

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY

(Near Jnana Bharathi Campus, Mallathalli, Bengaluru-560 056.)

(An Autonomous Institution, Affiliated to VTU, Belgaum, Aided by Government of Karnataka)



PROJECT REPORT ON “REAL TIME DATA ACQUISITION IN AN IOT BASED CONTROL UNIT”

Submitted in partial fulfillment for the requirements for the award of degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS & COMMUNICATION ENGINEERING

Submitted By

PRANAV JAYARAM :1DA15EC105

NAVEEN KUMAR N :1DA15EC093

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**

2017-18

Dr. AMBEDKAR INSTITUTE OF TECHNOLOGY

(Near Jnana Bharathi Campus, Mallathalli, Bengaluru-560 056.)

(An Autonomous Institution, Affiliated to VTU, Belgaum, Aided by Government of Karnataka)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



CERTIFICATE

This is to certify that the project work entitled “**REAL TIME DATA ACQUISITION IN AN IOT BASED CONTROL UNIT**” submitted in the partial fulfilment for the requirement of the 6th semester Mini project curriculum during the year 2017-18 is a result of bonafide work carried out by **PRANAV JAYARAM (1DA15EC105)** and **Late.NAVEEN KUMAR N (1DA15EC093)**.

PRANAV JAYARAM

[1DA15EC105]

Signature of guide

HEMALATHA K N

Asstistant Prof., Dept. of ECE, Dr. AIT

External Viva

Name of Examiners

1)_____

2)_____

Signature of H.O.D

Dr. GV JAYARAMAIAH

H.O.D, Dept. of ECE, Dr. AIT

Signature with date

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of this mini project would be complete only with the mention of the people who made it possible, whose support rewarded our effort with success.

We are grateful **to Dr. Ambedkar Institute of Technology** for its ideals and its inspirations for having provided us with the facilities that have made this mini project a success.

We are grateful to our Principal **Dr. C Nanjundaswamy** who gave a continuous support and provided us comfortable environment to work in.

We express our sincere gratitude to **Dr. G V Jayaramaiah**, Head of Department, Electronics and Communication.

We express our sincere thanks to our guide **Hemalatha K N**, Assistant Professor, Department of Electronics and Communication and project coordinator **Kavithadevi C S**, Associate Professor, Department of Electronics and Communication for their advice, supervision and guidance throughout the course of the project.

We are also grateful to all the other members of the faculty of Electronics and Communication Department for this cooperation.

Finally, we wish to thank our parents, all my dear friends and other people who have directly or indirectly been a support from the start of the project, for their whole-hearted co-operation, support and encouragement.

PRANAV JAYARAM
Late. NAVEEN KUMAR N

ABSTRACT

This project aims to design and implement a real-time data acquisition and batch-processing technique on an Internet of Things (IOT) based customizable control unit. The scope of the project is to incorporate the use of interconnectivity between devices, which generate data and to process the generated data onboard with the use of a real-time batch data processing algorithm. The processed data is then logged to the control unit, which has a continuous feedback to an IOT based platform. This platform includes online and offline nodes, which can monitor and control the devices present in the control unit. The project incorporates the design of a hardware circuitry, which includes the control panel and the processing unit.

The online and offline nodes are the interconnected devices, which use an instance of the same control unit application that can be set up on mobile and wireless devices. The project also aims to design the native control application, which can be deployed on mobile devices.

TABLE OF CONTENTS

TITLE.....	i
CERTIFICATE.....	ii
ACKNOWLEDGEMENT.....	iii
ABSTRACT.....	4
1 CHAPTER 1: INTRODUCTION.....	9
1.1 AIM OF THE PROJECT	10
1.2 PROBLEM STATEMENT	10
1.3 PROJECT SCOPE AND OBJECTIVES	11
2 CHAPTER 2: BLOCK DIAGRAMS AND ILLUSTRATIONS	12
2.1 OVERVIEW OF THE EMBEDDED SYSTEM.....	13
2.2 FUNCTIONAL BLOCK DIAGRAM.....	14
2.3 COMPLETE USE-CASE SCHEMATIC	15
3 CHAPTER 3: HARDWARE AND SOFTWARE DESIGN AND DEVELOPMENT	16
3.1 HARDWARE BLOCK	17
3.1.1 PROCESSOR BLOCK	18
3.1.2 IN-BOARD	19
3.1.3 OUT –BOARD	20
3.1.4 POWER SUPPLY AND PROTECTION CIRCUIT	21
3.1.5 PERIPHERAL COMPONENTS	22
3.2 COMPONENT SPECIFICATIONS	23
3.2.1 Broadcom BCM2873 Processor (Raspberry pi 3)	23
3.2.2 ATmega2560 Controller.....	24
3.2.3 5” TFT HDMI DISPLAY	25
3.2.4 DHT (DIGITAL HUMIDITY AND TEMPERATURE) SENSOR	28
3.2.5 Bluetooth module HC-05	30
3.3 HARDWARE DESIGN USING EAGLE.....	32

