to the box they have is the LCD display which can display the arrival of the letter so that the mail man knows that letter is deposited in the box properly. The system block diagram created is shown in the Figure 2.2

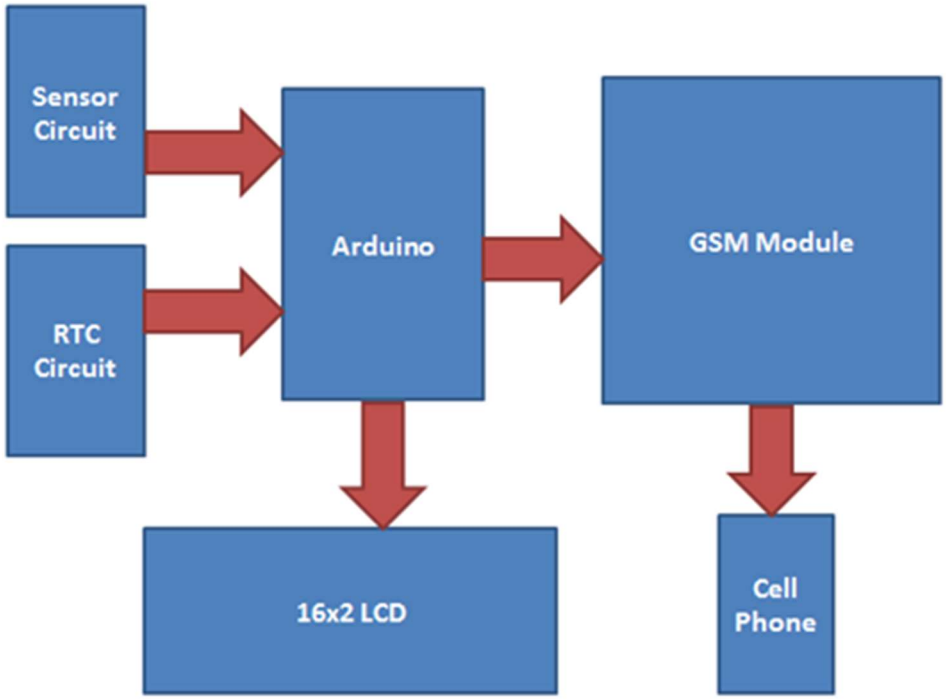


Figure 2.2 *Existing Arduino Based System*

In a paper published by Anjali Devi Pujari, Priyanka Bansode, Pragati Girme, Harshal Mohite, Anirudha Pande titled "***Smart Letter Box System Using Obstacle Sensor For Notifies The User By Android Application***" [6] they have just extended the project block diagram in Figure 2.2 by just creating an Android application which will inform the user for the arrival.

In another journal publication titled "***Design & Development of Electronic Letterbox using LDR***" by P.A. Ambresh, M. Ashwini, Rickson Wilson Rodrigues and Vikesh. The publication uses IC 555 based circuit and LDR and monitors the voltage level controller circuits. The intensity of light that falls on the LDR varies the resistance and hence the voltage level. The LEDs that are placed are connected in forward and reverse bias respectively. When the voltage level reaches around 1/3 of the total voltage the LDR is triggered and sends a signal to LED and it turns on. Thus looking at this ideas we combined all of them to generate our own letterbox which is smart and running on some of the latest trends and technologies.

Chapter III
Requirements / Analysis

Hardware

This intelligent letter box requires host processor which can run Debian Linux and Wi-Fi, Bluetooth capabilities along with some sensors and GSM Modules. We used software and designed our own customized PCB using the schematic design which helped us connecting all the wires.

Product fits for both personal and environmental purposes, it can be used as an indication for the post services office to collect the letters that may be put in the box so as to eliminate unnecessary trips to the box and hence save the fuel emission. The box will also help one know about the letters being deposited in the box.

Thus, we require following list to make a complete intelligent letterbox:

- Beaglebone Wireless (BB-WI): Used as the main processing unit (Host Processor). All the sensors and devices shall be connected to this device
- IR Sensor (Obstacle Sensor)
- LDR (Light Dependent Resistor)
- GSM Module (SIM900)
- LED (Light Emitting Diode)
- Connecting Wires
- Jumper Wires
- Basic Electronic Components: Multimeter (to check on voltages and current at various nodes basically for debugging the circuit), Wire Cutters (Cut off the wires of the required lengths to make the circuit look clean), Strippers (To remove the insulation and connect)
- Laptop Device (Equipped with appropriate storage and capable to handle the Beaglebone having a Linux OS preferably)
- Breadboard (Used to make the circuit for testing before final PCB design and implementation)
- Soldering Equipment (Soldering Iron: Solder the components on the PCB, Desoldering Gun: Removes the excess of soldering or if soldered incorrectly, Soldering Wire: used as electrical conducting to join the connection)
- SD Card: To store the image and the data related to the Beaglebone

Software

Considering software requirement, for developing a prototype team will need a Linux machine or a Windows PC with Putty installed. We decided to forward with a Debian image for Beaglebone black from Beagleboard.org.

Following is a list of required software for this project:

- Debian image above Debian 10.3 from https://beagleboard.org/latest-images
  - This Debian image comes with all device tree pre-loaded to it with Wi-Fi services pre-installed.
- Putty (If using Windows PC for programming and testing)
- Linux Bash scripting, C/C++, and Python programming
  - C/C++ programming skills will be needed to program button on LCD cape to turn display ON/OFF.
  - Bash scripting will be needed to edit DTS files and run some scripts automatically on boot.
  - For the project purpose we will try to program majority in either C/ C++ but we may use python in situation where certain libraries may not be present in C/C++
- GCC (GNU (GNUs Not Unix) Compiler Collection)
  - Usually, pre-installed on the Linux distribution. It would be used to compile the code and generate the output file.
- Eclipse
  - Some of the programming for the hardware would be done using this software and the cross compiler may be used and the file may be sent on the Beaglebone for execution. It will only be used only if the programming cannot be done on Beaglebone directly.
- Nano (Editor)
  - Any coding if required to be done in Beaglebone itself thus, a nano editor will be used to write the program and save the file using appropriate extension and this file shall be compiled using the GCC to get the binary executable file.
- ThingSpeak
  - Open-Source Internet of Things (IoT) application and API used to store and retrieve data using the HTTP (Hypertext Transfer Protocol) and MQTT (Message Queuing Telemetry Transport). It works closely with MATLAB and MathWorks. For the project purpose we will use it as a cloud platform to transfer the data and store it in the cloud in real time.
- EasyEDA:
  - A PCB design and simulation tool enabling hardware engineers to write EDA (Electronic Design Automation). We use this tool for PCB design and layout. It could be used as simulation but Beaglebone simulation may not be possible as

SPICE libraries are currently not present in the software. Hence software simulation of the Beaglebone circuits cannot be done but the PCB can be designed using this software.

Power Requirement

In order to power up the entire system and get the system working and performing the task that we require we must provide it with sufficient amount of supply. Today in the market we have variety of options available to provide supply to our system. Thus, looking at the requirement for our project we find that we have two major power consuming components. The desired power requirement for the project are as follows:

1. Beaglebone Black Wireless – 2 A @ 5V
2. Arduino based GSM Shield – 2 A @ 5V

So estimated power requirements for full product assembly will be approximately 4V @ 4 A for smooth and interrupt free operation. Thus, for the system we choose a suitable power adapter as the system we aim for will be close to the user i.e., attached to individual houses. If required we can attach a Li-on battery in situations where one has to make it portable in nature like for community mail boxes/ postal offices etc.

As most of the sensor that we want to use for the project and when we look at the pin out diagram of the same, we realise that we can power them up directly from the Beaglebone thus no extra power supply is required for them.

Block Diagram

The figure below Figure 3.1 shows the basic layout of our project or in other words the circuit connection in the raw form which will help us build the letter box. For our project we use the SIM900 based Arduino shield as GSM Module which will receive the commands from the Beaglebone black wireless which will tell when to send message. In the project for communication with the cloud we use ThingSpeak as the cloud service interface. From the Beaglebone black wireless we intend to send the count of letters to the cloud and also reset the same counter when the letters are removed. We intend to use the free trial features for the project as they seem to be sufficient for our project. As the cellular network we choose “Chatr” service which will help us sending messages to the user defined numbers.

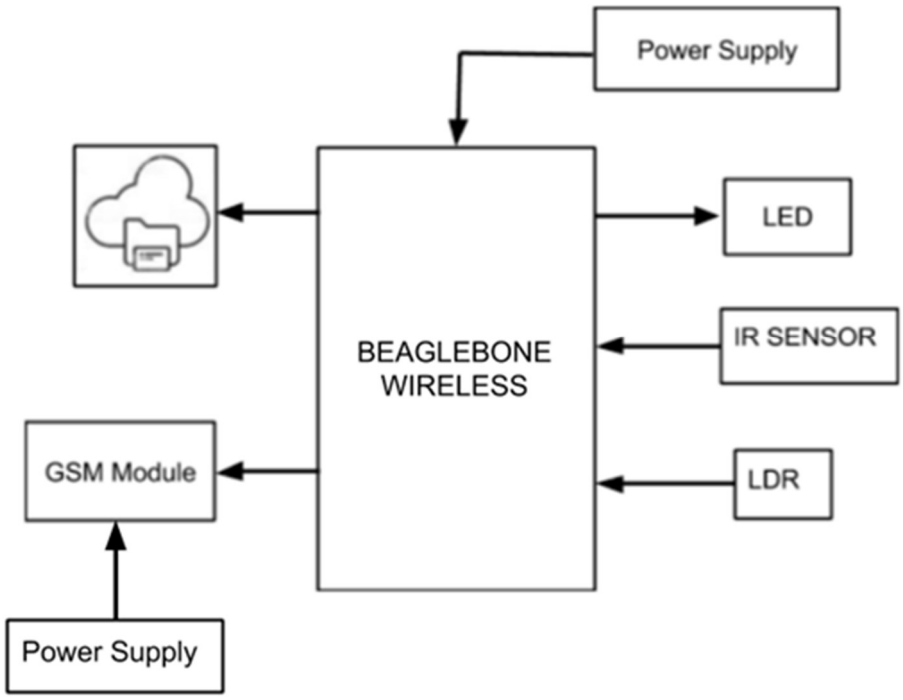


Figure 3.1 *Block Diagram of Proposed Project*

All testing of device was performed under Debian distro provided by Beagleboard.org for Beaglebone Black. The testing produced an executable binary file which we later used for our plug and play kind of deliverable. The binary file was produced by the GCC (GNU Complier Collection) which is pre-installed and available in Beaglebone Black Wireless. The product development stages have been shown in Figure 3.2.

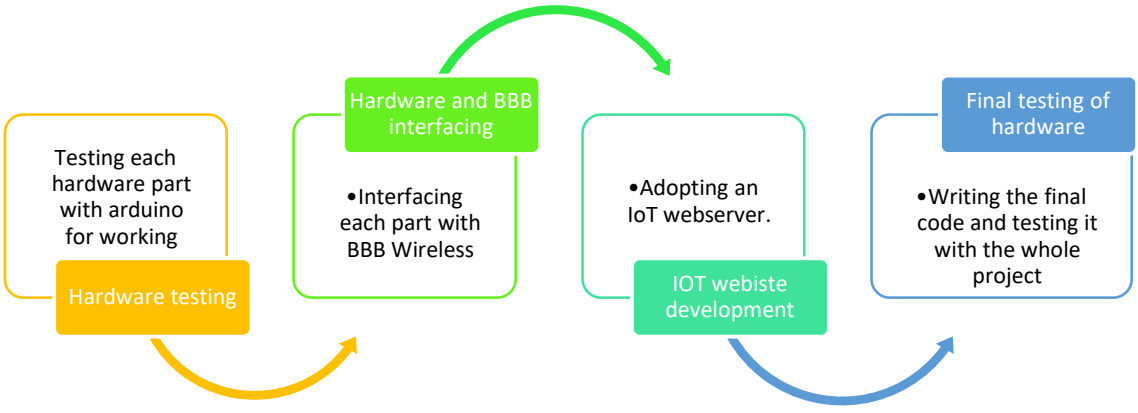


Figure 3.2 *Block Diagram of Product Development*

Project Flowchart

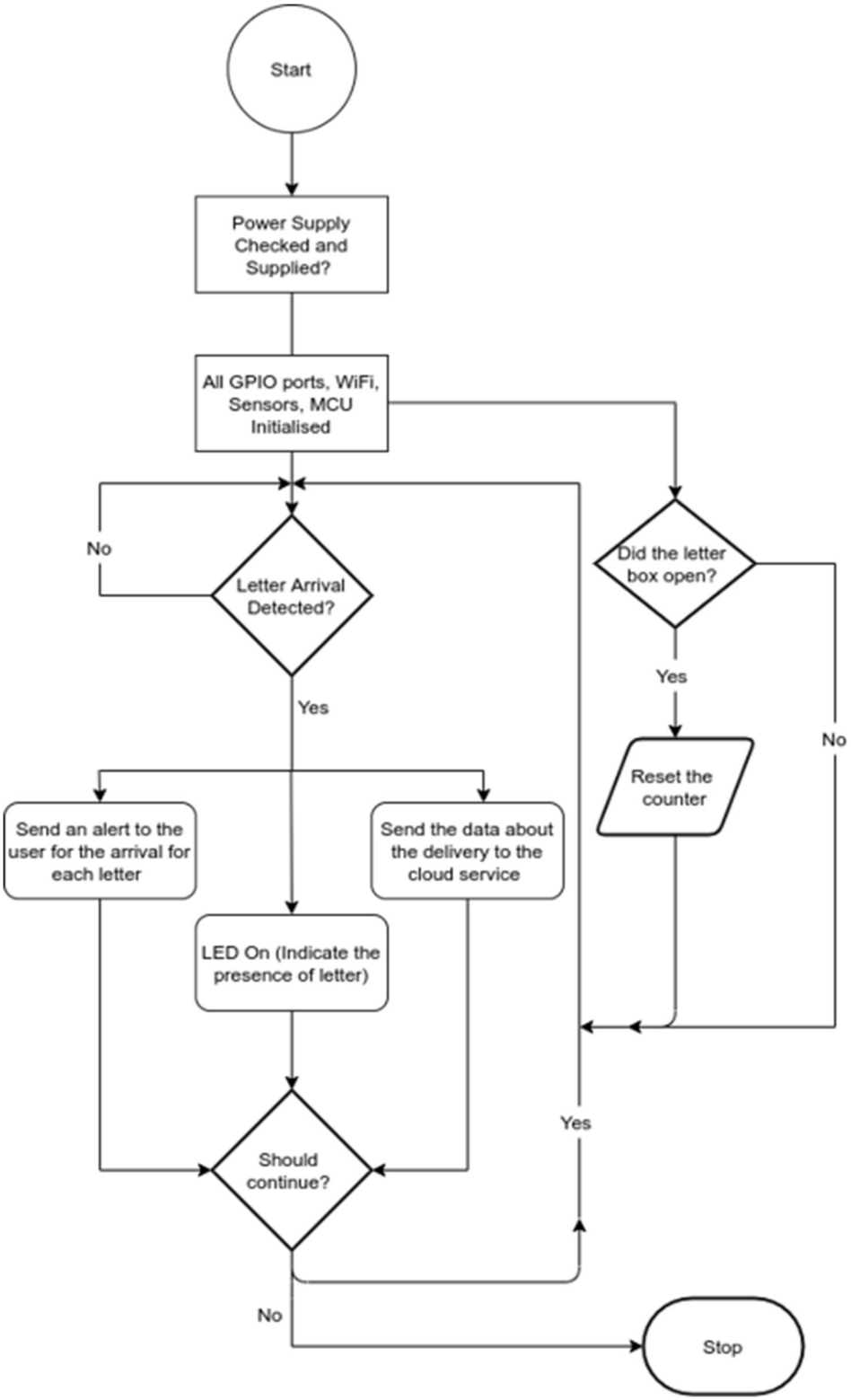


Figure 3.3 Flowchart for the project

Chapter IV
Design

Project Schematic

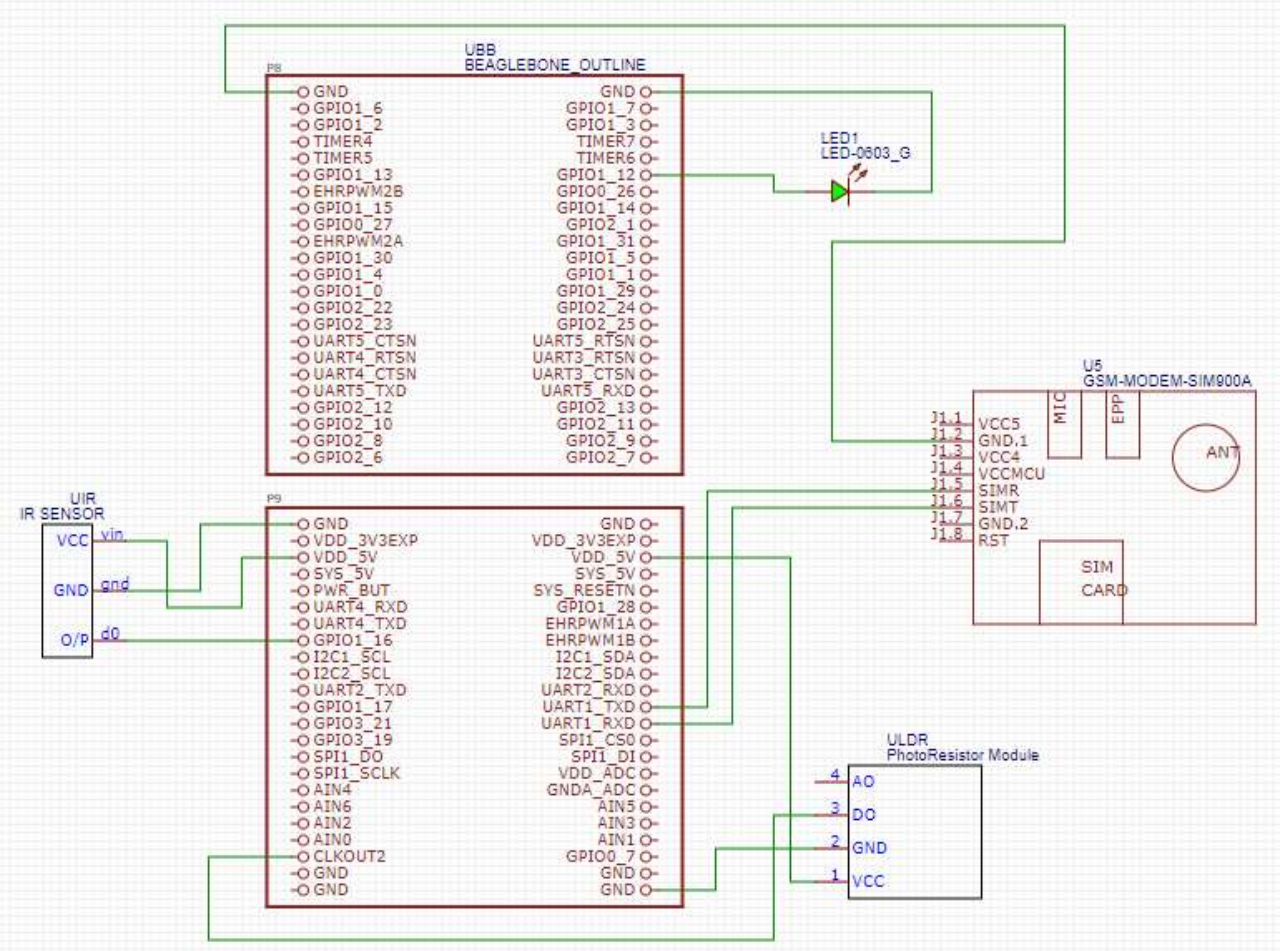


Figure 4.0 *Schematic Design for the Project*

Figure 4.0 shows the schematic capture for the project which will help us wire the same. To draw the schematic, we use EasyEDA software. EasyEDA is a free open-source platform and is best suited for beginners in schematic capture. As seen in the schematic one can see how the sensors like IR Sensor, LDR Sensor, LED and GSM Module get connected to the Beaglebone Black Wireless. A point to note here is that all Beaglebone Boards have the same pin configuration. Thus, for the schematic capture we used the layout of Beaglebone Black. At the later stage of the project, we used this schematic and converted into the PCB Layout which we can used to get PCB manufactured for the project. Before moving on further let us first discuss and know the hardware in detail which will help us in programming the system and reason for our connection.

- **Host Processor (Beaglebone Black Wireless)**

To about Beaglebone Black Wireless one just needs to refer the manual for Beaglebone Black as they both are the same with just a small difference between the two being that Beaglebone Black Wireless has Wi-Fi capabilities in place of the ethernet port which the Beaglebone Black provides instead of Wi-Fi feature. For reference one can see Figure 4.1 (a) and Figure 4.1 (b) to compare their design apart from the that all the functionalities are the same.

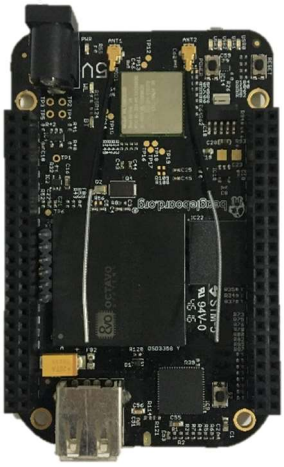


Figure 4.1 (a) *Beaglebone Black Wireless*

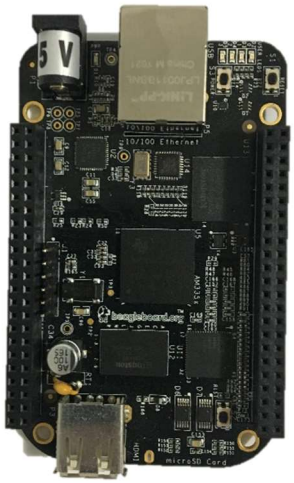


Figure 4.1 (b) *Beaglebone Black*

For connecting all the sensors and other peripheral devices we choose Beaglebone Black Wireless (BB-WI) as the main processing unit also known as the Host processor. The main features of the Beaglebone Black Wireless include [ ]:

It has a processor Octavo Systems OSD3358 1GHz ARM Cortex A8 Family manufactured by Texas Instruments.

- In terms of storage, it has 512 MB DDR3 RAM with 4GB 8-bit eMMC on board flash storage
- To perform any real time operation, it has built in 2 x PRU units which are 32-bit micro-controllers
- It has pre-installed Debian Image which is a kind of Linux distribution with support for Cloud9 IDE on Node.js with Bonescript Library. The Cloud9 service helps one to control

the device wireless over the local Wi-Fi Network which the Beaglebone provide we shall discuss this more in detail at later stage.

