A Project Report

on

IOT BASED BANK LOCKER SECURITY SYSTEM

For the award of

ONTARIO GRADUATE CERTIFICATE

Submitted by

Kiranpreet Kaur Gill

Simran

Gurpreet Singh

Amandeep Singh

Jasmeet Singh

Under the guidance of

Dr. Mike Aleshams

ESE-4009 Embedded System Design Project

(January 2021 – April 2021)

Abstract

Bank locker plays an essential role, lockers are basically used to store the personal belongings and other stuff for security purposes. Basically, the lockers are operated by the key. As such a key is provided to locker owner. So, the other persons in spite of the main owner can use the locker too. So, many projects are made to enhance the security of the bank locker using technology. We also tried to enhance an existing project using all the skills and knowledge that we have learnt till now.

The aim of our project is to update an existing project, and to enhance the security of the locker 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 bank locker 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.

Acknowledgement

"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 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.

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.

~Kiranpreet Kaur Gill

~Simran

~Gurpreet Singh

~Amandeep Singh

~Jasmeet Singh

Table of Content

List of Figures

Chapter - 1

Introduction

Overview

This project is about the development of an existing bank locker system. As the locker operators can open their personal locker with the help of key and they can use the locker to store their personal belongings and stuff. Now, we are just making some amendments in this existing project, to make the bank locker system more secure. As such, with the help of our project, user can open their locker with the help of touch screen and fingerprint sensors that makes it more secure and reliable.

This smart bank locker security system has come with GSM, WI-FI capabilities. As firstly the user needs to enter the required password on the touchscreen to access the fingerprint sensor. Now, for this system we are having two users, one is the main owner and the second one is the co-owner. Now, if the password enter by the user is correct then the fingerprint sensor is enables and user can scan their fingerprint to open the locker. Similarly, the process repeats if the co-owner is trying to open the locker, but there is only one difference that is after the fingerprints matched perfectly, GSM will be activated, and SMS will be sent the main owner knows that the other user is using the locker.

Moreover, for the security purposes, we are giving the 3 attempts to the fingerprint sensor like user can use the fingerprint sensor 3 times. Now, if the user enters the wrong message on the touchscreen, immediately the buzzer will activate, and the camera will take picture and send it to the cloud. Though, we are using the vibrator sensor in the project, for security purposes. If anyone tries to smash the locker with hammer or something the vibrator sensor with activate and buzzer will trigger and again will again take the picture and send it to cloud. The whole locker system operates on the LINUX KERNEL, and cost ranges from $450-$500.

Problem Statement

Bank locker are very helpful for the users. As they can use the personal locker to save their personal belongings and stuff. The current system uses some microcontroller that help the user to operate the locker but it is lacking in some of important features like speed, WI-FI, cloud-services, security, and reliability. And the Owner did not get the information about who operate the locker at which time, because the system lacks in cloud services.

Goals and Objectives

Goal: Develop a smart bank locker security system with GSM and Cloud based technology running with Linux Kernel by the end of Winter 2021 academic term.

Objectives:

- ✓ Connecting a main micro-controller Beagle bone Black with the cloud and GSM to determines the required output.
- ✓ Interface various sensors using in the project which will help us getting know when the user is using the bank locker system or not.
- ✓ Using the WIFI facility to update the cloud every time with the user is using the bank locker system or not.
- ✓ Camera is used in the project to determine the unknow user to the bank locker.

Scope of Project

Deliverables:

- ✓ A smart bank locker security system to enhance the security of the personal locker of the user.
- ✓ Use of cloud services in the project to make the bank locker more secure.
- ✓ Using the GSM and camera facilities to send message if someone else is using the locker and the camera will take the picture of the unknown person.
- ✓ Using the online software for the PCB design and implement it on the ZERO PCB because of COVID conditions, the printed PCB taking a long time to deliver.
- ✓ Eventually, plug and play the project, which means no laptop is used in the final day.

Milestones:

TASK	START DATE	END DATE	PERSON INCHARGE
PROJECT PROPOSAL	18 Jan	12 Feb	
FINALISING AND ORDERING THE HARDWARE COMPONENTS	12 Feb	17 Feb	GURPREET
TESTING THE HARDWARE COMPONENTS(PART-1)	17 Feb	19 Feb	AMAN

TESTING THE HARDWARE COMPONENTS(PART-2)	19 Feb	22 Feb	JASMEET
DESIGNING SCHEMATIC	22 Feb	26 Feb	SIMRAN
INTERFACING BUZZER BEAGLEBONE	26 Feb	1 March	KIRAN
INTERFACING SERVO MOTOR WITH BEAGLEBONE	1 March	3 March	GURPREET
INTERFACING FINGERPRINT SENSOR WITH ARDUINO MEGA	3 March	5 March	SIMRAN
INTERFACING VIBRATION SENSOR	5 March	12 March	JASMEET
INTERFACING ARDUINO MEGA TO BEAGLEBONE BLACK	12 March	15 March	AMAN
INTERFACING GSM TO ARDUINO MEGA	15 March	17 March	KIRAN
INTERFACING ESP8266 WITH BBB	17 March	22 March	GURPREET
CONNECTING ESP8266 WITH ESP32 CAM, ESP32-CAM WIFI CONNECTIVITY	22 March	29 March	AMAN
ESP32 CAM PICTURE CLICKING AND STORING SENDING IMAGES FROM CAMERA BY EMAIL	29 March	31 March	JASMEET
INTERFACING TOUCH SCREEN WITH ESP8266	31 March	2 April	SIMRAN
INTERFACING REAL TIME CLOCK WITH BEAGLEBONE	2 April	5 April	KIRAN
DESIGNING PCB ONLINE	5 April	7 April	GURPREET

IMPLEMENTATION OF ZERO PCB(PART-1)	7 April	9 April	AMAN
IMPLEMENTATION OF ZERO PCB(PART-2)	9 April	12 April	JASMEET
GUI FOR THE TOUCHSCREEN FOR INPUT	12 April	15 April	SIMRAN
POWER MANAGEMENT	15 April	19 April	KIRAN
REPORT	19 April	23April	
FINAL DEMONSTRATION	23 April	28 April	

Limitations:

- ✓ Lots of Hardware Components, that makes the circuit complex.
- ✓ For better security of the locker, we can also add face recognition sensors.
- ✓ cloud-services for storing database will make it more perfect and reliable.

Outcomes and Benefits

- ✓ The system is powered up using a power adapter thus no need to worry about power unless there is a power cut in the bank.
- ✓ Cloud service used in the project is GMAIL, so the user can login from anywhere and check the updates.
- ✓ Updating to an existing project, as an existing project lacks in cloud service and speed. All these things are fulfilled in the purposed project.
- ✓ The basic procedure to open the locker is using the key, here we are making it completely digital using the present technology and procedures.
- ✓ The system helps the user to keep their locker secure by using cloud services and the camera. The SMS services that are sent via GSM.

Facilities and Resources

Laboratory:

- ✓ Embedded Systems Lab which we created as the classes were online thus a miniature lab was created at our place.
- ✓ Multimeter and Linux installed PC's present in our miniature home-made lab.

Intellectual Resources:

- ✓ Data Sheet of various sensors, GSM module, Beagle bone Black
- ✓ AT commands Catalogue
- ✓ Forums of Beagle bone and Element14 Communities
- ✓ Exploring Beagle bone Tools and Techniques for Building with Embedded Linux (Second Edition) from Derek Molloy.

Procedure and Methodology:

1) Getting the hardware requirements ready and ordered for the project sideways testing the components and understanding their working as they keep on arriving.
2) Install and update the devices such as the Beagle bone with the latest image OS to get to use some of the latest features available in terms of device trees and drivers.
3) Upon the getting the testing results positive and approved move and design the circuit schematic using software like EasyEDA, Fritzing, KiCad or Eagle, etc.
4) Installation of various libraries which will make our task of controlling the pins of Beagle bone easy and communication between peripherals and sensor simpler.
5) 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 of small components like buzzer with beagle bone.
6) Next, we start to interface all the hardware components with the required devices as per mentioned in the schematic.
7) Getting the user information intimation devices interfaced with the host processor i.e., Beagle bone.
8) To make the data available on the cloud service we need to establish a wireless connection so that Beagle bone can get connected to the cloud and send the appropriate data to the cloud and make it visible to the user.
9) At the end of all the interfacing we will integrate them and check if we get the desired functionality as mentioned in proposal.
10) 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 3).
11) Solder the devices on the PCB and test for the final demonstration of the project.

12) 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 - 2

Literature Review

(May,2020) J. Thirumalai, Gokul. R, Ganasekaran. P, Manellore Murali. M, Jackson Jublience Joseph. L Assistant Professor, Department of Electronic and Communication Engineering, Students, Department of Electronics and Communication Engineering, TJS Engineering College, Peruvoyal, Chennai - 601 206, Tamil Nadu, India conducted a research on an IOT (Internet of things) Bank locker security system which was published by International Journal of Engineering Research & Technology (IJERT).

The basic idea of the project is to secure the bank locker using the Fingerprint sensor. The main objective of the project is to effectively manage and control the bank locker by using a fingerprint sensor. It uses an Automated Safety Vault with Double Layered Defence Mechanism. This project was composed of an Electronic Lock that used fingerprint and password verification tools for scanning and sensing. Two layers helps to avoid any unauthorized entrance to the Vault. Biometric recognition is highly protected as one person's fingerprint never matches the other's as it usually includes fingerprints, ears, iris, recognition of the voice, signature, and hand geometry and confirmation. In this controller to analyse the sensor data and to generate desired controlled variables to ensure the security of the vault with the biometric. Pattern recognition is used as it ensures the secondary layer safety to the secured vault after successful adherence to the primary layer of password protection.

Figure 2.0: Fingerprint based Bank Locker Security System

The key objective of the existing project shown in figure 2.1 was to obtain the result of collective decision making based on the data provided by multiple sensors. To attain this performance, a controller is employed to analyze the sensor data and to generate desired controlled variables to ensure the security of the vault (J. Thirumalai, May 2020).

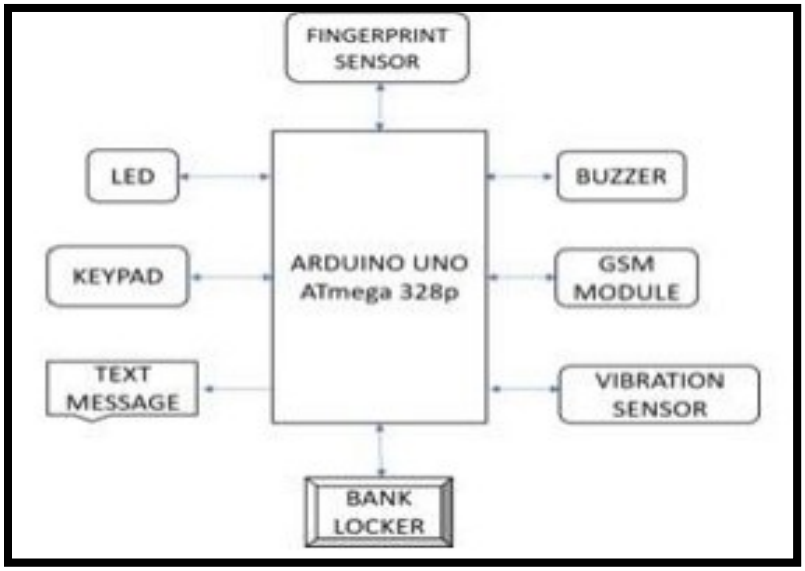

Figure 2.1: Existing block diagram of the project

The design and implementation of a prototype of an automated vault door locking device, which provides a double layer of security, was discussed in that research paper. It makes certain that by locking the vault door with a numeric password and biometric authentication, the proper user of the vault can be identified. It uses multiple sensors to track the state of the vault's activity from both the inside and the outside.

Chapter - 3

Requirements / Analysis

Overview

The IOT based Bank Locker Security System project is used to improvise the Bank locker security system. It uses various peripherals and sensors for this purpose. This system uses an automated Safety vault with layered defense mechanism. It uses passcode, fingerprint, and camera to manage and control the locker effectively.

In the project, we will use ESP 32 Camera module to upgrade the existing the project. As whenever an unauthorized person tries to open the locker then the camera will capture the picture and send it to the concerned person via mail and updating the user via an SMS too.

Initially, the Fingerprint Sensor will be deactivated. Once the use enters the password in the Touchscreen, the Fingerprint module will get activated. If the user then gives correct fingerprint, then the door will open which will be indicated by Servo motor. Although if wrong password is entered, or any vibration or loud sound is sensed, the beagle bone will send an alert SMS to the concerned person and also photos will be send via mail to the user. Overall, the system is quite reliable because if there's any issue in Internet connection, the concerned person will still be getting SMS alert.

Block Diagram

The figure below Figure 3.0 shows the basic layout of our project or in other words the circuit connection in the raw form which will help us build the IOT based Bank Locker Security System. 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 ESP32 CAM as the cloud service interface. From the Beaglebone black wireless we intend to send the picture to the Gmail. As the cellular network we choose "Chatr" service which will help us sending messages to the user defined numbers.

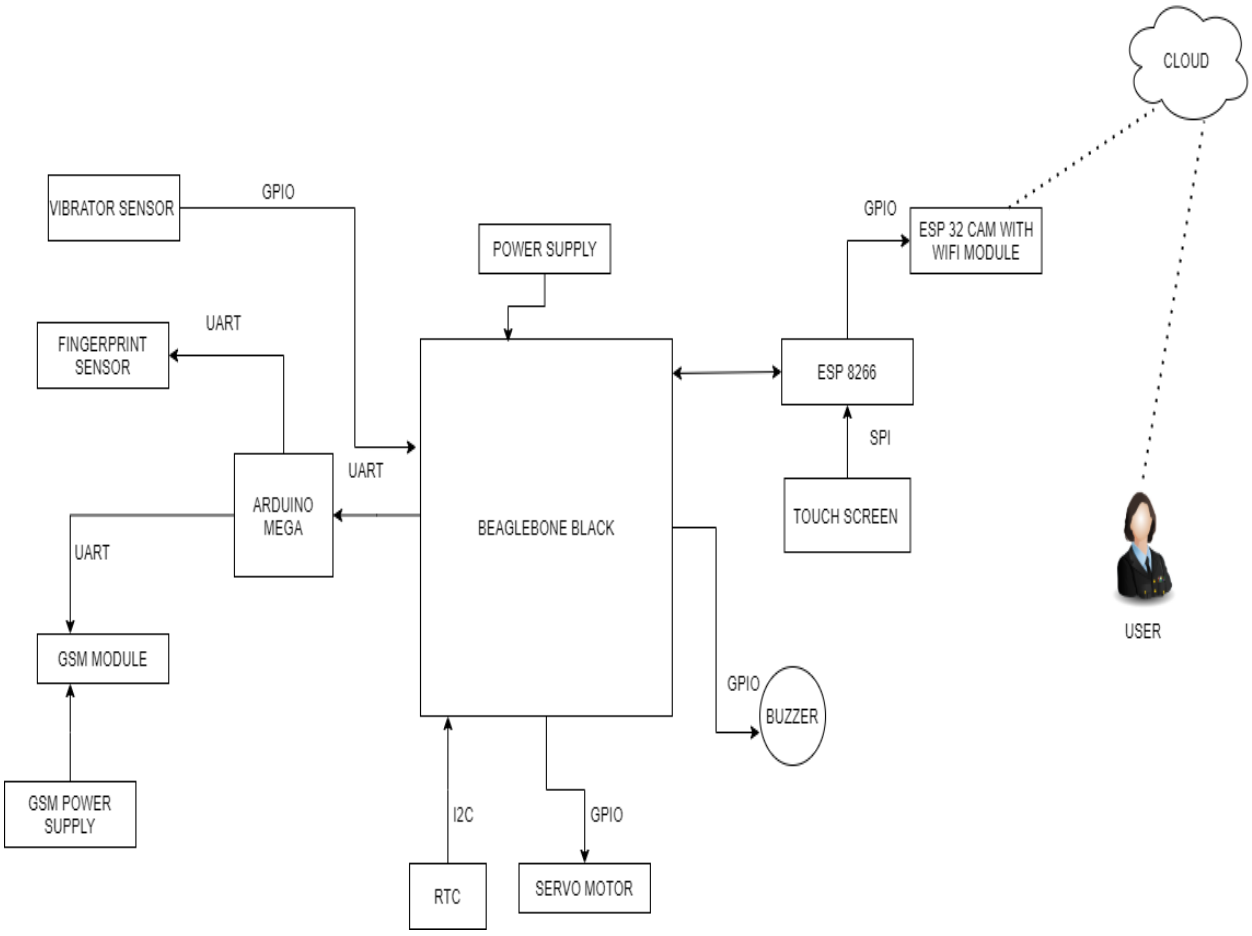

Figure 3.0: Block Diagram of proposed Project

Hardware

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

Thus, we require following list to make a complete IOT based Bank Locker Security System:

- Beagle Bone Black, microcontroller, work as master device for project.
- Vibration sensor: Vibration sensor measure the vibration around. The aim of vibration sensor in our system is to detect any impact or force which may be applied to force open the locker.
- ESP 8266: ESP8266 is 32-bit microcontroller that works as a slave module in our project to use for interfacing touchscreen.

- ESP32 Cam: The ESP32CAM is a tiny module based on ESP32 chip and OV2640. We are using it for capturing picture and sending it to mail via internet.
- Touch Screen: The touch screen will help to enter the password as a first layer security. And, to show useful information to the user.
- Piezo Buzzer: Buzzer is used as an audio output/ feedback for our system. The buzzer will start if anything abnormal like vibration, multiple wrong passwords etc. is detected
- Servo motors: Servo motor in our system is used as the lock/ unlock mechanism.
- Fingerprint sensor: For providing Biometric Authentication we are using Fingerprint sensor.
- RTC: RTC (Real-Time-Clock) is being used to keep track of Real-World Time.
- Camera module: Camera module will be used to take pictures when needed and the same will be sent to mail.
- GSM Module: GSM module is used for sending SMS to concerned person about various information.
- Arduino mega: Arduino Mega is being used an intermediate between the Beagle bone and GSM Module to increase the reliability of the system
- 5V Power adapters are used, one is used to supply power to the project and other is used to power up GSM module.
- 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.
- USB cable
- Connecting Wires

- Jumper Wires
- Zero PCB

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:

- Coding: Mainly preferable languages in our project is C, C++. Although Python Programming may be used in some cases where the feature is limited by C language.
    - MISRA - MISRA provides coding standards for developing safety-critical systems. MISRA C is the most widely used set of coding guidelines for C around the world. There have been three releases of the MISRA C standard -
        - MISRA C:1998
        - MISRA C:2004
        - MISRA C:2012
    - CERT - CERT is a secure coding standard. It's developed by the CERT division of the Software Engineering Institute at Carnegie Mellon University. This secure coding standard is available for C and C++.
- PCB designing PCB Designing will be done on EasyEDA Software.
    - 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.
- Assembler: GNU Assembler will be used which is default assembler for GCC Compiler.
    - 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.

- GCC is usually, pre-installed on the Linux distribution. It would be used to compile the code and generate the output file.

- Linux OS: Beagle bone Black support LINUX operating system. And we are using Linux to flash the latest image of Debian as per the requirement of the project. Usually, Beagle bone devices come with pre-installed Debian images, but we have to update it to meet our requirement.
  - 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++
- GNU NANO: It is a small and easy to use friendly text editor. This text editor totally supports the Beagle bone black. We can easily write the program and save it using a specific extension. For example, if we are making a txt file, we can save it with .txt extension. And lastly, we can make it executable using GCC command.
- Gmail: Gmail is very well-known service by Google. It can used to share message and media both.
  - The IMAP abbreviation stands for Internet Message Access Protocol and is one of the two most popular protocols for receiving email messages from the Internet. SMTP is an abbreviation that stands for Simple Mail Transfer Protocol. This protocol allows applications to transmit email messages over the Internet.
- SSH will be used for execution of commands on Beagle bone via Laptop while development.
- Arduino IDE is a coding software that makes the programming world more accessible to beginners with its simple interface and community-driven system.
- Eclipse IDE is used for executing the commands that is used in our project as it uses C/C++.

Communication Protocol

There are various communication protocols that are used for the interfacing of the components that are explained below:

- RTC Module – I2C protocol is used for interfacing the RTC as the module supports I2C protocol. Advantages of I2C – maintains low pin/signal count even with many devices on the bus, adapts to the needs of different slave devices, readily supports multiple master devices, and incorporates ACK/NACK functionality for improved error handling.
- ESP8266 – We have multiple protocol options as ESP8266 supports many. However, we are using UART as it has ease of communication. It requires only 2 wires for full duplex communication, and it does not require any address for data. Advantages of UART – hardware complexity is low and for one-to-one connection between devices.
    - ESP 8266 is interfaced with the touch screen using SPI.
    - ESP 8266 is interfaced with the BBB using UART Communication protocol.
- Wi-fi – ESP32 is a Wi-fi module with camera which can provide internet access to our system. Wi-fi is from the family of wireless network protocols, based on IEEE 802.11 family of standards, which is commonly used for local area networking of devices and internet access. Advantages of Wi-fi – Stable and faster signal than cellular data, ability to move the device while still in use and zero involvement of wires.
    - Camera module ESP32 is interfaced with ESP 8266 using GPIO Feature provided specially
- GSM Module: GSM module is interfaced using UART protocol as the module preferable provides UART support, and we are using Arduino mega to make connection between GSM and beagle bone.
    - GSM is connected to beagle bone black through Arduino mega are connected using UART protocol via Arduino Mega.
- Fingerprint Module: Fingerprint module is interfaced using UART protocol as the module preferable provides UART support, and we are using Arduino mega to make connection between Fingerprint and beagle bone.
- Servo Motor: PWM (Pulse Width Modulation), although it cannot be considered as protocol but PWM is being used to send signal to servo motor for rotating it to specific degree.
    - Servo motor is connected to BBB using GPIO pin.
- GMAIL: The IMAP abbreviation stands for Internet Message Access Protocol and is one of the two most popular protocols for receiving email messages from the Internet. SMTP

is an abbreviation that stands for Simple Mail Transfer Protocol. This protocol allows applications to transmit email messages over the Internet.

- The cloud service for this project will be Gmail, to store and send data.

- Buzzer is connected to the GPIO pin of the BBB.
- Vibrator sensor is interfaced with Beagle bone using GPIO pins.

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 enough 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 – 2 A @ 5V
2. GSM SIM900 Module – 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.

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.

Chapter - 4

Design

Project Schematic

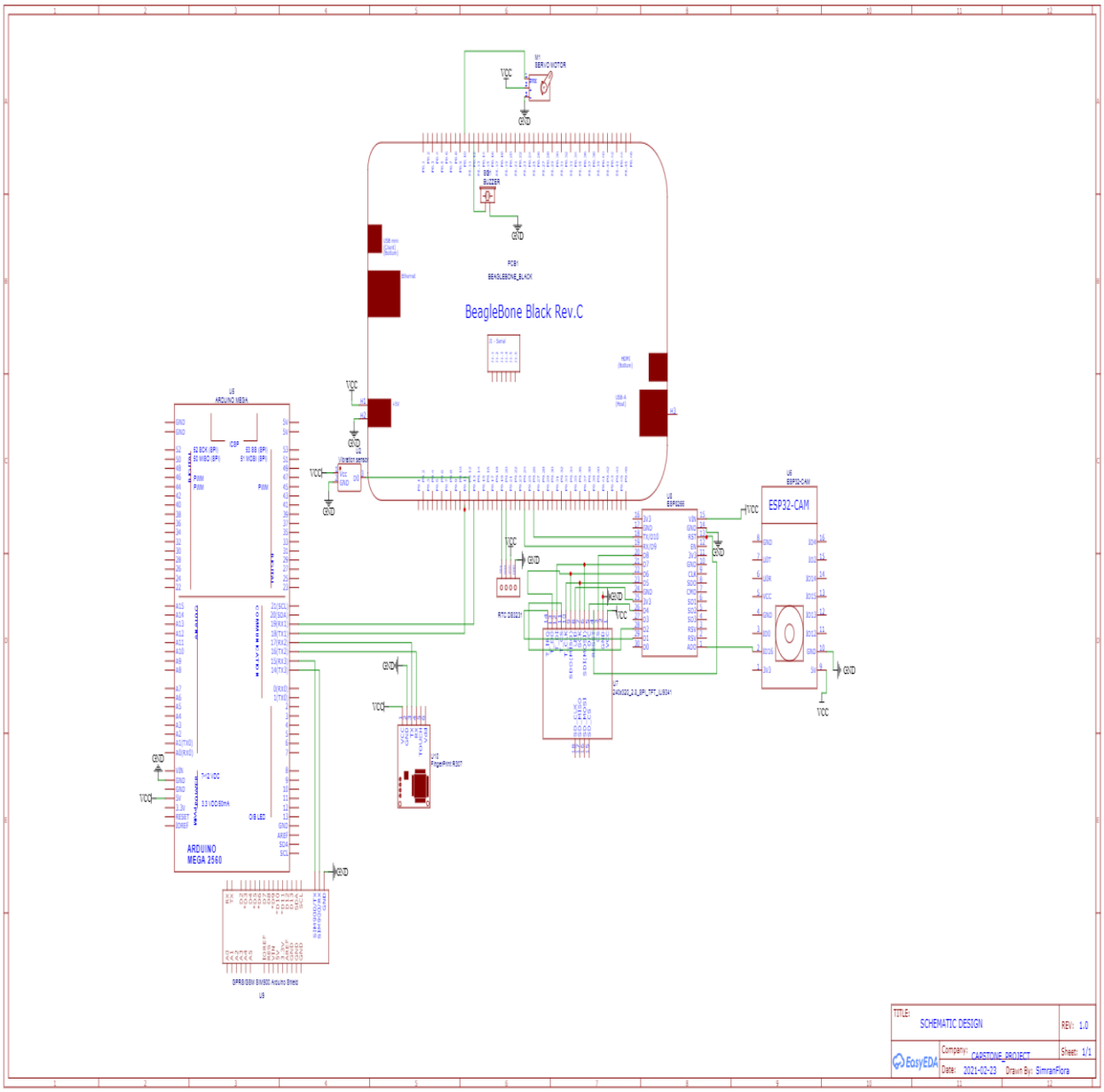

Figure 4.0 Schematic design

Here, we design the schematic design of the project using the EasyEDA online software. EasyEDA is a free open-source platform and is best suited for beginners in schematic capture. It

is an easier and powerful web-based Electronic Design Automation (EDA) tool that help the user to access it anywhere at any time, so software installation is not mandatory. Like user can install it on their personal computers as it is compatible with LINUX, WINDOWS, and MAC OS. EasyEDA is a very user friendly and simple to use software. All the components are easily available in the Library, user can select the desired components, drag and place it.