3.3.1	BOARD LAYOUT AND FABRICATION	33
3.3.2	POST FABRICATION	35
3.4	ASSEMBLING HARDWARE AND COMPONENT PLACEMENT	37
3.5	SOFTWARE BLOCK.....	39
3.5.1	TYPICAL CONTROL FLOW	39
3.5.2	SOFTWARE TOOLS USED.....	40
3.5.3	CONTROLLER FIRMWARE.....	41
3.5.4	PROCESSOR APPLICATION AND KERNEL	43
3.5.5	ONLINE APPLICATION	44
3.5.6	OFFLINE MOBILE APPLICATION.....	45
4	CHAPTER 4: RESULTS AND OUTPUTS	46
4.1	IN-BOARD OUTPUTS	47
4.2	OUT-BOARD RESULTS	49
5	CHAPTER 5: USE-CASE APPLICATIONS	52
6	CHAPTER 6: CONCLUSION AND SCOPE OF THE PROJECT	55
7	BIBLIOGRAPHY	57

TABLE OF FIGURES

Figure 1-Overview of the Embedded System	13
Figure 2-Functional Block diagram	14
Figure 3-Complete Circuit schemeatic	15
Figure 4-Processor Block.....	18
Figure 5-In-Board circuit	19
Figure 6-Out-Board circuit	20
Figure 7-Power supply circuit.....	21
Figure 8-Atmega2560 interfacing.....	22
Figure 9-BCM2873 Block diagram	23
Figure 10-ATMega2560 Pin Layout.....	24
Figure 11-5" TFT display panel.....	25
Figure 12-5" Display Pin layout	26
Figure 13-Interfacing Display and Raspberry Pi	27
Figure 14-DHT Sensor.....	28
Figure 15-Sensor working diagram	28
Figure 16-Bleutooth module HC-05	30
Figure 17-Board Schematic	33
Figure 18-Component Placement	33
Figure 19-Board file.....	34
Figure 20-Top Layer	34
Figure 21-Dimensions.....	35
Figure 22-Post fabrication.....	36
Figure 23-Arduino placement	36
Figure 24-Assembling components	37
Figure 25-Final Assembly.....	38
Figure 26-Control flow diagram	39
Figure 27-Online application front end.....	43
Figure 28-VCN Terminal application implementation.....	44
Figure 29-XML Emulated design	45
Figure 30-Main page emulation.....	45

Figure 31-COM PORT Output of Serial Monitor (Arduino IDE).....	47
Figure 32-Processor Data log.....	48
Figure 33-Relay 1,2,6-OFF , LED -OFF	49
Figure 34-Relay1-8 ON,LED-ON	50
Figure 35-Online trigger	50
Figure 36-Offline trigger.....	51
Figure 37-Android app.....	51
Figure 38-Industrial IoT vs Manual process	53

1 CHAPTER 1: INTRODUCTION

1.1 AIM OF THE PROJECT

The aim of this project is to design and implement a real time data acquisition and batch-processing algorithm on an Internet of Things based control unit. The fundamental concept of this project is to show the typical use-case applications of Internet of Things (IoT) and how it can help solve day-to-day issues in homes, industries and in many businesses. The project incorporates the use of an embedded system which comprises of a central processor, ancillary controllers and peripheral devices. The project demonstrates the application of IoT in the field of Home-automation, Industrial IoT, and Smart Monitoring.

The project runs a real time batch processing system on the processor which samples 15 bits of data at about 2000 samples of data per minute. This data is dynamic and customizable to the application in use. The use of DHT's (Digital Humidity and Temperature) sensors in the project is to demonstrate the use of temperature and humidity values to trigger ancillary controllers and to switch peripherals.

The main concept of (IoT) implemented in this project is the interconnection between devices and the control of these interconnected devices through online and offline nodes via the network and through a mobile application. This report elaborates the complete design and implementation of the project.