- It consists of 2 x 46 header pins that allows connection of various sensors and other peripherals.
- It has USB connectivity along with HDMI connectivity.
- It supports Bluetooth 4.1 with BLE and Wi-Fi IEEE 802.11 b/g/n. BeagleBone Black Wireless replaces the 10/100 Ethernet interface of BeagleBone Black with a high-performance flexible Wi-Fi/Bluetooth WiLink™ interface manufactured by Texas Instruments.
- Debug Support: Optional Onboard 20-pin CTI JTAG, Serial Header.
- External Storage: One can use microSD card attached at the bottom of the BB-WI near the HDMI port to store the OS and other data.Upto 64 GB cards are found to be supported.

The parts of typical Beaglebone Black Wireless is shown in Figure 4.2 below. Once that we have known the features of Beaglebone Black we now need to see how does it boot up. If one closely looks at the left side of the image Figure 4.2 one can notice that it consists of 4 USER LEDs while booting these LEDs glow up eventually but once the booting is completed these LEDs blinking pattern signifies somethings which are as follows:

1. USER0: Heartbeat Pattern from Linux Kernel
2. USER1: Turns on when the SD card is being accessed
3. USER2: Activity Indicator. Turns on when the Linux Kernel is not in the idle loop
4. USER3: Turns ON when the eMMC is being accessed.

The LEDs are shown clearly in the Figure 4.3 the image is adopted from Beaglebone Black as both the bords are the same with the exception that BB-WI doesn't consists of any on board ethernet port.

Figure 4.2 *Beaglebone Black Wireless with all parts shown*

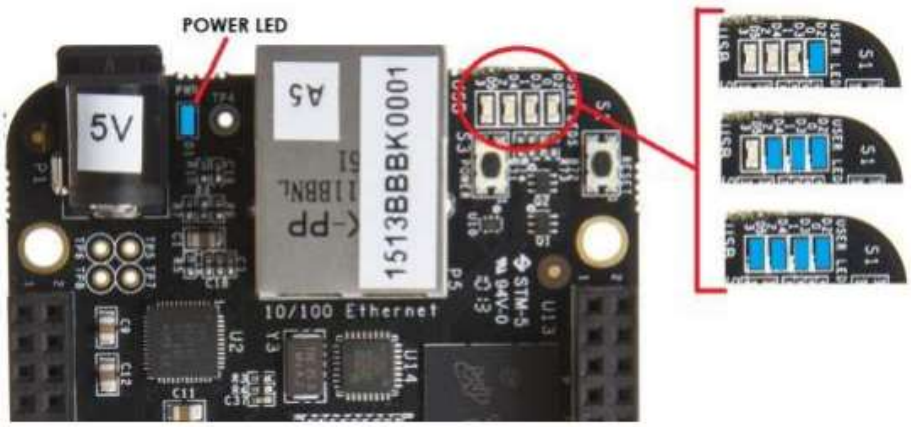


Figure 4.3 *Booting LEDs of Beaglebone Black*

To control the pins of Beaglebone one must know about the functionality set by default and about the functionalities that could be set using the different modes of the pins available for this purpose one needs to always refer the pin-out diagram of the Beaglebone family as shown in Figure 4.4. This pin diagram is common for all the families be it Beaglebone Black, Beaglebone AI or Beaglebone Wireless.

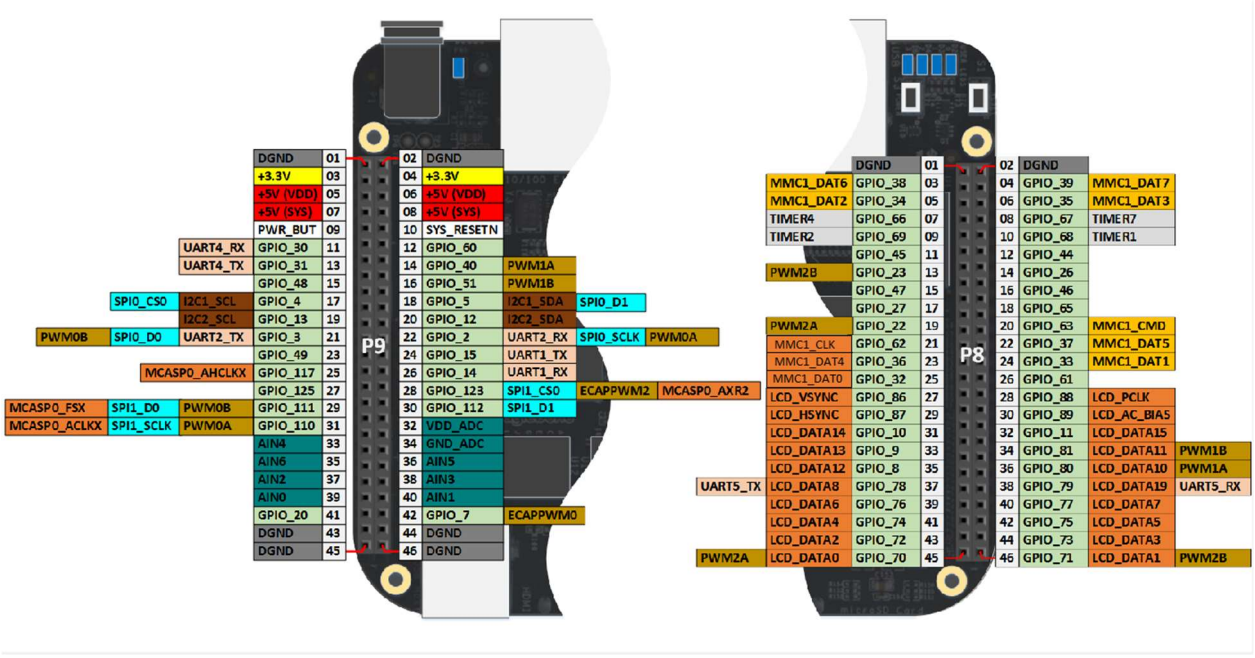


Figure 4.4 *Pin Diagram of Beaglebone Family*

For debugging process, we have a serial debug jumper J1 having 1x6 header. Serial debug is provided via UART0 on the processor via a single 1x6 pin header. In order to use the interface a

USB to TTL adapter will be required. The header is compatible with the one provided by FTDI. Signals supported are TX and RX. None of the handshake signals are supported. To gain the control of Beaglebone Black Wireless we have to get connected to the local Wi-Fi of Beaglebone as shown in the Figure 4.5 the default password for this is “BeagleBone”. Once we have connected, we can then open the web browser and type in the address as “192.168.8.1” this shall be connected via Cloud9 IDE as shown in the figure 4.6.

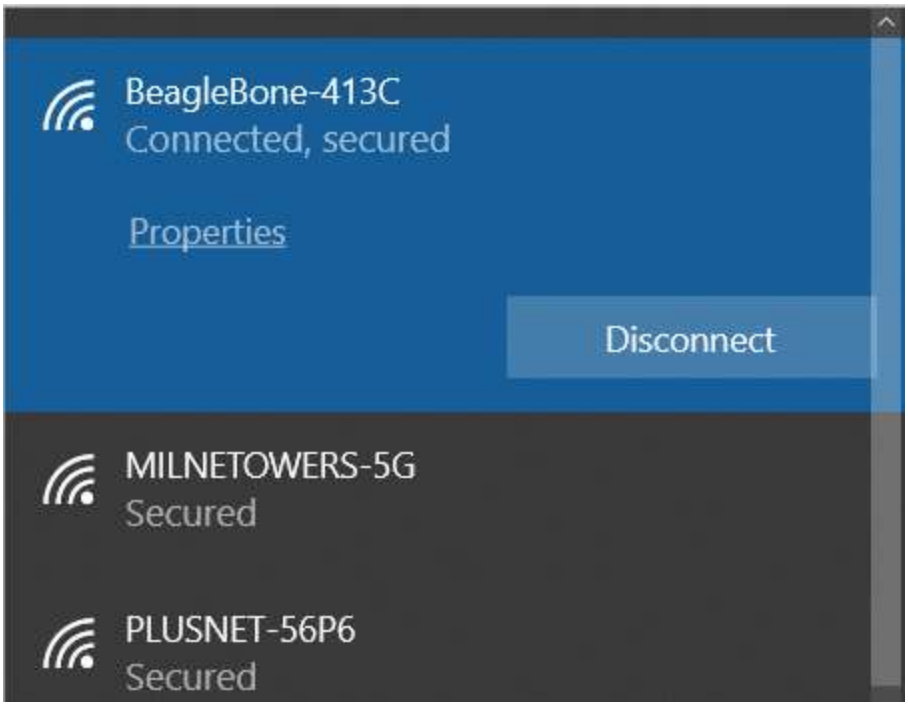
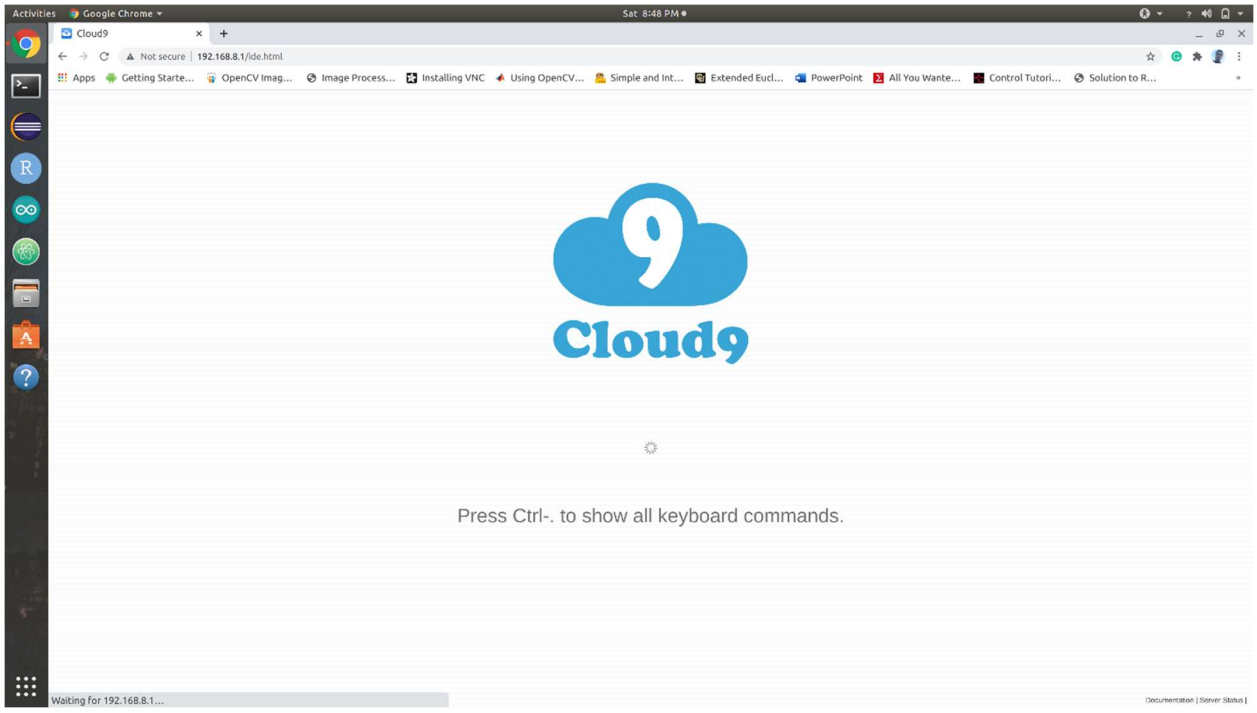
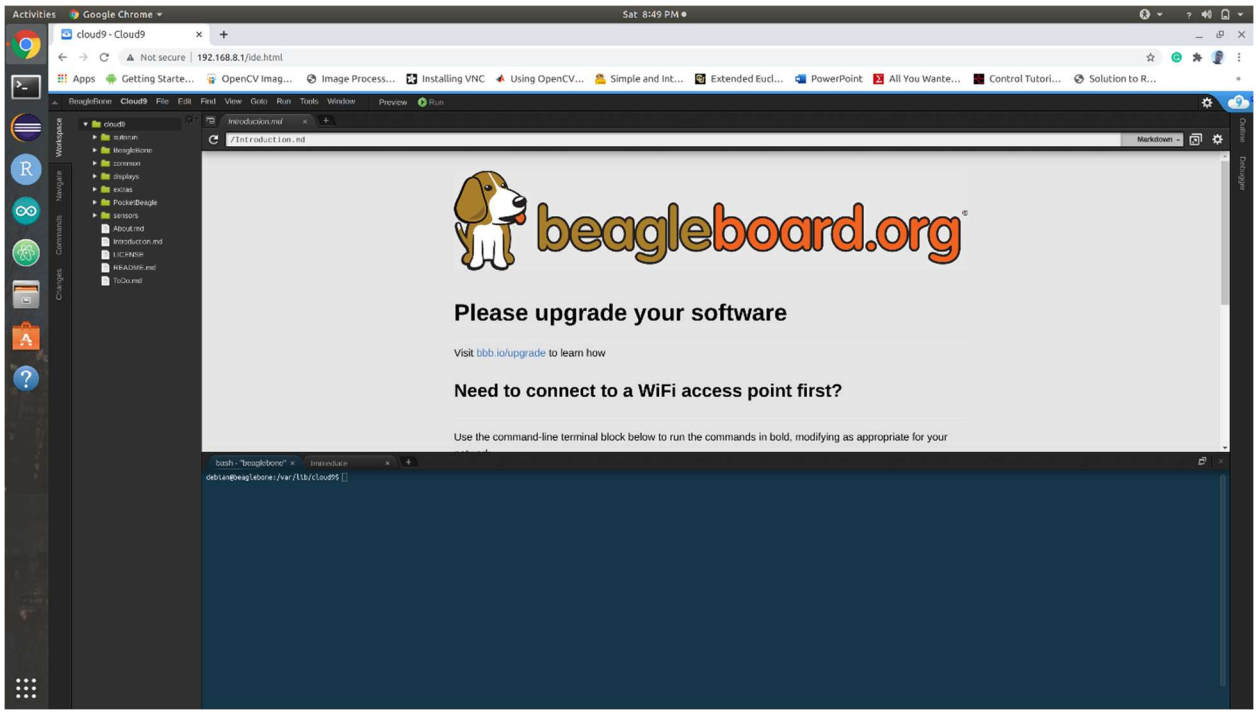


Figure 4.5 *Local Wi-Fi Network Available on computer*



(a)



(b)

Figure 4.6 *Cloud9 IDE Interface (a) Start up Screen (b) Beaglebone Black Wireless Terminal*

In order to get the Beaglebone Black Wireless powered up wireless using a battery we have a TP pins or battery pads present near the barrel jack on the board these pins are shown in Figure 4.7. Each of the pins have their meaning as follows:

PIN	DESIGNATION	FUNCTION
BAT	TP5	Battery connection point.
SENSE	TP6	Battery voltage sense input, connect to BAT directly at the battery terminal.
TS	TP7	Temperature sense input. Connect to NTC thermistor to sense battery temperature
GROUND	TP8	System Ground

Figure 4.7 *Battery Pads on the Beaglebone Wireless*

- **IR Sensor**

The next hardware that we require for this project is IR (Infrared) Sensor. For this sensor is available in the market as a module but in case one wishes to make a module on its own that is also possible. This is shown in the Figure 4.8.

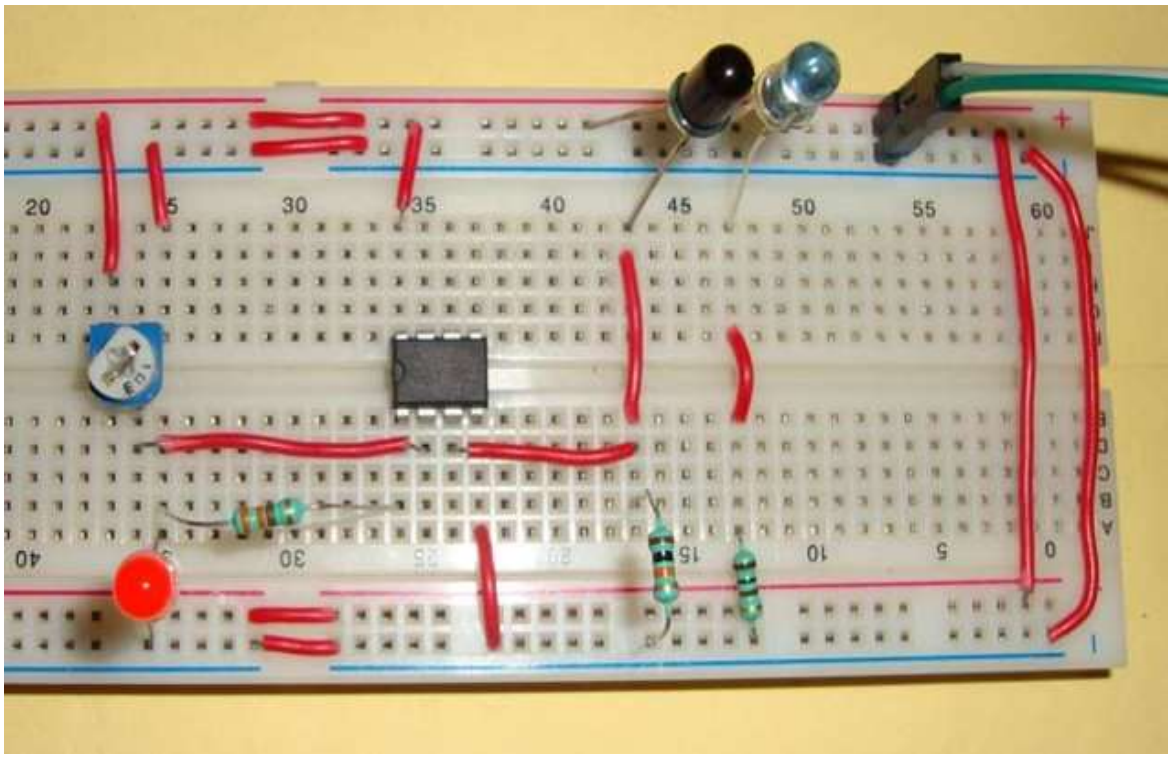


Figure 4.8 *IR Sensor Circuit Raw Circuit*

Since the sensor module is easily available in the market, we choose to opt for the module instead of building one for ourselves. Before going to connect the sensor module we must know what is the sensor and how does it work. The sensor module that we use for this project is shown in Figure 4.9 with all parts of the module being labelled.

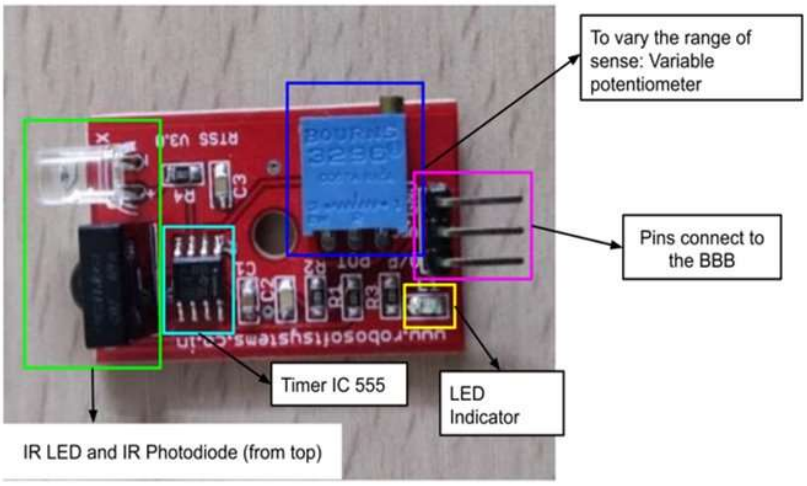


Figure 4.9 *IR Sensor Module Used in the Project*

There are two types of infrared sensors: active and passive. Here in the project since we only use active sensor thus, we shall be discussing only the active infrared sensor. Active infrared sensors both emit and detect infrared radiation. Active IR sensors have two parts: a light emitting diode (LED) and a receiver.

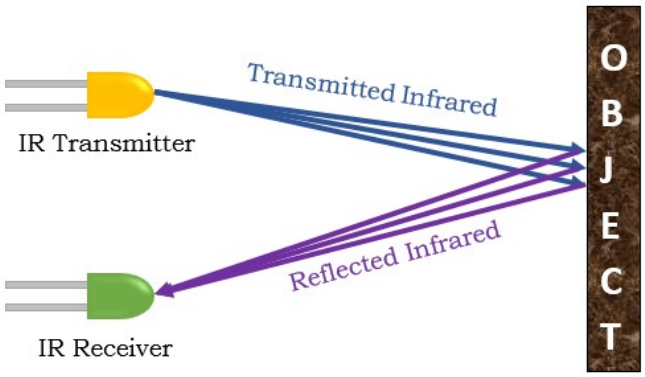


Figure 4.10 *Working Principle of IR Sensor*

When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver. Active IR sensors act as proximity sensors, and they are commonly used in obstacle detection systems (such as in robots). The working principle of this sensor is shown in Figure 4.10. Just as a small discussion Passive infrared sensor is also called as PIR sensor which is based on the Fresnel's lens and absorbs the radiation from the surrounding for detection. An IR sensor is usually governed by following three laws namely: Planck's Radiation Law, Veins Displacement Law and Stephen-Boltzmann Law. The only one drawback is that if this sensor is fitted or used in areas where dark surfaces are present more then when the rays are transmitted it would be absorbed by the surface instead of reflection to the receiver. Thus, for this project we needed to make sure that we have an equilibrium of the colours we used in the project.

- **LDR Module**

The reason one may want to use LDR is stop the wastage of power due to carelessness of human beings or unusual circumstances. We use the advantage of this objective in our project and get the counter to 0. To define an LDR or a photoresistor is a device that is made up of high resistance semiconductor material. It is basically a photocell that works on the principle of photoconductivity. An LDR is a passive component in nature as it doesn't generate energy. An LDR is a type of variable resistor which changes its resistance according to the intensity of light falling on its surface. This sensor senses the intensity of the surrounding light. The way in which the resistance varies depending upon the intensity of light is shown in Figure 4.11

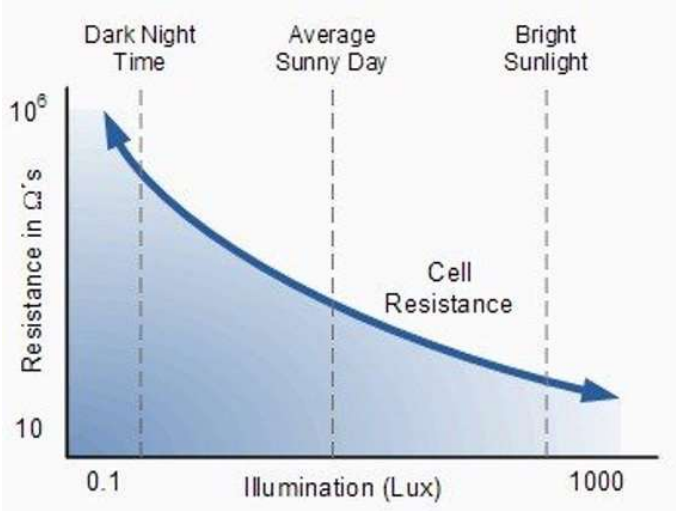


Figure 4.11 *Graph of Intensity v/s Resistance*

Resistance of an LDR is inversely proportional to the intensity of light that falls on LDR's surface. In other words, with an increase in light intensity, the resistance of photoresistor or LDR decreases. And that's why the graph between Resistance of LDR and intensity of light is hyperbolic in nature. The construction of an LDR includes a light-sensitive material that is placed on an insulating substrate like as ceramic. The material is placed in a zigzag shape in order to get the required power rating and resistance. The area of zigzag separates the metal placed areas into two regions. The structure of LDR is shown in Figure 4.12.