EasyEDA official sites comes with the tutorials that help the new user to understand the software easily for our project, we are using EasyEDA online. As seen in the schematic one can see hoe the components are connected to the host device. servo motor, RTC, Buzzer, vibrator sensor gets directly connected to the Beagle bone Black. Arduino mega and ESP8266 are used as salve device and required for the interfacing of fingerprint sensor, GSM module, touchscreen, and ESP32CAM module. A point to note here is that all Beagle bone Boards have the same pin configuration. Thus, for the schematic capture we used the layout of Beagle bone 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. But due to the pandemic situation, we are completing our project on ZERO PCB. 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.

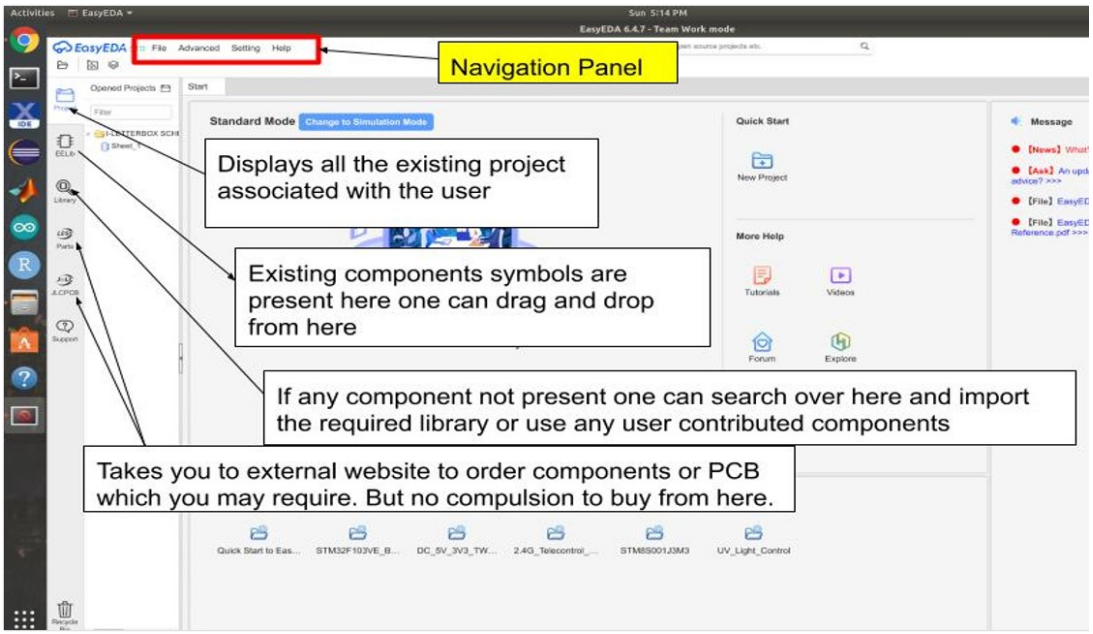

Figure 4.1 basic details about EasyEda Window

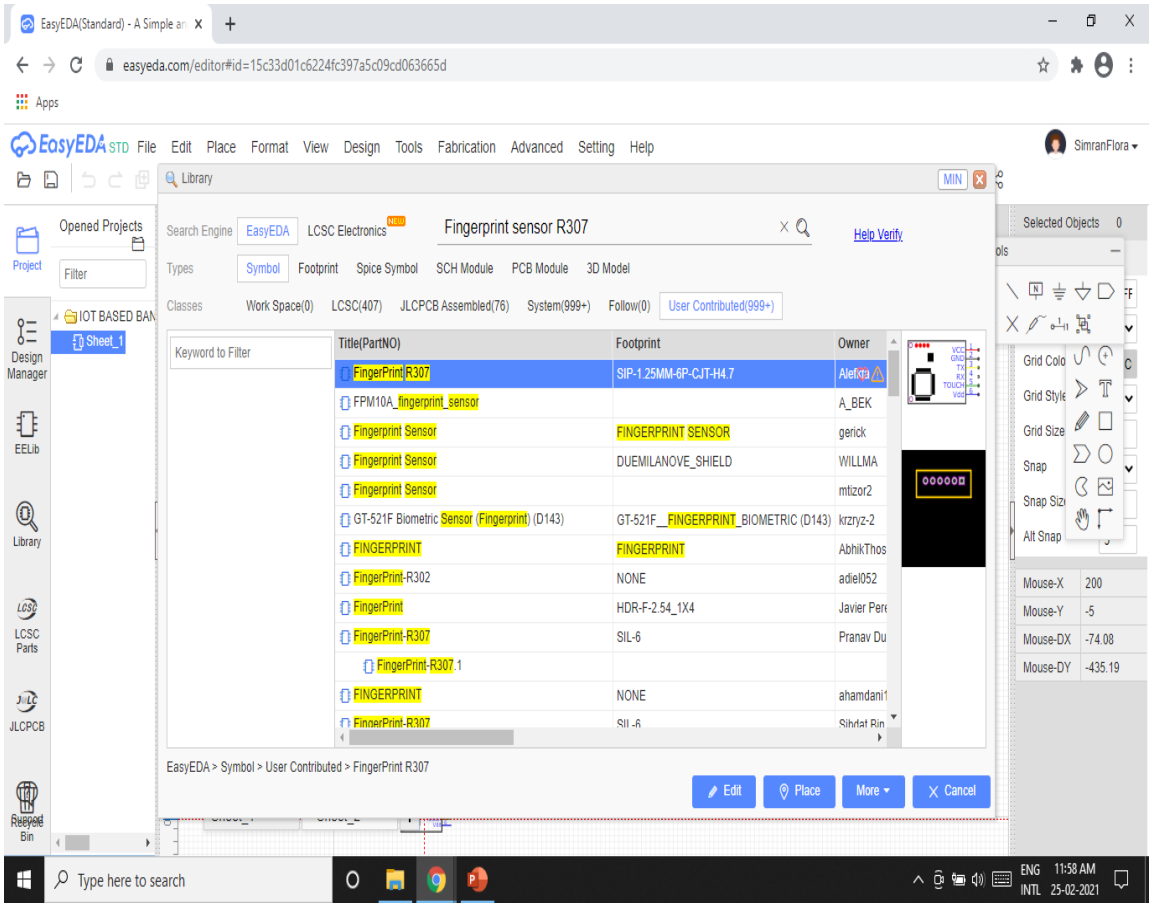

Figure 4.2 basic details about selecting a component from library

Knowing Hardware

1. **Beagle bone Black:** Beagle Bone Black is a low-cost, community-supported development platform for developers and hobbyists. Boot Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable.

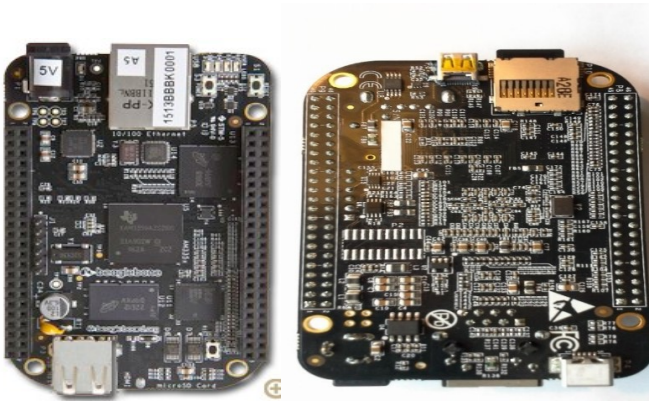

Figure 4.3 Beagle bone black

For connecting all the sensors and other peripheral devices we choose Beagle bone Black as the main processing unit also known as the Host processor. The main features of the Beagle bone Black 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 Bone script Library.
- 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.
- Debug Support: Optional Onboard 20-pin CTI JTAG, Serial Header.
- External Storage: One can use microSD card attached at the bottom of the BBB 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 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:

- ✓ USER0 is the heartbeat indicator from the Linux kernel.
- ✓ USER1 turns on when the SD card is being accessed
- ✓ USER2 is an activity indicator. It turns on when the kernel is not in the idle loop.
- ✓ USER3 turns on when the onboard eMMC is being accessed.

The LEDs are shown clearly in the Figure the image is adopted from Beaglebone Black.

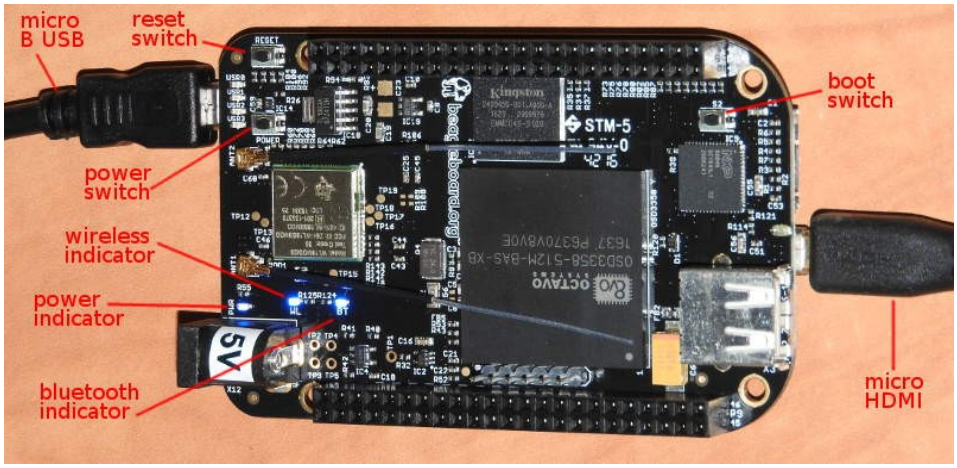

Figure 4.4 Beaglebone Black with all parts shown

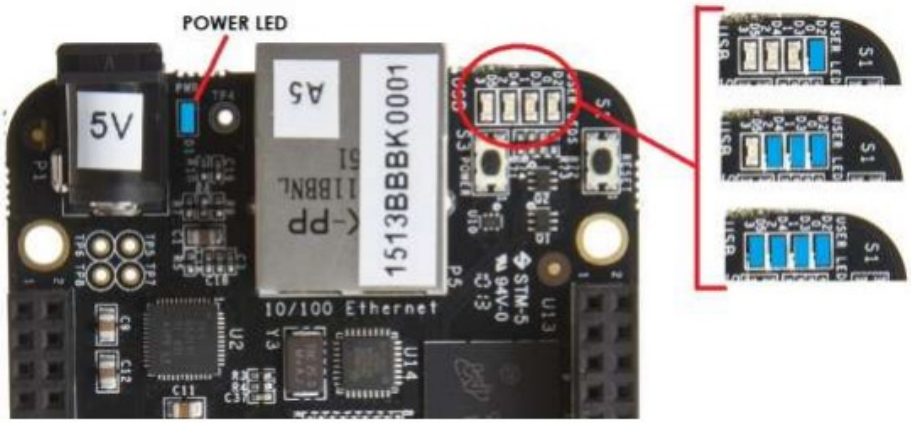

Figure 4.5 Booting LEDs of Beaglebone Black

To control the pins of Beagle bone 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 Beagle bone family as shown in Figure 4.5 This pin diagram is common for all the families be it Beagle bone Black, Beagle bone AI or Beagle bone Wireless.

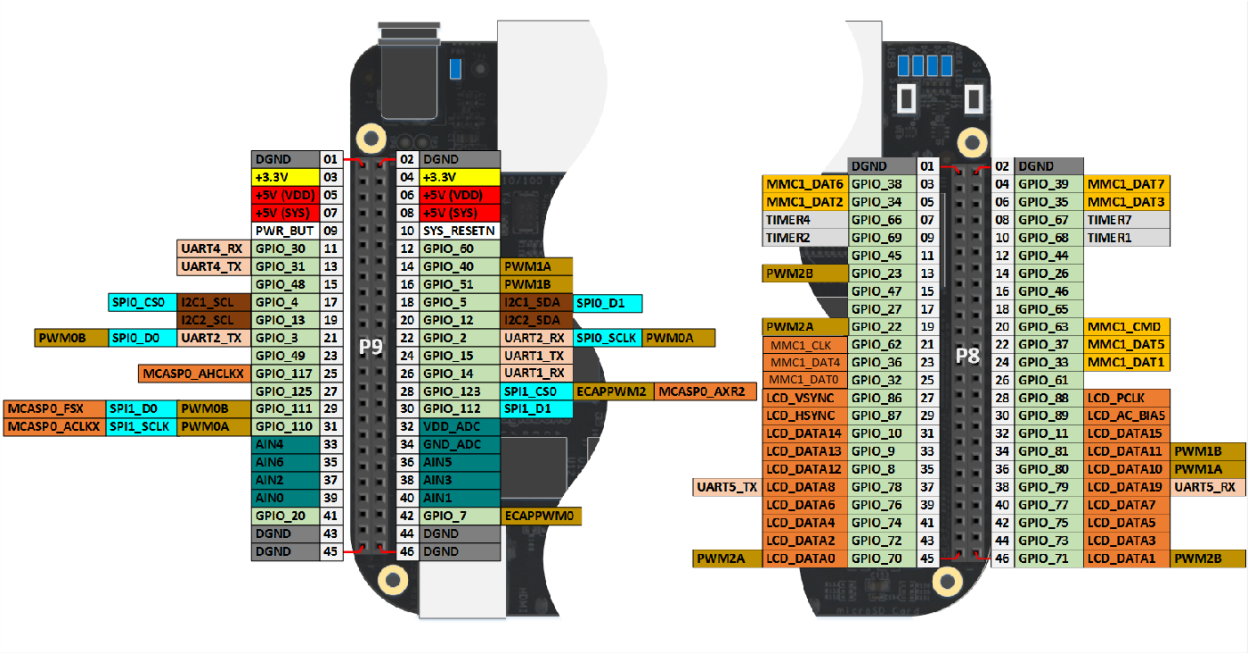

Figure 4.6 Pin Diagram of Beaglebone Family

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. 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

Connectors, LEDs, and Switches

Figure shows the locations of the connectors, LEDs, and switches on the PCB layout of the board. Figure 4.7

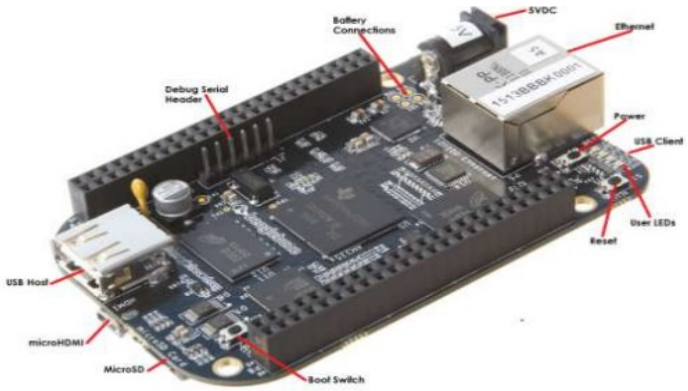

Figure 4.7 connector, Leds and Switches of BBB

- DC Power is the main DC input that accepts 5V power.

- Power Button alerts the processor to initiate the power down sequence.
- 10/100 Ethernet is the connection to the LAN.
- Serial Debug is the serial debug port.
- USB Client is a miniUSB connection to a PC that can also power the board.
- BOOT switch can be used to force a boot from the SD card.
- There are four blue LEDS that can be used by the user.
- Reset Button allows the user to reset the processor.
- uSD slot is where a uSD card can be installed.
- microHDMI connector is where the display is connected to.
- USB Host can be connected different USB interfaces such as Wi-Fi, BT, Keyboard, etc.

Key Components:

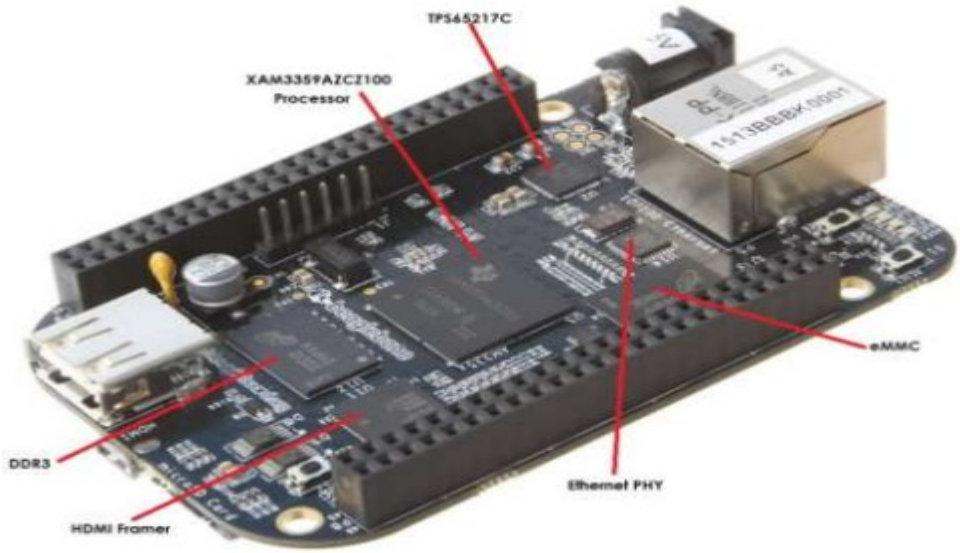

Figure 4.8 Key Components of BBB

- ✓ Sitara AM3359AZCZ100 is the processor for the board.
- ✓ Micron 512MB DDR3L is the Dual Data Rate RAM memory.
- ✓ TPS65217C PMIC provides the power rails to the various components on the board.
- ✓ SMSC Ethernet PHY is the physical interface to the network.
- ✓ Micron eMMC is an onboard MMC chip that holds up to 2GB of data.
- ✓ HDMI Framer provides control for an HDMI or DVI-D display with an adapter.

2. 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 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 4.9

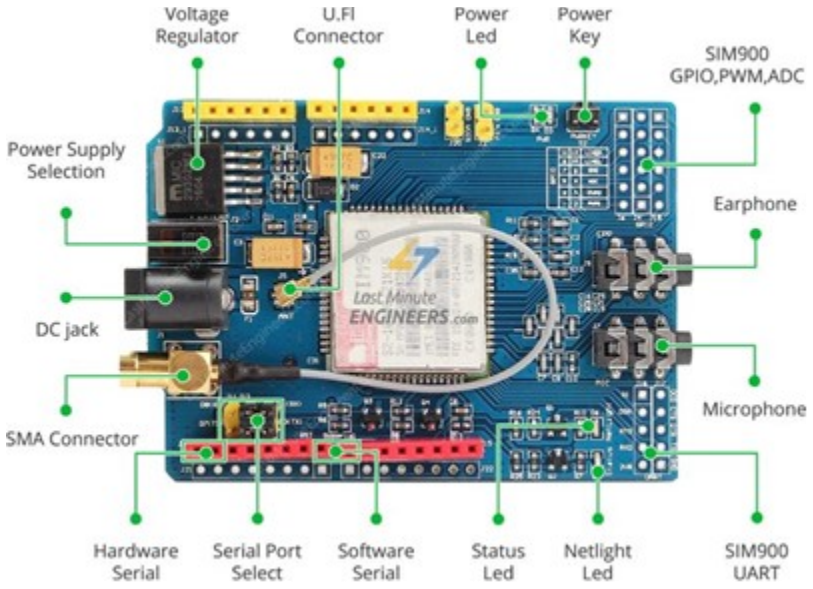

Figure 4.9 GSM SIM900 Shield for sending SMS

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.

3. 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 Beagle bone and other for GSM Module. The adapter used in the project are shown in Figure 4.15

Figure 4.10 Typical Power Adapter of 5V, 2A with barrel jack

4. Servo motor: We are using SG90 SERVO MOTOR in this project. Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement. The PWM sent to the motor determines position of the shaft and based on the duration of the pulse sent via the control wire; the motor will turn to the desired position. Servomotors are used in applications such as robotics, CNC machinery or automated manufacturing.

Figure 4.11 servo motor SG90

Operating features of Servo motor:

- ✓ Operating Voltage is +5V typically
- ✓ Torque: 2.5kg/cm
- ✓ Operating speed is 0.1s/60°
- ✓ Gear Type: Plastic
- ✓ Rotation: 0°-180°
- ✓ Weight of motor: 9gm

5. Buzzer: A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or k Buzzers come in a variety of construction, size, and specifications. Different types and sizes of buzzers are used for different applications. Based on construction, there are the following kinds of buzzers:

- ✓ Piezoelectric buzzers
- ✓ Magnetic buzzers
- ✓ Electromagnetic buzzers
- ✓ Mechanical buzzers
- ✓ Electromechanical buzzerseystroke.

Features of buzzer:

- ✓ In this project, Mechanical buzzer is used with the following features:
- ✓ Rated Voltage: 5V-12VDC
- ✓ 75dB a loud sound for alarm
- ✓ Active drive inside buzzer and simplify your design, only needs to control buzzer on or off
- ✓ Widely used in security and home alarm system, or other areas needs sound alarm.

Figure 4.12 buzzer

6. Vibration sensor: The Vibration sensor is used to detect if there is any vibration that is beyond the threshold. The threshold can be adjusted by the on-board potentiometer. When there no vibration, this module output logic LOW the signal indicates LED light, and vice versa. It can work from 3.3V to the 5V. The sensor uses LM393 comparator to detect the vibration over a threshold point and provide digital data, Logic Low or Logic High, 0 or 1. During normal operation, the sensor provides Logic Low and when the vibration is detected, the sensor provides Logic High. In this project, we will use 5V to power the module. For this project, the vibration sensor is placed in the locker door. It triggers the alarm when any pressure is applied on the locker to open it forcefully.

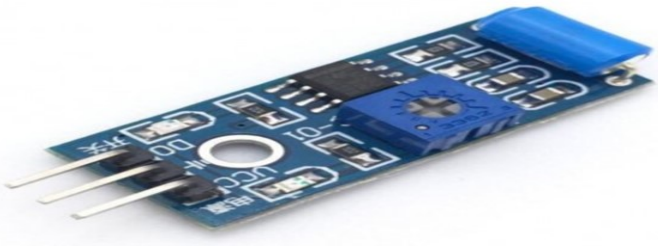

Figure 4.13 Vibration Sensor

Features of Vibrator sensor:

- ❖ The default state of the switch is close.

- ❖ Digital output Supply voltage:3.3V-5V
- ❖ On-board indicator LED to show the results.
- ❖ Dimension of the board: 3.2cm x 1.4cm

7. 2.8" TFT Touchscreen: TFTs, also called TFT screens, are a type of active-matrix LCD display capable of displaying millions of high-contrasts, clear and bright color pixels. TFTs are used in HDTV sets, computer monitors, laptop monitors, tablets, personal media players, smartphones and even feature phones. This is a 2.8-inch serial SPI color display module with touch function. It supports both analog SPI and hardware SPI. It is designed with PC plate including power supply IC and SD. better performance in terms of adjusting the pixels within the display to get better quality. 2.8- inch TFT LCD shield touch screen with SD card socket, easy to use Durable hard PCB ILI9341 board and metal, ideal for 5110 interface Support Serial Mode and hardware SPI, so very convenient to use. 5V compatible, use with 3.3V or 5V logic

Features of Touchscreen:

- ✓ Resolution: 240x320
- ✓ Driver IC: ILI9341
- ✓ Input Voltage: 5V/3.3V
- ✓ Color: 64k
- ✓ Display Module: 1
- ✓ Display area: 46(W) X 65(H)mm
- ✓ Size: 86 x 50 mm

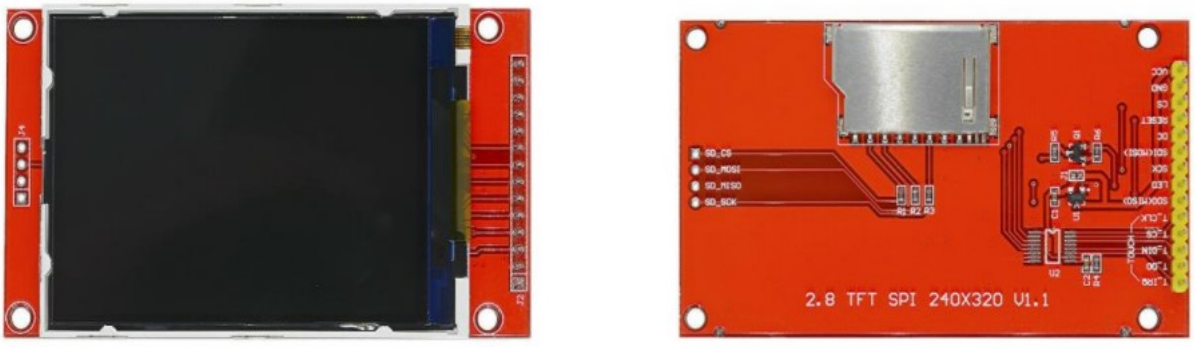

Figure 4.14 2.8" TFT touch screen front and back view

8. ESP8266: The NodeMCU (Node Microcontroller Unit) is an open-source software and hardware development environment built around an inexpensive System on-a-Chip (SoC) called the ESP8266. The ESP8266, designed and manufactured by Espressif Systems, contains the crucial elements of a computer: CPU, RAM, networking (Wi-Fi), and even a modern operating system and SDK.

Features of ESP8266:

- ✓ This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency.
- ✓ NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs.
- ✓ High processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating features make it ideal for IoT projects.
- ✓ NodeMCU can be powered using Micro USB jack and VIN pin (External Supply Pin).
- ✓ It supports UART, SPI, and I2C interface

Figure 4.15 ESP8266 nodemcu

9. ESP32CAM: The ESP32-CAM is a very small camera module with the ESP32-S chip that costs approximately $10. Besides the OV2640 camera, and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients. The ESP32-CAM doesn't come with a USB connector, so you need an FTDI programmer to upload code through the U0R and U0T pins (serial pins).