1.2 PROBLEM STATEMENT

“One of the main issues in today’s world is the requirement of multiple devices to perform multiple tasks, namely switching, monitoring, displaying of data and so on. The unification of all these tasks into a single control unit helps build efficient systems which consume less power, run more efficient data processing algorithms and are retro fit into many applications such as smart monitoring and surveillance, smart home and industrial automation and so on”

This is an important problem in today’s fast growing (IoT) sector, which requires a robust, efficient and less power consuming system and all to be integrated into one unit that runs a real time algorithm to schedule multiple such tasks.

1.3 PROJECT SCOPE AND OBJECTIVES

The scope of this project is to develop customizable data acquisition hardware and dedicated standalone controllers which perform unique tasks such as sensor data gathering, switching and inverting, display and multimedia and wireless control. The project also involves building some of the use-case applications using the hardware that can be deployed in industries and smart homes.

The data gathered after a subsequent amount of time can be used as training data for a future Machine learning based application that can also be incorporated into the same.

The full extent of batch processing helps a manufacturing based growth industry that can regularly monitor and control a complete production cycle/ product build.

This project has a product-oriented design that can be tapered to specific industry needs in the future.

The main objective of the project is to design and develop a generic hardware that supports up to 3 IoT based applications such as Industrial IoT, ADAS (Advanced driver assistance system) and Home Automation using a set of 2 standalone controllers with a processor.

Another important objective is to design and develop compatible onboard and mobile applications which run on a wide range of platforms such as Linux, Windows, and Android (Mobile).