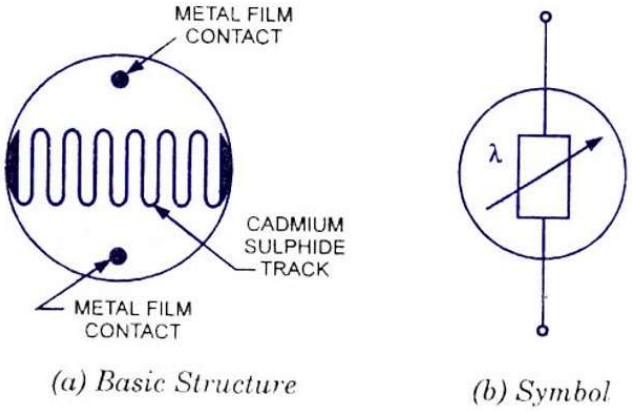


Figure 4.12 Structure of LDR

The structure is housed in a clear plastic or resin case, to provide free access to external light. As explained above, the main component for the construction of LDR is cadmium sulphide (CdS), which is used as the photoconductor and contains no or very few electrons when not illuminated. In the absence of light, it is designed to have a high resistance in the range of megaohms. As soon

as light falls on the sensor, the electrons are liberated, and the conductivity of the material increases. For the purpose of the project we use the LDR Sensor Module which is directly available in the market as shown in Figure 4.13

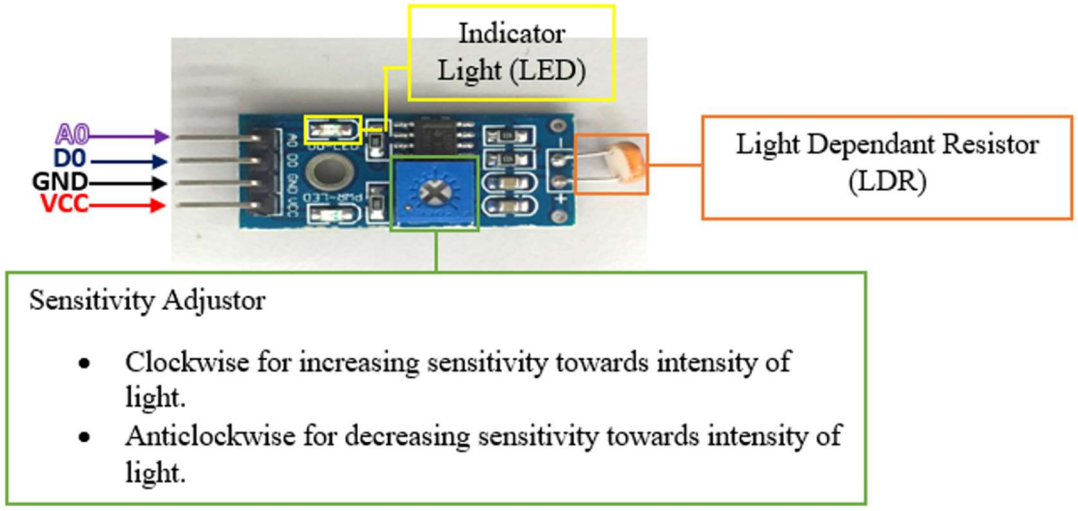


Figure 4.13 LDR Sensor Module

- **SIM900 GSM Shield**

GSM is a mobile communication modem; it stands for global system for mobile communication (GSM). It is a widely used mobile communication system in the world. GSM is an open and digital cellular technology used for transmitting mobile voice and data services operate at the 850MHz, 900MHz, 1800MHz, and 1900MHz frequency bands.

A GSM digitizes and reduces the data, then sends it down through a channel with two different streams of client data, each in its own particular time slot. The digital system has the ability to carry 64 kbps to 120 Mbps of data rates. The GSM module we are using is based on the SIM900. We are using the one which is like a shield for Arduino. This shield has an audio jack wherein one can connected earphones and talk like he/she would do using a normal phone. The GSM Shield which clear labelling of all its parts is shown in the Figure 4.14

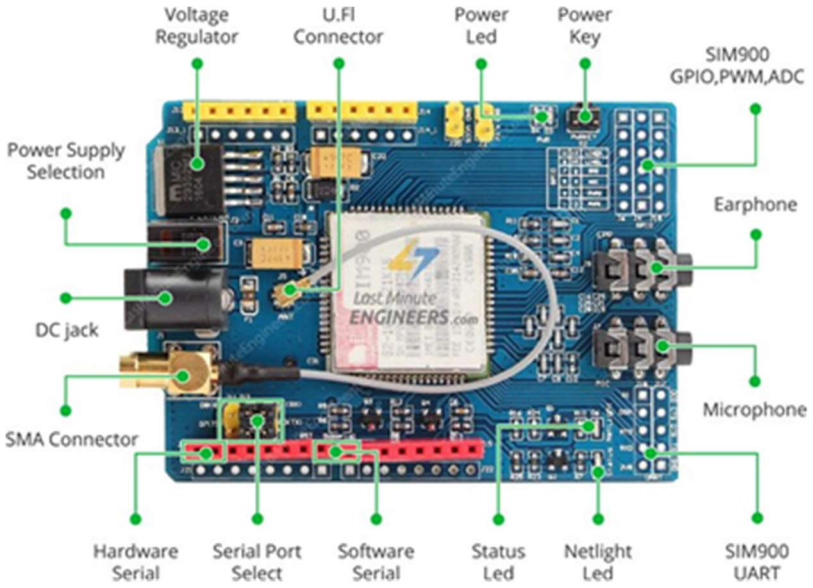


Figure 4.14 *GSM SIM900 Shield for sending SMS*

The GSM module used can sending/receive text and call using the antenna attached to it. If one wants to monitor time, then it can do so using the RTC present by simply connecting a CR1220 battery at the back of the shield. This module will be used to send the messages to the user in case there is an arrival of the letter along with when all the letters have been removed from the letter box. To do this functionality we will require AT (Attention) commands which are usually used by modems to communicate and perform various task.

AT commands are instruction used to control a modem. Every command starts with "AT" or "at" this is the reason why modem commands are called AT commands. Along with commands desired for the dial up modem the commands also support the GSM/GPRS modems and cell phones which include SMS related commands like AT+CMGS (Send SMS Message), AT+CMSS (Send Message from Storage), AT+CMGL (List SMS Messages) and AT+CMGR (Read SMS Messages). The "AT" at the start of the command informs the modem about the start of the command line. It is not a part of the command. All the GSM commands are extended commands as they carry a "+" sign ahead of them.

To power this module, we can either use a 5V adapter or some internal source, but whatever we do we not to select the power source using the switch present near the barrel jack.

- **Power Supply**

The power adapter converts the electrical outlet's electric currents into the low alternating current required by the device. Electrical specifications: Input voltage range – 90-264V. Output voltage – 5v. Average efficiency - >78.70%. Output current – 2A. Storage temperature - -20 degree to 60

degree Celsius. For our project we require two power supply adapters one for the Beaglebone and other for GSM Module. The adapter used in the project are shown in Figure 4.15

Figure 4.15 *Typical Power Adapter of 5V, 2A with barrel jack*

- **LED**

A light emitting diode (LED) is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with the holes releasing energy in the form of photons. The color of light is determined by the energy required for the electrons to cross the band gap of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light emitting phosphor on the semiconductor device. A typical LED parts and the LED are shown in Figure 4.16

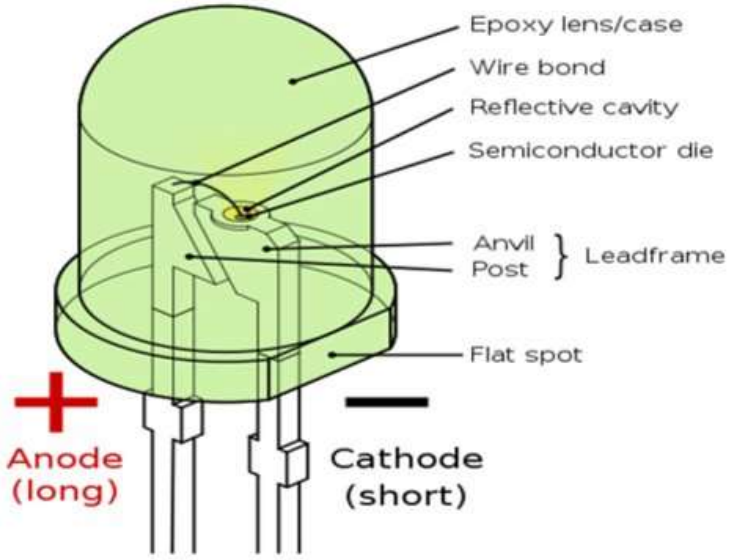


Figure 4.16 *LED Structure and its parts*

Cloud Service – ThingSpeak

Before we get to the ground lets understand ThingSpeak Cloud Service first. ThingSpeak is an open-source Internet of Things (IoT) application and API (Application Program Interface) to store and retrieve data from things using the HTTP (Hyper Text Transfer Protocol) and MQTT (Message Queuing Telemetry Transport) 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.

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. This platform enables the project to get the live status of the letterbox from anywhere. The basic working of this platform is by providing a channel with a set of unique secret write and read API key to the client. Figure 4.17 shows the basic channel for ThingSpeak which will be used in the project to get update about the count of letters.

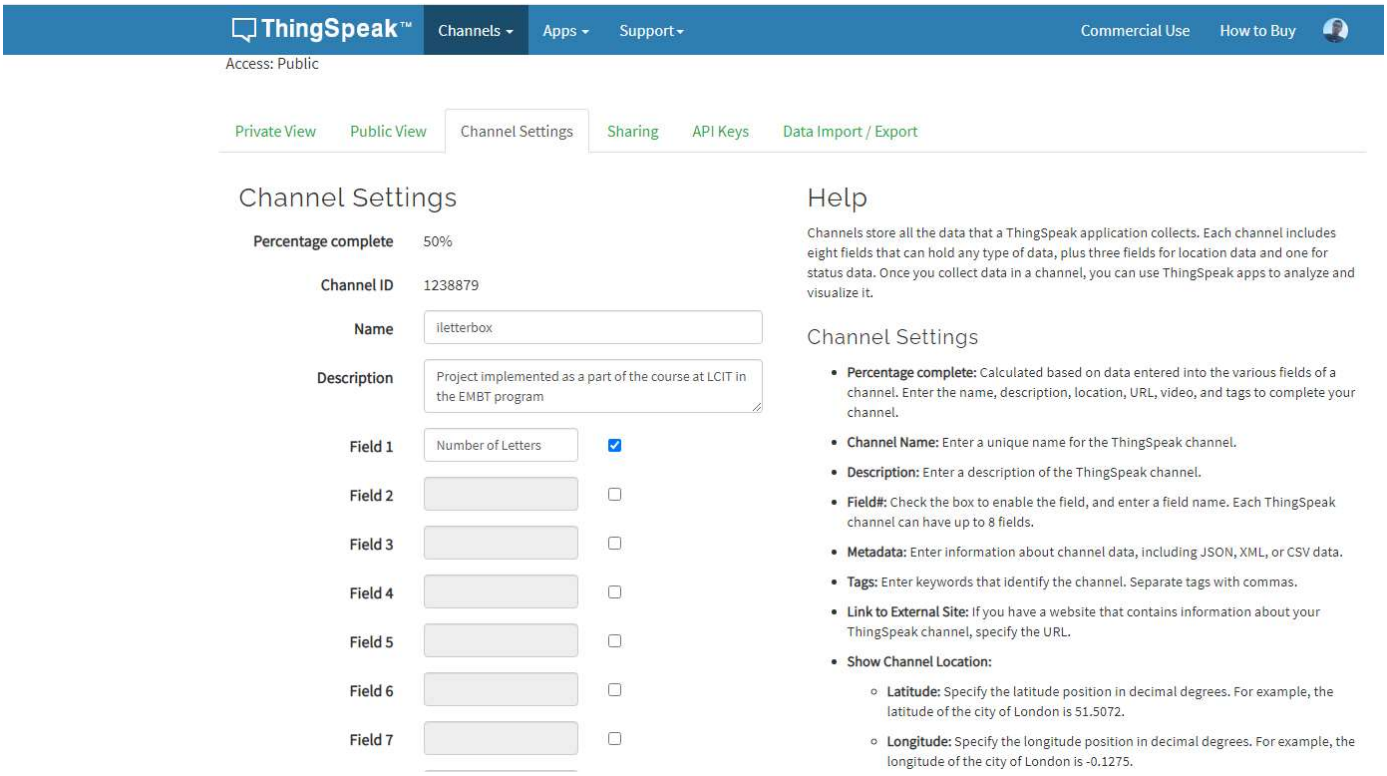


Figure 4.17 *ThingSpeak Channel Created for the Project*

As shown in code sample in Figure 4.18, the sensors read the data and push it to the hosted ThingSpeak server using an HTTP POST request using the SocketClient class.

```
/*---head data for server---*/
   head << "POST /update HTTP/1.1\n"
        << "Host: api.thingspeak.com\n"
        << "Connection: close\n"
        << "X-THINGSPEAKAPIKEY:7OE9JRDYIRM34MHQ \n"
        << "Content-Type: application/x-www-form-urlencoded\n"
        << "Content-Length:" << string(data.str()).length() << "\n\n";
```

Figure 4.18 *Commands used to connect to ThingSpeak*

Once when the data gets uploaded to this cloud service, we can view the data through various kinds of virtualisation methods like Graphs, Digit Display, Indicators and location where the channel is being used could also be added with the help of latitude and longitude coordinates. as per our requirements we can choose what we want to see. Also, the data can be exported into an excel sheet incase we want a history of data that was uploaded to the cloud service. Some of the visualization which we used for this project is shown in Figure 4.19.

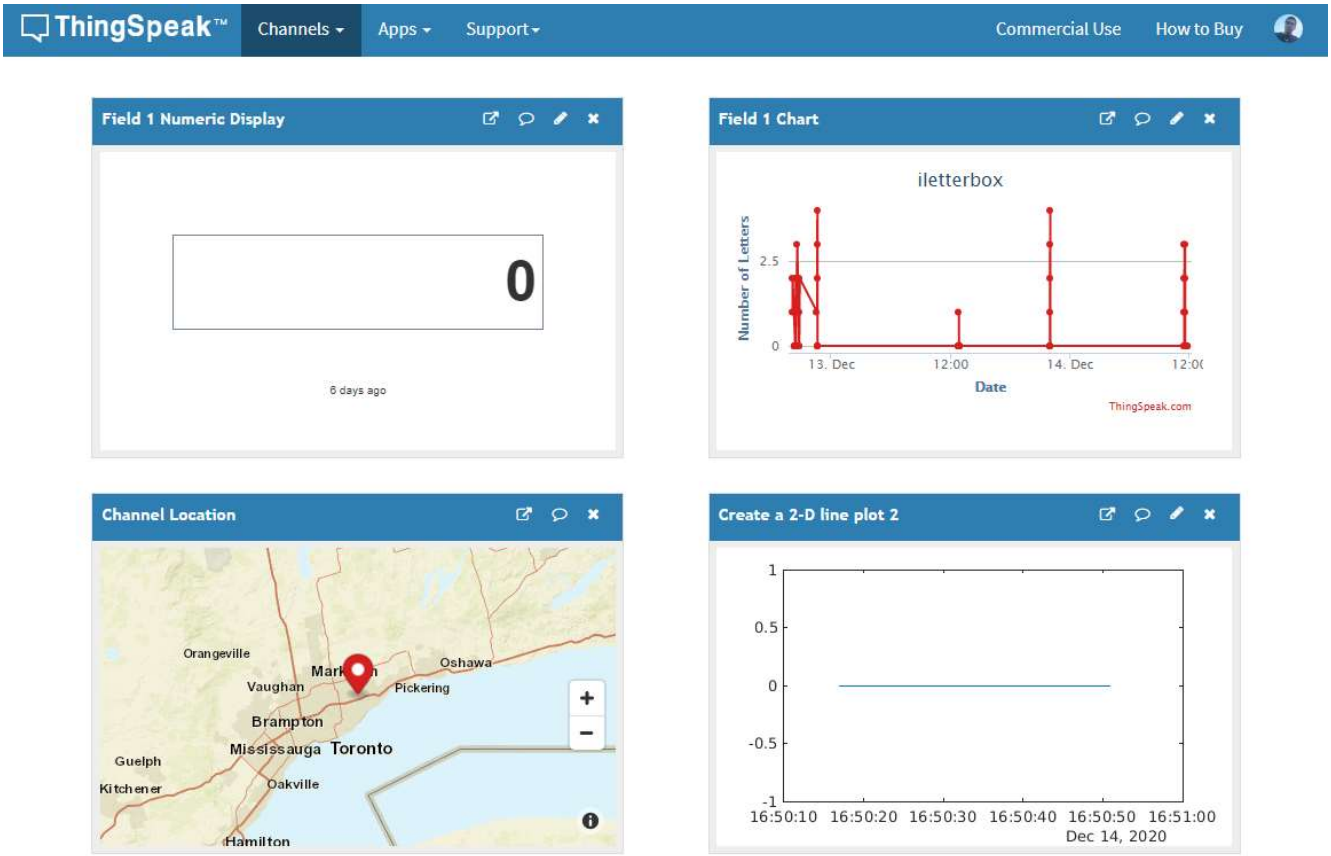
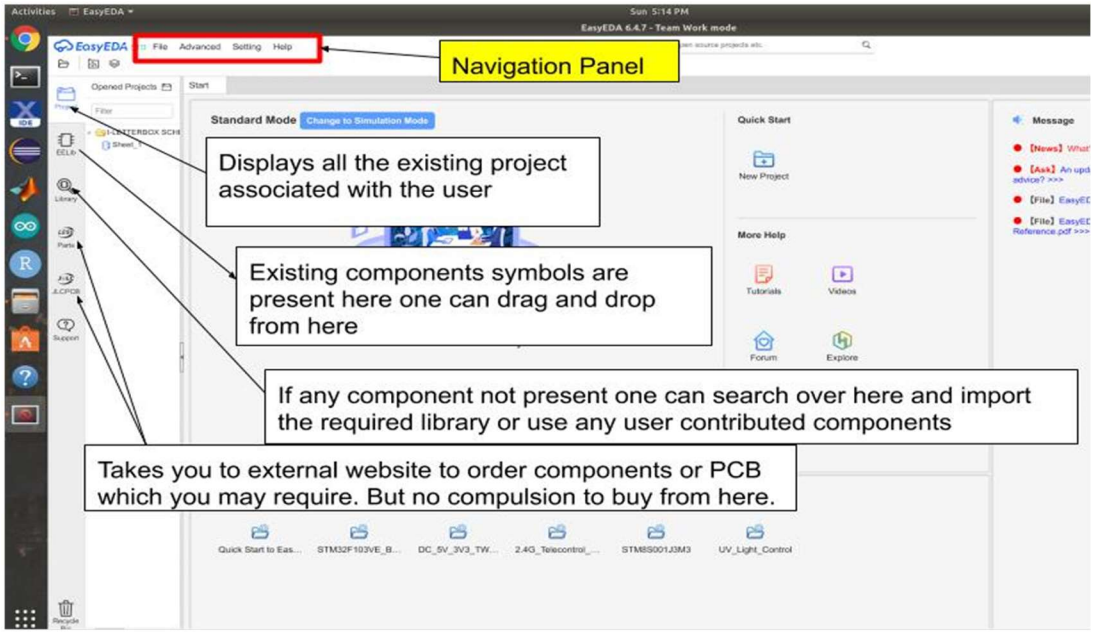


Figure 4.19 *Visualization in ThingSpeak*

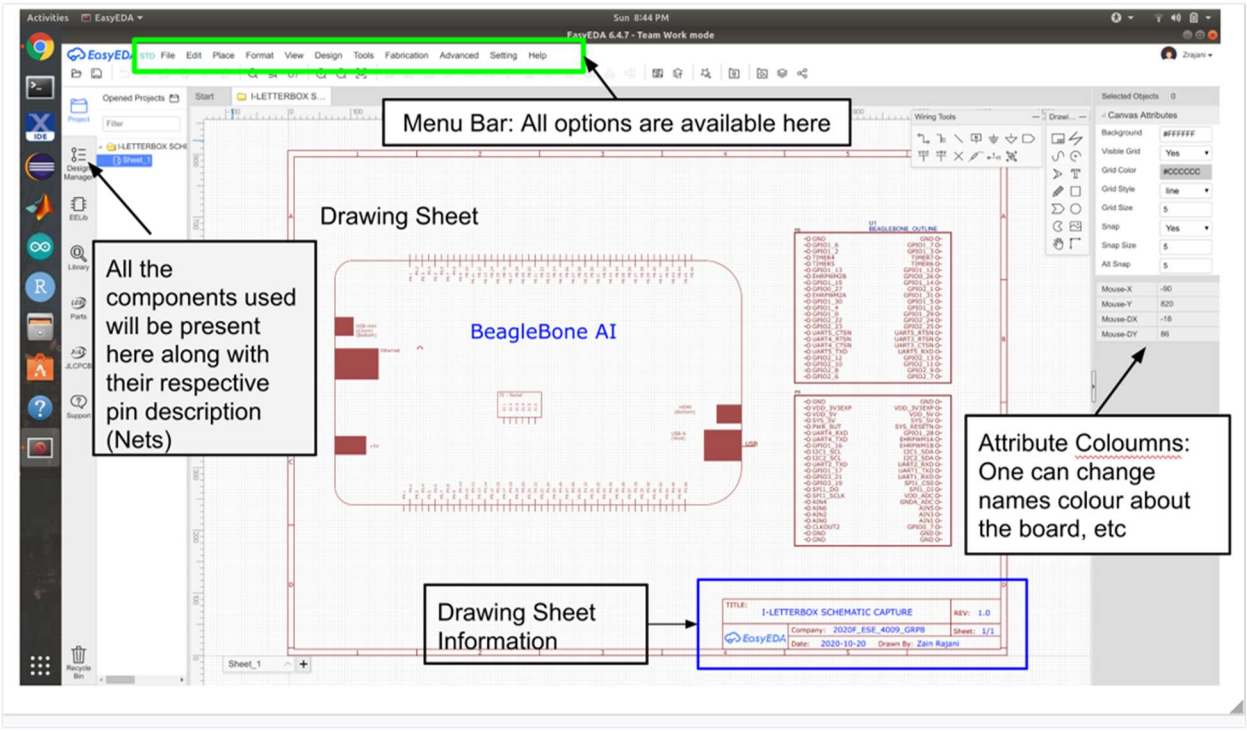
PCB Design

PCB design for our project is done on EasyEDA. It is an easier and powerful online PCB design tool that allows to design and share projects. Printable PCB layer image output is also supported in PDF, PNG, and SVG formats. Circuit designing is completed by using manual routing. We used the same schematics which we used to understand how we can draw out the connection between the Beaglebone Black Wireless and Sensors/ Devices.