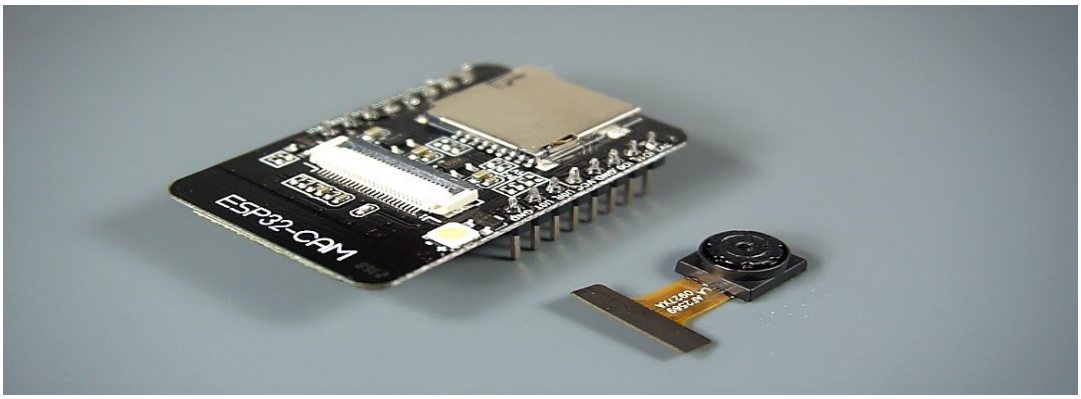

Figure 4.16 ESP32CAM module

Features of ESP32CAM:

- ✓ The smallest 802.11b/g/n Wi-Fi BT SoC module
- ✓ Low power 32-bit CPU, can also serve the application processor
- ✓ Up to 160MHz clock speed, summary computing power up to 600 DMIPS
- ✓ Built-in 520 KB SRAM, external 4MPSRAM
- ✓ Supports UART/SPI/I2C/PWM/ADC/DAC
- ✓ Support OV2640 and OV7670 cameras, built-in flash lamp
- ✓ Support image Wi-Fi upload
- ✓ Support TF card
- ✓ Supports multiple sleep modes
- ✓ Embedded Lwip and FreeRTOS
- ✓ Supports STA/AP/STA+AP operation mode
- ✓ Support Smart Config/AirKiss technology
- ✓ Support for serial port local and remote firmware upgrades (FOTA)

10. Fingerprint Sensor: Fingerprint Sensor is the type of technology that helps to identifies and authenticates the fingerprints of an individual to grant or deny an access to the system. Nowadays, Fingerprint scanner is used everywhere for security purposes. For this project, we are using R307 fingerprint sensor. R307 Fingerprint Module consists of high-speed DSP processor, high-performance fingerprint alignment algorithm, high-capacity FLASH chips and other hardware and software composition, stable performance, simple structure, with fingerprint entry, image processing, fingerprint matching, search and template storage and other functions. The user can store up to 127 fingers in this module. The FP module can directly interface with 3.3 or

5v Microcontroller. For this project, we are interfacing with Arduino Mega. There are 6 pins in the module with different color wires. The color of the wires determines the pins of the module.

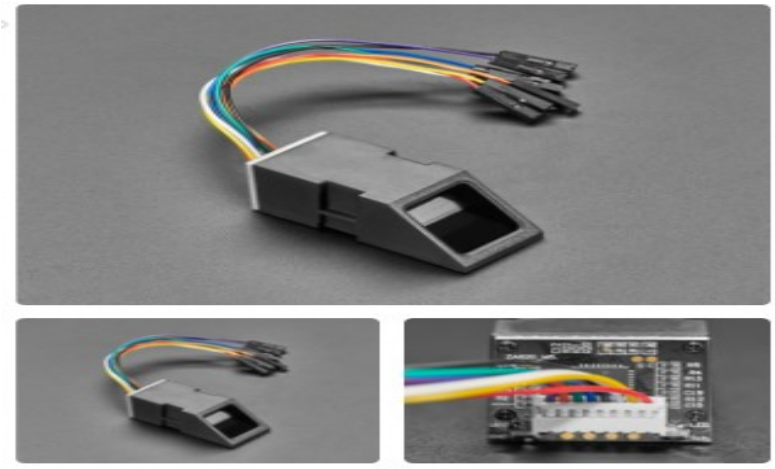

Figure 4.17 R307 Fingerprint sensor

Features of Fingerprint:

- Supply voltage: DC 4.2 ~ 6.0V
- Working current: 50mA (typical)
- Peak current: 80mA
- Fingerprint image input time: <0.3 seconds
- Window area: 14x18 mm
- Matching mode:1: 1 and 1: N
- Communication Protocol: UART

11. Real Time Clock (RTC): For this project, we are using the RTC because the beagle bone black does not come with the feature of real time clock. So, we are using DS3231 for this project. The DS3231 is a low-cost, extremely accurate I2C real-time clock (RTC) with an integrated temperature compensated crystal oscillator (TCXO) and crystal. The device incorporates a battery input and maintains accurate timekeeping when main power to the device is interrupted.

The RTC maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year.

RTC module battery backup: The DS3231 incorporates a battery input and maintains accurate timekeeping when main power to the device is interrupted. The bottom side of the board holds a battery holder for 20mm 3V lithium coin cells. Any CR2032 battery can fit well.

Features of RTC:

- Can function with low voltages
- A programmable Square-wave output as per requirement
- A battery backup to stay updated even if there is no power
- A dual-directional, 400 kHz of I2C interface for speedy transmission
- 32 bytes of EEPROM for to read/write or to save data
- A pushbutton to reset time
- RTC can be used either in 12hrs or 24hrs format
- An aging trim register to set a user-provided value as an offset with reference to the factory value
- Maintains seconds, minutes, hours, days, weeks, months, years information
- Switches automatically from a power source to an in-built battery source

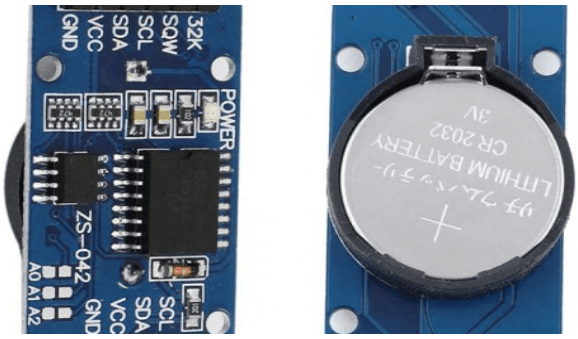

Figure 4.18 DS3231 RTC

12. Arduino Mega 2560: The Arduino MEGA 2560 is designed for projects that require more I/O lines, more sketch memory, and more RAM. With 54 digital I/O pins, 16 analog inputs and a larger space for your sketch it is the recommended board for 3D printers and robotics projects. This gives your projects plenty of room and opportunities maintaining the simplicity and effectiveness of the Arduino platform. This document explains how to connect your Mega2560 board to the computer and upload your first sketch.

The Arduino Mega 2560 is programmed using the Arduino Software (IDE), our Integrated Development Environment common to all our boards and running both online and offline. For

more information on how to get started with the Arduino Software visit the Getting Started page. If you want to program your Arduino Mega 2560 while offline you need to install the Arduino Desktop IDE. Connect your Mega2560 board with an A B USB cable; sometimes this cable is called a USB printer cable.

For this project, we are using Arduino Mega as a slave device, and it is used for the interfacing of GSM and fingerprint sensor with the beagle bone black and UART communication protocol is used.

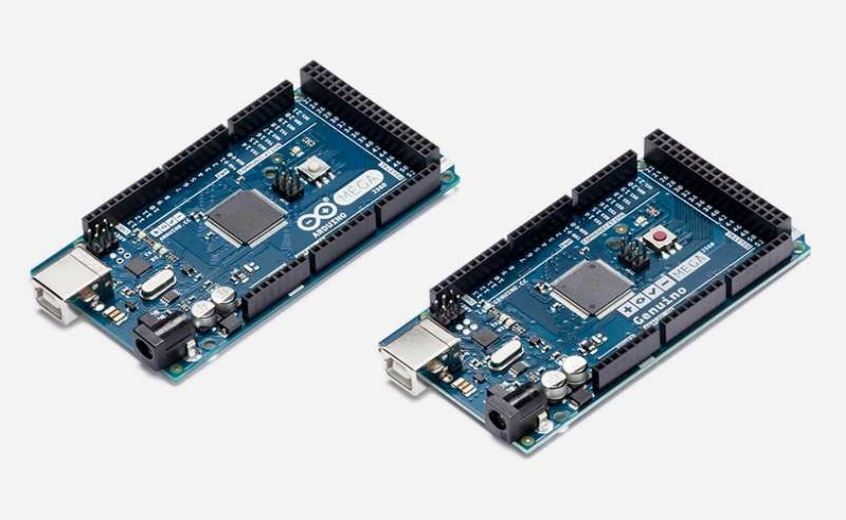

Figure 4.19 Arduino mega 2560

Features of Arduino Mega 2560:

- ✓ Microcontroller: ATmega2560
- ✓ Operating Voltage: 5V
- ✓ Input Voltage (recommended): 7-12V
- ✓ Input Voltage (limit): 6-20V
- ✓ Digital I/O Pins: 54 (of which 15 provide PWM output)
- ✓ Analog Input Pins: 16
- ✓ DC Current per I/O Pin: 20 mA
- ✓ DC Current for 3.3V Pin: 50 mA
- ✓ Flash Memory: 256 KB of which 8 KB used by bootloader
- ✓ SRAM: 8 KB

- ✓ EEPROM: 4 KB
- ✓ Clock Speed: 16 MHz
- ✓ LED_BUILTIN: 13

13. Zero PCB: Zero PCB is basically a general-purpose printed circuit board (PCB), also known as perfboard or DOT PCB. It is a thin rigid copper sheet with holes pre-drilled at standard intervals across a grid with 2.54mm (0.1-inch) spacing between holes.

Function of Holes: There are number of holes in the Zero PCB. Each hole is encircled by a round or square copper pad so that component lead can be inserted into the hole and soldered around the pad without short-circuiting the nearby pads and other leads. For connecting the lead of component with another lead, solder these together or join these using a suitable conducting wire.

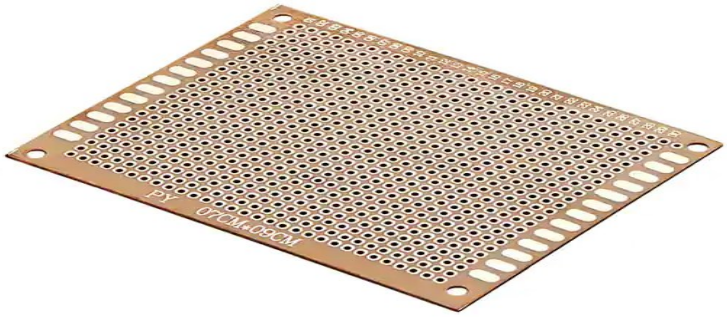

Figure 4.20 Zero PCB

Cloud Service - GMAIL

Gmail was a project started by Google developer Paul Buchheit, who had already explored the idea of web-based email in the 1990s, before the launch of Hotmail, while working on a personal email software project as a college student. Buchheit began his work on Gmail in August 2001.

The IMAP abbreviation stands for Internet Message Access Protocol and is one of the two most popular protocols for receiving email messages from the Internet. SMTP is an abbreviation that stands for Simple Mail Transfer Protocol. This protocol allows applications to transmit email messages over the Internet.

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. PCB designing is a computer-aided designing technology. It is used to plan and design circuit boards for electronics circuits design to deliver a specific output. It can be done for both through-hole and surface mount electronic components.

Typically, any electronic PCB design involves putting together a combination of power supply systems, control systems, communication systems, input, output systems together in a single or multiple PCBs that can work in interfacing with each other. Circuit designing is completed by using manual routing or auto router. We used the same schematics which we used to understand how we can draw out the connection between the Beagle bone Black and Sensors and other peripheral 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 would not be possible. One more advantage is, it can support other software libraries and schematic which includes Altium, EAGLE, LTspice, and DXF.

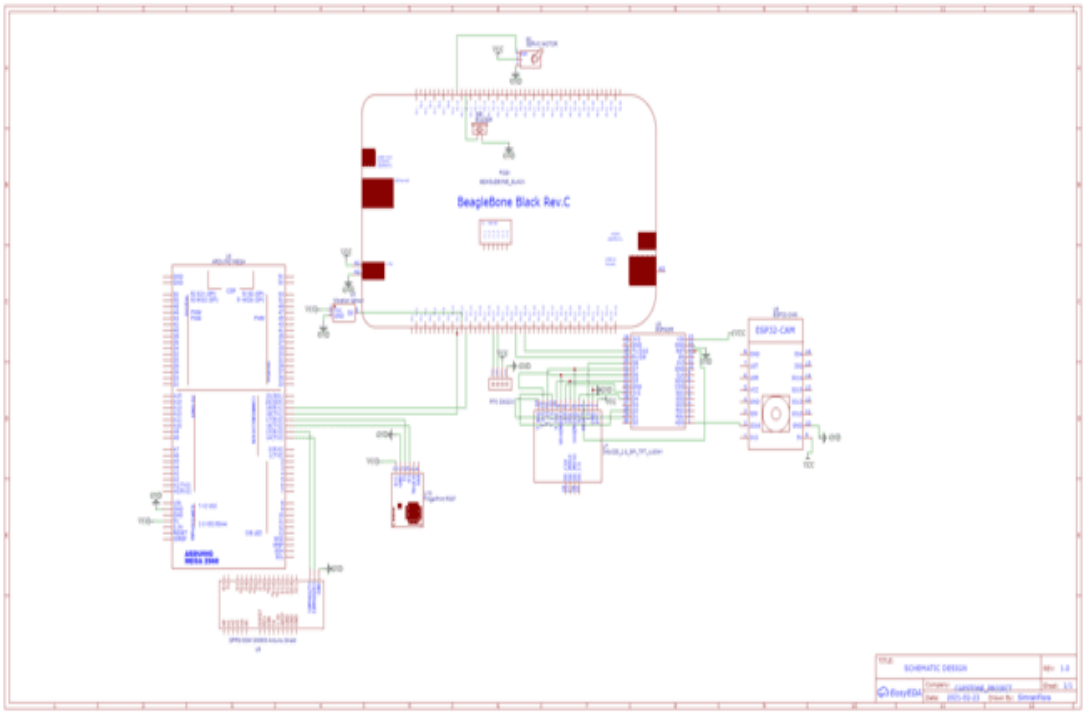

Figure 4.21 Schematic design of the project

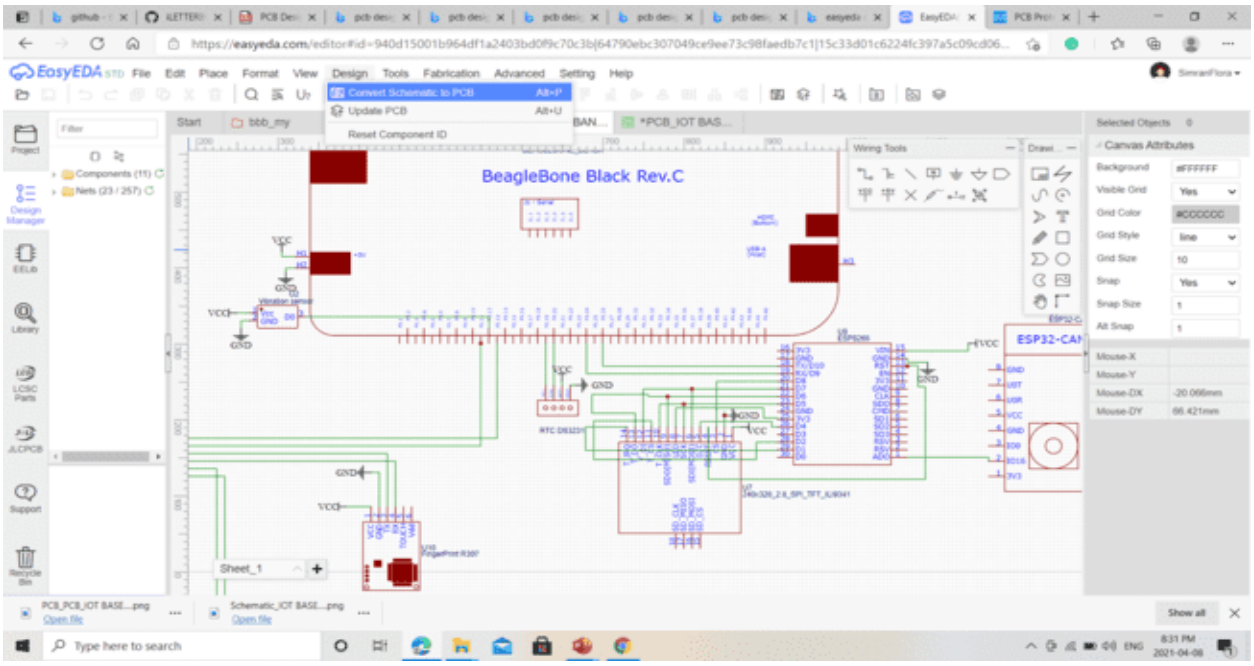

Figure 4.22 Steps to convert Schematic design to PCB design

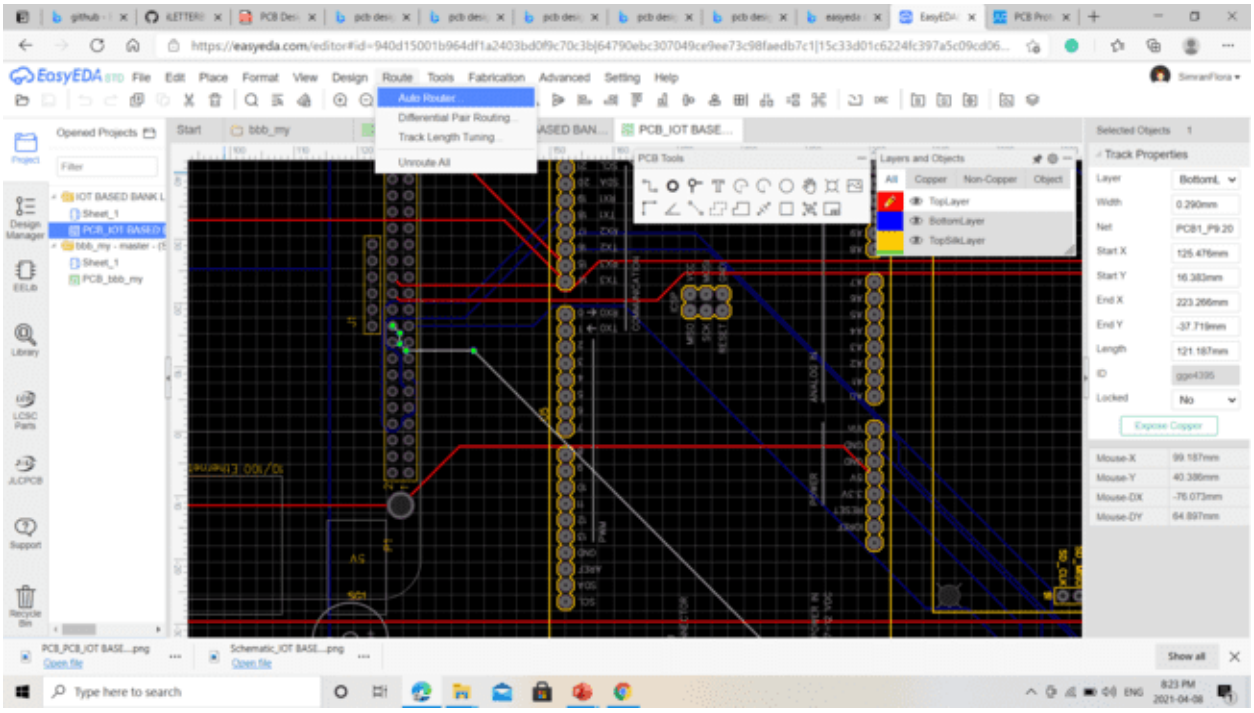

Figure 4.23 Steps to determine the Auto routing

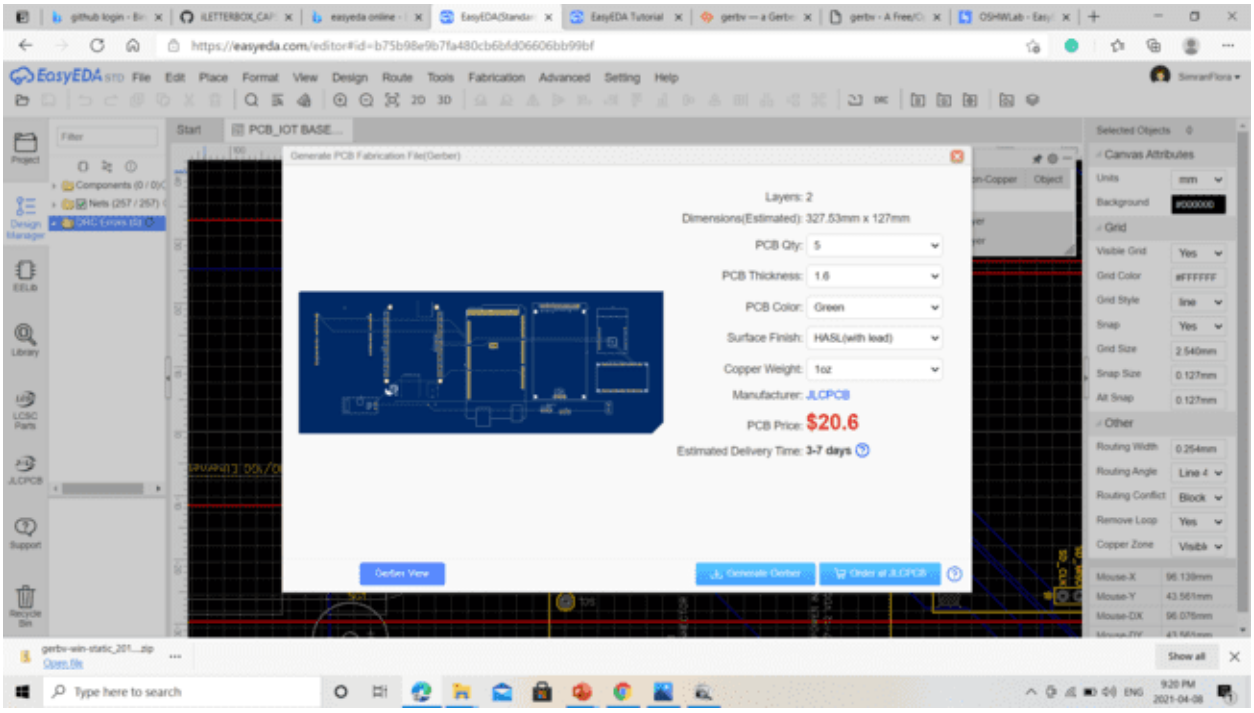

Figure 4.24 Steps to generate Gerber files, and to order the PCB online

From this part of the EasyEDA, online software, we can order the printed PCB and it cost only 20$ as shown in figure 4.24. We just need to send the Gerber files that are generated after the pcb is routed and then we can send to the company to obtain the exact replica of the designed PCB. For our project, we have only two layers that are top layer and bottom layer.

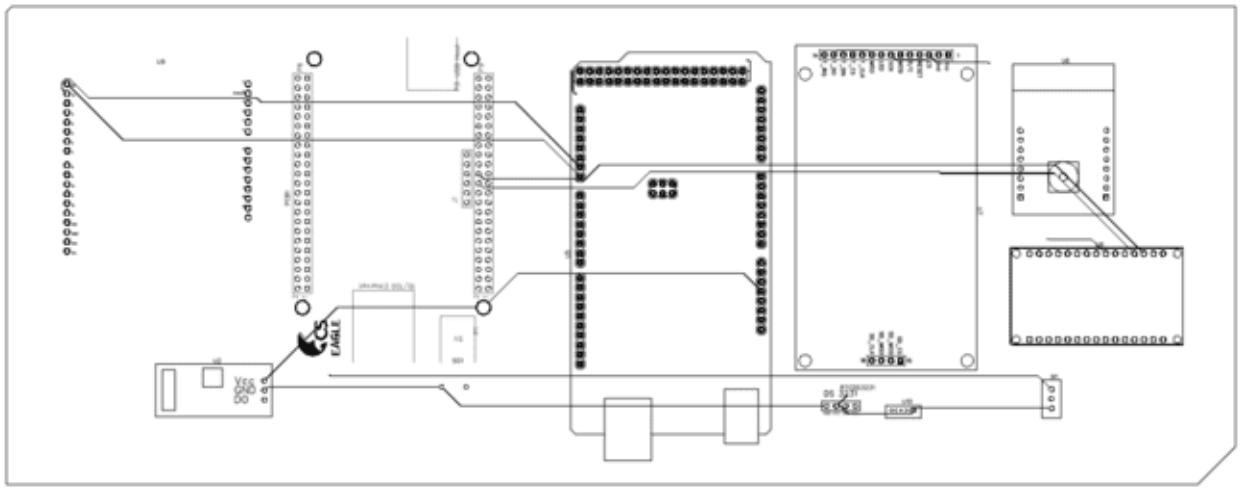

Figure 4.25 PCB design of the project

Chapter-5

Implementation and Test

Flow Chart

The flowchart of an IOT based Bank Locker Security System is shown in figure 5.0.