2 CHAPTER 2: BLOCK DIAGRAMS AND ILLUSTRATIONS

2.1 OVERVIEW OF THE EMBEDDED SYSTEM

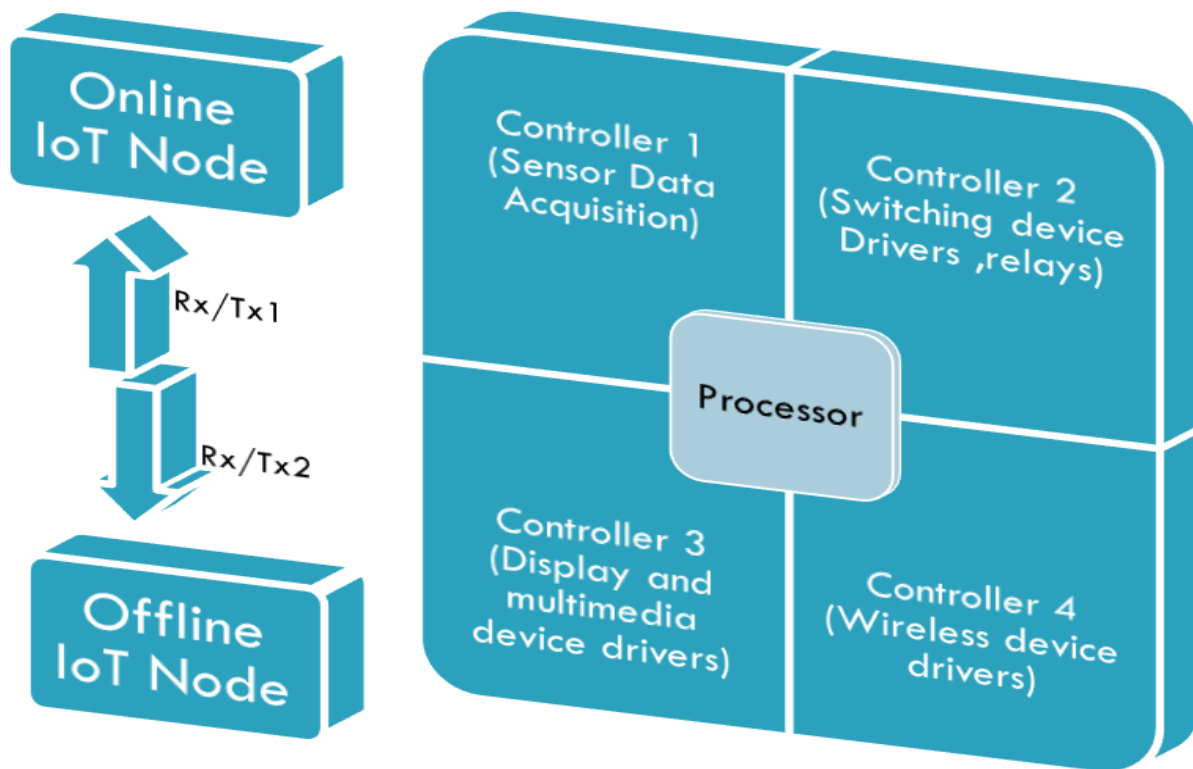


Figure 1-Overview of the Embedded System

1. The Block diagram shows the embedded system block diagram of the real-time systems that comprises the processor and standalone ancillary controller units present.
2. It shows the two Nodes that are present in the control of the IoT unit namely, the online node and the offline node. The Online node is any device that is connected to the same network as that of the embedded system, the offline node is a wireless node that isn't necessarily connected to the same network but, is interfaced with the embedded system via Bluetooth. BLE <4.0.
3. The processor runs a real-time data acquisition algorithm which acquires and stores data locally and can also be accessed from a remote node.
4. The controllers run independent firmware's pertain to the real-time working on tasks such as sensor data acquiring, Switching (relay control) and Display and multimedia operations.
5. The mobile offline node has a control application developed for the native Android OS.

2.2 FUNCTIONAL BLOCK DIAGRAM

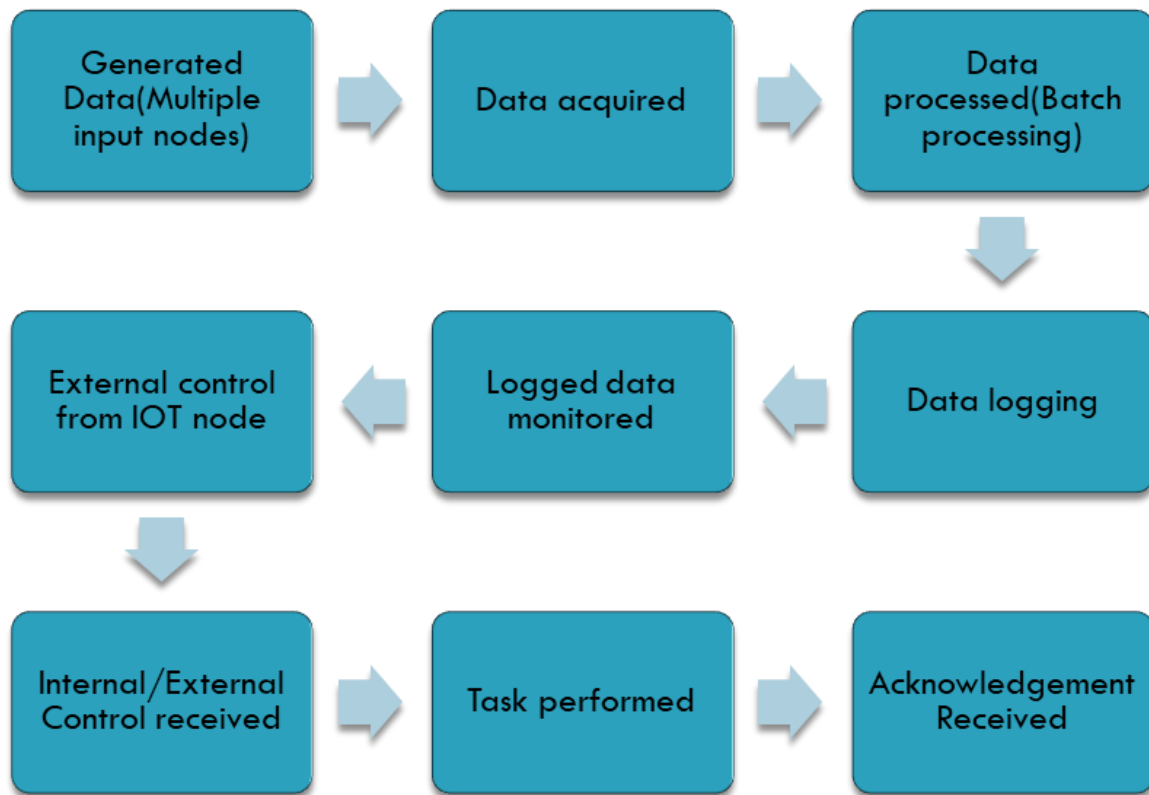


Figure 2-Functional Block diagram

1. The project involves the use of the adjacent functional blocks where an IoT node (Online/Offline) is used as a master device which targets controlling the main processor running the batch processing algorithm.
2. The concept of IoT is typically used by the processor, which interconnects with multiple subsystems and the master node.
3. The Data generated from the standalone controllers is acquired by the processor via the batch processing algorithm and available for future reference.
4. The same processing system acknowledges an event trigger from an offline or an online node. On receiving the event trigger, the processor immediately understands the request and performs the desired task and sends an acknowledgment of the same.
5. This is the real-time system that is always running in the background of the Real-time operating Kernel of the processor.
6. These functions performed by the processor are also logged. The standalone controllers are also equipped with an onboard EEPROM which stores the switching tasks performed.