Easy EDA has a large user contributed library thus people can easily find almost any component which they require for the project. To get a successful PCB design from a given schematic one just needs to be sure that the component has a suitable footprint else the design wouldn't be possible. One more advantage is, it can support other software libraries and schematic which includes Altium, EAGLE, LTspice, and DXF. Figure 4.20 shows the software of EasyEDA which we have used for PCB and schematic capture. All the parts of the screen have been labelled for better understanding.



(a) Introduction Screen



(b) Circuit Drawing Screen

Figure 4.20 *EasyEDA Software*

Figure 4.21 shows the component finding navigation screen which all the parts being labelled.

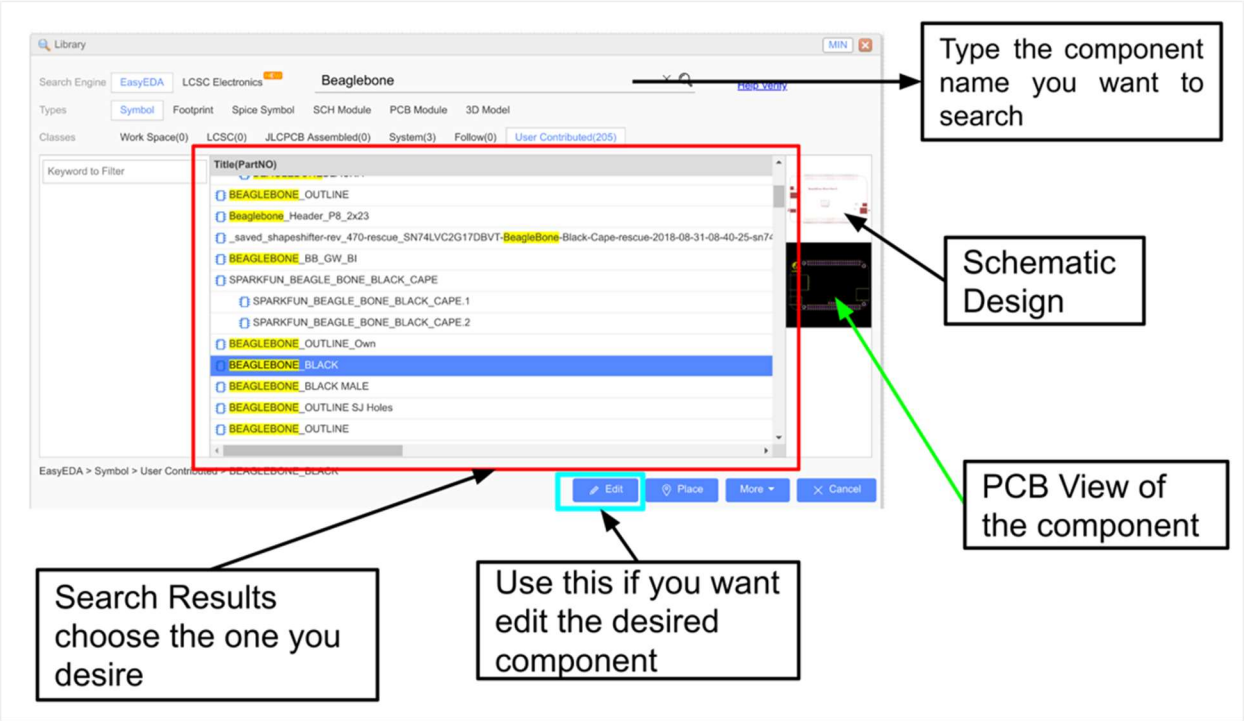


Figure 4.22 *Component Hunting and Navigation Screen*

Upon following the user manual guidelines for this software, we finally drew the PCB design for our project as shown in Figure 4.23. Through this software one can also generate the BOM (Bill of Material) for the project which will give you a list of components to be purchased to make this project. This BOM can be used as a checklist when one goes to the market to get the components.

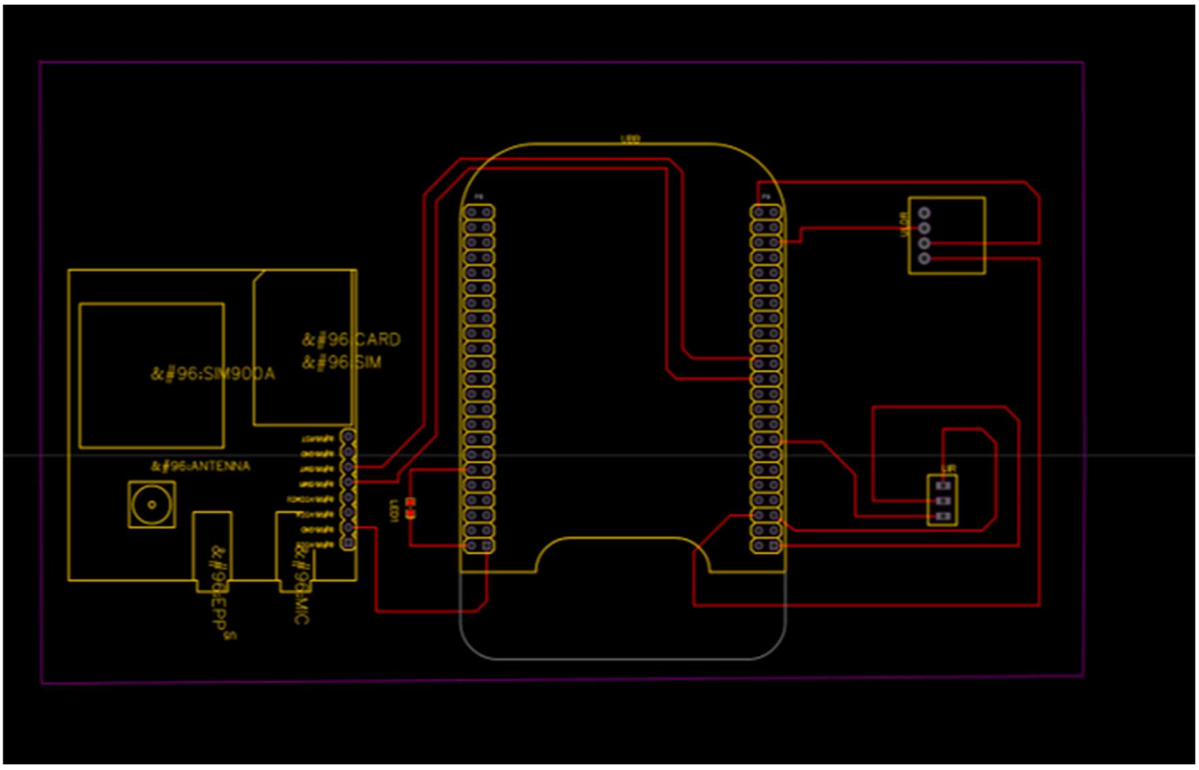
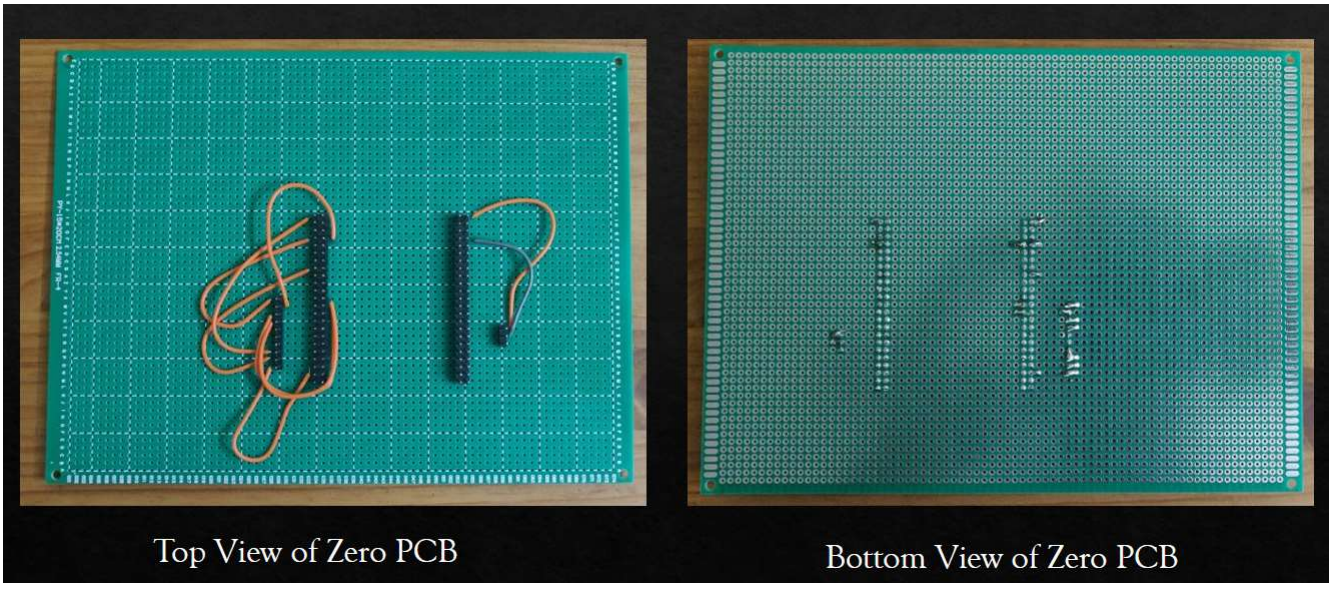


Figure 4.23 *PCB Design for the project*

Once when the PCB Design was ready for the project, we generated the Gerber files and gave it to the manufacturer to get it ready for us. The order was placed through EasyEDA website to JLCPCB which is a China based company the PCB got delivered to us within 7 days with cost approximately around $31.28 CAD for 5 PCB of the same design.



(a)

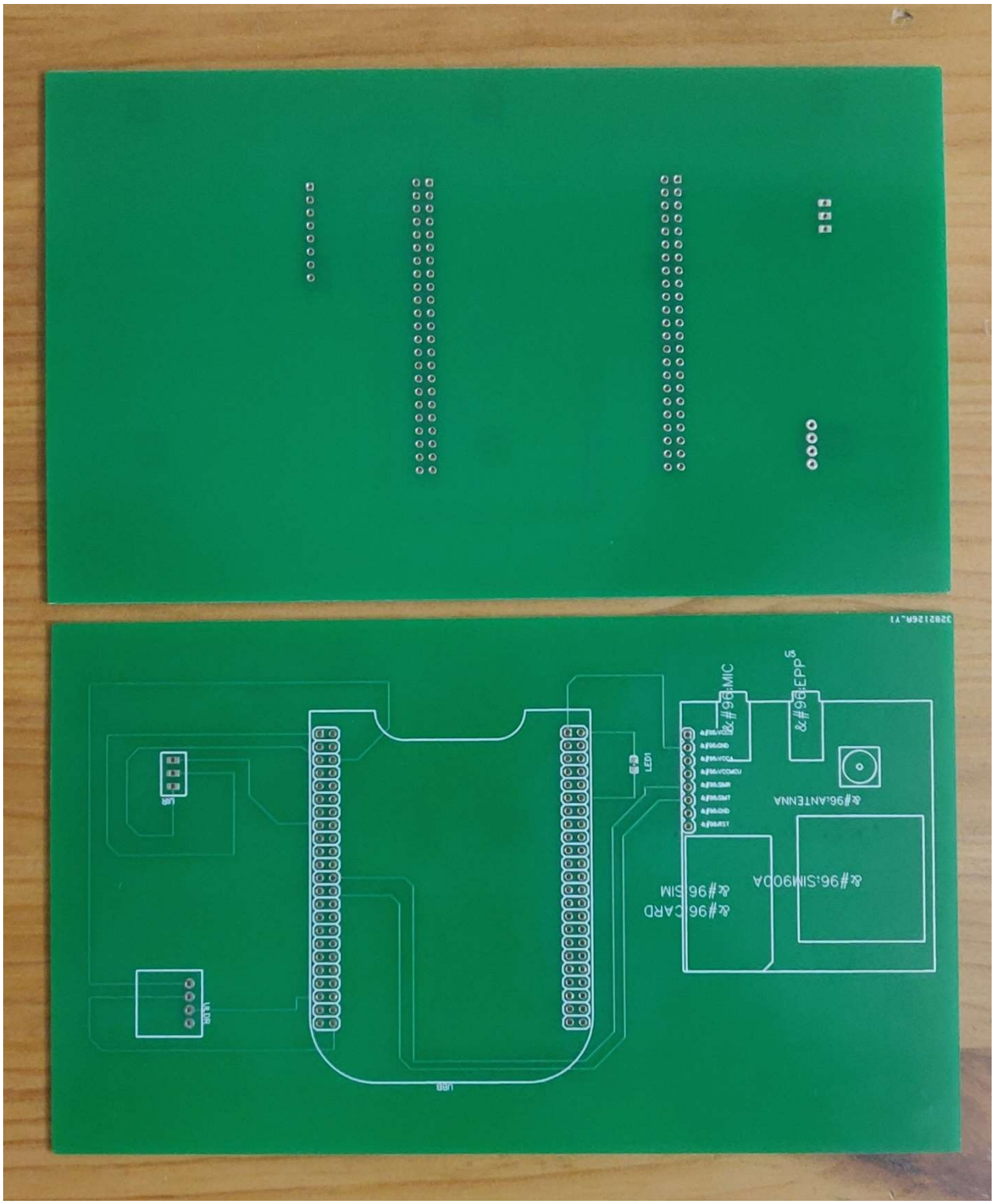


Figure 4.24 (a) *PCB Manufactured from JLCPCB (b) Zero PCB of the project*

The cost usually varies depending upon the size of the PCB we order. But in case one wishes to order then one needs to check for the files properly as when we order we didn't notice that the LED was surface mount and not through hole type thus it became a issue which we resolved it by

connecting a copper wire through it. The final PCB manufactured for the project is shown in Figure 4.24 along with the zero PCB we made as a backup in case the actual PCB if arrived late.

Product Design

Once the final project is done, we needed to get the design ready. Thus, for the user to understand our project working we drew some animated drawing and finally converted it into the letter box. The animated drawing also known as engineering drawing is shown in Figure 4.25.

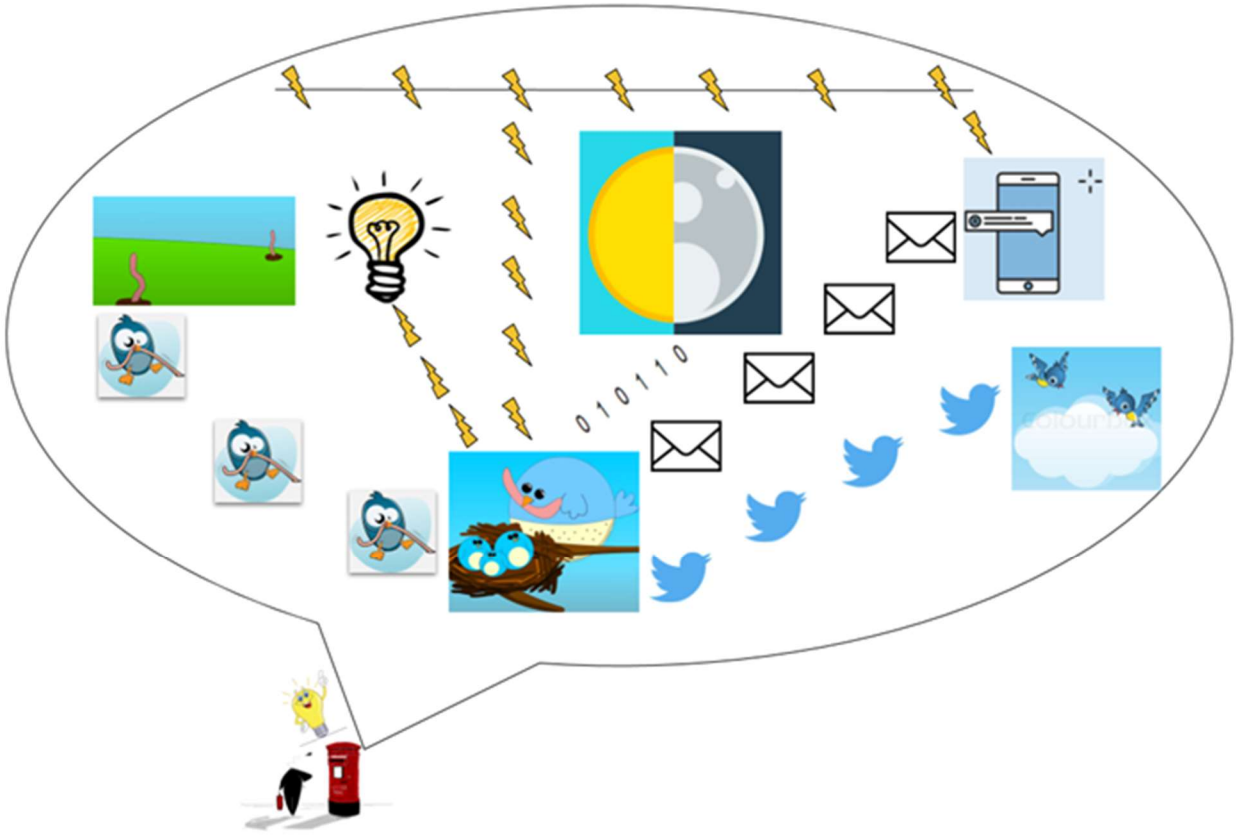
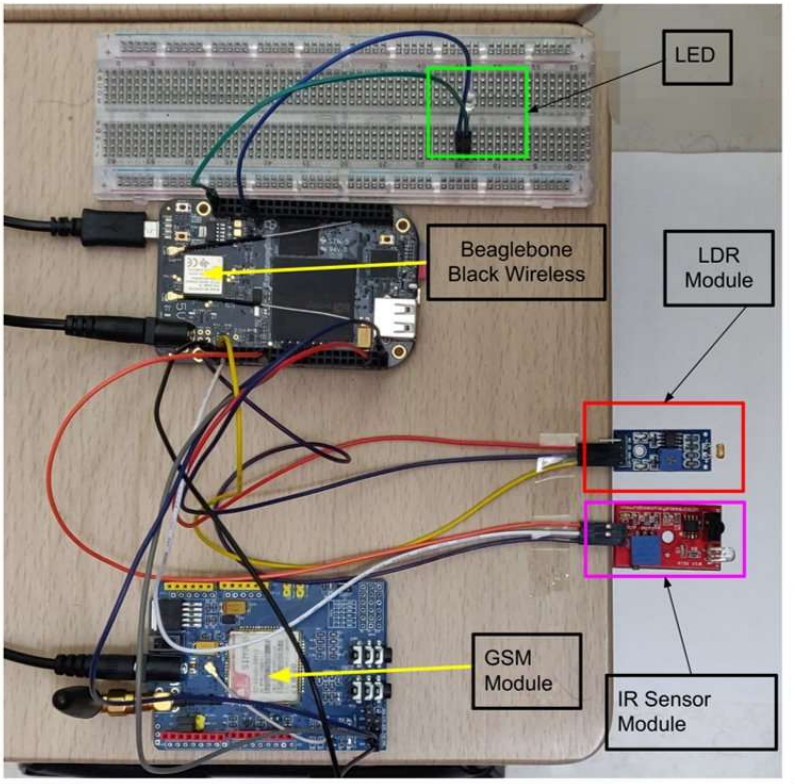


Figure 4.25 *Engineering Drawing of the project*

The final connection of the project along with the final product look is shown in Figure 4.26



(a)

Figure 4.26 *Product Design (a) Final Connections for the project (b) Final Product Look*

Chapter V
Implementation and Test

Interfacing with BB-WI

- **Interfacing Beaglebone Black Wireless with IR Sensor and LED:**

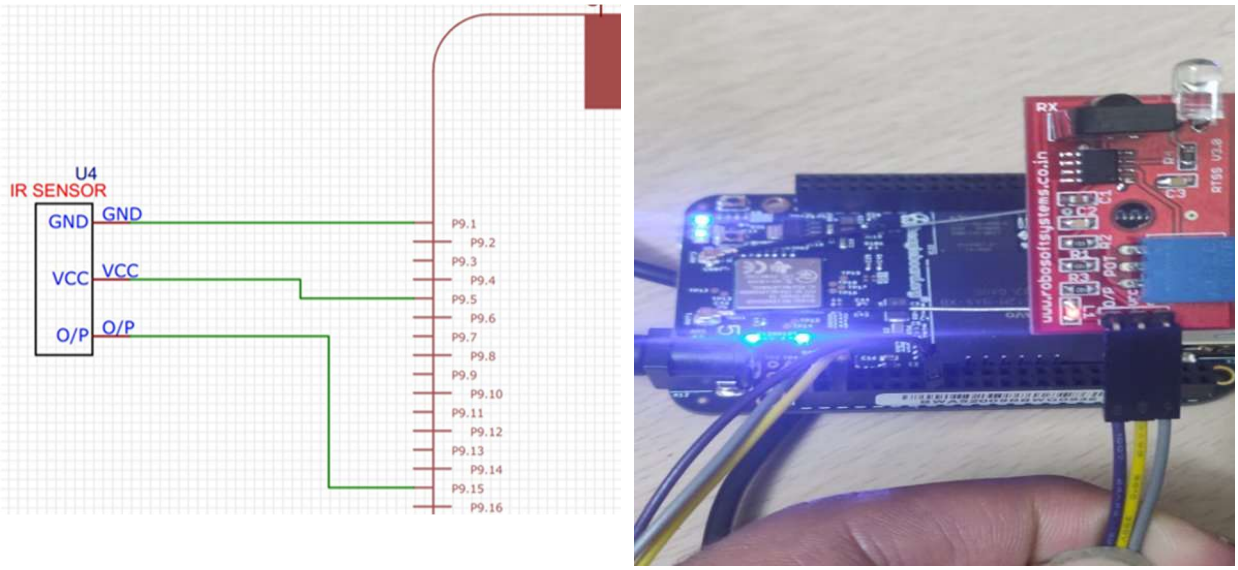


Figure 5.1 *Interfacing IR Sensor and LED with Beaglebone Black Wireless*

Figure 5.1 shows how the interfacing of IR Senor and LED is done with Beaglebone Black Wireless. The IR sensor works on 5V voltage. Since we power up the Beaglebone Black Wireless using power adapter we use P9.5 pin of the Beaglebone to power up the IR Sensor. The ground of the sensor is connected to ground of the Beaglebone Headers. Since we intended to use Digital communication, we want to use one of digital GPIO pins thus we use P9.15 pin of the Beaglebone.