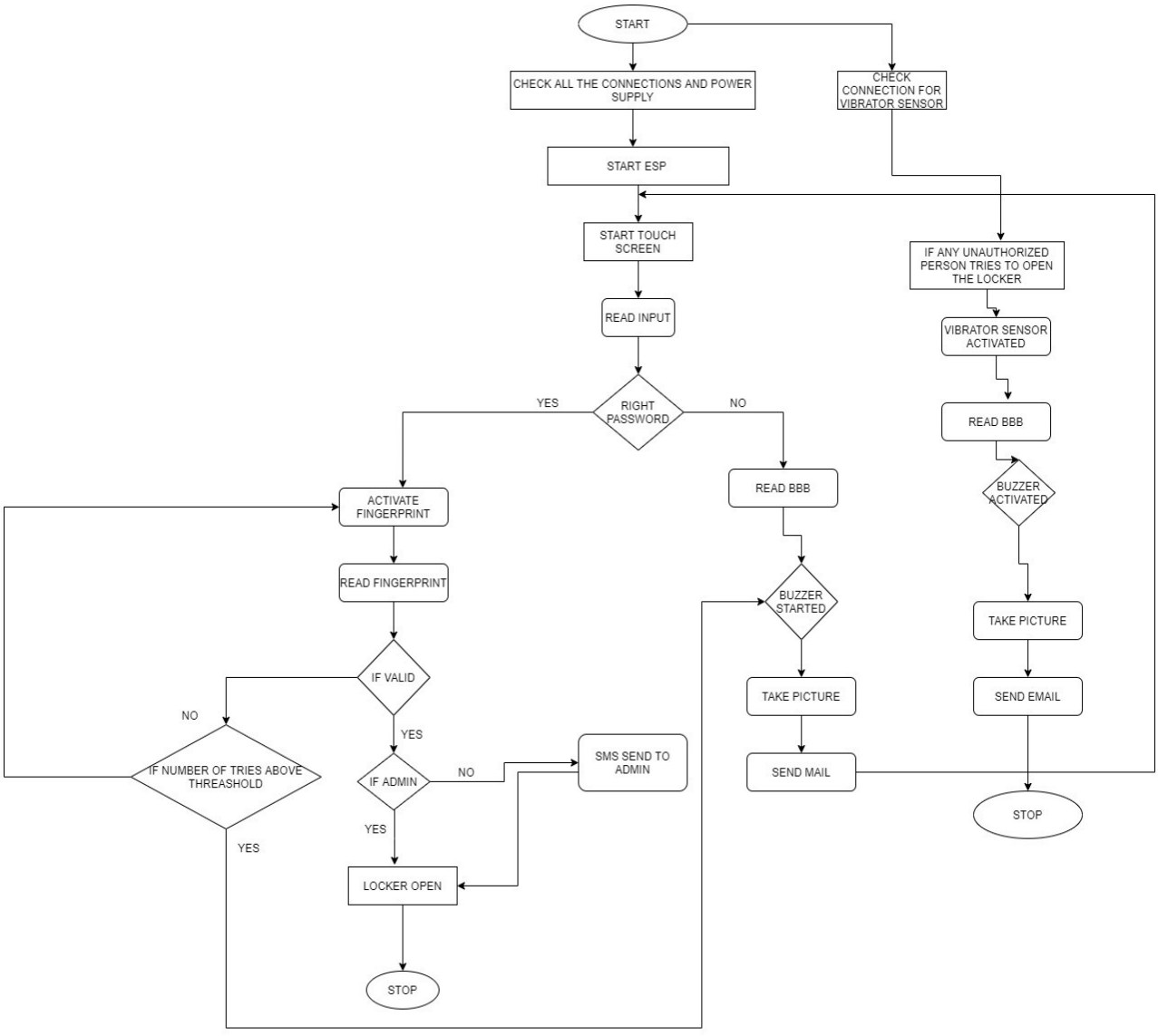

Figure 5.0 Flowchart of our proposed project

Here in the figure 5.0, we have the flowchart of the purposed project. In the flowchart, we have described that firstly check the power supply for the whole project. Check that beagle bone black and esp8266 are started first. The we have assigned two users to the locker system. Now the user

needs to enter the password first to enable the finger print sensor. When the user enters the correct password, there will be a welcoming message on the screen that determine will user is operating the locker. Now, the fingerprint sensor is ready to scan, we have given 3 attempts to the user to scan their finger prints. Now, here is the trick, if the main user scans the finger print, then he can directly open the locker. But when the co-owner tries to open the locker. And apply his finger print on the scanner, a SMS will be sent to the owner with the help of GSM module. However, when the user enters the wrong password, the buzzer will trigger and the camera will take picture and send it to the cloud. So, no chances are given for the password retry. One more important aspect is there, the vibration sensor, so if any intruder tries to open the locker forcefully, this sensor will work and buzzer will activities, and the camera will take picture and send it to the cloud.

Interfacing with Beaglebone Black

- **Interfacing Beaglebone Black with buzzer**

  Figure 5.1 shows how the interfacing of buzzer is done with Beaglebone Black. The Buzzer works on 5V voltage. Buzzer is directly connected to the beaglebone black with the help of GPIO pins as shown in figure 5.2. The function of buzzer is to make sound when something wrong happens.

Buzzer	Beaglebone Black
RED	P8.12
BLACK	P8.2

Figure 5.1 : Interfacing of beaglebone black with buzzer

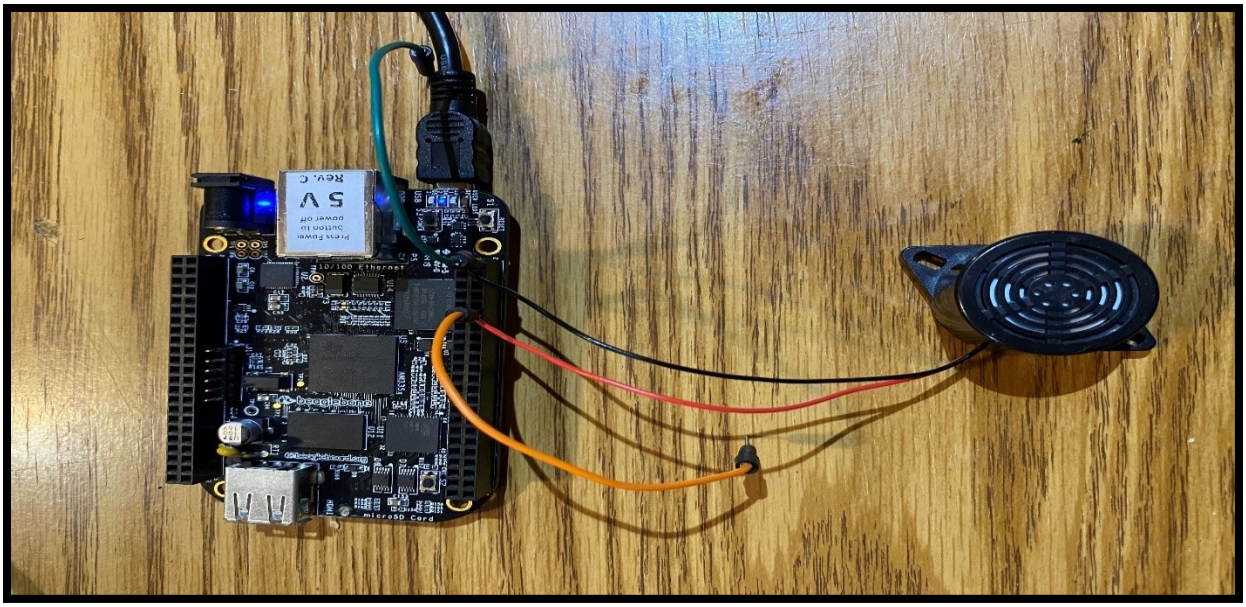

Figure 5.1(a): Connections of Beaglebone Black with the buzzer

- Installing the IOBB library

STEP1: To install the library, initially use the command “ls” to get the list of files. Then, enter the command “cd iobb.” Here, it is used to go to the directory iobb and to open the library folder.

```
debian@192.168.7.2's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Dec  3 19:22:32 2020 from 192.168.7.1
debian@beaglebone:~$ ls
bin  iobb  pthread.c
debian@beaglebone:~$ cd iobb
debian@beaglebone:~/iobb$ ls
ADC_VOICE  blog-apps  Lab      Makefile  README.md  Toolkit
```

Figure 5.2 installing IOBB library in bbb

STEP 2: “Make” command is entered to next make executable file, then it require some time to execute the command and open the file.

```
BBBio_lib  Demo       LICENSE  overlay   test-io
debian@beaglebone:~/iobb$ sudo make
[sudo] password for debian:
gcc -c ./BBBio_lib/BBBiolib_PWMSS.c -o ./BBBio_lib/BBBiolib_PWMSS.o -W
gcc -c ./BBBio_lib/BBBiolib_McSPI.c -o ./BBBio_lib/BBBiolib_McSPI.o -W
gcc -c ./BBBio_lib/BBBiolib_ADCTSC.c -o ./BBBio_lib/BBBiolib_ADCTSC.o -W
gcc -c ./BBBio_lib/i2cfunc.c -o ./BBBio_lib/i2cfunc.o
gcc -c ./BBBio_lib/BBBiolib.c -o ./BBBio_lib/BBBiolib.o
ar -rs ./BBBio_lib/libiobb.a ./BBBio_lib/BBBiolib.o ./BBBio_lib/BBBiolib_PWMSS.o
 ./BBBio_lib/BBBiolib_McSPI.o ./BBBio_lib/BBBiolib_ADCTSC.o ./BBBio_lib/i2cfunc.
o
ar: creating ./BBBio_lib/libiobb.a
cp ./BBBio_lib/libiobb.a ./
cp ./BBBio_lib/BBBiolib.h ./iobb.h
cp ./BBBio_lib/BBBiolib_ADCTSC.h ./
cp ./BBBio_lib/BBBiolib_McSPI.h ./
cp ./BBBio_lib/BBBiolib_PWMSS.h ./
cp ./BBBio_lib/i2cfunc.h ./
gcc -o LED ./Demo/Demo_LED/LED.c -L ./BBBio_lib/ -liobb
gcc -o ADT7301 ./Demo/Demo_ADT7301/ADT7301.c -L ./BBBio_lib/ -liobb
gcc -o SevenScan ./Demo/Demo_SevenScan/SevenScan.c  -L ./BBBio_lib/ -liobb
gcc -o SMOTOR ./Demo/Demo_ServoMotor/ServoMotor.c -L ./BBBio_lib/ -liobb
gcc -o LED_GPIO ./Demo/Demo_LED_GPIO/LED_GPIO.c -L ./BBBio_lib/ -liobb -pthread
```

Figure 5.2(a) installing IOBB library in bbb

STEP 3: To install the make file “sudo make install” command is entered. Finally, the library is installed in the Beaglebone Black and is ready to use.

```
debian@beaglebone: ~/iobb
File  Edit  View  Search  Terminal  Help
gcc -o Debouncing ./Demo/Demo_Debouncing/Debouncing.c -L ./BBBio_lib/ -liobb
gcc -o 4x4keypad ./Demo/Demo_4x4keypad/4x4keypad.c -L ./BBBio_lib/ -liobb
gcc -o ADC ./Demo/Demo_ADC/ADC.c -L ./BBBio_lib/ -liobb -lm
gcc -o ADC_VOICE ./Demo/Demo_ADC/ADC_voice.c -L ./BBBio_lib/ -liobb -lm -pthread
 -O3
gcc -o GPIO_CLK_status ./Toolkit/Toolkit_GPIO_CLK_Status/GPIO_status.c  -L ./BBB
io_lib/ -liobb
gcc -o EP_status ./Toolkit/Toolkit_EP_Status/EP_status.c -L ./BBBio_lib/ -liobb
gcc -o ADC_CALC ./Toolkit/Toolkit_ADC_CALC/ADC_CALC.c
gcc -o lcd3-test ./Demo/Demo_I2C/lcd3-test.c -I. -L ./BBBio_lib/ -liobb
gcc -o test-outputs test-io/test-outputs.c -I. -L. -liobb
gcc -o pb-test-outputs test-io/pb-test-outputs.c -I. -L. -liobb
gcc -o test-inputs test-io/test-inputs.c -I. -L. -liobb
gcc -o pb-test-inputs test-io/pb-test-inputs.c -I. -L. -liobb
debian@beaglebone:~/iobb$ sudo make install
rm -f /usr/local/include/BBBiolib.h
cp ./BBBio_lib/libiobb.a /usr/local/lib
cp ./BBBio_lib/BBBiolib.h /usr/local/include/iobb.h
cp ./BBBio_lib/BBBiolib_ADCTSC.h /usr/local/include
cp ./BBBio_lib/BBBiolib_McSPI.h /usr/local/include
cp ./BBBio_lib/BBBiolib_PWMSS.h /usr/local/include
cp ./BBBio_lib/i2cfunc.h /usr/local/include
ln -s /usr/local/include/iobb.h /usr/local/include/BBBiolib.h
debian@beaglebone:~/iobb$ ~~
```

Figure 5.2(b) using the install commands on terminal

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 <iobb.h>  // Library to access GPIO
#include <stdio.h> // Standard IO library

//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


int main(void)
{
  iolib_init();   //Initializing the iobb library
  iolib_setdir(8, 12, DigitalOut);   //Setting Pin as Output of a specific port
  while(1)  // Infinite Loop
  {
    pin_high(8, 12);   // Making the connected pin high
    printf("BUZZER ON \n"); // Message Print
    iolib_delay_ms(500);  //A delay of 500 ms

    pin_low(8, 12);   // Making the connected pin low
    printf("BUZZER OFF \n"); // Message Print
    iolib_delay_ms(500);   //A delay of 500 ms
  }
  iolib_free();  //Free up all the GPIOs
  return(0);  //End of Main Function
}
```

The below mentioned are the commands that should be entered on the terminal.

STEP 1: Enter the command “ssh debian@192.168.7.2” here ssh command instruct the system to establish an encrypted secure connection with the host machine. Debian here represent the user_name that is being accessed on the host and then it is followed by an IP address.

STEP2: Before continuing to next step, the password “temppwd” must be entered.

STEP 3: Enter the command “sudo nano buzzerTest.c” to open Nano text editor and to directly write, edit and navigate the code and to get immediate onscreen feedback. Here, buzzerTest.c is the file name. Then enter the same password “temppwd.” Here CTRL+O: save the code; then press enter; CTRL+X: to exit.

STEP4: To compile and execute the code, enter the command “gcc buzzerTest.c -o buzzerTest liobb.” Here, buzzerTest.c is the program_name and buzzerTest is the executable file name. Afterwards, press Enter and write the Executable_name : “sudo ./buzzerTest.”

STEP5: Again enter the password: temppwd.

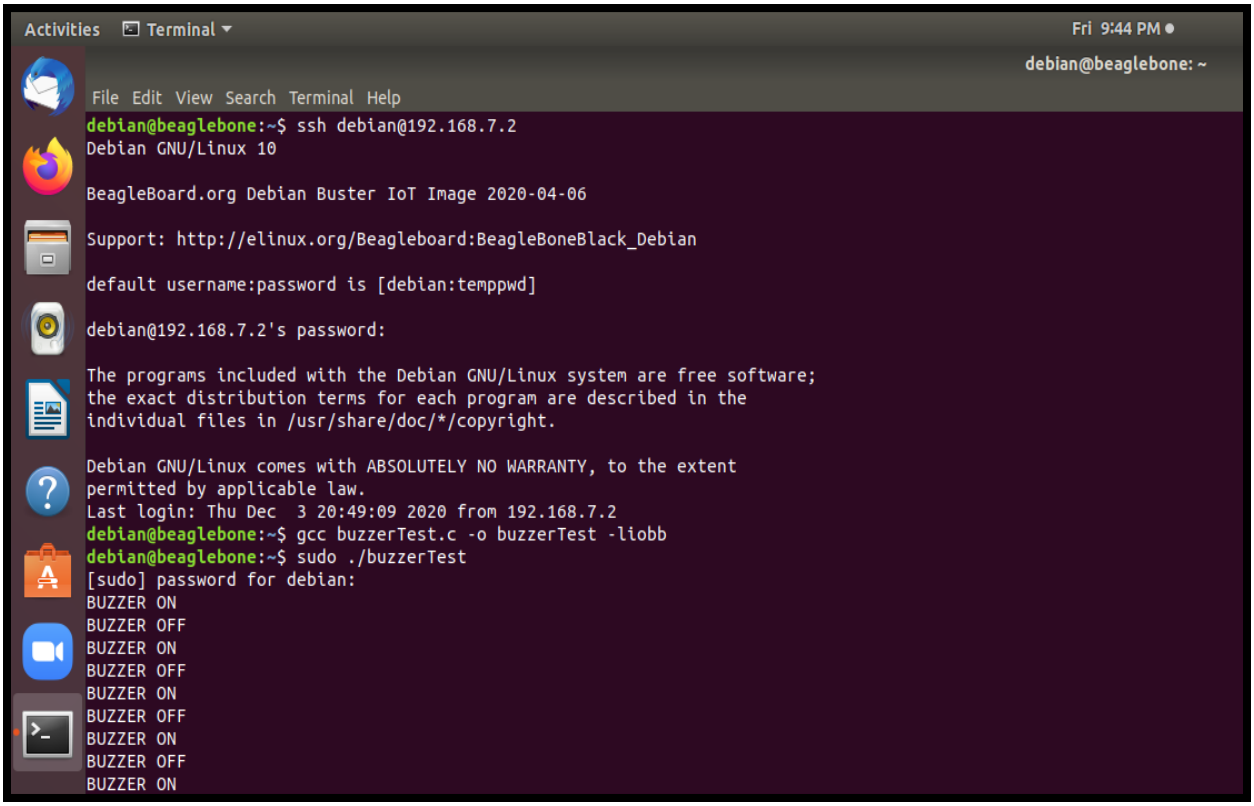

Figure 5.3: Output of the buzzer

- **Interfacing of beaglebone black with Servo motor**

  After the interfacing of buzzer, the Servo motor is connected directly to the beaglebone black. Connections are made with the help of GPIO pins.

Servo motor	Beaglebone Black
GND(BROWN)	P8.2
5V (RED)	P9.7
PWM(BROWN)	P8.10

Figure 5.4: Interfacing of beaglebone black with servo motor

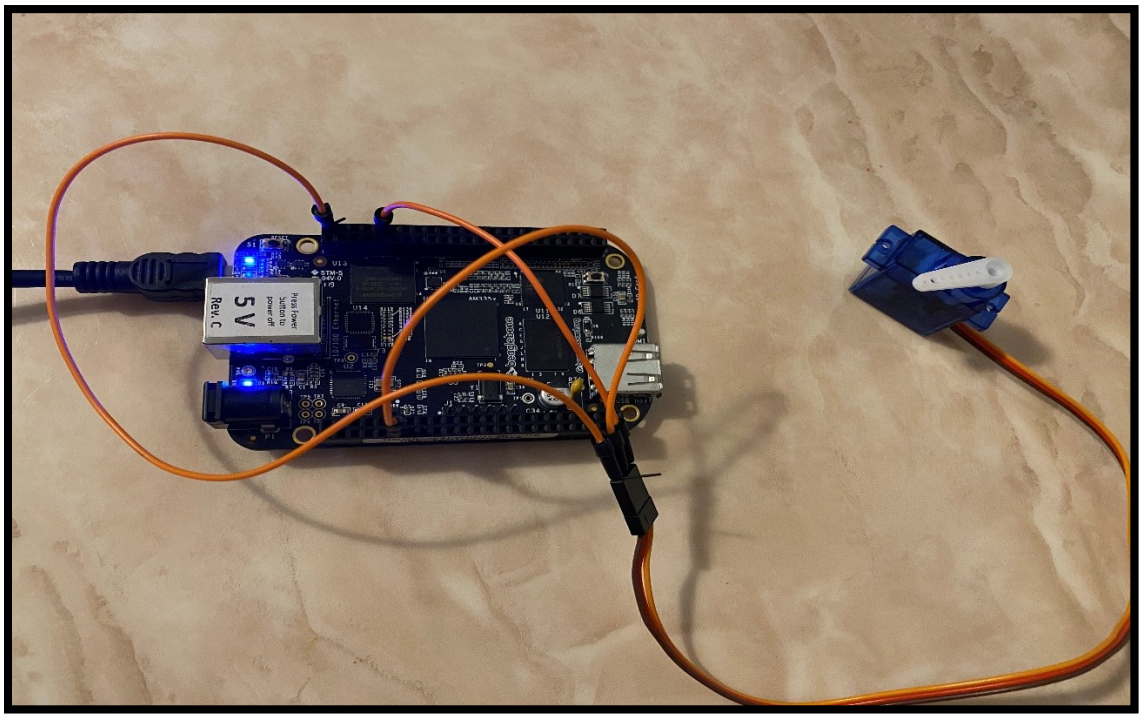

Figure 5.5: Connections of Beaglebone Black with Servo Motor

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 <iobb.h>   // A header library to control GPIOs of Beaglebone
#include <stdio.h>   //Standard C input Output Library
#include <time.h>  //Time Library
#include <unistd.h> //defines miscellaneous symbolic constants and types, and declares
miscellaneous functions
#include <sys/types.h> //definitions for types like size_t , ssize_t

void servo_move(int angle); //Defining a function

int pin = 10;

int  main(void)

{

  iolib_init();   //Initializing the iobb library
  iolib_setdir(8, pin, DigitalOut);   //Setting Pin direction of a specific pin of a
specific port
  printf("Program Started\n"); // A simple print message

  while(1)  //Infinite Loop

    {

      for(int i=0;i<180;i = i+10) // A for loop to servo_angle function repeatedly with
value 0 - 180
      {
        servo_move(i);  // Calling the Function to move servo at specific angle
        iolib_delay_ms(50);
      }

      for(int i=180;i>0;i= i-10) // A for loop to servo_angle function repeatedly with
value 180 - 0
      {
       servo_move(i); // Calling the Function to move servo at specific angle
       iolib_delay_ms(50);
      }

  }


  iolib_free();  // Free the GPIos
```

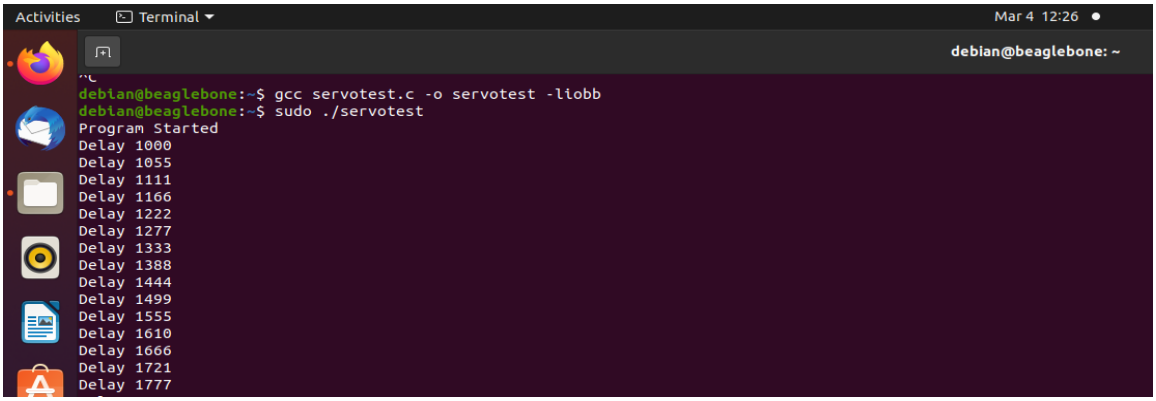

Figure 5.6: Output of the Servo Motor

- **Interfacing of vibration sensor with Beaglebone Black**

Vibration sensor is connected directly to the beaglebone black with the help of GPIO pins as shown in figure 5.7. This sensor is used to detect any unauthorized access by hammering the locker.