2.3 COMPLETE USE-CASE SCHEMATIC

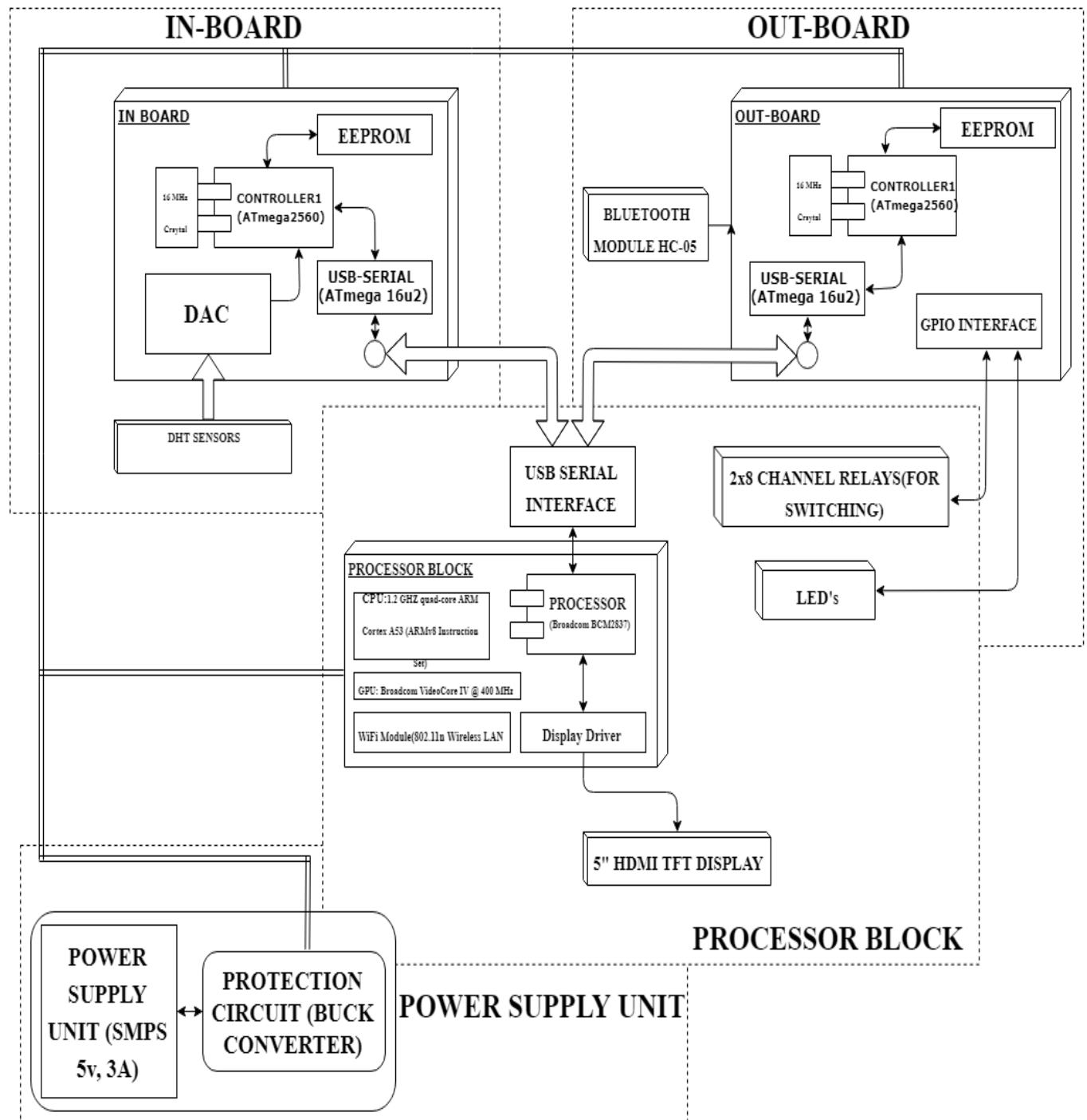


Figure 3-Complete Circuit schematic

The figure depicts the complete use-case schematic of the project which is a combination of the Power supply unit, the Processor block, In-board and the Out-board.