Once the connections were done properly, we just needed to write the code for the same. The code listing for the same is shown below.

```
#include <stdio.h>
#include <unistd.h>      // For usleep function
#include <iobb.h>       // To access the GPIO pins easily library already
downloaded and installed

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

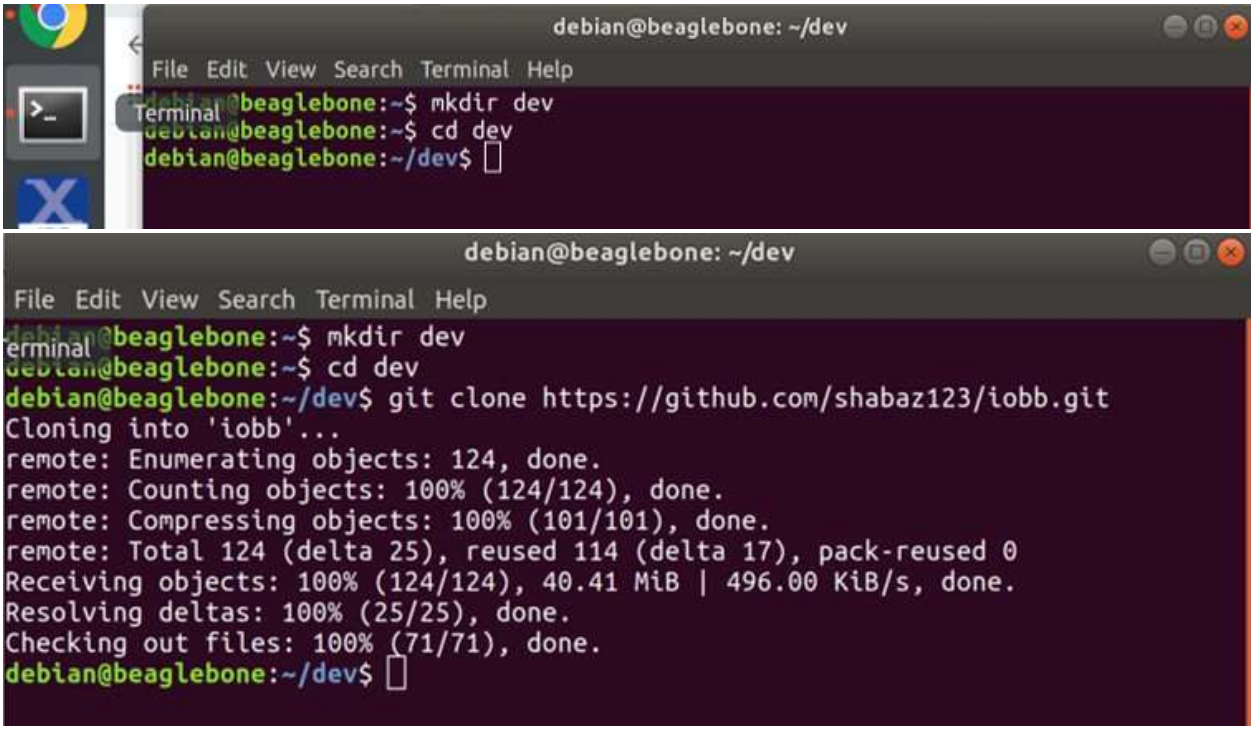
```
  //Initialize the GPIO function libraries
  iolib_init ();

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

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

  // Loops forever
  while (1)
  {
    // Check the input value of the pin
    if (is_low (9,15))
    {
      pin_high (8,11);
      counter++;
      printf ("Letter # %d/n", counter);
      usleep (120000); // Suspend the process for the given duration
    }
    if (is_high (9,15))
    {
      // If sensor detects no object do nothing get the LED reset and the
counter to 0
      pin_low (8,11);
    }
  }

// 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

}
```

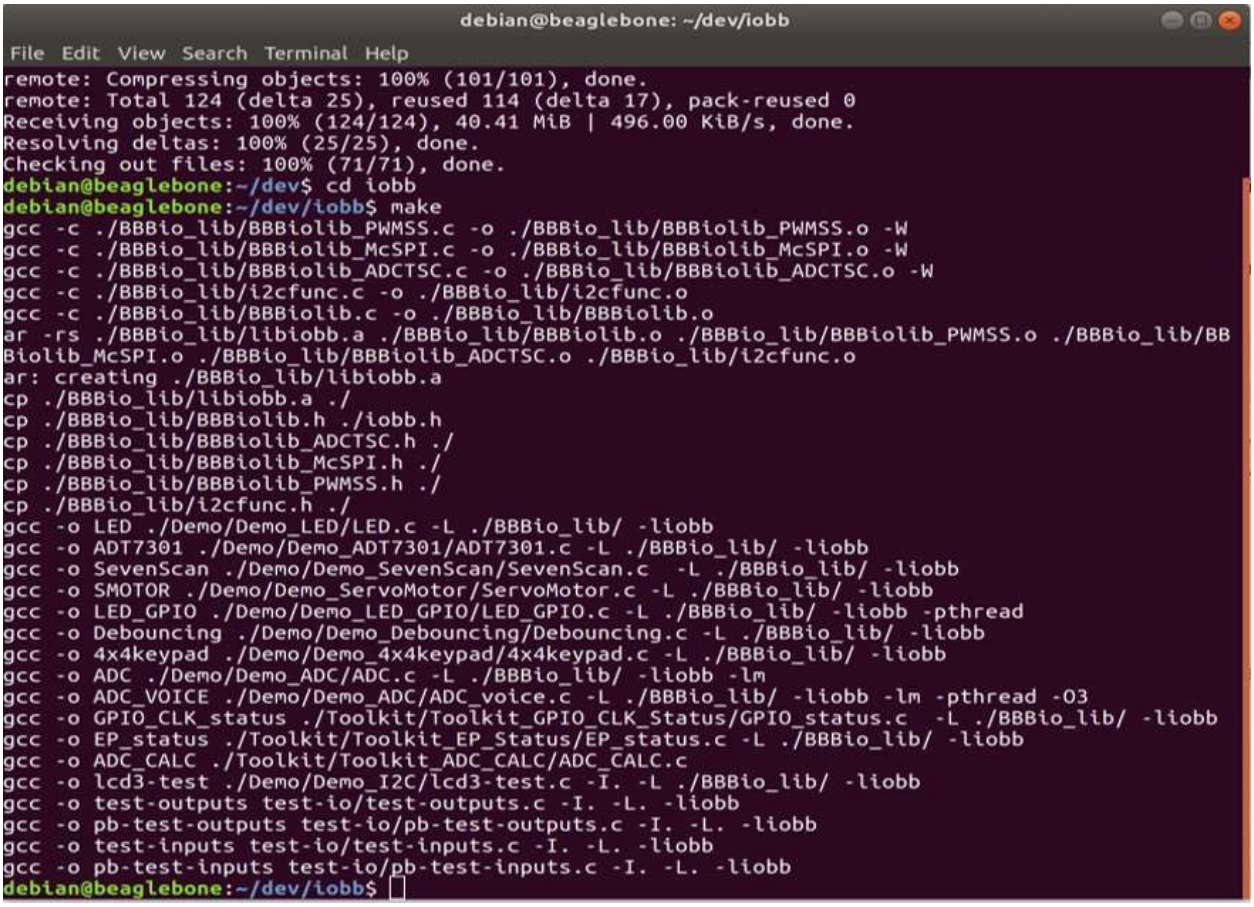
The code above as you see uses a user defined file which is named as iobb.h. Thus, to use this library we need to install this library. Once we install this library, we can use it for all future interfacing. The installation commands are as follows.

```
>mkdir folder_name
>cd folder_name
>git clone https://github.com/shabaz123/iobb.git
>cd iobb
>sudo make
>sudo make install
```

So, we can access the GPIO of the Beaglebone Black Wireless through this user made library which is available through GitHub platform. Once we clone this repository on our host processor

we then build this file and then using “make install” command we put the file in the by default path which is /usr/include. Figure 5.2 shows the steps on how we installed the library.





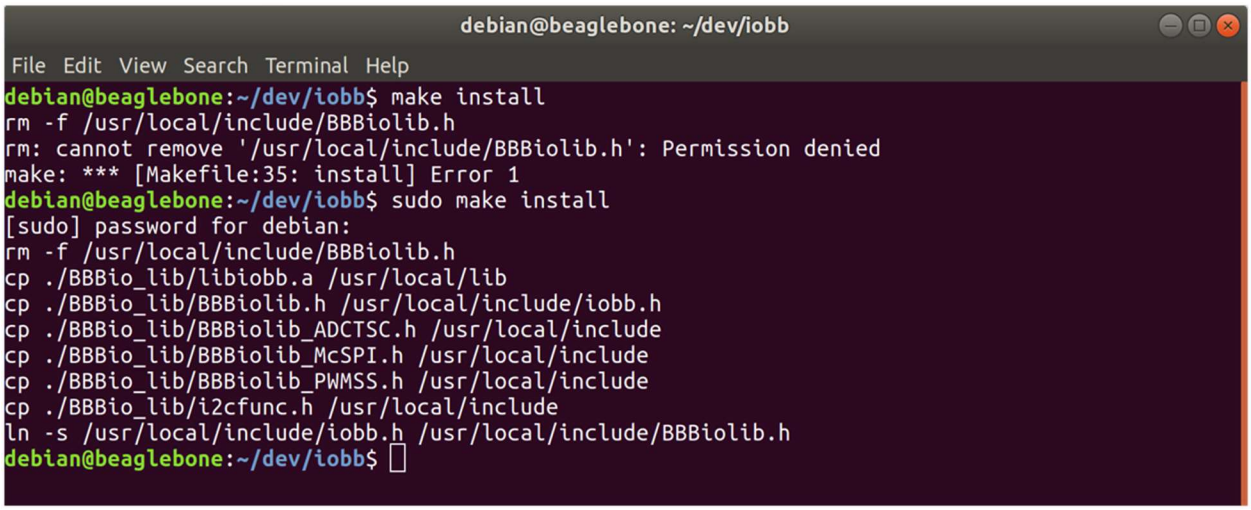


Figure 5.2 *Steps for installing the library to control BB-WI GPIO*

Next, we also need to connect LED for this purpose we use the P8 header. Thus, we directly connect the LED directly to the Beaglebone Black Wireless. We connect the positive leg of the

LED to the P8.12 pin of BB-WI which will supply 3.3 V and the negative leg to the ground of the BB-WI. The connection for the same and schematic for the same is shown in Figure 5.3.

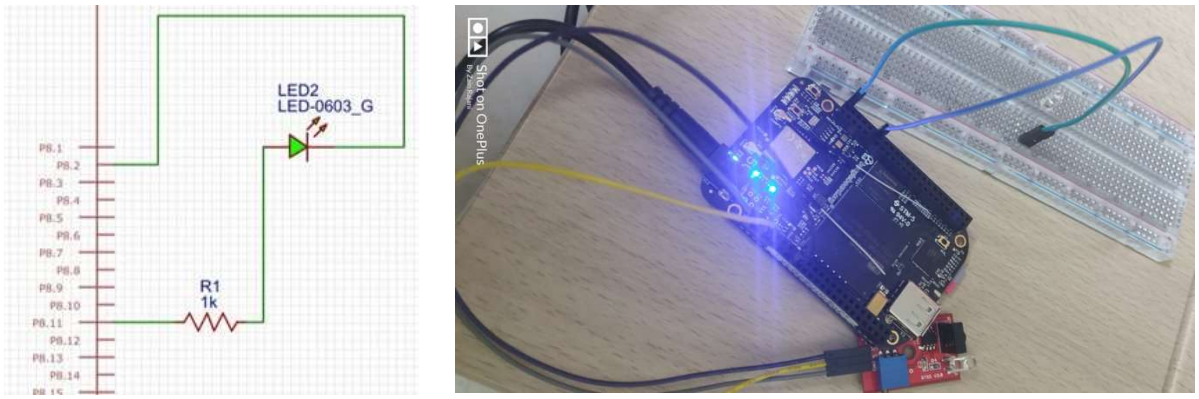
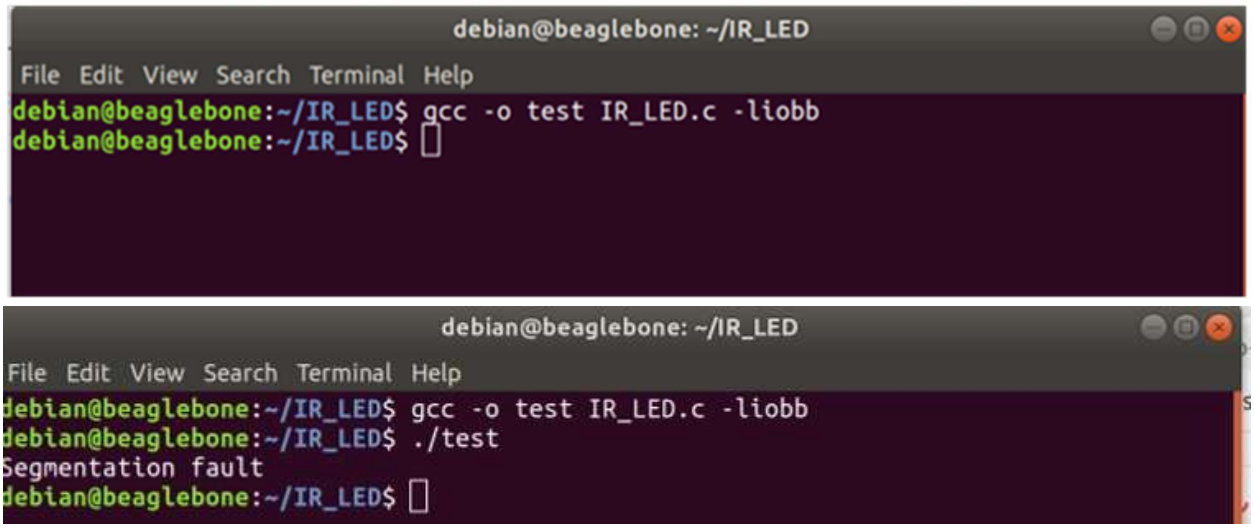


Figure 5.3 *Interfacing LED with Beaglebone Black Wireless*

Upon successful connection we have to compile the code and the results for the same can be seen in Figure 5.4. First, we compile the code using the "gcc" compiler and we also add the library to this compilation command. To compile the code, we need the following command:

```
gcc file_name.c -o output_file_name -liobb
```

The reason to add -liobb at the end is because we used a file which is not defined by the system thus the system should be aware that we have to inform the system about the same.



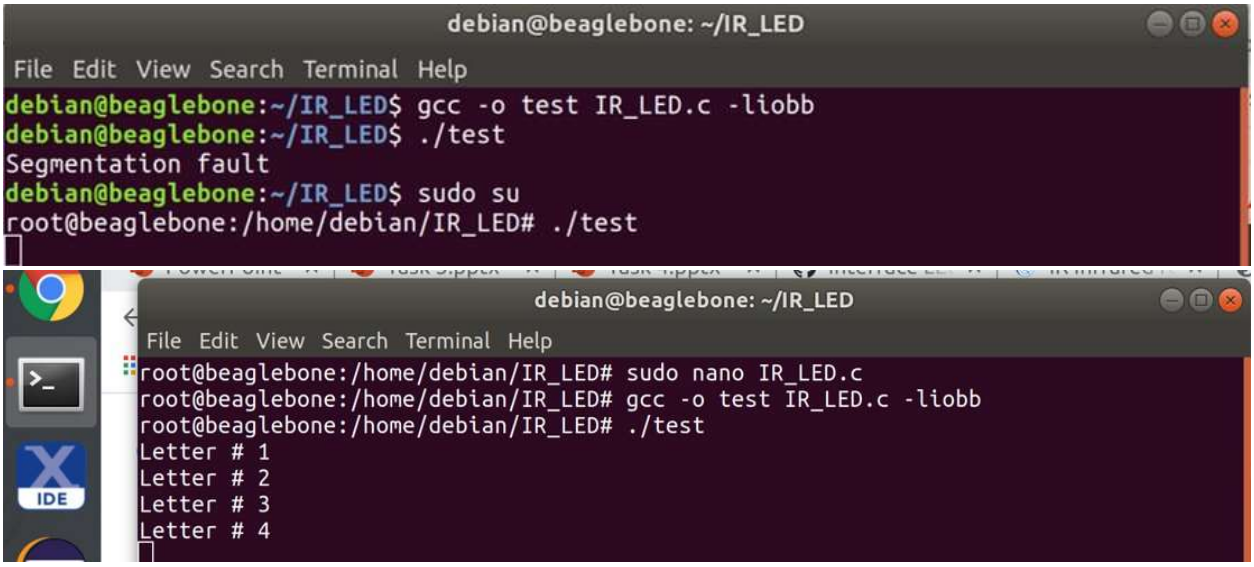


Figure 5.4 *Steps followed for executing the code for IR Sensor and LED*

As seen the letters are being counted by the IR Sensor and is being displayed on the console of the Begalebone Black Wireless. But notice clearly, we used the super user mode to run the executable file the reason being that if we want to gain the control of the GPIO pins we either need to be super user or admin as we want to access memory which is quite sensitive. The output on the hardware is shown in Figure 5.5

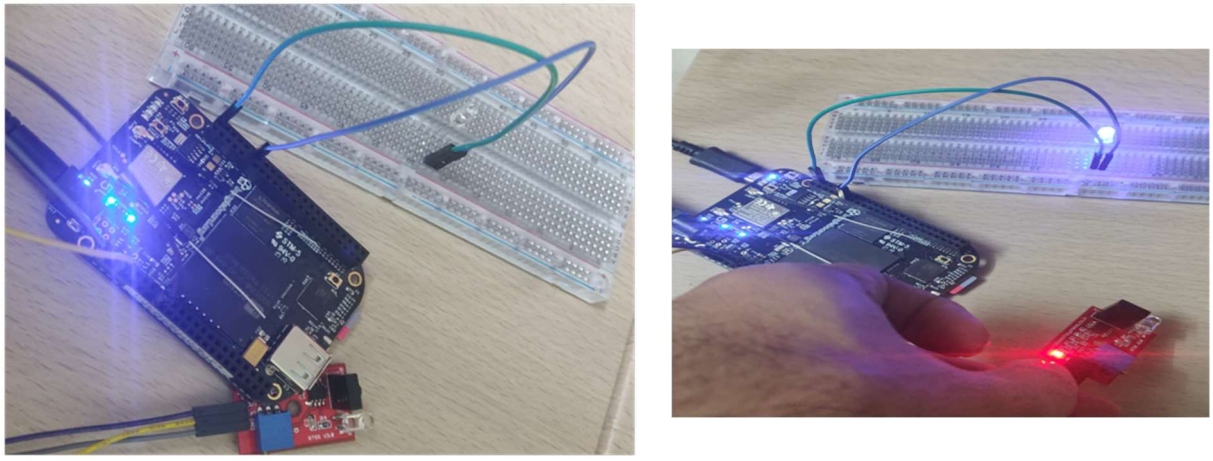


Figure 5.5 (a) *When program initially runs (Left)* (b) *When Letter arrives in the box (Right)*

- **Interfacing BB-WI with LDR**

After the previous interfacing which helped us in accomplishing the task for counting the letters the next thing was to get it reset to zero without any manual reset. Thus, we use LDR for this

purpose. Depending upon the intensity of light that falls on this sensor we get the counter value back reset to zero. For the purpose of the project, we assume that there is sufficient amount of light in the surrounding when we remove the letters. This light will help us in setting the counter value set to 0 which would mean that all letters have been removed from the box. The connection for the same have been done by following the schematic. The connection and schematic for the same is shown in Figure 5.6.

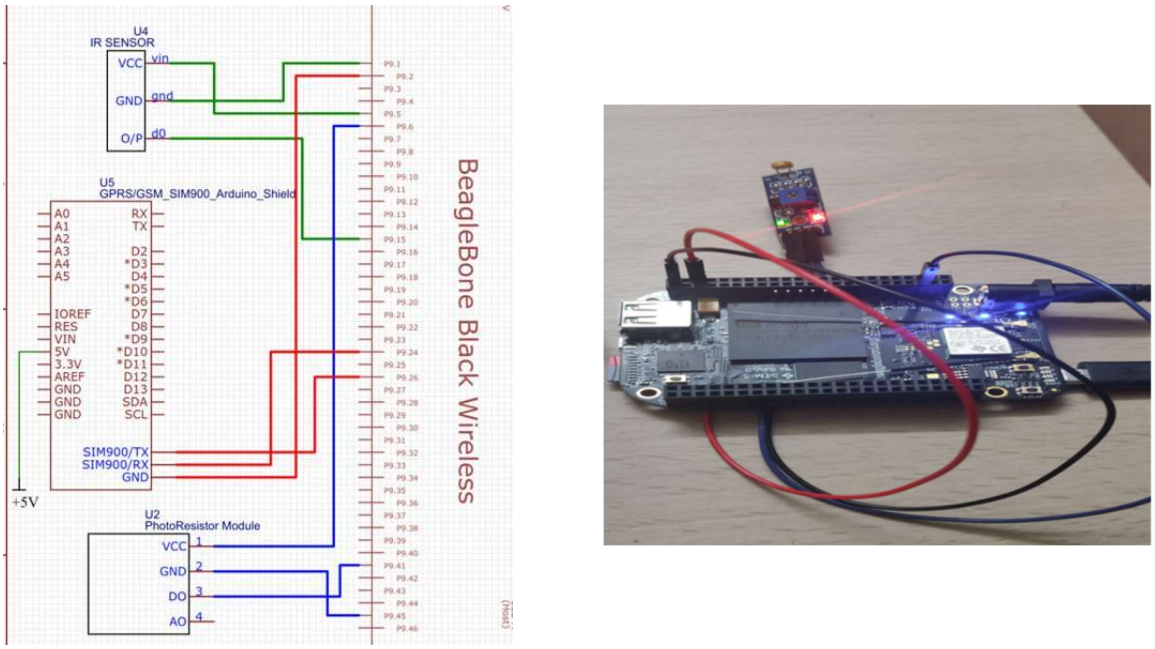


Figure 5.6. *Connection for LDR Module to BB-WI*

As seen from the Figure 5.6 we see that we use the bottom pins of the P9 header of the Beaglebone Black Wireless. Since we are concerned here with just the brightness or darkness not with the variations of the intensity, we use the digital pins of the module. The sensor is powered up using 5V and unlike the IR sensor this module too is powered up using P9.6 as we use power adapter. The output of the LDR would be available through P9.41 and the ground of the sensor module is connected to the ground of the Beaglebone Black Wireless like P9.46. Upon successful connection of the sensor, we the next require to code the sensor to perform its respective task this would be accomplished by the code listing shown below.