VIBRATOR SENSOR	BEAGLEBONE BLACK
DO	P9.12
VCC	VCC
GND	GND

Figure 5.7: Interfacing of vibration sensor with Beaglebone Black

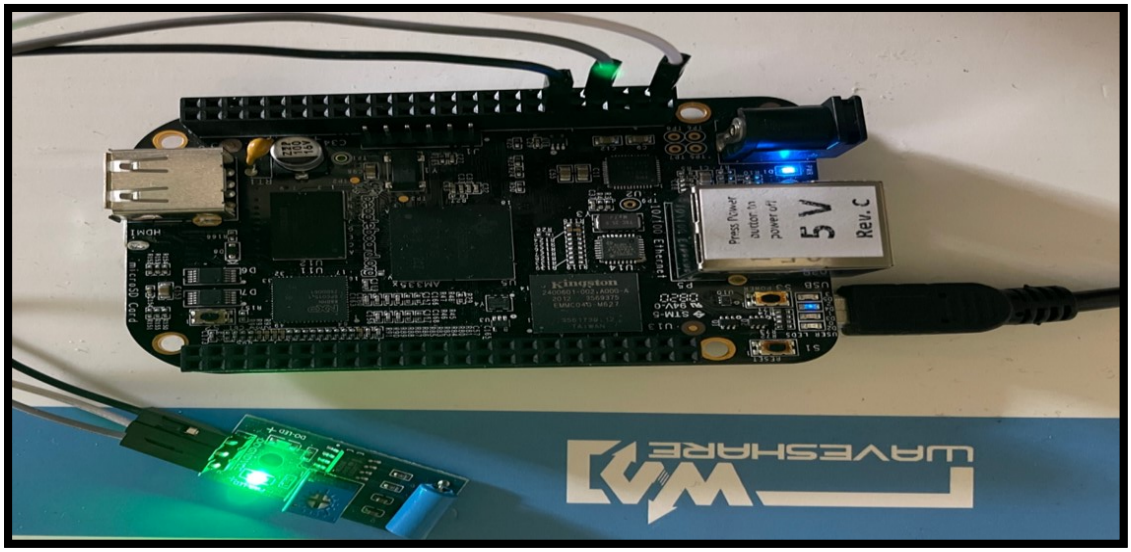

Figure 5.8: Interfacing of Vibration sensor

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 <iobb.h>  // Library to access GPIO
#include <time.h>  //Time Library
#include <stdio.h> // Standard IO library


int main(void)
{
  iolib_init();  //Initializing the iobb library
  iolib_setdir(9, 12, DigitalIn);  //Setting Pin as Input of a specific port

  while(1)   // Infinite Loop
  {
    if (is_high(9, 12)) // Checking if pin 12 of port 9 is high
    {
                printf("Vibration Detected \n");  // Print if Vibration Detected
           }

    else if (is_low(9, 12))  // Checking if pin 12 of port 9 is low
    {
                printf("No Vibration Detected \n");  // Print if No Vibration Detected
        }

        iolib_delay_ms(900);

  }
  iolib_free();
  return(0);
}
```

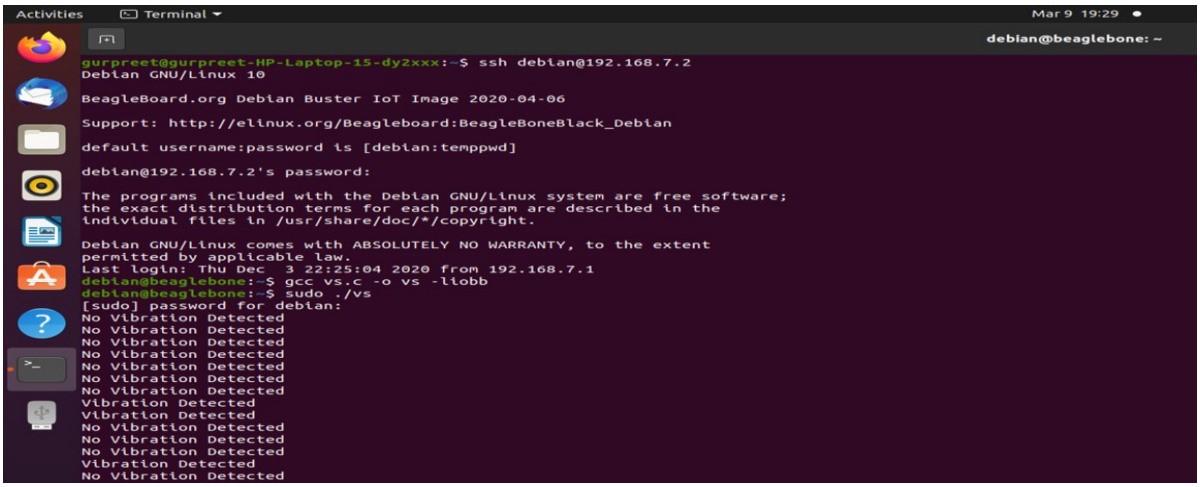

Figure 5.9: Output of the Vibration Sensor

- **Interfacing of RTC Module with Beaglebone Black**

Real time clock is connected using I2C communication protocol. RTC is connected directly to the beagle bone to determine the real time on screen as BBB does not have inbuilt RTC.

RTC DS3231	BEAGLEBONE BLACK
SCL	P9.19
SDA	P9.20
VCC	VCC
GND	GND

Figure 5.10: Interfacing of RTC Module with Beaglebone Black

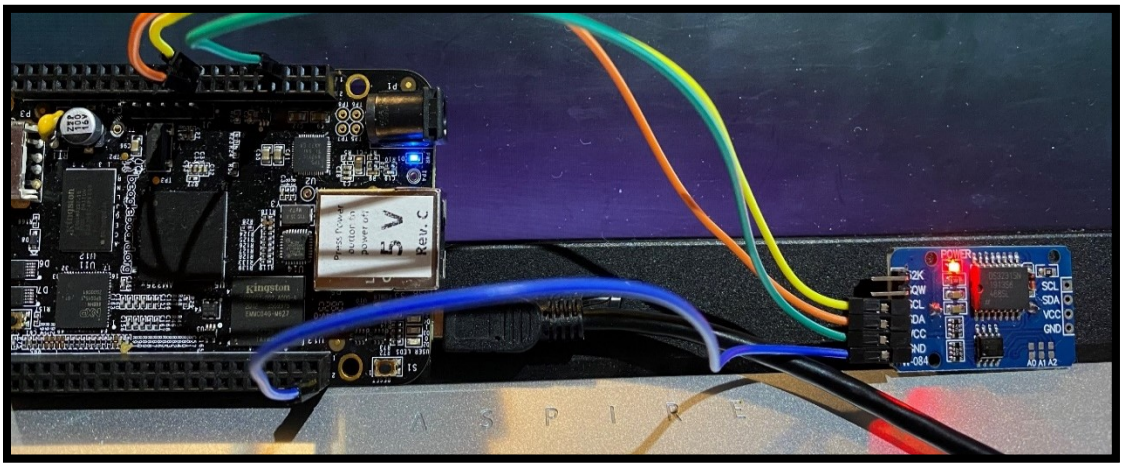

Figure 5.11: Connection of RTC module with Beaglebone Black

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:

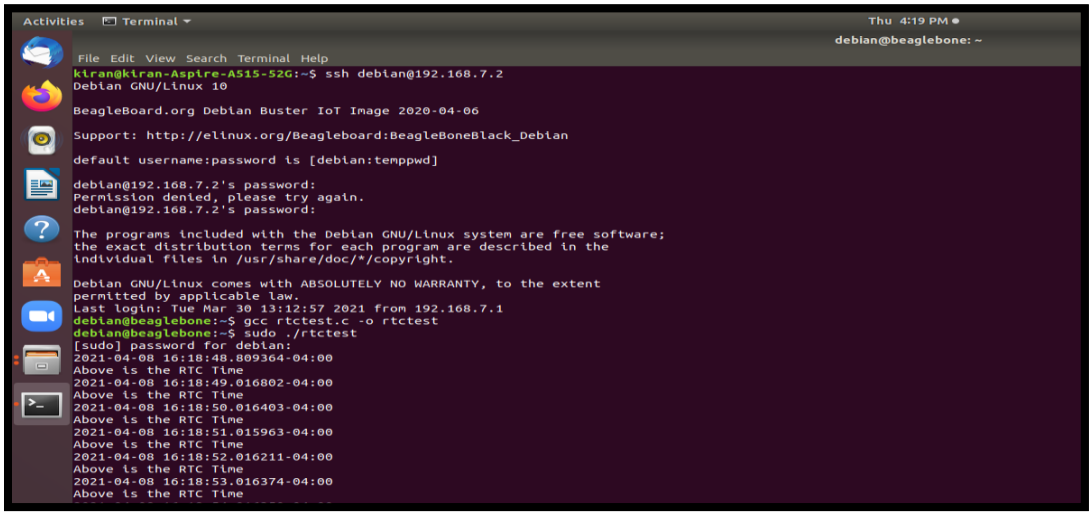

Figure 5.12: Output of the RTC Module

```
#include <stdio.h> // Standard IO library
#include <unistd.h> //defines miscellaneous symbolic constants and types, and d$
#include <stdlib.h> //Includes some Standard C Libraries used for trigerring Co$



int main(void)
{
        while(1)
        {
        system("hwclock -r -f /dev/rtc0"); // Ping the Web link using Command l$
        printf("Above is the RTC Time \n");
        }

        return(0);
}
```

- **Interfacing of Arduino Mega with Beaglebone Black**

  Arduino Mega is used as a slave as the fingerprint sensor and GSM Module is connected to it. Arduino mega is connected to the beaglebone using UART communication protocol.

BEAGLEBONE	ARDUINO
P9 11	19 RX1
P9 13	18 TX1
GND	GND

Figure 5.13: Interfacing of Arduino Mega with Beaglebone Black

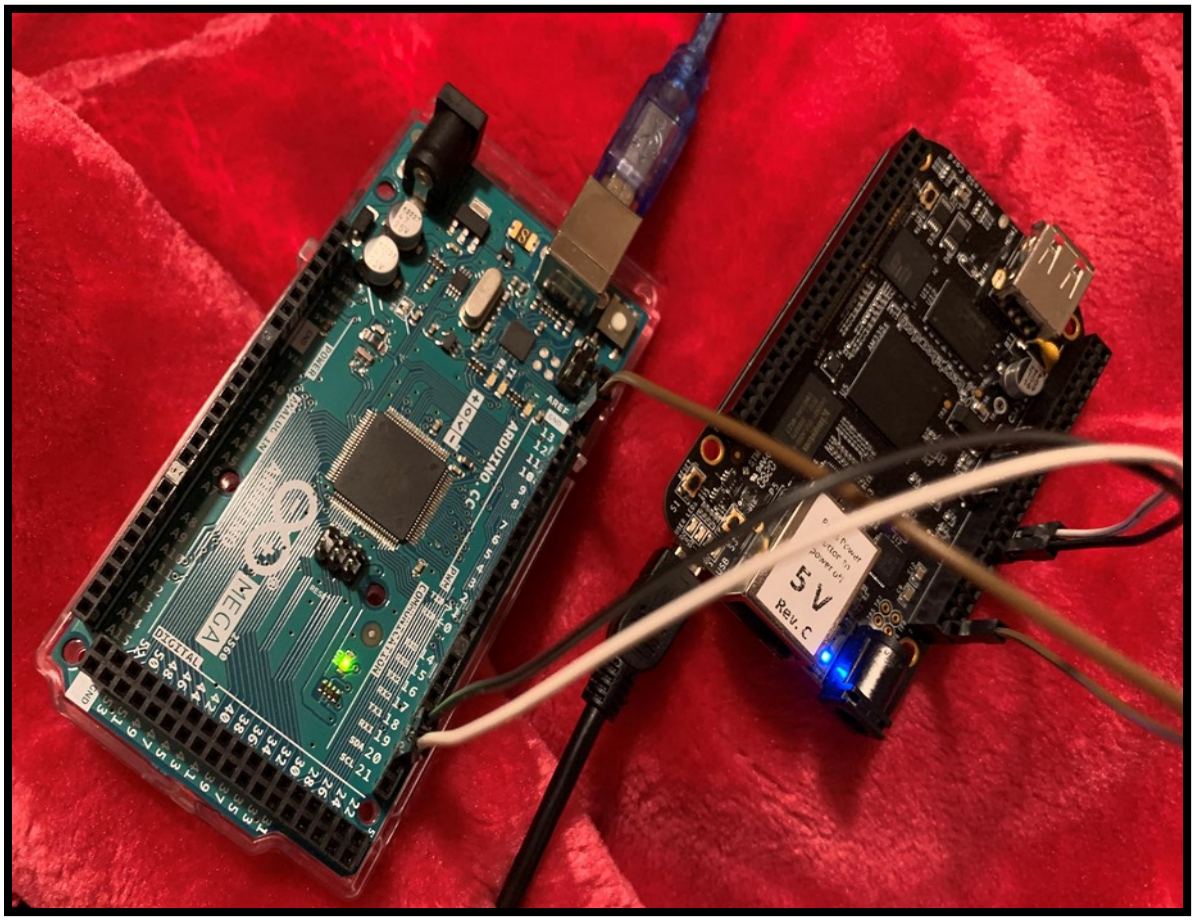

Figure 5.14: Connections of Arduino Mega with Beaglebone Black

- Enabling the UART port of the Beaglebone Black

  We need to follow few steps to accomplish this task:

  - Make a file under the name setTty.sh.
  - Command used at the terminal is:

    nano /usr/local/sbin/setTty.sh
  - In this file bash script is written to configure the ports on the beaglebone side.
  - After saving the file next step is to make this file executable that can be done by the following command on the terminal.
  - chmod 755 /usr/local/sbin/setTty.sh
  - Create a new file with the name testTty.sh.
  - Open the text editor with the sudo nano command.
  - nano /usr/local/sbin/testTty.sh.

- Similarly make this file also executable like the last one.
- chmod 755 /usr/local/sbin/testTty.sh.
- Create the service configuration file that makes the system run it on startup
- Command used : nano /lib/systemd/system/setTty.service.

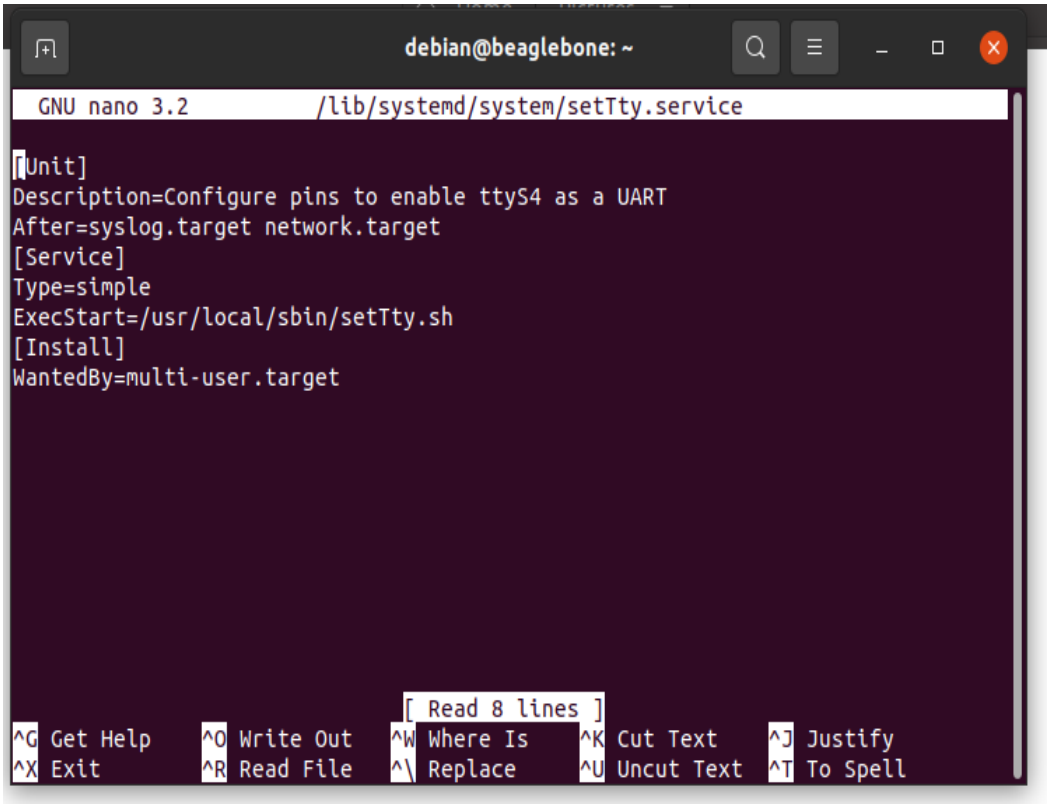

Figure 5.14(a): enabling the uart port for bbb

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:

CODE 1: Beaglebone side

```
#include <stdio.h> //Standard C input Output Library
#include <unistd.h> //defines miscellaneous symbolic constants and types, and declares
miscellaneous functions
#include <string.h> //C Library for various String Operations
#include <termios.h> // Contains the definitions used by the terminal I/O interfaces
#include <fcntl.h> // FIle control, Open, close
#include <time.h>
#define BAUDRATE B9600
 int main (void) // Main function
{
    int f1, i; // Variable integers
    char incomingbuffer[1000]; // declare a char array for receiving data
    char buf[20]; // A buffer char array to store temporary data

    size_t nbytes; //size_t is an unsigned integer data type used for storing size
    ssize_t outgoing_data; // //size_t is an signed integer data type used for storing size

       // /dev/ttyO4 is linked with UART4 Port of Beaglebone
    if ((f1 = open("/dev/ttyO4", O_RDWR | O_NOCTTY | O_NONBLOCK))<0){ // Try opening the
UART4 f1 in Read Write mod

        printf("UART: Failed to open the f1.\n"); //A message Print
        return 0;
    }
    else{
    printf("UART: Started\n"); //A message Print
       }
    struct termios options;
    /* set new port settings for canonical input processing */
    options.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; //  CLOCAL - It will
ignore Data Carrier Signal
       //CS8 = Select 8 data bits
       // CREAD = Enable Receiving Characters
       // CRTSCTS = Automaticall handle RTS (Request to Send) and CTS(Clear to Send)
    options.c_iflag = IGNPAR | ICRNL; // IGNPAR = IGNORE PARITY, ICRNL - Use CR & NL as
Break Line
    options.c_oflag = 0; //output mode flag
    options.c_lflag = ICANON; //Waits for complete line to come
    options.c_cc[VMIN]=2; // Minimum wait Time
    options.c_cc[VTIME]=20; //Maximum Wait Time
    tcflush(f1, TCIFLUSH); //Flush all the Input Data
    tcsetattr(f1,TCSANOW,&options);  // TCSANOW - Any change made should be applied
immediately
```

```
sleep(1); // Wait for a second
    while(1)
    {

        memset(buf, 0, sizeof(buf)); // Clear any old data in buffer
         strcpy(buf, "WE ARE GROUP 2\n"); // Copy a string in buf char array
         nbytes = strlen(buf); // Store size of buf array in nbytes
         outgoing_data = write(f1, buf, nbytes); // Sending message to Arduino Module
         printf("Sent Bytes: ");
         sleep(2);// Allowing time to make sure that ARduino has read the data
         printf("%d \n", outgoing_data);// print the bytes sent
        int incoming_data = 0;
        incoming_data = read(f1,&incomingbuffer,100); // Read the incoming Message from
Arduino Module
        //Read the f1 and store the data in incomingbuffer[, read 100 bytes max
        //Will only execute if newline or carriage return is received
        printf("\n\nBytes Received - %d",incoming_data); // Print how many bytes was
incomingbuffer[
        printf("\n");

        if(incoming_data > 1) //If no. of bytes are read is more than 1
        {
                for(i=0;i<incoming_data;i++) //a for loop to print data byte by byte
                {
                        printf("%c",incomingbuffer[i]); //print a byte of message from
Arduino Module
                }
                printf("\n");

             sleep(2);
        }


        }
    close(f1); //Close the f1 at last
}
```

CODE 2: Arduino side

```
void setup() {
  // initialize both serial ports:
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
  // read from port 1, send to port 0:
  if (Serial1.available()) {
    int inByte = Serial1.read();
    Serial.write(inByte);
  }

  // read from port 0, send to port 3:
  if (Serial.available())
  {
    received = 1;
    while(Serial.available())
    {
      int inByte = Serial.read();
      Serial1.write(inByte);
      delay(2);
    }
  }

  if(received == 1)
  {
    Serial.println("Data Sent to Beaglebone.");
    Serial1.println();
    received = 0;
  }

}
```

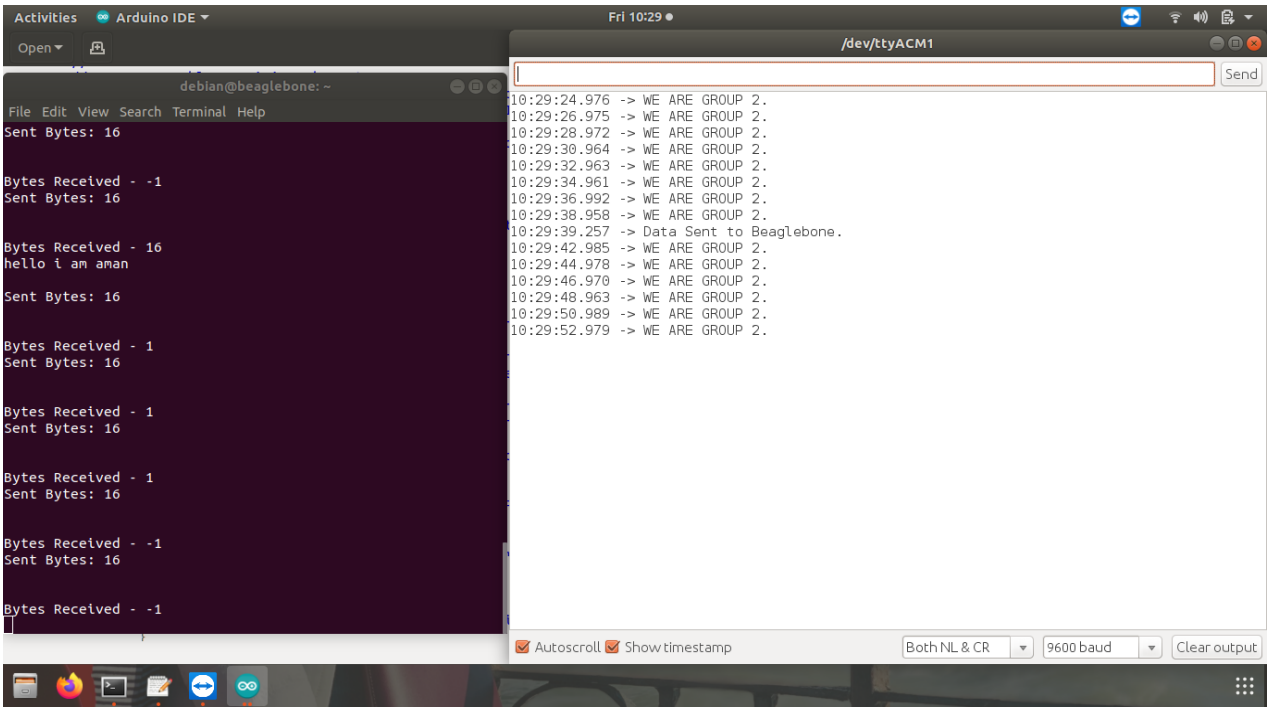

Figure 5.15: Output of BBB and Arduino Mega

- **Interfacing of ESP8266 Module with Beaglebone Black**

  ESP8266 is communicating with beagle bone by UART communication. We are using the ESP8266 in this project just for the interfacing of touch screen.

ESP8266 Module	Beaglebone Black
18 Tx	P9.26
19 Rx	P9.24
GND	GND

Figure 5.16: Interfacing of ESP8266 Module with Beaglebone Black

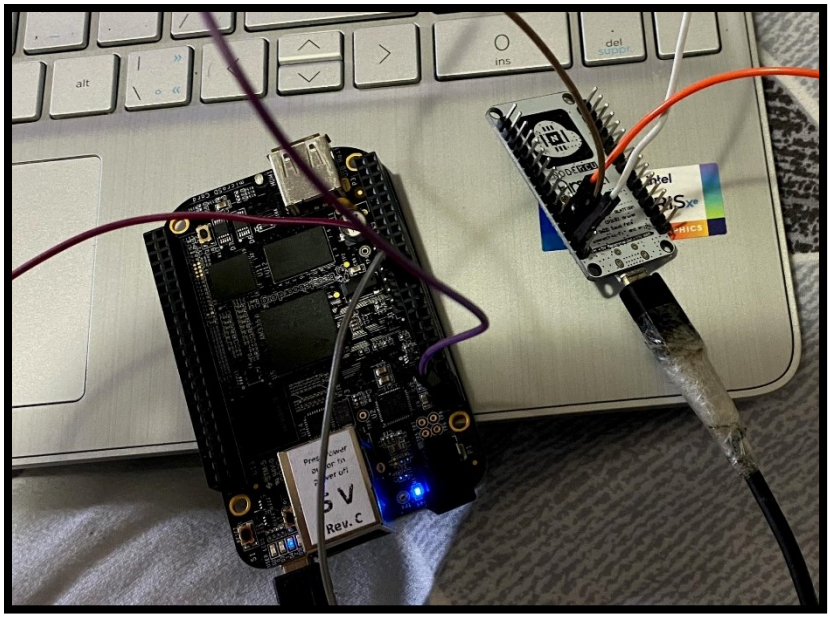

Figure 5.17: Connections of ESP8266 Module with Beaglebone Black

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:

CODE 1: Beaglebone Black Side

```
#include <stdio.h> //Standard C input Output Library
#include <unistd.h> //defines miscellaneous symbolic constants and types, and declares
miscellaneous functions
#include <string.h> //C Library for various String Operations
#include <termios.h> // Contains the definitions used by the terminal I/O interfaces
#include <fcntl.h> // FIle control, Open, close
#include <time.h>
#define BAUDRATE B9600
 int main (void) // Main function
{
    int f1, i; // Variable integers
    char incomingbuffer[1000]; // declare a char array for receiving data
    char buf[50]; // A buffer char array to store temporary data

    size_t nbytes; //size_t is an unsigned integer data type used for storing size
    ssize_t outgoing_data; // //size_t is an signed integer data type used for storing size

        // /dev/ttyO4 is linked with UART4 Port of Beaglebone
    if ((f1 = open("/dev/ttyO1", O_RDWR | O_NOCTTY | O_NONBLOCK))<0){ // Try opening the
UART4 f1 in Read Write mod
        printf("UART: Failed to open the f1.\n"); //A message Print
        return 0;
    }
    else{
    printf("UART 1: Started\n"); //A message Print
        }
    struct termios options;
    /* set new port settings for canonical input processing */
    options.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; //  CLOCAL - It will
ignore Data Carrier Signal
        //CS8 = Select 8 data bits
        // CREAD = Enable Receiving Characters
        // CRTSCTS = Automaticall handle RTS (Request to Send) and CTS(Clear to Send)
    options.c_iflag = IGNPAR | ICRNL; // IGNPAR = IGNORE PARITY, ICRNL - Use CR & NL as
Break Line
    options.c_oflag = 0; //output mode flag
    options.c_lflag = ICANON; //Waits for complete line to come
    options.c_cc[VMIN]=2; // Minimum wait Time
    options.c_cc[VTIME]=20; //Maximum Wait Time
    tcflush(f1, TCIFLUSH); //Flush all the Input Data
    tcsetattr(f1,TCSANOW,&options);  // TCSANOW - Any change made should be applied
immediately
```

```
  while(1)
 {

        memset(buf, 0, sizeof(buf)); // Clear any old data in buffer
        strcpy(buf, "We are group 2 performimg ESP8266 Test\n"); // Copy a string in buf
char array
        nbytes = strlen(buf); // Store size of buf array in nbytes
        outgoing_data = write(f1, buf, nbytes); // Sending message to Arduino Module
        printf("Sent Bytes: ");
        sleep(2);// Allowing time to make sure that ARduino has read the data
        printf("%d \n", outgoing_data);// print the bytes sent
        int incoming_data = 0;
        incoming_data = read(f1,&incomingbuffer,100); // Read the incoming Message from
Arduino Module
        //Read the f1 and store the data in incomingbuffer[, read 100 bytes max
        //Will only execute if newline or carriage return is received
        printf("\n\nBytes Received from ESP8266 - %d",incoming_data); // Print how many
bytes was incomingbuffer[
        printf("\n");

        if(incoming_data > 1) //If no. of bytes are read is more than 1
        {
                for(i=0;i<incoming_data;i++) //a for loop to print data byte by byte
                {
                        printf("%c",incomingbuffer[i]); //print a byte of message from
Arduino Module
                }
                printf("\n");

            sleep(2);
        }


        }
    close(f1); //Close the f1 at last
}
```

Code 2: ESP8266 side

```
String a = "";
void setup() {
  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:

  if(Serial.available())
  {
    a = Serial.readString();
    delay(1000);
    Serial.print(a);
    Serial.println(" - ESP8266");