3 CHAPTER 3: HARDWARE AND SOFTWARE DESIGN AND DEVELOPMENT

3.1 HARDWARE BLOCK

The Hardware block consists of the following blocks which are the different components present in the embedded system.

Sl. No	FUNCTIONAL BLOCK
3.11	PROCESSOR BLOCK
3.12	IN-BOARD
3.13	OUT-BOARD
3.14	POWER SUPPLY AND PROTECTION CIRCUIT
3.15	PERIPHERAL COMPONENTS

3.1.1 PROCESSOR BLOCK

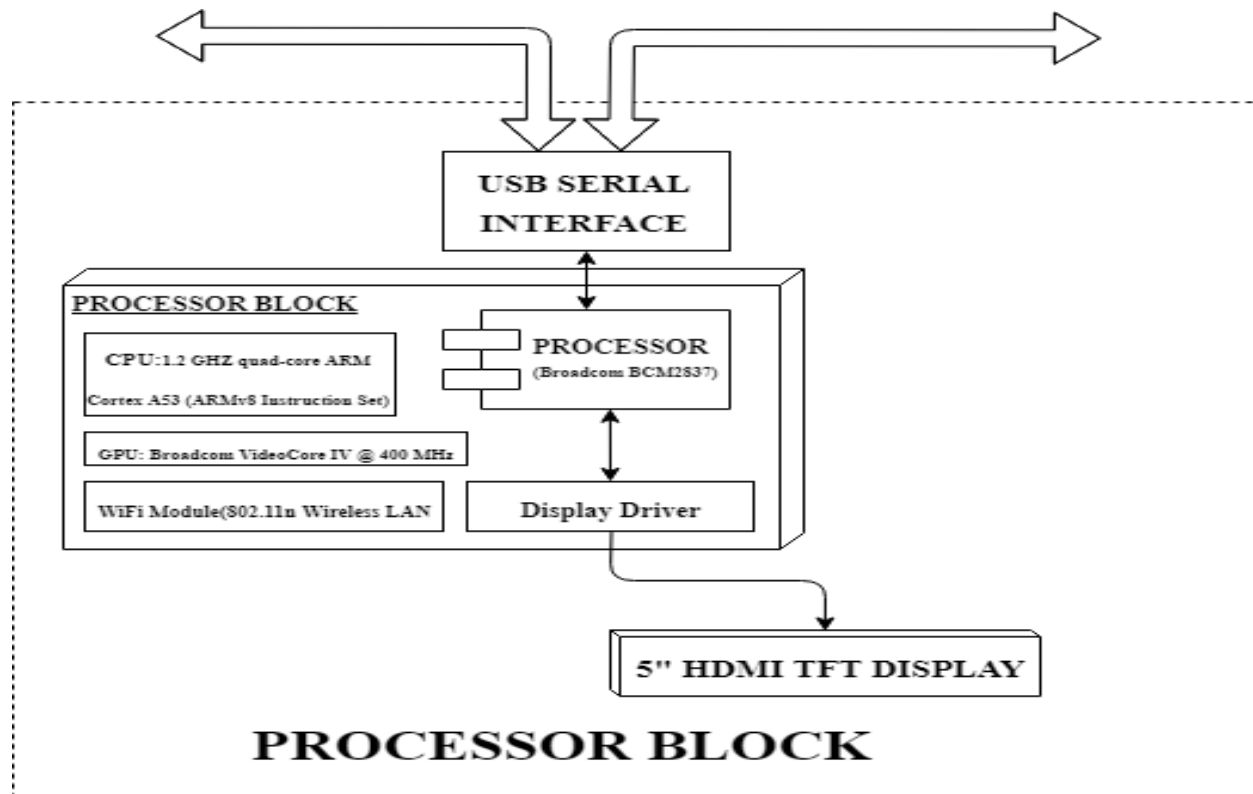


Figure 4-Processor Block

PROCESSOR BLOCK SPECIFICATIONS:

System on Chip: Broadcom BCM2837

CPU: 4× ARM Cortex-A53, 1.2GHz

GPU: Broadcom Video Core IV

RAM: 1GB LPDDR2 (900 MHz)

Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless

Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy

Storage: micro SD

GPIO: 40-pin header, populated

Ports: HDMI, 3.5mm analog audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