```
#include <stdlib.h>
// For system(...) command to run some external commands in C Program
#include <stdio.h>
// For Standard Input Output Function like printf (), scanf () and more
#include <unistd.h>      // For usleep function
#include <iobb.h>       // To access the GPIO pins easily library already
downloaded and installed

int main ()
```

```
{
  // Variable Declarations;
  int counter=0;

  //Initialize 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_high (9,41)))
    {
      pin_high (8,11);
      counter++;
      printf ("Letter # %d \n", counter);
      usleep (100000);
   }
    // If no letter is sensed and there is darkness then
    if ((is_high (9,15)) && (is_low (9,41)))
    {
      // If sensor detects no object do nothing only reset the counter if
there is darkness and send an alert for removal of the letters
      counter=0;
      printf ("Counter Reset # %d\n", counter);
      pin_low (8,11);
    }

  }
// 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
}
```

From the above code listing we see that first we use the same library which we used for the IR Sensor to get control of the GPIO pins. Next, we use the logic that if IR sensor detects object (sensor functions as active low) and if brightness is low (sensor behaves as active low) we increment the count of letters else we let the counter be the same unless and until the brightness goes high. For this purpose, we use logical operation i.e., and operation which means the counter will be incremented only when both the conditions are true.

Once we write the code using nano editor, we can then compile the same code using “gcc” compiler. The syntax for the same is shown in the previous interfacing. Figure 5.7 shows the compilation of the code and output on the console.

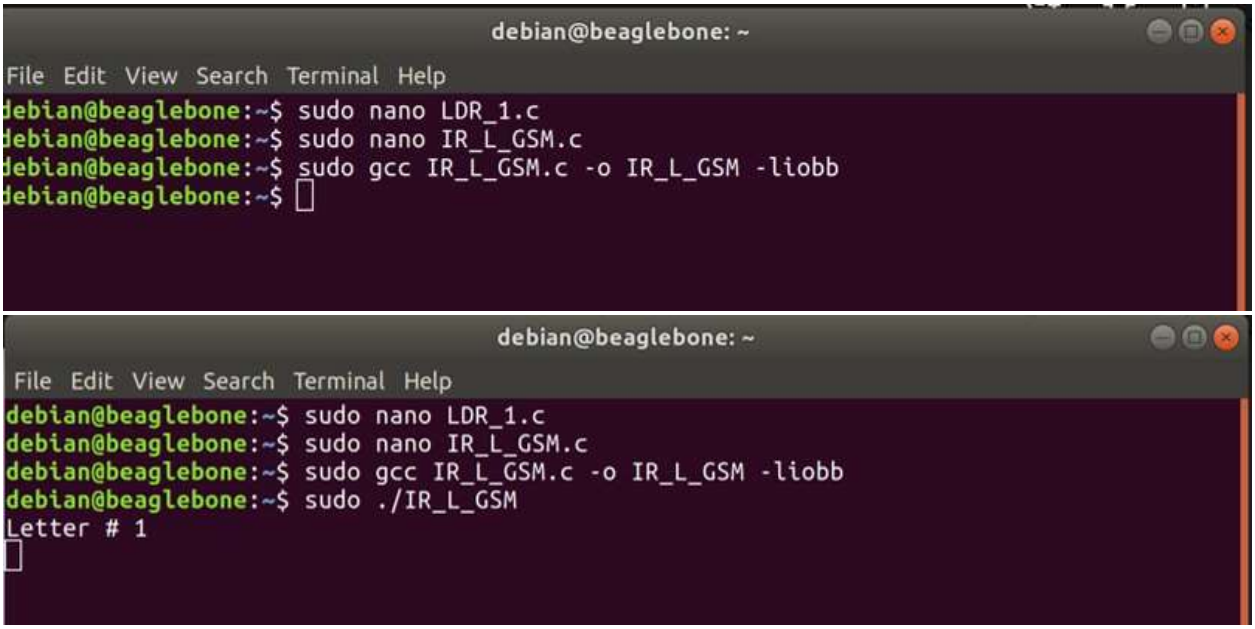


Figure 5.7 *Program Execution of interfacing LDR with BB-WI*

The program when executed also will be shown through hardware as well this is shown in Figure 5.8.

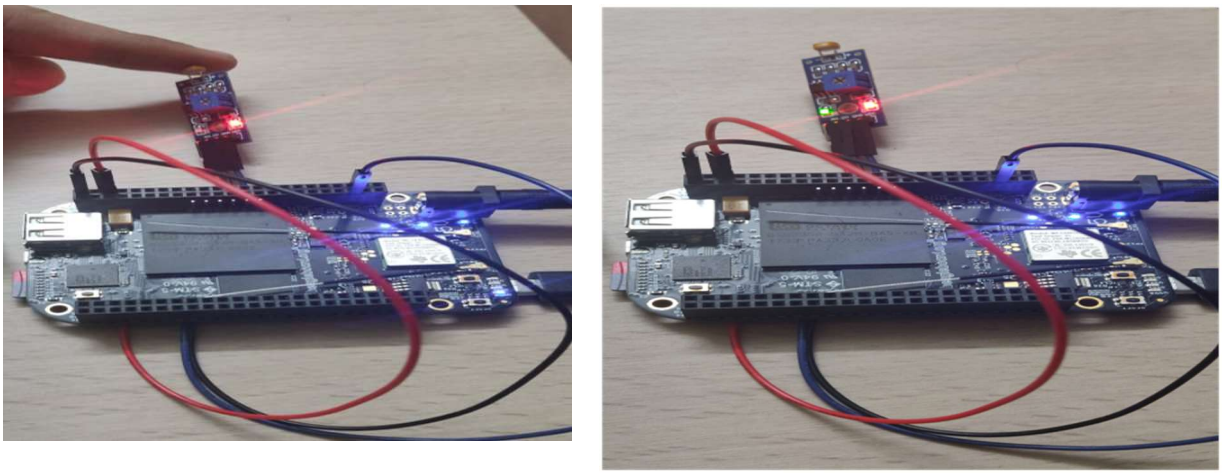


Figure 5.8 *Execution of LDR with BB-WI on hardware*

- **Interfacing BB-WI with GSM SIM900 Module**

Once we settled up the counters which perform their tasks as they have to. We further on moved in the next phase of the project to sure that user is informed when the counter value changed via a SMS through the GSM Module. The user receives a message as soon as the letter is deposited in the box or when the letter box is opened and we again assume here that when the letter box is opened all the letters from the box will be removed. For this interface we used the UART communication protocol. In simple terms UART is related to receiver and transmitter section of the module thus when the Beaglebone transmits a command to the GSM module when the command is received the module will send the message through the cellular network.

In order to use the UART pins first we need to configure the pins of the Beaglebone Black Wireless to function in the UART mode. For this purpose, we first need to look for the pin out diagram of the Beaglebone Black and see which pins can function as UART and thus configure the pins using the following command:

```
>config-pin Px.y mode_name
```

Where x stands for header number, y stands for pin number on that header and mode_name stands for the modes which the pin can be configured into like UART, I2C, SPI, etc. For this interface purpose we use UART1 which is located on the header 9 of the Beaglebone at pin 24 and 26 for receiver and transmitter respectively.

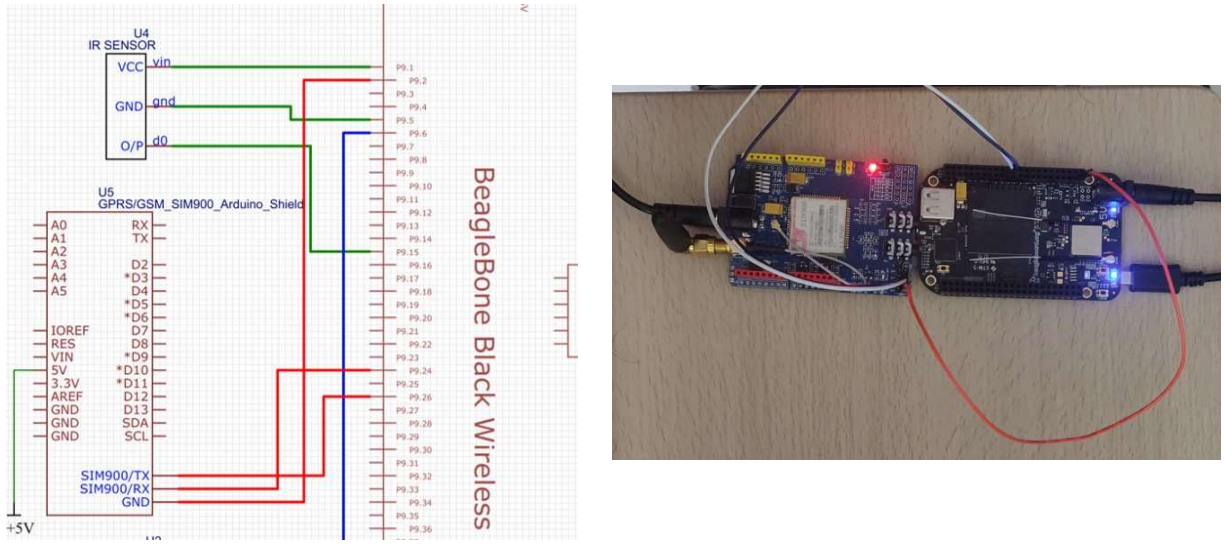


Figure 5.9 *GSM Interfaced with Beaglebone Black Wireless*

Thus, to summarise the connection are like P9.24 which is the UART transmitter of the Beaglebone is connected to the GSM SIM900 receiver, P9.26 which is the UART receiver pin of the Beaglebone is connected to the GSM SIM900 receiver pin, Ground of the GSM Module is connected to ground of the Beaglebone Black Wireless. The module is powered up using the power adapter having rating of 5V and 2A. These connections are shown in Figure 5.9 by following the schematic.

Once the connections are completed by following the schematic and accomplishing the entire setup shown in Figure 5.9, we have start to program so that we can communicate with the GSM Module to send SMS to the user. For this purpose, we used python programming message to send message when the letter arrives and a separate python programming code when the letters are removed. The code listing for the same is as below.

```
from time import sleep
import serial
from curses import ascii

##set serial
ser = serial.Serial()

##Set port connection to USB port GSM modem
ser.port = '/dev/ttyO1'

## set older phones to a baudrate of 9600 and new phones and 3G
modems to 115200
## ser.baudrate = 9600
ser.baudrate = 115200
ser.timeout = 1
ser.open()
def sendsms(number,text):
    ser.write('AT+CMGF=1\r\n')
    sleep(2)
    ser.write('AT+CMGS="%s"\r\n' % number)
    sleep(2)
    ser.write('%s' % text)
    sleep(2)
    ser.write(ascii.ctrl('z'))


sendsms('+16477868334','ALERT: NEW LETTER HAS ARRIVED')
```

As seen in the above code listing, we see that unlike to gain the control of GPIO in C program we used a iobb.h library thus in the python code we have a serial module which will help us gaining the control of the GPIO. Once done with that we next have to set the UART port we wish to use and then set the baud rate of the module since we used a 4G in the module we set the baud rate to be 115200 and then using AT commands we send the SMS to the desired mobile number. A similar code listing can be seen below for python programming when the letters are removed from the letterbox.

```
from time import sleep
import serial
from curses import ascii

##set serial
ser = serial.Serial()

##Set port connection to USB port GSM modem
ser.port = '/dev/ttyO1'

## set older phones to a baudrate of 9600 and new phones and 3G modems to
115200
ser.baudrate = 9600
ser.baudrate = 115200
ser.timeout = 1
ser.open()
def sendsms(number,text):
    ser.write('AT+CMGF=1\r\n')
    sleep(2)
    ser.write('AT+CMGS="%s"\r\n' % number)
    sleep(2)
    ser.write('%s' % text)
    sleep(2)
    ser.write(ascii.ctrl('z'))


sendsms('+16477868334','ALERT: LETTERS REMOVED FROM THE LETTERBOX')
```

Upon writing these codes we need compile them up and see if there is any error if no errors are found then we need to get these codes integrated into C programming language. The reason for doing this is because as promised in our proposal we will do majority of the coding in C language and for very few parts we will use python programming language. Thus, to integrate these codes together we use the **system ('command')**. The system command will suspend the current on going program and execute the command present in the system command and return back to the execute the code after its completion. This can be seen in the code listing below.

```
#include <stdlib.h>        // For system(...) command to run some external
commands in C Program
```

```
#include <stdio.h>        // For Standard Input Output Function like
printf(), scanf() and more
#include <unistd.h>      // For usleep function
#include <iobb.h>       // To access the GPIO pins easily library already
downloaded and installed

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 /home/Debian/pysms.py");
      counter++;
      printf("Letter # %d \n",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 an alert for removal of the letters
      counter=0;
      printf("Counter Reset # %d\n",counter);
      pin_low(8,11);
      system("/usr/bin/python2.7 /home/Debian/pysend.py");
    }

  }
```

```
// 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
}
```

The above code is used and is running while the previously connected sensors are still on. This will help us in knowing if we have a proper code running. The above codes are written in nano editor and the C program is complied using the "gcc" compiler. When the code generates the binary file and when executed we see that when the letter arrives in the box and when the brightness is low a text message is received on the cell phone and also when the letter is removed from the box again a text message shall be received on the cell phone. For each entry of letter, a message shall be sent to the cell phone. Figure 5.10 shows how the code is executed and Figure 5.11 shows the message is received on the cell phone when program is executed.

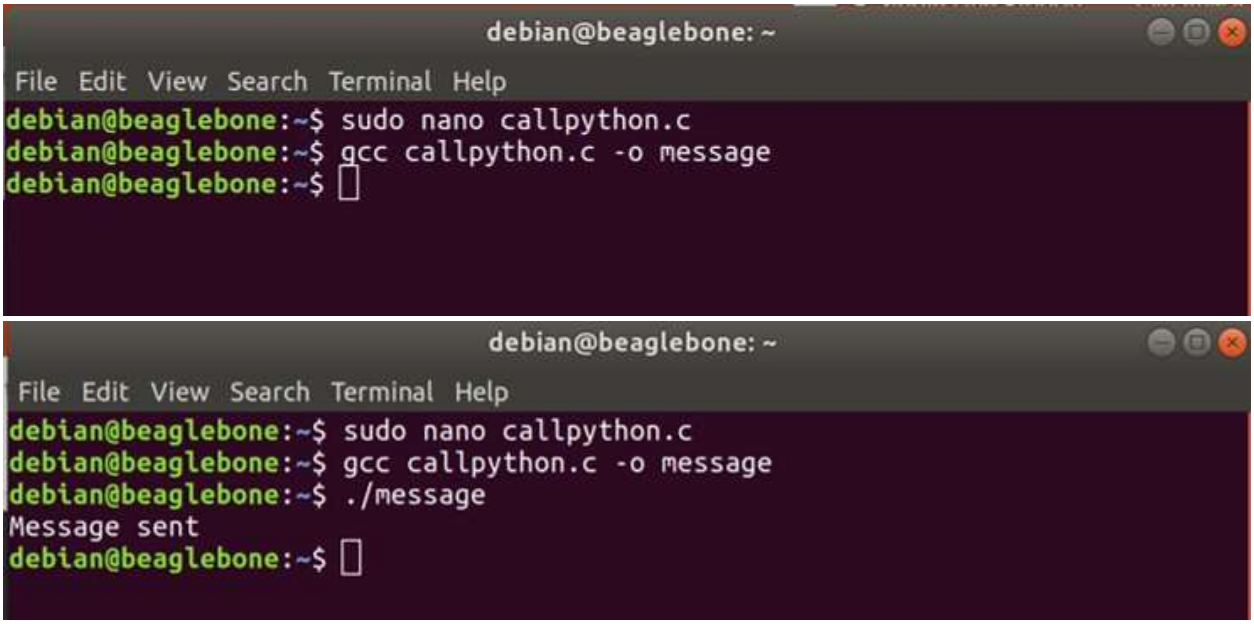


Figure 5.10 *Program Execution for interfaced GSM Module*

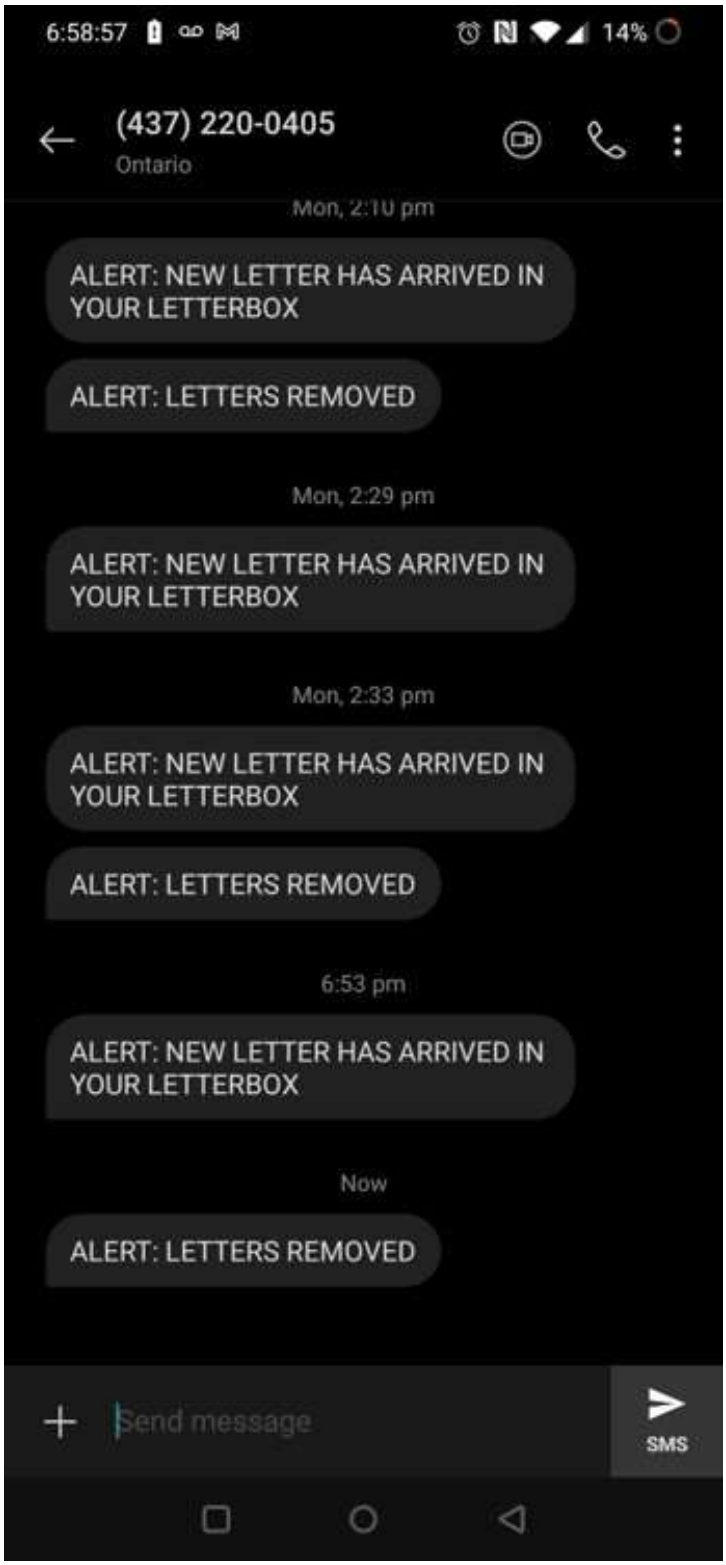


Figure 5.11 *Message Received using GSM Module*

- **Interfacing BB-WI with the cloud service (ThingSpeak)**

As the next and final step of the project as promised in the proposal, we have to upload the data to the cloud service. The cloud service that we have chose for this project is ThingSpeak but to get connected to this cloud service we first require a suitable internet connection with an acceptable internet speed. Figure 5.11 shows how we start the connection manager in order to get connected to the Wi-Fi.

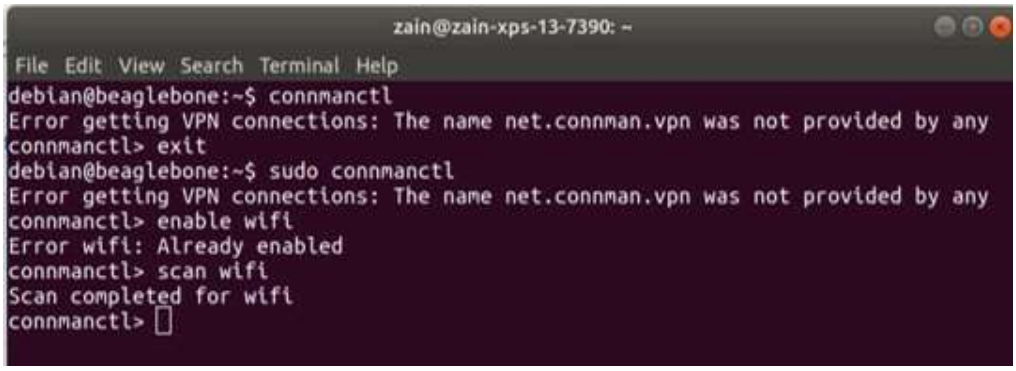


Figure 5.11 *Starting the Connection Manager*

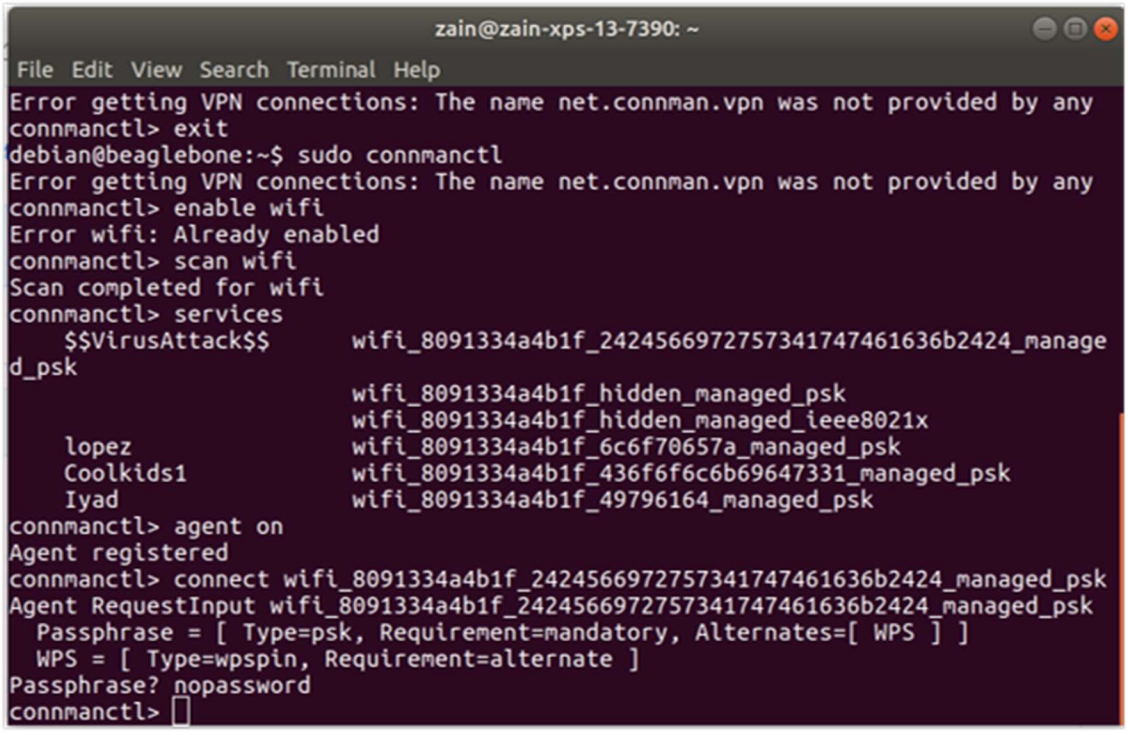


Figure 5.12 *Steps to get enable and connect to Wi-Fi*

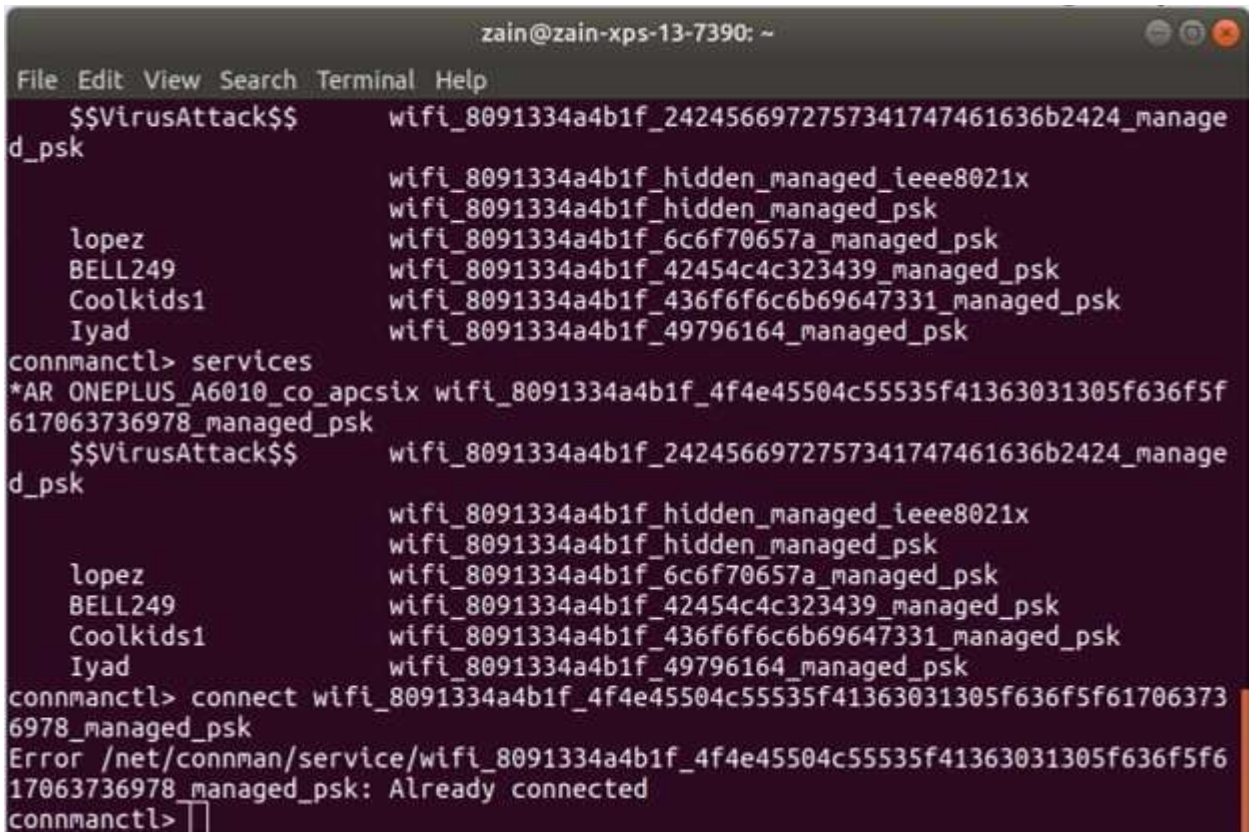


Figure 5.13 Steps to Get Connected to Internet

These instructions to get connected to the Wi-Fi can be found easily on the internet. The steps are shown in the Figure 5.12 and 5.13 for reference purpose we write it down as follows:

1. Connect to the BBW
2. Run the network manager