  }
```

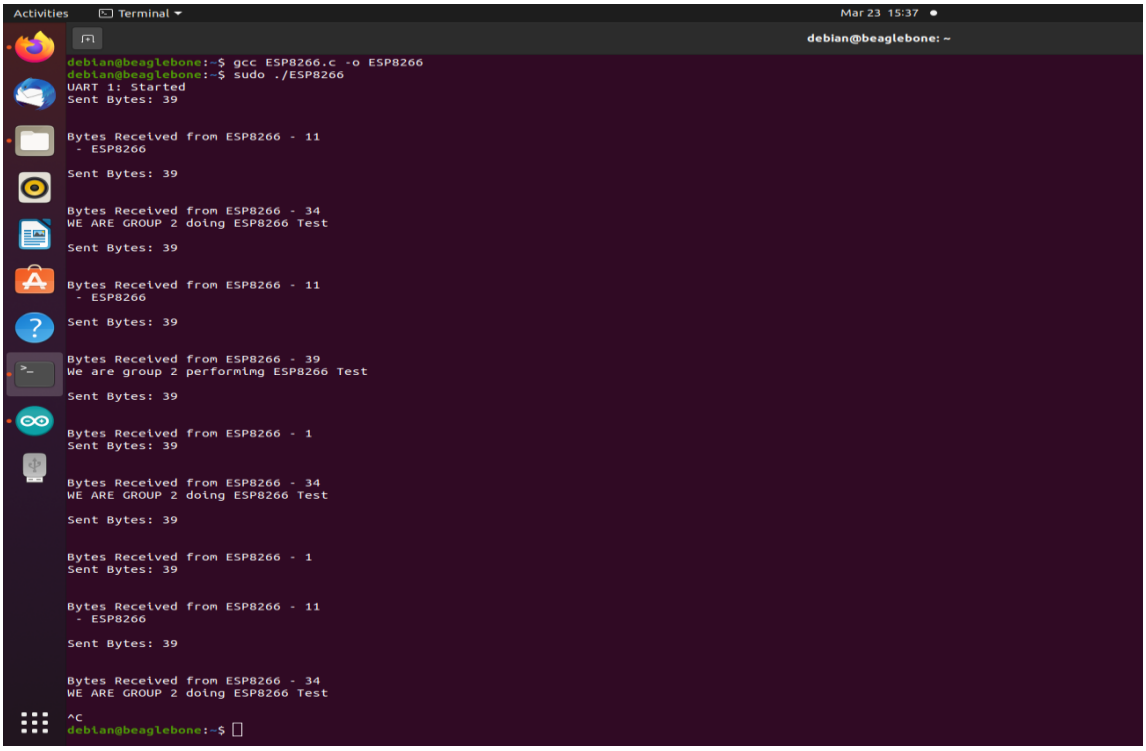

Figure 5.18: Output of the ESP8266 Module

Interfacing with Arduino Mega

- **Interfacing of Arduino Mega with GSM SIM900 Module**

  To interface the GSM (Global System for Mobile Communication) with the Arduino Mega. The coding will be done to it compatible with the mentioned objective. The interfacing is done to send the SMS (Short Message Services) when the locker is opened.

  - Before interfacing GSM module, it's important to understand the working of GSM module.
  - Initially, the power adapter is required to power up the GSM module. Next, we need to check with the LED status. If there is no LED status, then the perpendicular power button should be hold for 2 seconds approximately.
  - PWR: This LED is connected to the shield's power supply line. If it is "ON" then the shield is receiving power.
  - Status: It indicates the working status of SIM900. If it is "ON" the chip is in working mode.
  - Netlight: It indicates the status of cellular network. It will blink at various state it's in.
    - OFF: the SIM900 is not working.
    - 64ms ON, 800ms OFF: The SIM900 chip is running but not register to the cellular network.
    - 64ms ON, 3seconds OFF: The SIM900 chip is registered to the cellular network and can send/receive voice and SMS.
    - 64ms ON, 300ms OFF: The GPRS data connection you requested is active.
  - When supplying power externally through an adapter make sure that you have made the switch on the external mode.

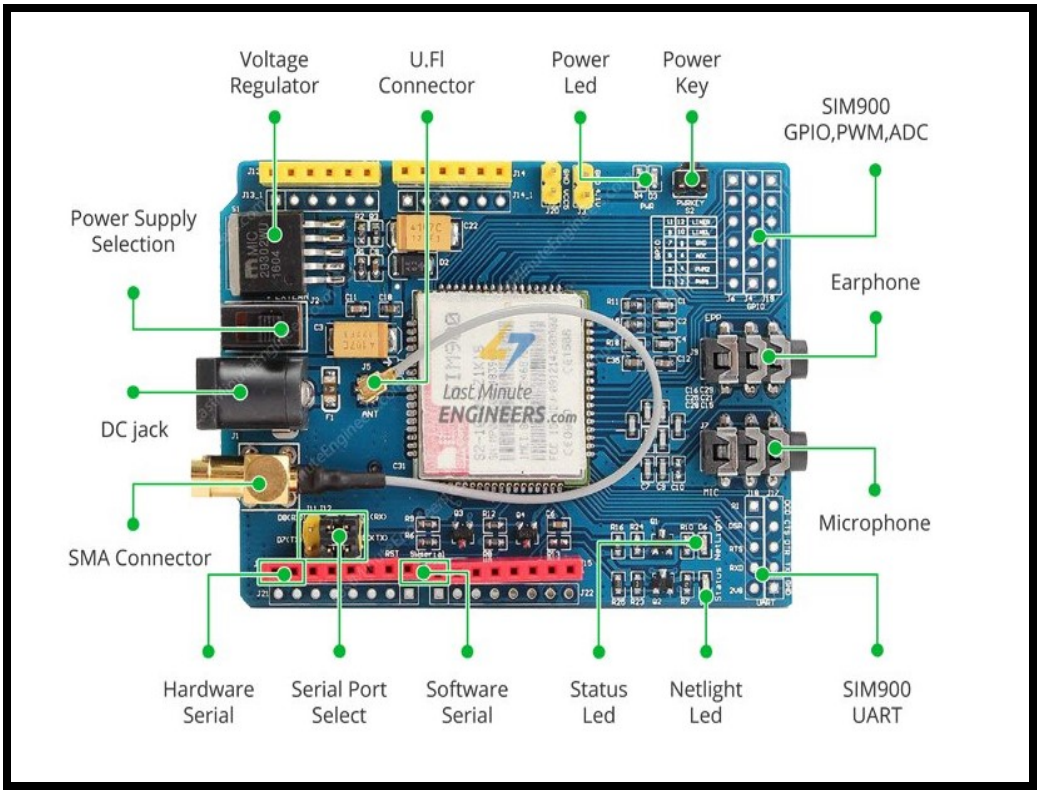

Figure 5.19: GSM SIM900 Module

➢ **Getting Started with GSM SIM900 Module**

- STEP 1: Insert the SIM card into the SIM card holder – make sure you've read the preliminary steps in the previous section.
- STEP 2: Make sure the antenna is well connected.
- STEP 3: On the serial port select, make sure the jumper cap is connected as shown in figure below to use software serial.
- STEP 4: Power the shield using an external 5V power supply. Make sure you select the external power source with the toggle switch next to the DC jack.
- STEP 5: To power up/down the shield press the power key for about 2 seconds.
- STEP6: Then, the Status LED will light up and the NetLight LED will blink every 800 ms until it finds the network. When it finds the network, the NetLight LED will start blinking every three seconds.

ARDUINO MEGA	GSM MODULE
PIN 15	SIM Tx
PIN 14	SIM Rx
GND	GND

Figure 5.20: Interfacing of Arduino Mega with GSM SIM900 Module

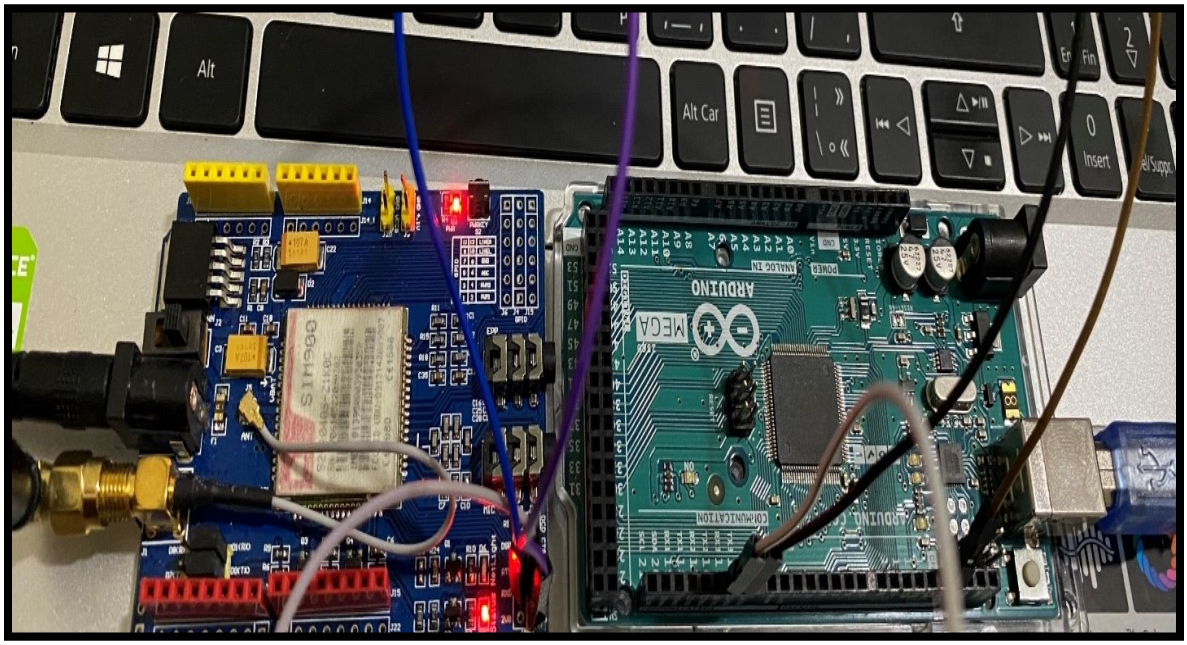

Figure 5.21: Connections of Arduino Mega with GSM SIM900 Module

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 <SoftwareSerial.h>
//
//// Configure software serial port
```

```
//SoftwareSerial GSMS(7, 8);  // RX, TX

#define GSMS Serial3
#define DEBUG 1
#define message "TEST SMS . IT's WORKING"

String number1 = "+14372478350";

String current_num;
String check;

void setup() {

  // Open serial communications and wait for port to open:
  Serial.begin(19200);
  GSMS.begin(19200);

  Serial.println("STARTING");

  delay(20000);
  Serial.println("STARTED");

  Serial.println("Initializing GSM");
  delay(5000); // Boot Up time for GSM

  Serial.println("STARTED\n\n");

  Serial.print("Number 1: ");
  Serial.println(number1);

  Serial.println("\n\n\n");

}

void loop()
{

  current_num = number1;
  sendSMS();
  delay(10000);

}


/////////////// Function to Send SMS ////////////////////////

int sendSMS()
{
```

```
  Serial.print("Trying to send SMS to : ");
  Serial.println(current_num);
  delay(500);

  GSMS.println("AT+CMGS=\""+current_num+"\"");
  delay(100);
  check = GSMS.readString();

  GSMS.print(message); //text content
  delay(100);
  check += GSMS.readString();

  GSMS.write(26);
  delay(100);
  check += GSMS.readString();

  if(DEBUG) Serial.println(check);

  if(DEBUG)
  {
    Serial.print("Check Data: ");
    Serial.println(check);
  }

  if(check.indexOf("OK") != -1 and check.indexOf("ERROR") == -1)
  {
    if(DEBUG) Serial.println("SMS sent successfully");
    return 1;
  }

  if(DEBUG) Serial.println("SMS sending Done");

  delay(5000);
  GSMS.println("ATD + +14372478350;");

  delay(30000);
  // AT command to hang up
  GSMS.println("ATH"); // hang up

  return 0;
}
```

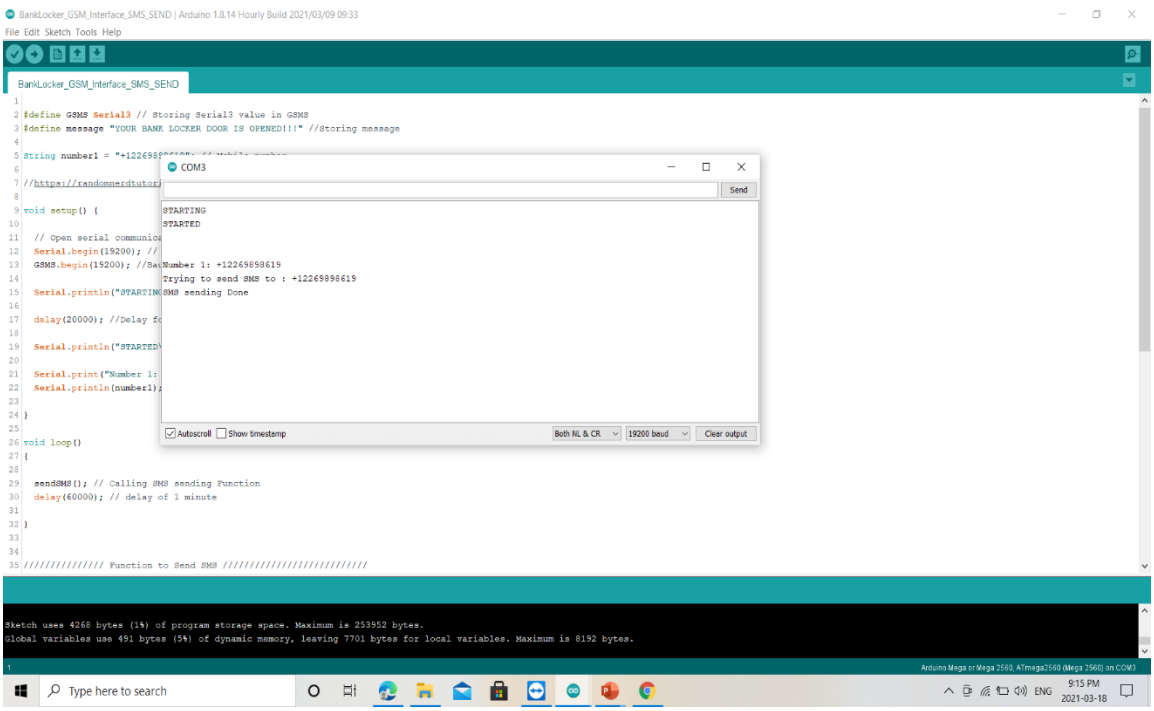

Figure 5.22: Output of GSM SIM900 Module

- **Interfacing of Fingerprint Sensor with Arduino Mega**

As we are working on the Project "IOT BASED BANK LOCKER SECURITY SYSTEM", here in this project, fingerprint sensor is used as a "Second Layer" security for the locker. The user, need to scan the required finger to open the locker. As per the software coding, three chances will be given to the user. If the user is somehow unable to scan the core finger, then the user jumps to the first layer of security: touchscreen.

Fingerprint Sensor	Arduino mega
GND	GND
VCC	VCC
Rx (WHITE WIRE)	PIN 16 (TX2)
Tx (YELLOW WIRE)	PIN 17 (RX2)

Figure 5.23: Interfacing of Arduino Mega with Fingerprint Sensor

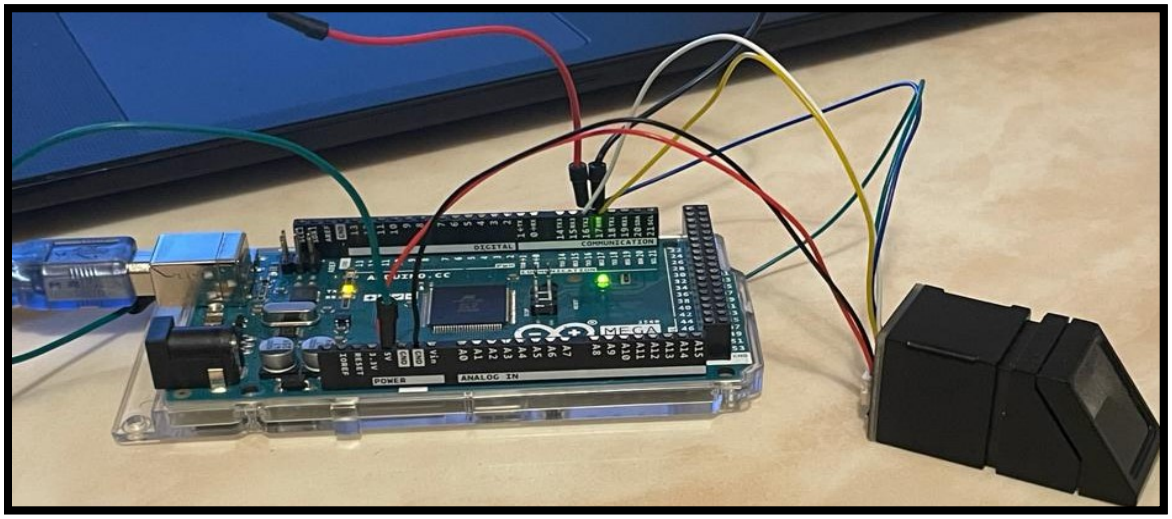

Figure 5.24: Connections of Arduino Mega with finger print

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 <Adafruit_Fingerprint.h> // Adafruit Fingerprint Sensor Library

#define mySerial Serial2 //Serial Port where Fingerprint is connected

Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);// Initialize
Fingerprint Port

// Below 3 values needs to be updated manually
int pcount = 3; // Total Enrolled person
int ids[3] =          {1,      2,        3}; //Enrolled Fingerprint Id's of
individual person in Serial order
String names[3] = {"Simran", "FLORA", "SAINI"}; // Name of Individuals in
Serial order


uint8_t id = 0; //To store Finger ID
int mode = 0; //Mode to decide if Fingerprint Sensor is in enroll mode or
detect mode
int detected = 0;
void setup()
{
  Serial.begin(9600);//Initialize Serial Communication
  delay(100);
  Serial.println("\n\nAdafruit Fingerprint sensor enrollment");

  // set the data rate for the sensor serial port
  finger.begin(57600);//Initialize Serial Communication with Fingerprint
Module

  if (finger.verifyPassword()) {
    Serial.println("Found fingerprint sensor!"); //FIngerprint Found
  } else {
    Serial.println("Did not find fingerprint sensor :("); //Fingerprint not
Found
    while (1) { delay(1); } //Wait if Fingerprint not found
  }

  finger.getTemplateCount(); //Get how many fingerprints are enrolled till
now

  if (finger.templateCount == 0) { //If Template Count is 0
    Serial.print("Sensor doesn't contain any fingerprint data. Please run
the 'enroll' example.");
  }
  else {
      Serial.println("Waiting for valid finger...");
      Serial.print("Sensor contains ");
      Serial.print(finger.templateCount);
      Serial.println(" templates");
```

```
  }

  delay(1000); // Wait for a second

  Serial.println("Enter 'enroll' for enrollment or 'detect' for
detection."); // A message to USer

}

uint8_t readnumber(void) {
  uint8_t num = 0;

  while (num == 0) {
    while (! Serial.available());
    String input = Serial.readString();
    if(input.indexOf("enroll") != -1)
    {
      Serial.println("Enroll Mode Activated");
      mode = 1;
    }
    else if(input.indexOf("detect") != -1)
    {
      Serial.println("Detect Mode Activated");
      mode = 0;
    }
    else if(input.indexOf("delete") != -1)
    {
      Serial.println("Deleting all");
      mode = 2;
    }
    else
    {
      num = input.toInt();
    }
  }
  return num;
}

void loop()                     // run over and over again
{
  if (Serial.available())
  {
    String input = Serial.readString();

    if(input.indexOf("enroll") != -1)
    {
      Serial.println("Enroll Mode Activated");
      mode = 1;
    }
```

```
    else if(input.indexOf("detect") != -1)
    {
      Serial.println("Detect Mode Activated");
      mode = 0;
    }
    else if(input.indexOf("delete") != -1)
    {
      Serial.println("Deleting all");
      mode = 2;
    }
  }

  if(mode == 0) //If Mode is 0 - means Detection mode
  {
    getFingerprintID();
    if(detected > 0)
    {
    for (int i = 0; i < pcount; i++)
    {
      if(ids[i] == detected)
      {
        Serial.print("Person Name is: ");
        Serial.println(names[i]);
        break;
      }
    }
    }
    detected = 0;
    delay(200);            //don't ned to run this at full speed.
  }


  if(mode == 1) // If mode is 1 - means enrollment mode
  {
    Serial.println("Ready to enroll a fingerprint!");
    Serial.println("Please type in the ID # (from 1 to 127) you want to
save this finger as...");
    id = readnumber();
    if (id == 0) {// ID #0 not allowed, try again!
       return;
    }
    Serial.print("Enrolling ID #");
    Serial.println(id);

    while (!  getFingerprintEnroll() );

  }

  if (mode == 2)
```

```
  {
    finger.emptyDatabase();
    Serial.println("Now database is empty :)");

    delay(1000);
    mode = 1;
  }

}

uint8_t getFingerprintEnroll() {
  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #");
Serial.println(id);
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.println(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      break;
    default:
      Serial.println("Unknown error");
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
```

```
      //Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      //Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  Serial.println("Remove finger");
  delay(2000);
  p = 0;
  while (p != FINGERPRINT_NOFINGER) {
    p = finger.getImage();
  }
  Serial.print("ID "); Serial.println(id);
  p = -1;
  Serial.println("Place same finger again");
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.print(".");
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      break;
    default:
      Serial.println("Unknown error");
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(2);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
```

```
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      //Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      //Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  // OK converted!
  Serial.print("Creating model for #");  Serial.println(id);

  p = finger.createModel();
  if (p == FINGERPRINT_OK) {
    Serial.println("Prints matched!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_ENROLLMISMATCH) {
    Serial.println("Fingerprints did not match");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  Serial.print("ID "); Serial.println(id);
  p = finger.storeModel(id);
  if (p == FINGERPRINT_OK) {
    Serial.println("Stored!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_BADLOCATION) {
    Serial.println("Could not store in that location");
    return p;
  } else if (p == FINGERPRINT_FLASHERR) {
    Serial.println("Error writing to flash");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }
```

```
  return true;
}

uint8_t getFingerprintID() {
  uint8_t p = finger.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.println("No finger detected");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  // OK success!

  p = finger.image2Tz();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      break;
    case FINGERPRINT_IMAGEMESS:
      Serial.println("Image too messy");
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      //Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      //Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }

  // OK converted!
```

```
  p = finger.fingerSearch();
  if (p == FINGERPRINT_OK) {
    Serial.println("Found a print match!");
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    //Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_NOTFOUND) {
    Serial.println("Did not find a match");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }

  // found a match!
  Serial.print("Found ID #"); Serial.print(finger.fingerID);
  Serial.print(" with confidence of "); Serial.println(finger.confidence);

  detected = finger.fingerID;
  return finger.fingerID;
}

// returns -1 if failed, otherwise returns ID #
int getFingerprintIDez() {
  uint8_t p = finger.getImage();
  if (p != FINGERPRINT_OK)  return -1;

  p = finger.image2Tz();
  if (p != FINGERPRINT_OK)  return -1;

  p = finger.fingerFastSearch();
  if (p != FINGERPRINT_OK)  return -1;