Interfaces Used: 2 USB-Serial ports are used to connect to the IN-BOARD and OUT-BOARD. 5" TFT display connected via HDMI port

3.1.2 IN-BOARD

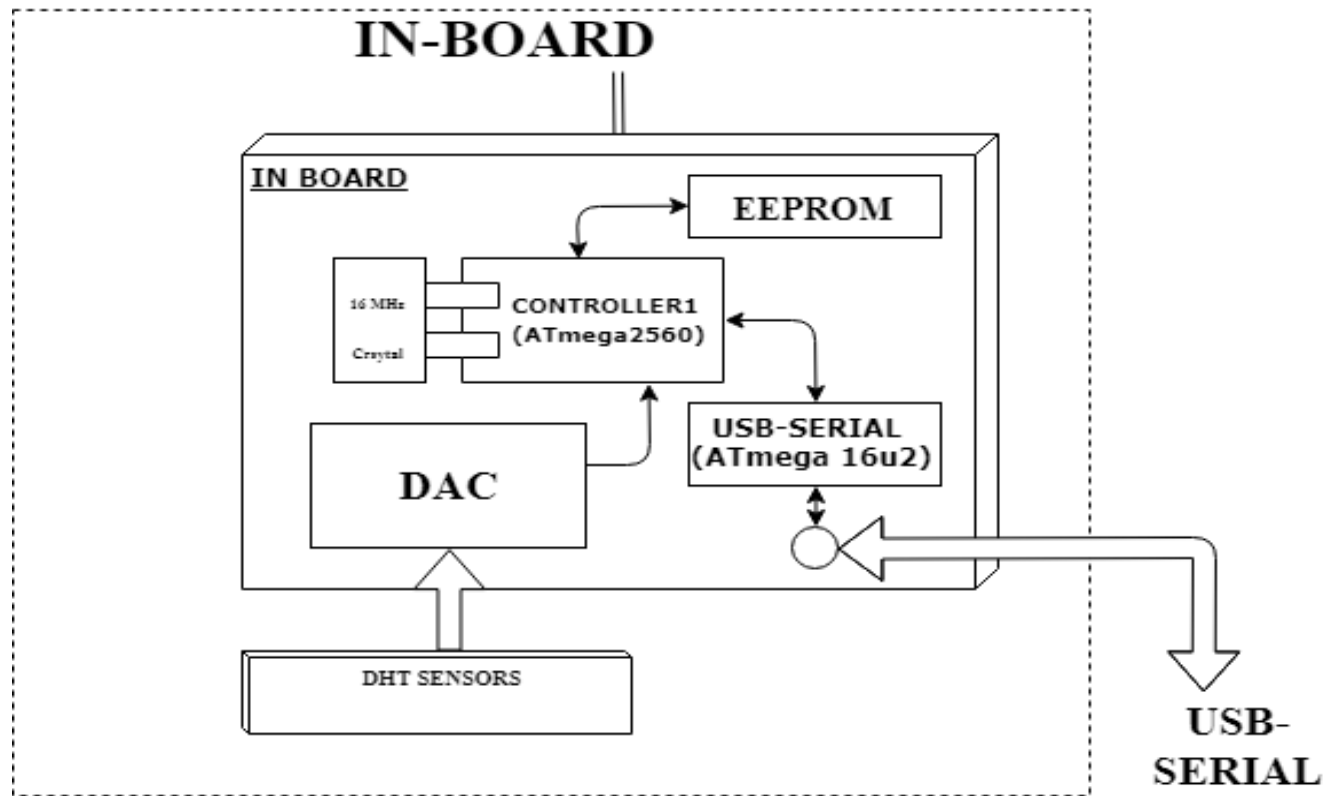


Figure 5-In-Board circuit

1. The IN-BOARD has an ATmega2560 microcontroller.
2. It has 54 digital input/output pins (of which 15 can be used as PWM outputs)
3. 16 analog input pins
4. 4 UARTs (hardware serial ports)
5. A 16 MHz crystal oscillator
6. A USB connection
7. A power jack
8. An ICSP header
9. 1 reset button.
10. 5 DHT (Digital Humidity and Temperature) Sensors connected to the digital PWM pins as sensor data Input port. The sensor data sampled from each DHT is 2 bits. So, 10 bits of data is sampled at 2000 samples/minute, resulting in 20000 bits of samples/minute.