```
debian@beaglebone:~$ sudo connmanctl
[sudo] password for debian:
Error getting VPN connections: The name net.connman.vpn was
not provided by any
```

3. Enable Wi-Fi

```
connmanctl> enable wifi
Error wifi: Already enabled
```

4. Scan for Wi-Fi

```
connmanctl> scan wifi
Scan completed for wifi
```

5. Show nearby available services

```
connmanctl> services
```

This command will list all the close by networks to which the Beaglebone Black Wireless can connect to.

6. Turn on the Agent

```
connmanctl> agent on
Agent registered
```

7. Connect to the suitable network: Copy network ID of interest from step 5 and paste after ‘connect’.

```
connmanctl> connect wifi_f45eab2e3825_47546f74686572_managed_psk
```

8. Enter Wi-Fi password if needed

```
Passphrase?
Connected to wifi_8030dcea18e4_464953_managed_psk
```

9. Quit/ close the connection Manager

```
connmanctl> quit
```

10. Check the connection

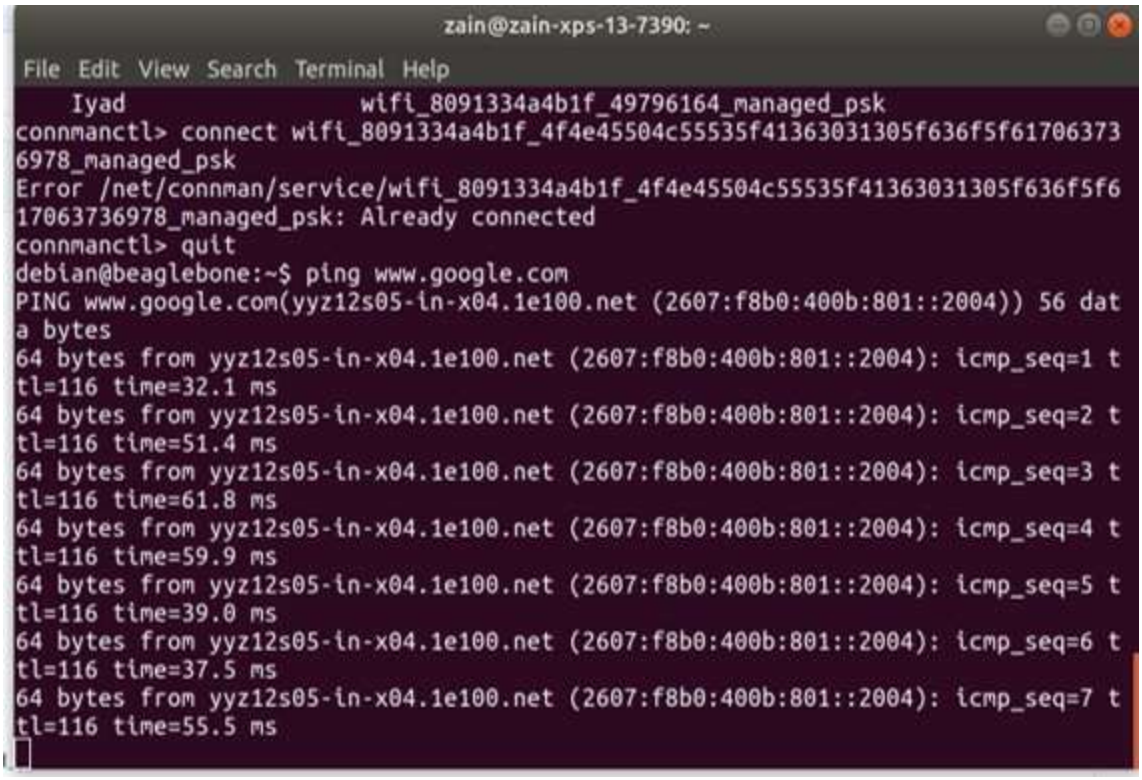


Figure 5.14 *Connected to the internet*

Now that we are connected to the internet, we can connect to the cloud service via HTTP protocol. Since we require the SocketClient class which is directly adapted from book "Exploring Beaglebone" by Derek Molloy. Thus, we saw a need that we might have to deviate this time to C++ programming language which was fine as we still don't have to change any codes which we previously wrote in C language. The main reason for using C++ language is due to ostringstream function which we require to get communicated to the cloud service. The code listing to get connected to the cloud is shown below.

```
#include <stdlib.h>          // For system(...) command to run some external
commands in C Program
#include <stdio.h>          // For Standard Input Output Function like
printf(), scanf() and more
#include <unistd.h>        // For usleep function
#include <iobb.h>         // To access the GPIO pins easily library already
downloaded and installed
#include <iostream>      //  For input output cin, cout functions
#include <sstream>      //  string stream classes like ostringstream (output
string)
#include <fstream>     //  Read and write file system **Can be ignored as we
dont deal with any files
#include "exploringBB/chp11/thingSpeak/network/SocketClient.h"   // Internet
Protocol Family Header

int thingspeak (int count)
```

```
{
   ostringstream head,data;
   SocketClient sc ("api.thingspeak.com",80);    // object for API to the
ThingSpeak Website using HTTP port 80
   data << "field1="<<count<<endl;               // Data to be sent to the
cloud and which feild to store it
   sc.connectToServer();                         // Connect to the server
(request)
   // Use the HTTP methods to get the request sent to the cloud
   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";
   // Send Data
   sc.send (string(head.str()));
   sc.send (string(data.str()));
   sleep(15);
}


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);     // Call the cloud function created
      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 an alert for removal of the letters
```

```
      counter=0;
      printf("Counter Resetted # %d\n",counter);
      pin_low(8,11);
      system("/usr/bin/python2.7 pysend.py");
      thingspeak(counter);     // Call the function to connect and send data
to the cloud
    }