  // found a match!
  Serial.print("Found ID #"); Serial.print(finger.fingerID);
  Serial.print(" with confidence of "); Serial.println(finger.confidence);
  return finger.fingerID;
}
```

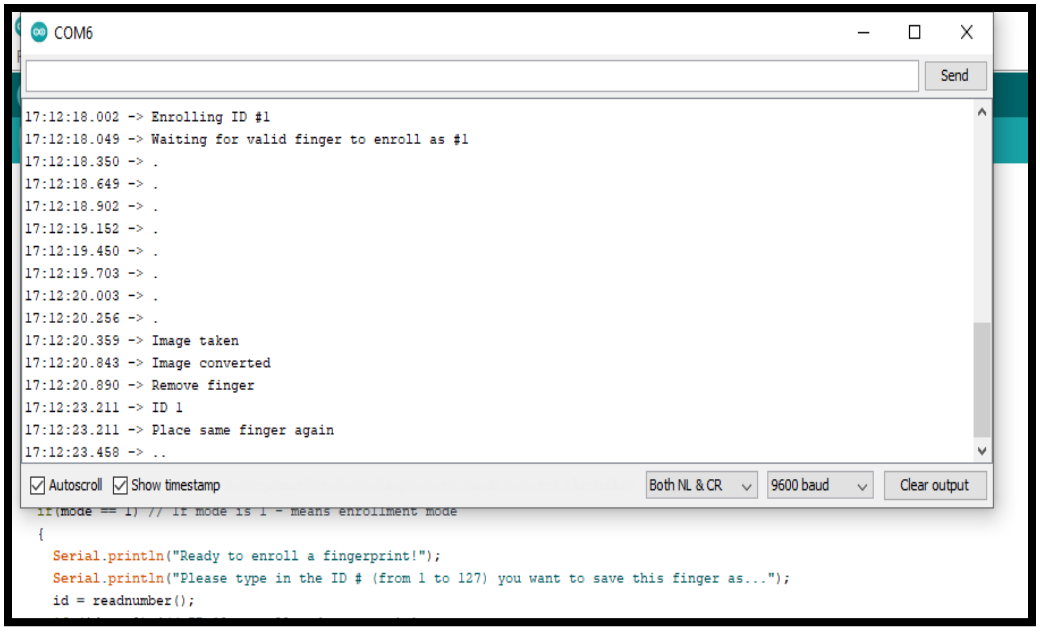

Figure 5.25: Output of the Enrolled Fingerprint

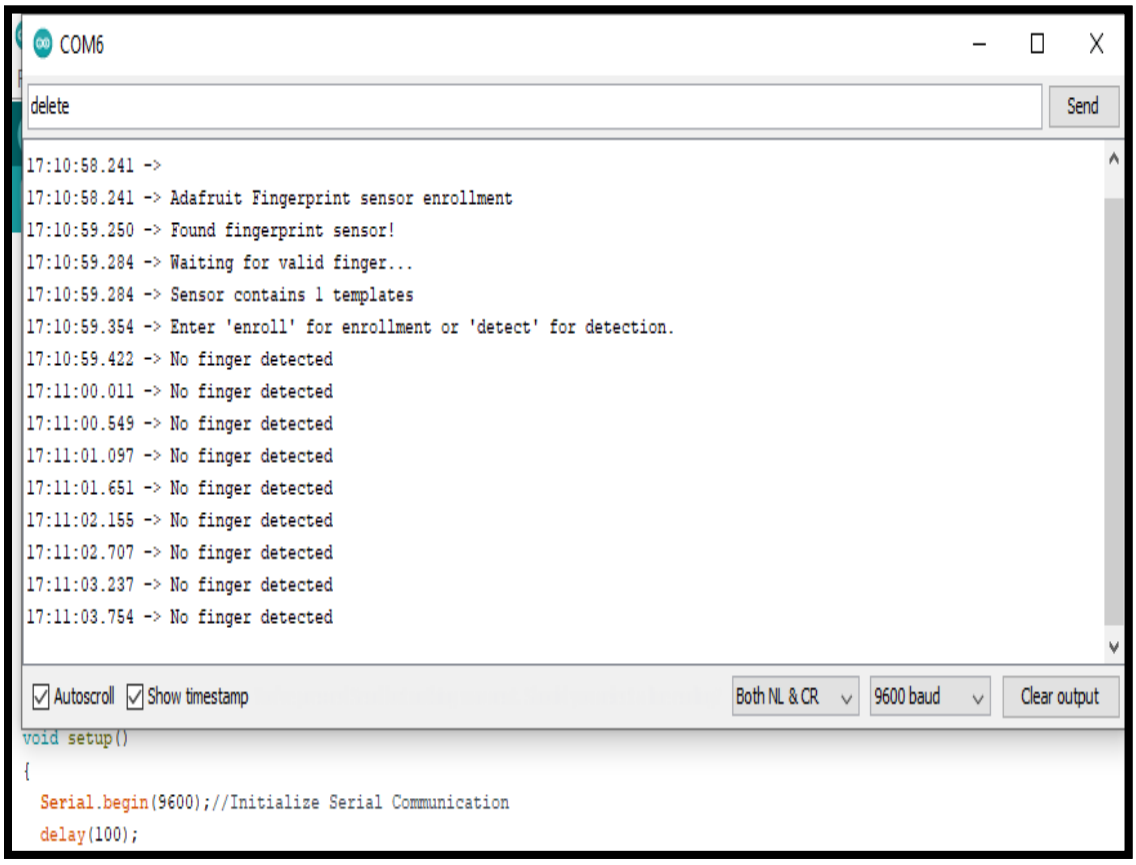

Figure 5.26: Output of the detected Fingerprint

Interfacing with ESP8266 Module

- **Interfacing of ESP8266 with ESP32 CAM module**

  ESP32 CAM module is being used for two tasks in this project and these are:

  - To provide the Wi-Fi to the system so that Internet access is possible.
  - To capture the picture of the unauthorized person who tries to access the locker.
  - Mode of communication is GPIO and how these two devices are connected can be shown below in figure 5.25.

ESP 8266	ESP 32 CAM MODULE
Vin	5V
GND	GND
AD0	I016

Figure 5.27: Interfacing of ESP8266 with ESP32 CAM Module

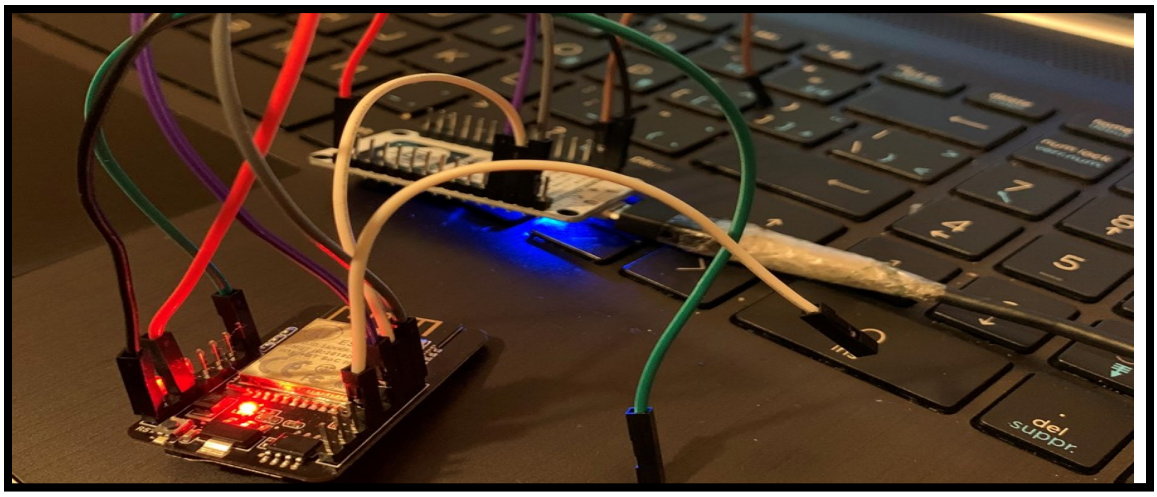

Figure 5.28: Connections of ESP8266 WITH ESP32 CAM Module

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:

CODE 1: Coding for ESP8266 Module

```
#define pin A0
void setup() {
pinMode(pin,OUTPUT);
Serial.begin(9600);
}

void loop() {
digitalWrite(pin,HIGH);
delay(2000);
digitalWrite(pin,LOW);
delay(2000);
}
```

CODE 2: Coding for ESP32 CAM Module

```
#include "WiFi.h"
#define pin 16
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
void setup() {

  Serial.begin(115200);
 pinMode(pin,INPUT_PULLUP);
  int numberOfNetworks = WiFi.scanNetworks();

  for(int i =0; i<numberOfNetworks; i++){

      Serial.print("Network name: ");
      Serial.println(WiFi.SSID(i));
      Serial.print("Signal strength: ");
      Serial.println(WiFi.RSSI(i));
      Serial.println("-----------------------");

  }
    WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi ..");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
    delay(1000);
  }
  Serial.println("connected.");
  Serial.println(WiFi.localIP());
}



void loop() {
  int DATA = digitalRead(pin);
  if(DATA == 1){
    Serial.println("GOT SOME DATA FROM THE ESP8266:");
  }
  else{
    Serial.println("waiting for data:");
  }
}
```

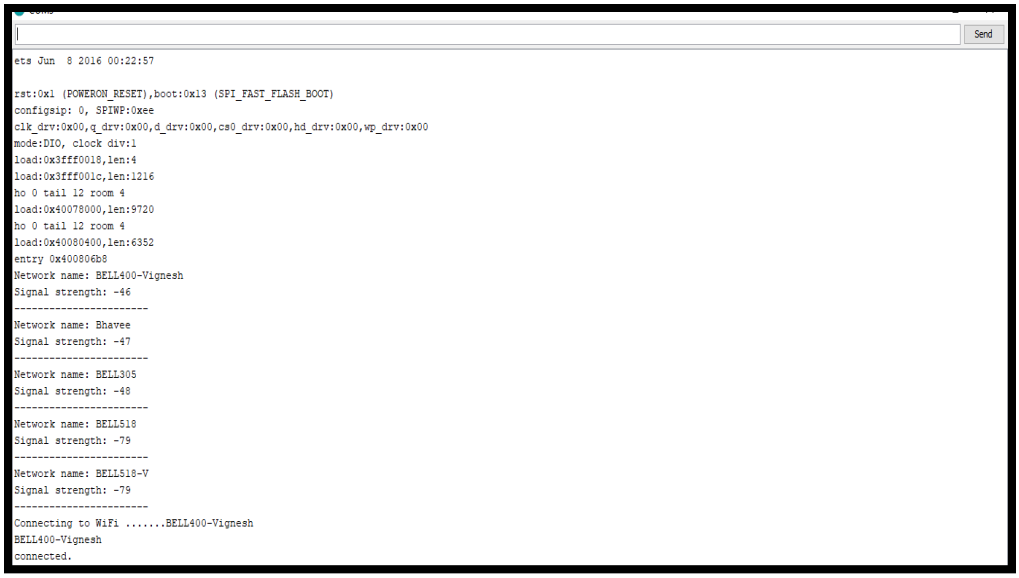

Figure 5.27: Output of ESP32 for Wi-Fi Connectivity

CODE 3: Coding for taking and sending picture to the Email.

```
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>
#include <EEPROM.h>
#include "SD_MMC.h"
const char* ssid = "gaetonde";
const char* password = "Trivedii";

#define emailSenderAccount    "group27572@gmail.com"
#define emailSenderPassword   "asdf@1234"
#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ESP32-CAM Photo Captured"
#define emailRecipient        "gillpreet0996@gmail.com"

#define CAMERA_MODEL_AI_THINKER

#if defined(CAMERA_MODEL_AI_THINKER)
  #define PWDN_GPIO_NUM     32
  #define RESET_GPIO_NUM    -1
  #define XCLK_GPIO_NUM      0
  #define SIOD_GPIO_NUM     26
  #define SIOC_GPIO_NUM     27
```

```
  #define Y9_GPIO_NUM       35
  #define Y8_GPIO_NUM       34
  #define Y7_GPIO_NUM       39
  #define Y6_GPIO_NUM       36
  #define Y5_GPIO_NUM       21
  #define Y4_GPIO_NUM       19
  #define Y3_GPIO_NUM       18
  #define Y2_GPIO_NUM        5
  #define VSYNC_GPIO_NUM    25
  #define HREF_GPIO_NUM     23
  #define PCLK_GPIO_NUM     22
#else
  #error "Camera model not selected"
#endif

SMTPData smtpData;

#define FILE_PHOTO "/photo.jpg"
int button = 12;
#define EEPROM_SIZE 1
int pictureNumber = 0;
bool takeNewPhoto = true;

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  Serial.begin(115200);
  Serial.println();
pinMode(button,INPUT_PULLUP);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    ESP.restart();
  }
  else {
    delay(500);
    Serial.println("SPIFFS mounted successfully");
  }

  Serial.print("IP Address: http://");
```

```
  Serial.println(WiFi.localIP());

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
////////////////////////////////////////////////////////////////////////////
//////////////////////////
  if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
  }

  uint8_t cardType = SD_MMC.cardType();
  if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
```

```
    return;
  }
////////////////////////////////////////////////////////////////////////
//////////////////////

  capturePhotoSaveSpiffs();
}

void loop() {
  int a = digitalRead(button);
  Serial.println(a);


    capturePhotoSaveSpiffs();
      SEND_TO_MAIL();
      delay(8000);
      SPIFFS.remove(FILE_PHOTO);
            delay(2000);
}

bool checkPhoto( fs::FS &fs ) {
  File f_pic = fs.open( FILE_PHOTO );
  unsigned int pic_sz = f_pic.size();
  return ( pic_sz > 100 );
}


// Capture Photo and Save it to SPIFFS
void capturePhotoSaveSpiffs( void ) {
  camera_fb_t * fb = NULL; // pointer
  bool ok = 0;
    takeNewPhoto = true;

  do {
    Serial.println("Taking a photo...");

    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      return;
    }

     File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

    if (!file) {
      Serial.println("Failed to open file in writing mode");
    }
    else {
      file.write(fb->buf, fb->len);
```

```
      Serial.print("The picture has been saved in ");
      Serial.print(FILE_PHOTO);
      Serial.print(" - Size: ");
      Serial.print(file.size());
      Serial.println(" bytes");
     Serial.printf("Saved file to path: %s\n", FILE_PHOTO);
    }
    file.close();

    esp_camera_fb_return(fb);

    ok = checkPhoto(SPIFFS);
  } while ( !ok );

}


void STORE_SDCARD(){
  camera_fb_t * fb = NULL;
  fb = esp_camera_fb_get();
  if(!fb) {
    Serial.println("Camera capture failed");
    return;
  }
  EEPROM.begin(EEPROM_SIZE);
  pictureNumber = EEPROM.read(0) + 1;

  // Path where new picture will be saved in SD Card
  String path = "/picture" + String(pictureNumber) +".jpg";

  fs::FS &fs = SD_MMC;
  Serial.printf("Picture file name: %s\n", path.c_str());

  File file2 = fs.open(path.c_str(), FILE_WRITE);
  if(!file2){
    Serial.println("Failed to open file in writing mode");
  }
  else {
    file2.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
    EEPROM.write(0, pictureNumber);
    EEPROM.commit();
  }
 // SPIFFS.remove(FILE_PHOTO);
  file2.close();
  esp_camera_fb_return(fb);

  // Turns off the ESP32-CAM white on-board LED (flash) connected to GPIO 4
```

```
  delay(2000);
  Serial.println("Going to sleep now");
  delay(2000);
  Serial.println("This will never be printed");
}


void  SEND_TO_MAIL( void ) {
  // Preparing email
  Serial.println("Sending email...");
  // Set the SMTP Server Email host, port, account and password
  smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);

  // Set the sender name and Email
  smtpData.setSender("ESP32-CAM", emailSenderAccount);

  smtpData.setPriority("High");

  // Set the subject
  smtpData.setSubject(emailSubject);

  // Set the email message in HTML format
  smtpData.setMessage("<h2>Photo captured with ESP32-CAM and attached in
this email.</h2>", true);
  // Set the email message in text format
  //smtpData.setMessage("Photo captured with ESP32-CAM and attached in this
email.", false);

  smtpData.addRecipient(emailRecipient);
  //smtpData.addRecipient(emailRecipient2);

  // Add attach files from SPIFFS
  smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
  smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

  smtpData.setSendCallback(sendCallback);

  // Start sending Email, can be set callback function to track the status
  if (!MailClient.sendMail(smtpData))
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());

  smtpData.empty();
}

// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {
  //Print the current status
```

```
  Serial.println(msg.info());
}
```

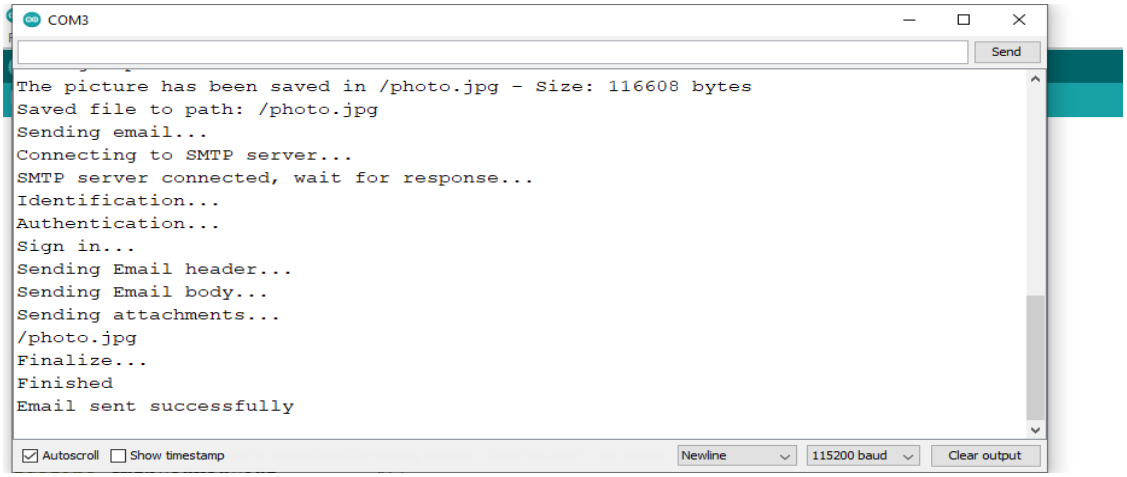

Figure 5.29: Output of ESP32 CAM for sending and taking pictures

- **Interfacing of ESP8266 with Touchscreen**

ESP8266, is used in this project only as a bridge between the touchscreen and Beagle bone black. ESP8266 is being interfaced with the 2.8" TFT touch screen using the SPI communication protocol.

ESP8266	TOUCH SCREEN
3.3V	VCC
GND	GND
D8 (15)	CS
RST	RESET
D4 (2)	DC
D7 (13)	SDI
D5 (14)	SCK
3.3V	LED
D6 (12)	SDD

Figure 5.30: Interfacing Touchscreen with ESP8266 Module

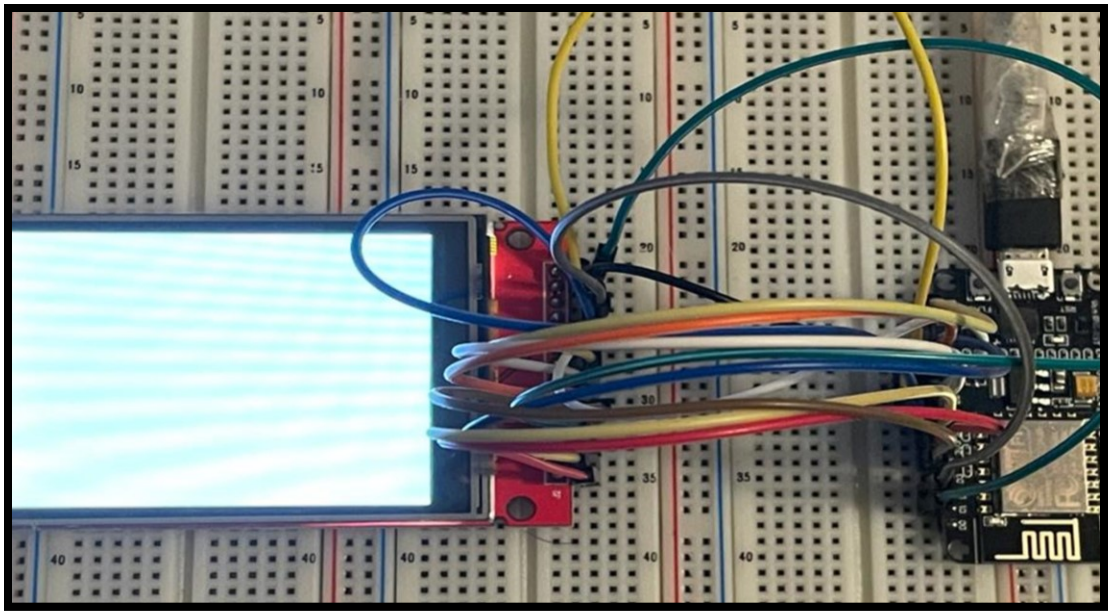

Figure 5.31: Connections of Touchscreen with ESP8266 Module

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:

Code 1: Displaying Text on Touchscreen

```
#include <Adafruit_GFX.h>       // include Adafruit graphics library
#include <Adafruit_ILI9341.h>   // include Adafruit ILI9341 TFT library
#define TFT_CS    D2     // TFT CS  pin is connected to NodeMCU pin D2
#define TFT_RST   D3     // TFT RST pin is connected to NodeMCU pin D3
#define TFT_DC    D4     // TFT DC  pin is connected to NodeMCU pin D4
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  tft.begin(); //Intializing the screen
  tft.fillScreen(ILI9341_BLACK);//filling the screen with black colour
  tft.setCursor(10, 140); //setting the cursor for the desired output
  tft.setTextColor(ILI9341_RED); //setting the textcolour
  tft.setTextSize(5);//setting the text size
  tft.print(" !!GROUP 2 ROCKS!! ");//printing the text on touch screen
 tft.setCursor(15, 175);//setting the cursor
 tft.setTextColor(ILI9341_WHITE); //setting the textcolour
 tft.setTextSize(2.5);//setting the text size
 tft.println(" TOUCHSCREEN IS ON  ");//printing the text on screen
}
void loop(void)
{}
```

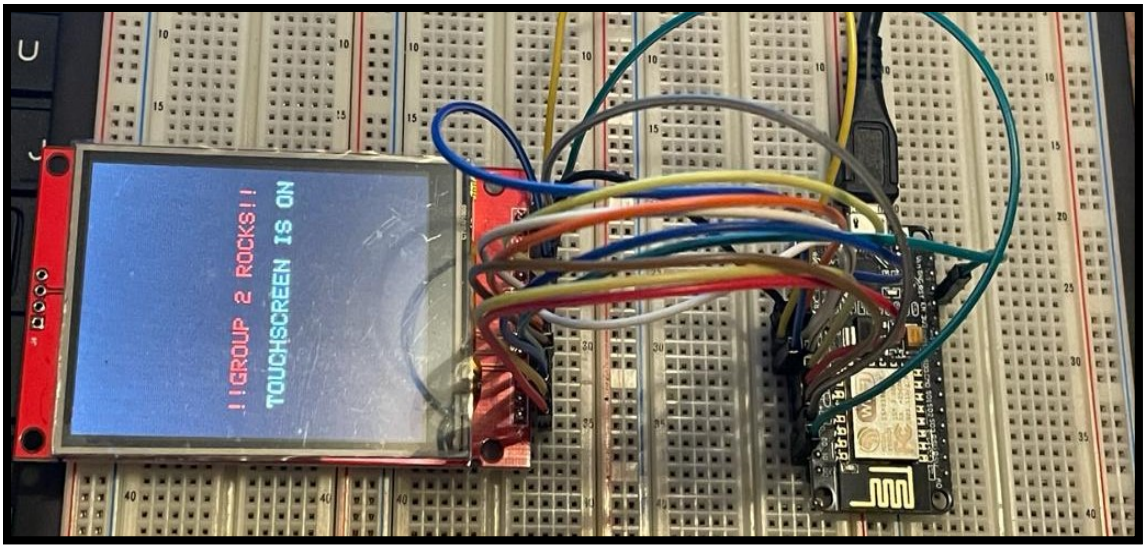

Figure 5.32: Output of displaying text on the Touchscreen

CODE 2: GUI for Touchscreen

```
#include <Arduino.h> //to handle functions and other stuff
#include <SPI.h> //for spi communication

#include "Adafruit_ILI9341esp.h" //for TFT screen
#include "Adafruit_GFX.h"   //for graphics
#include "XPT2046.h" //touch screen controller

#define TFT_DC 2 // connections of screen with nodemcu
#define TFT_CS 15
int e = 0; //declaring the integer and string as variable
String d ;
/* UI details for buttons */
#define BUTTON_X 40
#define BUTTON_Y 100
#define BUTTON_W 60
#define BUTTON_H 25
#define BUTTON_SPACING_X 20
#define BUTTON_SPACING_Y 20
#define BUTTON_TEXTSIZE 2

// defining text box where numbers go
#define TEXT_X 10
#define TEXT_Y 10
#define TEXT_W 220
#define TEXT_H 50
#define TEXT_TSIZE 2.5
#define TEXT_TCOLOR ILI9341_GREEN
```

```
#define TEXT_LEN 12 //defining the length for text field
char textfield[TEXT_LEN + 1] = "";
uint8_t textfield_i = 0; //initializing text field with zero

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC); //making the tft
obeject
XPT2046 touch(/*cs=*/ 4, /*irq=*/ 5); // defining the parameters for touch

char buttonlabels[15][5] = {"ENT", "Clr", "-", "1", "2", "3", "4", "5",
"6", "7", "8", "9", "*", "0", "#" }; //declaring the buttons
uint16_t buttoncolors[15] = {ILI9341_DARKGREEN, ILI9341_DARKGREY,
ILI9341_YELLOW,
                             ILI9341_RED, ILI9341_RED, ILI9341_RED,
                             ILI9341_RED, ILI9341_RED, ILI9341_RED,
                             ILI9341_RED, ILI9341_RED, ILI9341_RED,
                             ILI9341_ORANGE, ILI9341_RED, ILI9341_ORANGE
                            }; // defining the colour
Adafruit_GFX_Button buttons[15]; // total count of buttons
  String c; // initializing the string
void setup() {
  delay(1000);

  Serial.begin(115200);
  //SPI.setFrequency(ESP_SPI_FREQ);
  fg(); // defining the function


}
void fg() {
  #define TEXT_LEN 12 ////defining the length for text field
char textfield[TEXT_LEN + 1] = "";  // entered content should increment by
1
uint8_t textfield_i = 0;
  tft.begin(); // initializing the screen
  touch.begin(tft.width(), tft.height());  // Must be done before setting
rotation
  Serial.print("tftx ="); Serial.print(tft.width()); Serial.print(" tfty
="); Serial.println(tft.height());
  tft.fillScreen(ILI9341_BLACK);

  // calibration values for my screen
  touch.setCalibration(1832, 262, 264, 1782);
  // create buttons
  for (uint8_t row = 0; row < 5; row++) {
    for (uint8_t col = 0; col < 3; col++) {
      buttons[col + row * 3].initButton(&tft, BUTTON_X + col * (BUTTON_W +
BUTTON_SPACING_X),
                                        BUTTON_Y + row * (BUTTON_H +
BUTTON_SPACING_Y), // x, y, w, h, outline, fill, text
```

```
                                        BUTTON_W, BUTTON_H, ILI9341_WHITE,
buttoncolors[col + row * 3], ILI9341_WHITE,
                                        buttonlabels[col + row * 3],
BUTTON_TEXTSIZE);
      buttons[col + row * 3].drawButton();
    }
  }

  tft.drawRect(TEXT_X, TEXT_Y, TEXT_W, TEXT_H, ILI9341_WHITE); // creating
the text field

}
void loop() {
  uint16_t x, y;
  if (touch.isTouching())
    touch.getPosition(x, y);

  for (uint8_t b = 0; b < 15; b++) {
    if (buttons[b].contains(x, y)) {
      buttons[b].press(true);  // tell the button it is pressed
    } else {
      buttons[b].press(false);  // tell the button it is NOT pressed
    }
  }

  for (uint8_t b = 0; b < 15; b++) {
    if (buttons[b].justReleased()) {
      buttons[b].drawButton();  // draw normal
    }

    if (buttons[b].justPressed()) {
      buttons[b].drawButton(true);  // draw invert!

      if (b >= 3)// help in reading the values for screen  {
        if (textfield_i < TEXT_LEN) {
          textfield[textfield_i] = buttonlabels[b][0];
          textfield_i++;
          textfield[textfield_i] = 0; // zero terminate
        }
      }
      if (b == 1) // required for the functioing of clr button  {

        textfield[textfield_i] = 0;
        if (textfield > 0) {
          textfield_i--;
          textfield[textfield_i] = ' ';
        }
      }
```

```
    // update the current text field
    Serial.println(textfield);
    tft.setCursor(TEXT_X + 2, TEXT_Y + 10);
    tft.setTextColor(TEXT_TCOLOR, ILI9341_BLACK);
    tft.setTextSize(TEXT_TSIZE);
    tft.print(textfield);
     c = textfield;
    if (c == "690" || c == "137") // setting the password for two users {
       d = textfield;
      if (d == "690" || d == "137") {
        e += 1;
      }
    }
    else{
      if(c.length() >= 3){
        tft.fillScreen(ILI9341_BLACK);
          c = textfield;
        tft.print(" SORRY WRONG ONE");//msg display on screen
        delay(5000); //delay of 5 secs

     int lengthr = c.length(); //help to clear the screen
     Serial.print(lengthr);
      for(int j = 0;j<lengthr;j++){
      textfield[textfield_i] = 0;
      if (textfield > 0) {
        textfield_i--;
        textfield[textfield_i] = ' ';
      }