  }
// 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
}
```

To the cloud service we send is the value of the counter which would be sent to the cloud as seen in the code listing. Thus, one can see that we have created a separate function which allows us to get communicate with the cloud. Thus, when the sensor is activated it increments the data or gets the counter back to zero and send the same thing to the cloud service depending upon the loop that is running. The compilation of the code now takes place through "g++" compiler as now the code is written in C++ programming rather than C programming this is illustrated in Figure 5.15. The Figure 5.16 shows how the cloud displays the value of the counter.

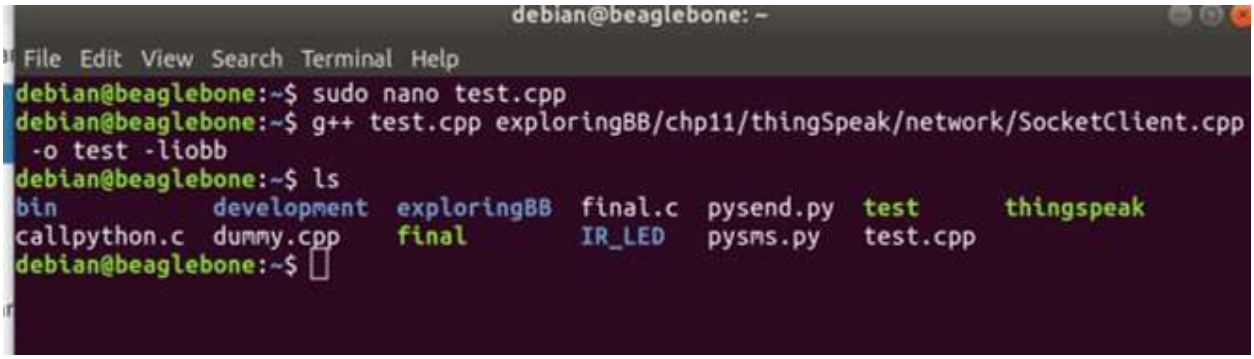


Figure 5.15 *Execution of the code to connect to cloud service*

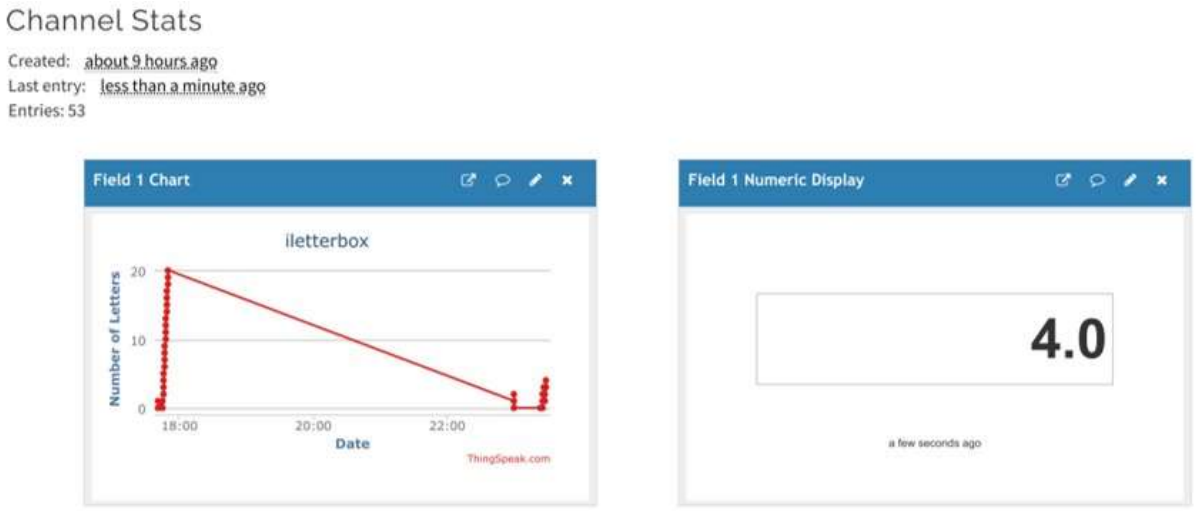


Figure 5.16 (a) *Executing the Code* (b) *Display on the Cloud*

The cloud service will get updated only every 15 seconds as we use a trial version of the software. Thus, due to the same reason we have a delay of 15 seconds set in the code this delay can be reduced if one purchases and pays for the cloud service.

- **Plug and Play the System**

Upon successful interfacing of all the hardware and software with the Beaglebone Black Wireless our last requirement was that we needed to make sure that the system runs without the help of any other devices like the laptop or anything. Thus, to do this we followed the following steps:

1. Compile the code and generate an executable file from that code which you want to execute when the system boots up
2. Create a bash script that will launch the code at boot/ start-up. To do so type in the commands as shown in the same order.
   ```
   cd /usr/bin
   nano scriptname.sh
   ```
3. Type the following into the nano script
   ```
   #!/bin/bash
   config-pin P9.24 uart
   config-pin P9.26 uart
   /home/Debian/output_file_name
   ```
   The output file name is the name of the file obtained in step 1. Remember to give the entire path where the executable file is present.
4. Save and grant execution permission
   ```
   chmod u+x /usr/bin/scriptname.sh
   ```

5. Create the service using

   ```
   nano /lib/system/scriptname.service
   ```

6. Edit the file as necessary. We added the following lines into the service

   ```
   [Unit]
   Description=description of code
   After=syslog.target network.target
   [Service]
   Type=simple
   ExecStart=/usr/bin/scriptname.sh
   [Install]
   WantedBy=multi-user.target
   ```

7. Create a symbolic link and get access to the location of the service.

   ```
   cd /etc/systemd/system/
   ln /lib/systemd/scriptname.service scriptname.service
   ```

8. Reboot the system and check if it works as desired.

   ```
   reboot
   ```

What and Why or Why not?

In the proposal we had mentioned some of the terms and now we will discuss some of the adopted keywords and explain what is the keyword and if we have used it in the project.

Keyword	Define Keyword	Why or Why not used the keyword?
GSM Module	GSM stands for Global System for Mobile Communication. This module is capable of acting like a mini mobile device and can perform all the functions which a normal cell phone would perform	In our project we use this module to send text message to the user to keep him informed about the activity that happens with letterbox
RTC	RTC stands for Real time clock. This is a clock which provides time and date which is in the real world. It is mainly used to get the network time. Its like the clock which we have in our cell phones	We have not used any real time clocks in the project as we have the cloud service which provides time and also the message when delivered on the phone has minimal delay.

IR Sensor	IR Sensor is an obstacle sensor mainly found in robots. It has a transmitter and receiver pair. The transmitter transmits the rays if any obstacle is present the rays get reflected from there and are sensed by the receiver thus, detecting that there is any presence of obstacle in the path.	We have used this sensor to sense and count the number of letters being deposited in the letterbox.
Arduino	Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online.	Arduino in the project has been used only for testing the sensors and other devices. This will confirm that the sensors are working fine and have no problem.
IoT	The internet of things, or IoT, is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. Thing in this matter could be anything a sensor, human or anything. The things connect and identify each other using IP addresses. Thus, one can control anything sitting from any corner of the globe.	In our project since we have a cloud service which is a part of the IoT architecture thus we use IoT technology to transfer the data to the cloud.
Beaglebone Black Wireless	Beaglebone Black Wireless is a modification of Beaglebone Black board. It replaces the 10/100 Ethernet port with onboard 802.11 b/g/n 2.4GHz Wi-Fi and Bluetooth, the popular open source BeagleBone Black computer. It is a 32-bit micro-computer which can be used as a standalone computer or as heart of the system. It is suitable for IoT based applications and runs Linux OS Debian Distribution for all its functionality. We can control this device over its local Wi-Fi network.	This is the main processor of the project. All the sensors and peripherals get connected to it and perform their respective tasks. It is a master device of the project.

LDR	LDR stands for Light Dependent Resistor. This is a light sensitive sensor. The resistance of the sensor varies with the intensity of light falling on the sensor. During the day light or bright light, it has a minimum resistance and has maximum resistance in the complete darkness or in the night.	Based on the working principle of the device we use this sensor in the resetting the counter. That is when the letter box in closed i.e., complete darkness the counter increments and while open it has some light thus helps in reset of the counter using some of the logical operation.
Twilio	Twilio, the cloud communications company, is reinventing telecom by merging the worlds of cloud computing, web services and telecommunications. Twilio provides a telephony infrastructure web service in the cloud, allowing web developers to integrate phone calls, text messages and IP voice communications into their web, mobile and traditional phone applications. It is a paid subscription but allows 25 messages to be sent for free per month	Since we use a GSM in our project, we do not require this third-party application for the project.
IFTTT	IFTTT derives its name from the programming conditional statement "if this, then that." What the company provides is a software platform that connects apps, devices and services from different developers in order to trigger one or more automations involving those apps, devices and services. From the name itself one can understand what it can do. It can be triggered by a large number of apps like Twitter, Gmail, Facebook, Phone Calls, etc. The app runs the trigger services through applets. These applets have pre-defined statements or they already know what to do if something happens. For example, if the applet is said that if "cake" is mentioned in the mail you receive then the applet should trigger something or do something.	Since we use a GSM in our project, we do not require this third-party application for the project.
GPIO Pins	GPIO stands for General Purpose Input Output. These pins help one to connect	For most of the sensors and LED in the project we have

	various general-purpose sensor. They are usually digital in nature. A general-purpose input/output (GPIO) is an uncommitted digital signal pin on an integrated circuit or electronic circuit board which may be used as an input or output, or both, and is controllable by the user at runtime. GPIOs have no predefined purpose and are unused by default. If used, the purpose and behavior of a GPIO is defined and implemented by the designer of higher assembly-level circuitry: the circuit board designer in the case of integrated circuit GPIOs, or system integrator in the case of board-level GPIOs.	used these pins as they are very easy to use and since we want to have a serial communication with the board.
IEEE 802.11	IEEE 802.11 refers to the set of standards that define communication for wireless LANs (wireless local area networks, or WLANs). The technology behind 802.11 is branded to consumers as Wi-Fi. As the name implies, IEEE 802.11 is overseen by the IEEE, specifically the IEEE LAN/MAN Standards Committee (IEEE 802).	This is the communication protocol which we use in the project to get communicated to the cloud service and the internet.
ThingSpeak	ThingSpeak is an open-source Internet of Things (IoT) application and API to store and retrieve data from things using the HTTP 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.	We have used this platform as the cloud service in our project. We send all the data about the number of letters in the box to the cloud so that other users who have no access to SMS service can look upon the cloud to check if there is any letter in the box.
TCP	TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation through which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.	Since it is related with the devices and with basic working principle of Wi-Fi working thus all the devices when want to connect to internet have TCP layer present thus in our project since we have BB-WI connecting to internet it sets

		its own way of exchanging the data via this protocol.
SSH	SSH or Secure Shell is a network communication protocol that enables two computers to communicate (c.f http or hypertext transfer protocol, which is the protocol used to transfer hypertext such as web pages) and share data. An inherent feature of SSH is that the communication between the two computers is encrypted meaning that it is suitable for use on insecure networks. SSH is often used to "login" and perform operations on remote computers but it may also be used for transferring data. SSH usually by default uses port 22 to establish such communication over TCP.	For the basic and initial setup for the Beaglebone to Wi-Fi and also if we wanted to code the board or use it via USB we used SSH. As to login into the Beaglebone we use the command ssh debian@192.168.7.2 thus it signifies that we use SSH to control the BB-WI remotely.
Pre-emptive Scheduler	Pre-emptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state. Algorithms that are backed by pre-emptive Scheduling are round-robin (RR), priority, SRTF (shortest remaining time first).	We externally have not used the pre-emptive scheduler but when the Beaglebone boots or performs any of its task it does it through the pre-emptive scheduler.
Semaphore	A semaphore is a variable that controls access to one or more resources. It is a tool-developers use to ensure functions only access valid data and run at the right time. A semaphore can prevent a deadlock or race condition by disallowing access to a resource when it is not available.	We have not used any semaphores in the project as we have no issues with resource allocation as leave it to the internal system of host processor to schedule our task and give access to resources
Mutex	A mutual exclusion object (mutex) is a program object that allows multiple program threads to share the same resource, such as file access, but not	We have not used any data structure of mutex and we

	simultaneously. When a program is started, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. The mutex is set to unlock when the data is no longer needed or the routine is finished.	have left this work for the internal functioning of BB-WI
Queues	Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.	We have not used any data structure of mutex and we have left this work for the internal functioning of BB-WI
Linux OS	Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply, the operating system manages the communication between your software and your hardware. Without the operating system (OS), the software wouldn't function. It is a open-source free to use OS with a large community contributing to it trying to make it more user friendly. The best point is it is secured as compared to Windows and it used by programmers and embedded engineers widely.	Our main processor runs on Linux Kernel thus we have to use and operate on Linux system. Thus, to program and control the BB-WI we require a basic knowledge of this OS. This OS comes by default installed on the BB-WI
Debian	Debian also known as Debian GNU/Linux, is a Linux distribution composed of free and open-source software, developed by the community-supported Debian Project. It is a kind of distribution under Linux like we have Ubuntu for laptops or RedHat, etc. Debian is one of the oldest	Our host processor BB-WI comes with pre-installed Debian thus we just update the image if we require. Debian is well suited for our application and project.

	operating systems based on the Linux kernel.	
GCC Compiler	The GNU Compiler Collection is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain and the standard compiler for most projects related to GNU and Linux, including the Linux kernel. GNU Compiler Collection (GCC): a compiler suite that supports many languages, such as C/C++ and Objective-C/C++. GNU Make: an automation tool for compiling and building applications. GNU Binutils: a suite of binary utility tools, including linker and assembler.	We require this compiler to compile our written codes and generate a suitable executable file in order to check if the code runs as per our logic. As all the codes were written by us on the BB-WI nano editor thus we used this pre-installed compiler.
Nano Editor	GNU nano is a text editor for Unix-like computing systems or operating environments using a command line interface. It emulates the Pico text editor, part of the Pine email client, and also provides additional functionality. Unlike Pico, nano is licensed under the GNU General Public License. It is like Notepad that we have in Window OS. Nano is included with many Linux distributions by default, but some users may need to install it through their distribution's package management tool.	To write the codes and program the BB-WI we had to use some editor hence we used this editor to write our codes.
MQTT	Message Queuing Telemetry Transport is a message protocol for restricted networks (low bandwidth) and IoT devices with extremely high latency. Because Message Queuing Telemetry Transport specializes in low bandwidth, high latency environments, it is an ideal protocol for machine-to-machine (M2M) communication. MQTT is used in IoT and IIoT up to the connection of cloud environments. It stands for MQ Telemetry Transport. It is an extremely simple and lightweight messaging protocol (subscribe and	This protocol may be used for cloud communication but there are other alternatives available thus we haven't used it in our project

	publish) designed for limited devices and networks with high latency, low bandwidth or unreliable networks.	
HTTP	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. 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. 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.	Since we didn't use MQTT to get our data to the cloud thus we use HTTP to get the data to the cloud as it was much easier to use and program.
API	API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.	In our project we have used functions from the POSIX API (usleep and sleep) in order to get the necessary delay by suspending all the process currently on-going on the processor. It provides a clean delay without wasting memory.
MathWorks	MathWorks is an American privately held corporation that specializes in mathematical computing software. Its major products include MATLAB and Simulink, which support data analysis and simulation. In the coding community, MathWorks hosts MATLAB Central, an online exchange where users ask and answer questions and share code. MATLAB Central currently houses around than 145,000 questions in its MATLAB Answers database while actively supporting numerous academic institutions to advance STEM education.	MathWorks is a company thus we don't use the company but we use one of its indirect products the cloud service of the project ThingSpeak which will help us in achieving in one of features promised in proposal communication with cloud.

EasyEDA	EasyEDA is a free and easy to use circuit design, circuit simulator and pcb design that runs in your web browser. EDA in the word stand Electronic Design Automation. EasyEDA allows the creation and editing of schematic diagrams, SPICE simulation of mixed analogue and digital circuits and the creation and editing of printed circuit board layouts and, optionally, the manufacture of printed circuit boards.	Being an open-source software and easy to use we use it to draw schematic and PCB Design for our project as when tried using other software we were not quite comfortable with them and also this software has large user community thus finding help would be easy.
C/C++	C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs. C++ is portable and can be used to develop applications that can be adapted to multiple platforms. C++ is fun and easy to learn! It is an extension of C programming language. C is a general-purpose programming language that is extremely popular, simple and flexible. It is machine-independent, structured programming language which is used extensively in various applications.C was the basic language to write everything from operating systems (Windows and many others) to complex programs like the Oracle database, Git, Python interpreter and more. It is said that 'C' is a god's programming language. Both these languages are widely used in embedded industry.	As promised during the proposal and also the set instructions said that one must use C/C++ programming language as the main language thus we use this programming language
MATLAB	MATLAB is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.	Since we do not perform any kind of analytical or numerical data operation thus, we don't require to use MATLAB in our project. Also, we don't use much mathematical concepts thus we don't require this software or tool.

Chapter VI
Evaluation

Introduction

This chapter discuss about success of solution meeting the user requirement. It will also evaluate testing method used during development. Finally, chapter will go on discussing methods of solving discrepancies or issues related Beaglebone Black Wireless, IR Sensor, LDR, GSM Module and Cloud Service (ThingSpeak).

Minimum Requirements

Minimum product output will be a Linux running letter box which can sense and count the letters arriving into the box and also reset the count when needed by the user. Along with this inform the user about the activity of the arrival and removal of the letters to and from the box. While updating the user about the activity via GSM based SMS the system must be capable to get the data updated to the cloud service. As the last requirement the system must be running without the help of any laptop or other devices.

Troubleshooting

This section of the chapter will tell you about the various we faced during the entire course of the project. The issues and their solutions are as follows:

1. Getting the hardware ready was a problem as some of the GSM may or may not be compatible with the Beaglebone and since this was the first time, we were using GSM thus we had some confusion. To resolve this issue, we had to search many issues and thus after looking links we found that either SIM900 or SIM800 would be compatible thus since SIM900 would be delivered first thus we used SIM900
2. EasyEDA software is quite good enough to have almost all of the components present but some of the components like the BB-WI were not present thus with help of the Beaglebone Black we edited the components as per our requirement. For example, we removed the Ethernet port from the Beaglebone Black as this is not present on the Beaglebone Black Wireless (BB-WI). Also, the Beaglebone Black is not available directly thus we imported the same using the library as we searched for Beaglebone and we got Beaglebone Black cape in the search result which we further edited as required for the project. The remaining

components we are yet to find else we may have to edit some other components which may suit our project.

3. At the time when one had to download the required GPIO libraries on the Beaglebone itself as we were using nano and GCC complier thus we wanted the libraries to be present on the BB-WI and for the same reason we required the internet connectivity. Thus, we made several attempts to connect it but later realised that since the router was far away from the BB-WI we had to go a little close in distance and hence it connected perfectly. When one wishes to run the code, he/she shall must be logged in as a root user (super user) . The reason being since we were trying to get the access some files which require administrative permission i.e., we were trying to change or set the functionality of the header pins which can be done only from the system files and thus we required full permission for it and this can be obtained only by being the admin/ root user.
4. While coding the GSM module we couldn't find a suitable C library thus we had to deviate to python programming which was fast and easy to use and code. But we had to install appropriate libraries and set the baud rates which was not known thus later found that baud rate for modern phone was 115200 and other phones with 3G and modem work at baud rate of 9600.
5. During the execution of the code, we had the IR sensor loop running twice thus when a single letter was given in the box, we had a count of 2 not 1 which was not good for the project. Thus, we decided to re-wire complete project and this was completely resolved. So, we assume that we had some wiring issues here.
6. For the GSM Module since we always must initialize the pins of the UART thus to make our task simple easy and relax free we took the help of the uEnv.txt file where we inserted the line "capemgr.enable_partno=BB-UART4" and reboot the BB-WI.
7. 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. Thus, we would conclude here that the antenna of the Beaglebone Wireless is weak and very sensitive.
8. Also, in the case of ThingSpeak, we use a header file called sstream to establish the http request. But it is not available for C language. So, we were compelled to change our system to the C++ language.
9. For the cloud service since we were replicating the code from the book “Exploring Beaglebone” by Derrek Molloy we when typed the code by modifying as per our requirement we could get the successful compilation and thus we later on realised that we need to have his developed socketclient library loaded in the Beaglebone and then specify the path where this library is present and hence, we were able to generate the executable file.

10. The last issue that we faced was that we were seeing that the data was getting up to the cloud but was unable to get it display thus, later on we realised that we had few spelling mistakes because of which we couldn't see the data. Once when the spelling was verified, we could see the data on the cloud.

Chapter VII
Conclusion

The aim of this project was to develop a product to meet the need of smart and intelligent letter box. The development focused on saving time and reminding people about the letters that arrive in their letterbox. This letterbox was aimed to provide the data it had regarding the letters to cloud service making it an IoT based feature which could be added in smart home kind of systems. Establishing a connection between the IoT physical system and the web platform was inevitable for accomplishing the objective.

Many designs were proposed all having different logics and different ideas using Arduino or Raspberry Pi. This system was made combining all those ideas with help of smart and fast host processor the Beaglebone Black Wireless. This was just a start to make a letterbox smarter more features to this project can be added or modified to suit individual requirements. For now, when the current requirements are met the user can see the status of the number of letters present in the letter box on the cloud service of ThingSpeak platform and also the user receive a text alert regarding the arrival and removal of the letters from the letter box.

Future Work

The project has been completed meeting all the objectives, though the future scope of this cannot be ignored. Modifications or further updates can be done on the project. Some of it is listing below:

1. The letterbox lacks the feature of identification which can be improved or added in the project using a camera and using some of the image processing concepts in OpenCV this feature will help the user identify the sender address and intimate the user about the same using GSM or via the existing cloud service being used in the project.
2. Using the same concept of computer vision/ OpenCV one can maintain the that is if the user removes one letter and keeps the remaining two letters in the box then what would happen. Which could be resolved using either a camera or ultrasonic sensor.
3. During the project implementation, we found that the design of the project matters thus, one can use this project and hence make a universal design that is if one can the same concepts that we used but for a universal design.
4. During the course of the project, we also didn't monitor the feature of security and thus we can think about that as well by capturing the photo of the delivery person and hence uploading those images to the cloud thus making sure about the security of any suspicious letter is delivered in the box.

5. One could also take care of the task where if the mailman may deposit a wrong letter in the box then what would happen if he wants to remove the box like how will count of the letter be affected.
6. Lastly, one can also think about a solution to monitor the situation when the same two letters are deposited in the box.

These are just a few scopes which may improve this project. More could be found as one progress and adds new features simultaneously. With complete dedication and time one can make a beautiful and efficient letter box.

Chapter VIII
User's Guide

About the Product

The product is a smart letterbox that can be a great addition to your smart home. The system will provide you an update with the number of letters being deposited in your letterbox also it will update when the letters are removed from the letterbox. The information about the letterbox will be communicated to the user via SMS Service and also through ThingSpeak Cloud Service.

Powering Up the System

To power up the system you will require 2 power adapters with rating of 5V, 2A. One adapter would be able to power up the Beaglebone Black Wireless and other will be used to power up the GSM Module. When the adapter is connected to Beaglebone Black Wireless is connected to the power supply and switched on the Power LED will be ON and after some time the top LEDs must start blinking in a repeated heartbeat pattern.

To make the GSM module connect to the suitable cellular network. Insert a SIM card at the bottom of the module and switch on the supply. When the supply is on the power LED turns on. Next, next to the power LED press the power key for approximately 2 seconds this will connect the system to the desired network. You can verify this from network and status LED that is when connected properly the network LED will continuously blink whereas the status LED will blink every 3 seconds.

Connecting Wi-Fi

When using the product for the first time you need to connect it to your preferred network. Connecting to available Wi-Fi will need some commands in terminal. To connect to Wi-Fi for the first time you have the following options:

1. Using Terminal in Linux System
2. If you have Windows OS the you can install PuTTY software and use it connect to Wi-Fi
3. Use the local Beaglebone Wi-Fi network and configure it to connect it to appropriate Wi-Fi

For first setup with Wi-Fi connect the main device (Beaglebone [Black colored device]) with micro-USB cable. Following is step by step procedure for Wi-Fi connection:

A. If you have Linux System
   1. Open terminal shell by pressing **Ctrl+Alt+T**.
   2. Type in the command **sudo ssh debian@192.168.7.2**
   3. Use the following login credentials: **Password: temppwd**
   4. Run Command **connmanctl** which will open an interactive mode shell.
   5. Run following commands in line:
      a. **enable wifi**
      b. **scan wifi**
      scan completed for wifi
      c. **services**
      Rogers123 wifi_0126874295ca_84135186135163516 8135_managed_psk
      Bell123 wifi_0142698595ca_321685438452369765236_managed_psk
      d. **agent on**
      e. **connect** wifi_0142698595ca_321685438452369765236_managed_psk
      Paraphrase? **Your_password_here**
      f. **quit**

B. For Windows system
   1. Install PuTTY software and launch the software.
   2. In the place for Host Name (IP address) enter 192.168.7.2 and click on open
   3. When prompted for Login Creditials enter the following
      a. Login As: debian
      b. Password: temppwd
   4. Run Command **connmanctl** which will open an interactive mode shell.
   5. Run following commands in line:
      a. **enable wifi**
      b. **scan wifi**
      scan completed for wifi
      c. **services**
      Rogers123 wifi_0126874295ca_84135186135163516 8135_managed_psk
      Bell123 wifi_0142698595ca_321685438452369765236_managed_psk
      d. **agent on**
      e. **connect** wifi_0142698595ca_321685438452369765236_managed_psk
      Paraphrase? **Your_password_here**
      f. **quit**

If you do not have a mico-USB cable not to worry you can connect to Wi-Fi using the following steps:

1. On any device like your laptop or mobile phone open your Wi-Fi settings and connect to local Beaglebone network which would be displayed in available networks as **BeagleBone-**

XXXX. When connecting it will ask for password enter **BeagleBone** which is your default password for that network.

2. Now open the Web Browser on that system and type in the following IP address 192.168.8.1 in finding the URL bar (its like how you enter the website name on the browser)
3. If the connection was successful Cloud 9 IDE will open. Now it will open the terminal of your device as shown in Figure 8.1

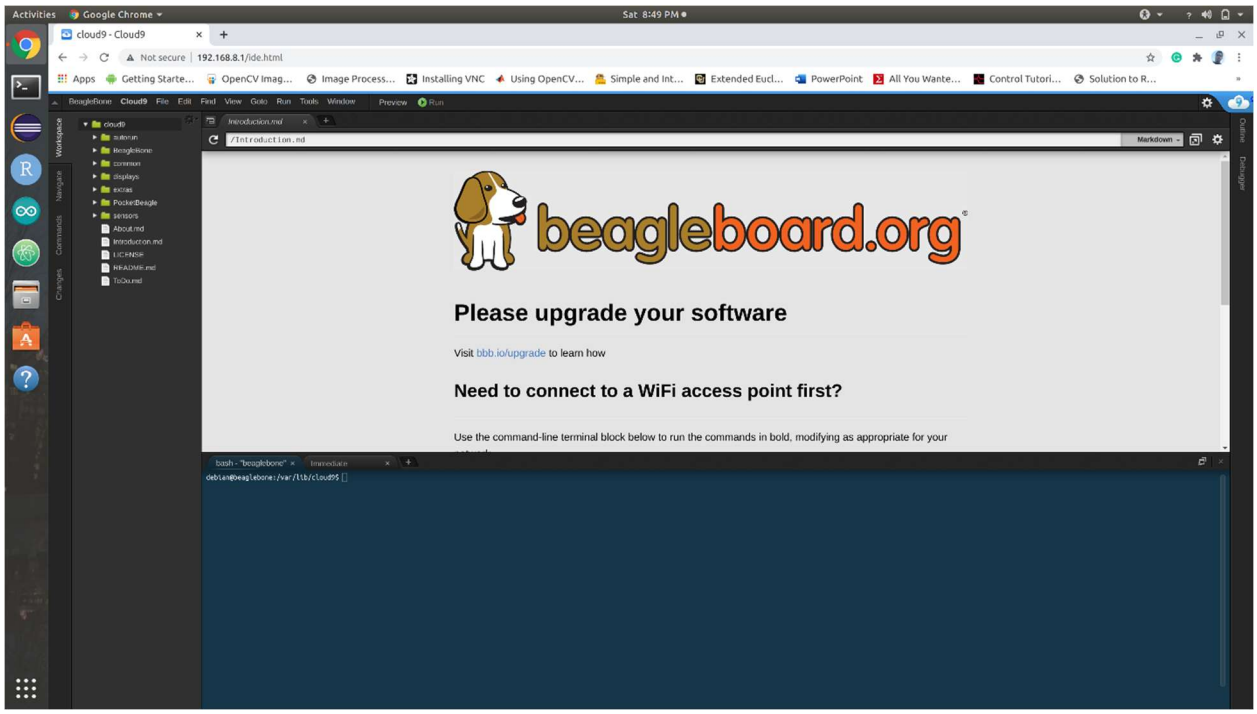


Figure 8.1 *Cloud9 IDE*

4. At the bottom of the screen (Aqua Colored) type in the following commands:
    a. Run Command **connmanctl** which will open an interactive mode shell.
    b. Run following commands in line:
        i. **enable wifi**
        ii. **scan wifi**
        scan completed for wifi
        iii. **services**
        Rogers123 wifi_0126874295ca_84135186135163516 8135_managed_psk
        Bell123 wifi_0142698595ca_321685438452369765236_managed_psk
        iv. **agent on**
        v. **connect** wifi_0142698595ca_321685438452369765236_managed_psk
        Paraphrase? **Your_password_here**
        vi. **quit**

Once connected to suitable Wi-Fi network for the first-time power off the Beaglebone (Black Colour) Device and power it up again.

Connecting to the web platform

- The link to the ThingSpeak account is: https://thingspeak.com/channels/1238879
- This link can be shared among the drivers to get the live updates of parking lot availability and to view the status.
- The features of the website are:
  - Can view the live status from anywhere.
  - Status is available in graph format as well

Location of the lot can be obtained from the channel.

Chapter IX
References

[1] (n.d.). Retrieved November 09, 2020, from https://www.tutorialspoint.com/system-function-in-c-cplusplus

[2] 14. Introduction to AT Commands. (n.d.). Retrieved November 08, 2020, from https://www.developershome.com/sms/atCommandsIntro.asp

[3] An Easier and Powerful Online PCB Design Tool. (n.d.). Retrieved October 26, 2020, from https://docs.easyeda.com/en/Introduction/Introduction-to-EasyEDA/index.html

[4] Cocker, B. (2018, October 31). Light Dependent Resistors (LDR) - Working, Construction, Symbol, Applications. Retrieved November 16, 2020, from https://www.circuitstoday.com/ldr-light-dependent-resistors

[5] Cooper, J. (n.d.). Setting up IO Python Library on BeagleBone Black. Retrieved November 09, 2020, from https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/uart

[6] Devi Pujari, A., Bansode, P., Girme, P., Mohite, H., & Pande, A. (2016). Smart Letter Box System Using Obstacle Sensor For Notifies The User By Android Application. International Research Journal of Engineering and Technology (IRJET), 3(10), 823-825.

[7] Electronic Letter Box: Embedded Systems Project Topics. (n.d.). Retrieved September 26, 2020, from https://www.seminarsonly.com/Engineering-Projects/Embedded/Electronic-Letter-Box.php

[8] Electronic Letter Box Project Circuit and its Working. (2018, May 26). Retrieved September 26, 2020, from https://www.electronicshub.org/electronic-letter-box-project-circuit/

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

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

[11] 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/

[12] Khan, S. (2015, September 23). Intelligent Letter Box using Arduino and GSM. Retrieved September 26, 2020, from https://www.engineersgarage.com/contributions/intelligent-letter-box-using-arduino-and-gsm/

[13] Kumar, A. (2019, January 31). IR-Based Counter System with IoT. Retrieved November 01, 2020, from https://www.hackster.io/kashwani893/ir-based-counter-system-with-iot-6c69d0

[14] LDR Sensor Module Interface With Arduino. (2017, September 27). Retrieved November 16, 2020, from https://www.instructables.com/LDR-Sensor-Module-Users-Manual-V10/

[15] Light-emitting diodes Circuit, Working Principle and Application. (2018, May 16). Retrieved November 02, 2020, from https://www.elprocus.com/light-emitting-diode-led-working-application/

[16] Mathworks. (n.d.). BeagleBone Black Pin Map. Retrieved October 26, 2020, from https://in.mathworks.com/help/supportpkg/beagleboneio/ug/beaglebone-black-pin-map.html

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

[18] PCB Concepts and Materials. (2017, September 21). Retrieved November 29, 2020, from https://www.instructables.com/PCB-Concepts-and-Materials/

[19] PCB Design Fundamentals: Overview. (n.d.). Retrieved November 29, 2020, from https://www.ni.com/tutorial/10580/en/

[20] PCB Designing Using EasyEDA. (2017, September 18). Retrieved November 30, 2020, from https://www.instructables.com/id/PCB-Designing-Using-EasyEDA/

[21] Raithatha, D. (2017, October 02). Smart Letter Box. Retrieved September 26, 2020, from https://www.instructables.com/id/Smart-Letter-Box/

[22] S. (2020, February 13). BeagleBone Black (BBB) and PocketBeagle I/O (GPIO), SPI and I2C Library for C – 2019 Edition. Retrieved November 02, 2020, from https://www.element14.com/community/community/designcenter/single-board-computers/next-genbeaglebone/blog/2019/08/15/beaglebone-black-bbb-io-gpio-spi-and-i2c-library-for-c-2019-edition

[23] Shetty, R. (Producer). (2013, November 27). Beagle bone black GSM modem interface part 2 [Video file]. Retrieved October 20, 2020, from https://www.youtube.com/watch?v=814CbzFG6JE

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

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

[26] Usleep(3) — Linux manual page. (2017, September 15). Retrieved November 01, 2020, from https://man7.org/linux/man-pages/man3/usleep.3.html