      }
      fg();  // again the function is called
    }
    }


  if (e >= 2) // here 2 means we are defining the two users  {
      e = 0;
      tft.setCursor(TEXT_X , TEXT_Y);
      tft.fillScreen(ILI9341_BLACK);
      if(d == "690") //first user
      {
      tft.print(" WELCOME OWNER "); //msg printed on screen
      }
      else if(d == "137") //second user
      {
     tft.print(" WELCOME CO-OWNER "); //msg printed on screen

      }
      delay(5000);  //delay of 5 secs
```

```
       int lengthr = c.length(); // again update the screen with cleaaring
everything
       Serial.print(lengthr);
        for(int j = 0;j<lengthr;j++){
        textfield[textfield_i] = 0;
        if (textfield > 0) //text field is >zero
        {
          textfield_i--; //decrement starts
          textfield[textfield_i] = ' ';
        }
      }
       fg();
      }

    }
  }

}
```

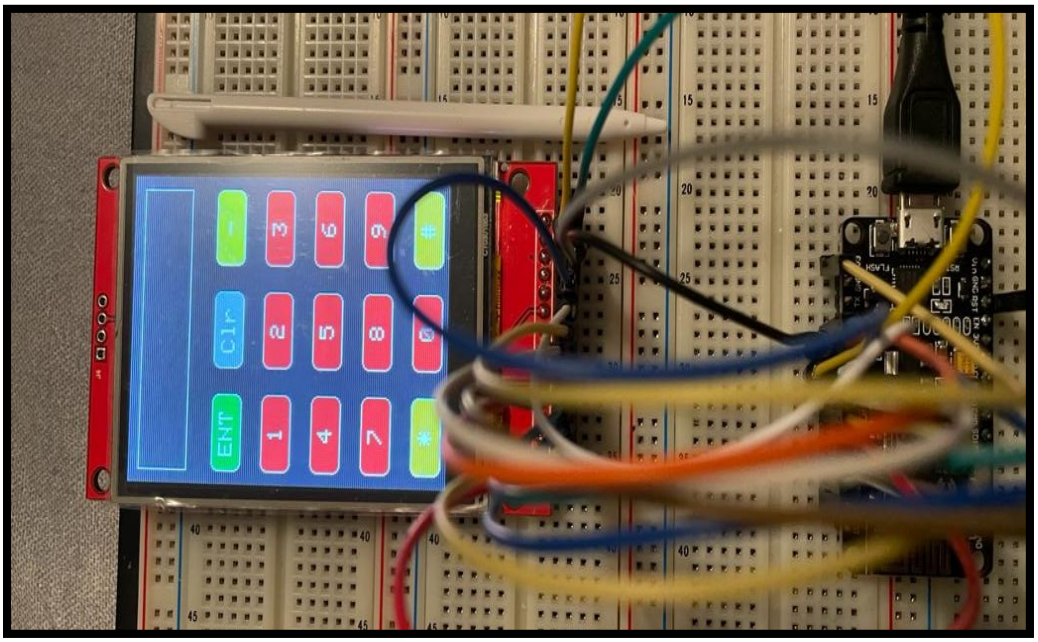

Figure 5.33: Output of the GUI

Chapter - 6

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, Vibration Sensor, RTC Module, Fingerprint Sensor, Arduino Mega, ESP32 CAM Module, ESP8266 Module GSM Module and Touchscreen.

Minimum Requirements

Minimum product output will be a locker that opens when the biometric credentials of the user will meet. Alongside this if anybody tries to open the locker with physical force it will turn on the buzzer and activates the camera module. The image will be sent to the owner of the locker with the help of the gmail services. In 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. For the fingerprint sensor and Arduino mega, we had to use the UART ports of the beaglebone black. Firstly, while doing that we were getting the error and then we realised that for the beaglebone black we have to enable the UART ports first. We fixed that issue with the help of the link that we have provided in the references.
2. While the interfacing of vibration sensor, the vibration sensor stopped working then we had to replace it with the new one.
3. During Zero PCB implementation, we faced problem during the problem while soldering as the holes of the PCB were close to each other and some connections were short then we need to desolder and solder it again. For that, we used the marker before making the connections also; instead of soldering the component we soldered the headers.
4. We are using ESP32 CAM module for the wi-fi and camera and it does not have any USB port to upload the code into it. So we had two options to proceed with. First one was to use the FTDI programmer cable and second one was using the ESP8266 so as we are using

node MCU already in the project so went with second one. We also ordered the FTDI cable in case any problem occurs.

5. 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 SIM900 would be compatible and we connected the GSM with the BBB using Arduino Mega.
6. EasyEDA software is quite good enough to have almost all of the components present but some of the components like 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.
7. 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 Beaglebone Black We had few components interfaced directly with the beaglebone by the GPIO pins so to use these pins we had to use the iobb library. 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.
8. While coding the GSM, 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.

Integrated code

This section is all about the final code that we have developed using our previous codes that we used while interfacing of the components. So, this code basically follows a pattern that we want on the output side.

```
#include <iobb.h>  // Library to access GPIO
#include <time.h>  //Time Library
#include <stdio.h> // Standard IO library
#include <stdio.h> //Standard C input Output Library
#include <unistd.h> //defines miscellaneous symbolic constants and types, and declares
miscellaneous functions
#include <string.h> //C Library for various String Operations
#include <termios.h> // Contains the definitions used by the terminal I/O interfaces
#include <fcntl.h> // FIle control, Open, close
#include <time.h>



#define BAUDRATE B9600

void servo_move(int angle); //Defining a function

//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
int vibration, openFlag, fingerFlag, pinFlag, wrongpinFlag;
int buzzerPin = 12;
int servoPin = 10;
int vibrationPin = 12;

int arduino, esp; // Variable integers
char arduinoincoming[100]; // declare a char array for receiving data
char arduinooutgoing[100]; // A buffer char array to store temporary data
char espincoming[100]; // declare a char array for receiving data
char espoutgoing[100]; // A buffer char array to store temporary data
int aonum, ainum;
```

```
int eonum, einum;

char *users[] = {"Aman", "Simran"};
int pinuser = 0;
int fingeruser = 0;



int wrongFingerFlag;

int main(void)
{
   iolib_init();  //Initializing the iobb library
   iolib_setdir(9, vibrationPin, DigitalIn);  //Setting Pin as Input of a specific port
   iolib_setdir(8, buzzerPin, DigitalOut);
   iolib_setdir(8, servoPin, DigitalOut);

   servo_move(0);

   pin_high(8, buzzerPin);
   sleep(1);
   pin_low(8, buzzerPin);

   // /dev/ttyO4 is linked with UART4 Port of Beaglebone
   if ((arduino = open("/dev/ttyO4", O_RDWR | O_NOCTTY | O_NONBLOCK))<0){ //
Try opening the UART4 f1 in Read Write mod

      printf("UART: Failed to open the arduino port.\n"); //A message Print
      return 0;
   }
```

```
    else
    {
        printf("UART arduino: Started\n"); //A message Print
    }

    struct termios options;
    /* set new port settings for canonical input processing */
    options.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD; //  CLOCAL - It will ignore Data Carrier Signal
    //CS8 = Select 8 data bits
    // CREAD = Enable Receiving Characters
    // CRTSCTS = Automaticall handle RTS (Request to Send) and CTS(Clear to Send)
    options.c_iflag = IGNPAR | ICRNL; // IGNPAR = IGNORE PARITY, ICRNL - Use CR & NL as Break Line
    options.c_oflag = 0; //output mode flag
    options.c_lflag = ICANON; //Waits for complete line to come
    options.c_cc[VMIN]=2; // Minimum wait Time
    options.c_cc[VTIME]=20; //Maximum Wait Time
    tcflush(arduino, TCIFLUSH); //Flush all the Input Data
    tcsetattr(arduino,TCSANOW,&options);  // TCSANOW - Any change made should be applied immediately
    tcflush(arduino, TCIOFLUSH); // Flush ALl the Input and Outputs
    sleep(1); // Wait for a second

    // /dev/ttyO1 is linked with UART4 Port of Beaglebone
    if ((esp = open("/dev/ttyO1", O_RDWR | O_NOCTTY | O_NONBLOCK))<0)
    { // Try opening the UART4 f1 in Read Write mode
        printf("UART1 esp: Failed to open the esp UART.\n"); //A message Print
        return 0;
    }
```

```
else
{
   printf("UART esp: Started\n"); //A message Print
}

struct termios espoptions;
/* set new port settings for canonical input processing */
espoptions.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
espoptions.c_iflag = IGNPAR | ICRNL;
espoptions.c_oflag = 0;
espoptions.c_lflag = ICANON;
espoptions.c_cc[VMIN]=2;
espoptions.c_cc[VTIME]=20;
tcflush(esp, TCIFLUSH);
tcsetattr(esp,TCSANOW,&espoptions);
tcflush(esp, TCIOFLUSH);
sleep(1);



while(1)   // Infinite Loop
{
   //Getting Sensor Data
   if (is_low(9, 12)) // Checking if pin 12 of port 9 is high
   {
      vibration = 1;
      printf("Vibration Detected \n");  // Print if Vibration Detected
   }

   else
```

```
        {
            vibration = 0;
            //printf("No Vibration Detected \n");  // Print if No Vibration Detected
        }



        //////// ARDUINO UART
        if(vibration == 1)
        {
            pin_high(8, buzzerPin);
            //memset(arduinooutgoing, 0, sizeof(arduinooutgoing)); // Clear any old data in
buffer
            //strcpy(arduinooutgoing, "vibration"); // Copy a string in buf char array
            //aonum = strlen(arduinooutgoing); // Store size of buf array in nbytes
            //aonum = write(arduino, arduinooutgoing, aonum); // Sending message to Arduino
Module
            //printf("Sent Bytes: ");
            sleep(5);
            //printf("%d \n", aonum);// print the bytes sent
        }
        else if(wrongpinFlag == 1)
        {
            pin_high(8, buzzerPin);
            memset(arduinooutgoing, 0, sizeof(arduinooutgoing)); // Clear any old data in
buffer
            strcpy(arduinooutgoing, "wrongpin"); // Copy a string in buf char array
            aonum = strlen(arduinooutgoing); // Store size of buf array in nbytes
            aonum = write(arduino, arduinooutgoing, aonum); // Sending message to Arduino
Module
```

```
            printf("Sent Bytes: ");
            printf("%d \n", aonum);// print the bytes sent
            wrongpinFlag = 0;
        }
        else
        {
            pin_low(8, buzzerPin);
        }


        ainum = read(arduino,&arduinoincoming,100); // Read the incoming Message from
Arduino Module


        if(ainum > 1) //If no. of bytes are read is more than 1
        {
            printf("\n\nBytes Received from Arduino - %d \n",ainum); // Print how many bytes
was i
            for(int i=0;i<ainum;i++) //a for loop to print data byte by byte
            {
                printf("%c",arduinoincoming[i]); //print a byte of message from Arduino Module
                if(arduinoincoming[i] == 'f')
                {
                        if(pinFlag == 1)
                        {
                                fingerFlag = 1;
                                fingeruser = (int)arduinoincoming[i+1] - 48;
                                printf("Fingerprint Verified for User: %s \n", users[fingeruser]);
                        }
                        else
```

```
                {
                        printf("Enter and Verify Pin First\n");
                        fingeruser = (int)arduinoincoming[i+1] -48;
                    printf("Fingerprint for User: %s \n", users[fingeruser]);
                }

                printf("fingeruser: %d \n", fingeruser);
            }
            if(arduinoincoming[i] == 'e')
            {
                wrongFingerFlag = 1;
                printf("Several Wrong Fingerprint Detected.\n");
            }
        }
        printf("\n");

        sleep(2);
    }

    //ESP UART
    if(vibration == 1)
    {
        memset(espoutgoing, 0, sizeof(espoutgoing));
        strcpy(espoutgoing, "vibration"); // Copy a string in buf char array
        eonum = strlen(espoutgoing); // Store size of buf array in nbytes
        eonum = write(esp, espoutgoing, eonum); // Sending message to Arduino Module
        printf("Sent Bytes: ");
        sleep(1);
        printf("%d \n", eonum);
    }
```

```
    else if(wrongFingerFlag == 1)
    {
      memset(espoutgoing, 0, sizeof(espoutgoing));
      strcpy(espoutgoing, "wrongfin\n"); // Copy a string in buf char array
      eonum = strlen(espoutgoing); // Store size of buf array in nbytes
      eonum = write(esp, espoutgoing, eonum); // Sending message to Arduino Module
      printf("Sent Bytes: ");
      sleep(1);
      printf("%d \n", eonum);
    }

    einum = read(esp,&espincoming,100); // Read the incoming Message from Arduino Module



    if(einum > 1) //If no. of bytes are read is more than 1
    {
      printf("\n\nBytes Received from ESP - %d",einum); // Print how many bytes was incomingbuffer[
      printf("\n");
      for(int i=0;i<einum;i++) //a for loop to print data byte by byte
      {
        printf("%c",espincoming[i]); //print a byte of message from Arduino Module
        if(espincoming[i] == '1')
        {
          pinFlag = 1;
          pinuser = 0;
          printf("\nTouchscreen Passcode Verified by User: %s \n", users[pinuser]);
          printf("Pin User: %d \n", pinuser);
        }
```

```
        else if(espincoming[i] == '2')
        {
          pinFlag = 1;
          pinuser = 1;
          printf("\nTouchscreen Passcode Verified by User: %s \n", users[pinuser]);
          printf("Pin User: %d \n", pinuser);
        }
        else if(espincoming[i] == 'e')
        {
           wrongpinFlag = 1;
           printf("Several Wrong Pin Attempts Detected.\n");
           pin_high(8, buzzerPin);
           sleep(5);
           pin_low(8,buzzerPin);
        }
    }
    printf("\n");

    sleep(1);
}

if (pinFlag == 1 && fingerFlag == 1)
{
    if(fingeruser == pinuser)
    {
            openFlag = 1;
    }
    else
    {
            printf("Pin and Finger is of Different User\n");
```

```
                pinFlag = 0;
                fingerFlag = 0;
                openFlag = 0;
        }
    }

    if(openFlag == 1)
    {
        printf("Opening the Locker\n");

        sleep(2);

        for(int an = 10; an <= 180; an = an+10)
        {
                servo_move(an);
                sleep(0.5);
        }

        if (fingeruser == 0)
        {
                memset(arduinooutgoing, 0, sizeof(arduinooutgoing)); // Clear any old data
in buffer
                strcpy(arduinooutgoing, "-OA\n"); // Copy a string in buf char array
                aonum = strlen(arduinooutgoing); // Store size of buf array in nbytes
                aonum = write(arduino, arduinooutgoing, aonum); // Sending message to
Arduino Module
                printf("Sent Bytes: ");
                sleep(1);
                printf("%d \n", aonum);// print the bytes sent
        }
```

```
        if (fingeruser == 1)
        {
                memset(arduinooutgoing, 0, sizeof(arduinooutgoing)); // Clear any old data
in buffer
                strcpy(arduinooutgoing, "-OS\n"); // Copy a string in buf char array
                aonum = strlen(arduinooutgoing); // Store size of buf array in nbytes
                aonum = write(arduino, arduinooutgoing, aonum); // Sending message to
Arduino Module
                printf("Sent Bytes: ");
                sleep(1);
                printf("%d \n", aonum);// print the bytes sent
        }
        openFlag = 0;
        pinFlag = 0;
        fingerFlag = 0;

        sleep(5);

        for(int an = 170; an >= 0; an = an-10)
        {
                servo_move(an);
                sleep(0.5);
        }
    }

    iolib_delay_ms(200);

  }

  close(esp); //Close the f1 at last
```

```
    close(arduino); //Close the f1 at last
    iolib_free();
    return(0);
}

void servo_move(int angle) // Function for moving servo to angle
{

  int delay = 0; // An integer to store delays

  delay =( (5.55 * angle)+1000); // A formula to calculate delay for Servo motor
  // We got the delay and other details from Servo motor (SG90) datasheet

  for(int i=0;i<50;i++) // A for loop running 50 times for setting angle of Servo
  {
    pin_high(8,servoPin); // Making the servo connected pin high
    usleep(delay); //Delay in microseconds which we calculated above
    pin_low(8,servoPin); // Making the servo connected pin low
    usleep(20000-delay); //Delay in microseconds which we calculated above subtracted by
20000
  }

  iolib_delay_ms(10); // Small delay to give time to Servo to settle

}
```

Zero PCB implementation

Soldering is one of the most fundamental skills needed to dabble in the world of electronics. The two go together like peas and carrots. And, although it is possible to learn about and build electronics without needing to pick up a soldering iron, you'll soon discover that a whole new

world is opened with this one simple skill. Soldering is the only permanent way to ‘fix’ components to a circuit.

- **STEPS FOR SOLDERING**
  - Collecting Materials for Soldering
  - Preparing to Solder
  - Soldering a PCB

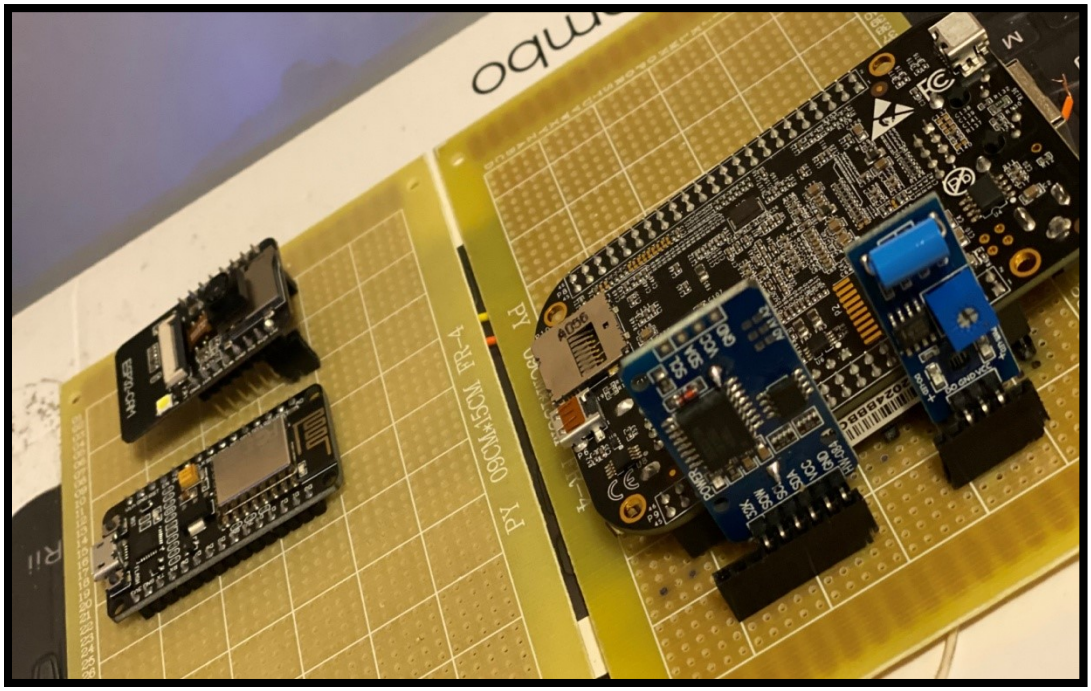

Figure 6.0: Front View of Zero PCB

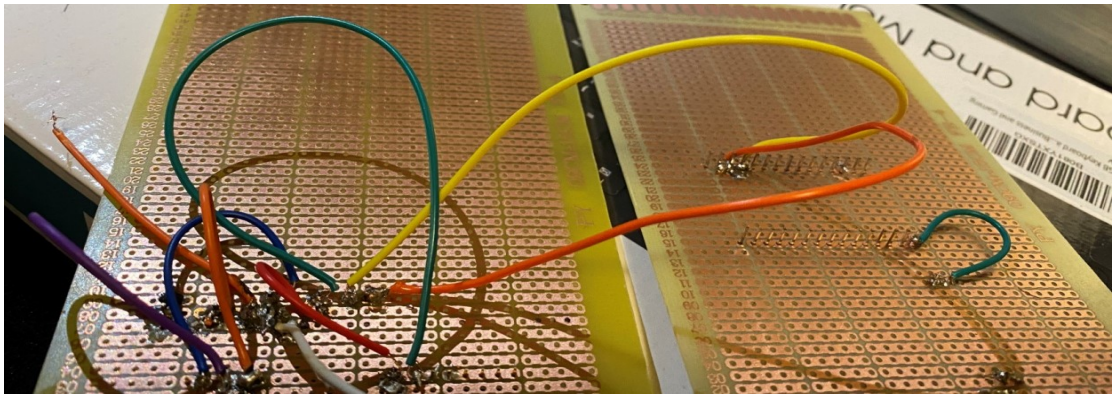

Figure 6.1: Back view of Zero PCB

Chapter - 7

Conclusion

Purpose of the project is to make a locker that acquires the advanced features of security. Features added are mainly addressing the security aspect of the locker. Various sensors have been used to achieve the expectations. Cloud services has been used to share the important data related to the project that makes this system as a smart system. This was just a startup idea to make a locker with advanced security features further components can be added to modify this project that can serve the individual requirements in a better way. As of now the user can get the image of the unauthorized user who tries to access the locker and also a text message is received by the owner of the locker.

Future work

To satisfy the individual requirements many changes can be suggested. So, in the future scope of this project these things can be considered:

OTP (one time Password) can be used to replace the permanent password as a new password will be send to the main owner by the means of e-mail or text message using GSM.

1. Facial Recognition can be used that can add one more layer of security to the system.
2. One can consider this as a basic idea to explore more dimensions to this existing one.

Chapter - 8

User's Guide

About the product

The product is about the bank locker security. As the user's personal locker is being secure by this system. A two-layer security is given to the user's personal locker with a password and then fingerprint scanner. Any odd activity is being monitored by the camera and then an email is sent to the cloud.

Features

- ✓ The system has 4 microcontroller and out which Beagle bone black is the host controller.
- ✓ The system is a plug and play system, like no laptops are required to run the code, just the load the code once's and then wall adapter are there to give power to the main micro controller.
- ✓ If anyone tries to open the locker forcefully, then there is a vibration sensor in the system that will aware the main user while taking an image with the camera and send it to the cloud.
- ✓ The system is insured with a two-layer security, the first one is the password entering that should be done on touchscreen and the 2nd layer is fingerprint sensor. So, somehow if any unknown person knows your password, still the person cannot open the locker because the finger prints are unique for each individual.
- ✓ The touchscreen password is linked with the finger print scanner, so with every correct password and correct finger print scanned there is a name of the user on the screen.

How to place the components?

Now, placing or installing the components for the system is quite important. For the final modelling the product, all the wires and pcb's and the integrated circuits are camouflaged from the user, like Beagle bone black, Arduino mega, Esp8266, RTC, Buzzer, GSM module. We are just displaying the touchscreen that is required to enter the password and finger print sensor to scan finger, the rest of the circuit is kept underground.

Chapter - 9

References

- Sending email using SMTP https://randomnerdtutorials.com/esp32-send-email-smtp-server-arduino-ide/
- Node MCU features : https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet
- Pin description of Touch screen: http://www.lcdwiki.com/res/MSP2807/2.8inch_SPI_Module_MSP2807_User_Manual_EN.pdf
- GUI for the Screen : http://www.lcdwiki.com/res/MSP2807/2.8inch_SPI_Module_MSP2807_User_Manual_EN.pdf
- Enabling the UART ports of beaglebone: https://electronics.trev.id.au/2018/02/09/get-uart-serial-ports-working-beaglebone-black.
- Reading data from serial ports: https://stackoverflow.com/questions/6947413/how-to-open-read-and-write-from-serial-port-in-c.
- Arduino mega with GSM: SIM900 GSM GPRS Shield with Arduino | Random Nerd Tutorials
- AT Commands: http://www.developershome.com/sms/atCommandsIntro.asp
- Vibration Sensor details : SW-420 Vibration Sensor Module Pinout, Datasheet, Features & Specifications (components101.com)
- GPIO for beaglebone : BBB - BeagleBone Black I/O (GPIO) Library for C | element14 | BeagleBoard
- I2C communication protocol: Basics of the I2C Communication Protocol (circuitbasics.com)
- RTC Module: https://datasheets.maximintegrated.com/en/ds/DS3231-DS3231S.pdf
- RTC with beaglebone black: https://learn.adafruit.com/adding-a-real-time-clock-to-beaglebone-black
- Fingerprint Sensor manual: https://www.openhacks.com/uploadsproductos/r307_fingerprint_module_user_manual.pdf

- Fingeerprint sensor with the Arduino: https://learn.adafruit.com/adafruit-optical-fingerprint-sensor?view=all
- PCB Designing: https://www.lydnow.com/pcb-designing#:~:text=What%20is%20PCB%20designing%3F%20PCB%20designing%20is%20a,for%20both%20through-hole%20and%20surface%20mount%20electronic%20components
- Extracting Gerber files: gerbv — a Gerber (RS-274X) viewer download | SourceForge.net
- Beaglebone black : https://stackoverflow.com/questions/38601376/hibernate-a-beaglebone-black
- https://cdn-shop.adafruit.com/datasheets/BBB_SRM.pdf
- Usleep : https://man7.org/linux/man-pages/man3/usleep.3.html
- Previous similar project:
- https://www.ijert.org/research/an-iot-based-bank-locker-security-system-IJERTCONV8IS07008.pdf
- Easy EDA : https://www.wellpcb.com/easyeda.html
- https://easyeda.com/editor#id=15c33d01c6224fc397a5c09cd063665d
- https://www.youtube.com/watch?v=30BYc0ikP6c
- Interfacing ESP8266 With ESP32: https://www.instructables.com/Programming-ESP32-CAM-With-ESP8266/
- Interfacing beaglebone with Arduino: How to Make a BeagleBone and an Arduino Communicate : 4 Steps – Instructables
- Easy EDA editor: https://easyeda.com/editor
- Zero PCB design: https://www.instructables.com/Simple-PCB-soldering/
- Testing Soldering with multimeter: https://multimeterexpert.com/use-multimeter-test-solder-joint/
- Servo motor manual: https://components101.com/motors/servo-motor-basics-pinout-datasheet