3.1.3 OUT –BOARD

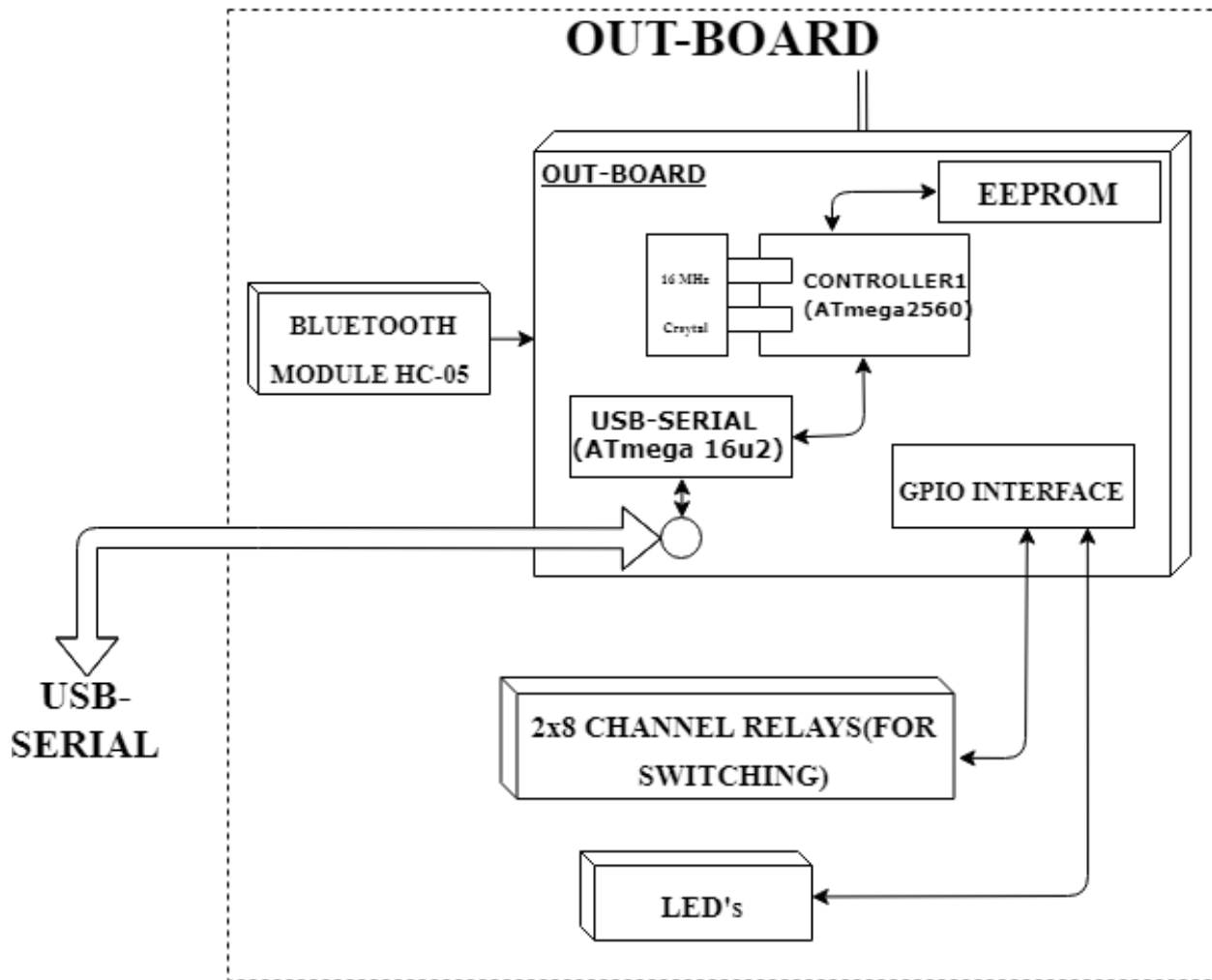


Figure 6-Out-Board circuit

The OUT-BOARD has the same specifications as that of the IN-BOARD, The difference is that it is used to perform different tasks.

1. The OUT-BOARD is connected to the peripheral devices that must be accessed using the processor and the online and offline IoT nodes.
2. The Bluetooth module HC-05 is also connected to the OUT-BOARD for wireless communication from mobile devices.
3. The peripherals connected to the OUT-BOARD are, LED's, 2x8 Channel relay module, and the USB-Serial port.

3.1.4 POWER SUPPLY AND PROTECTION CIRCUIT

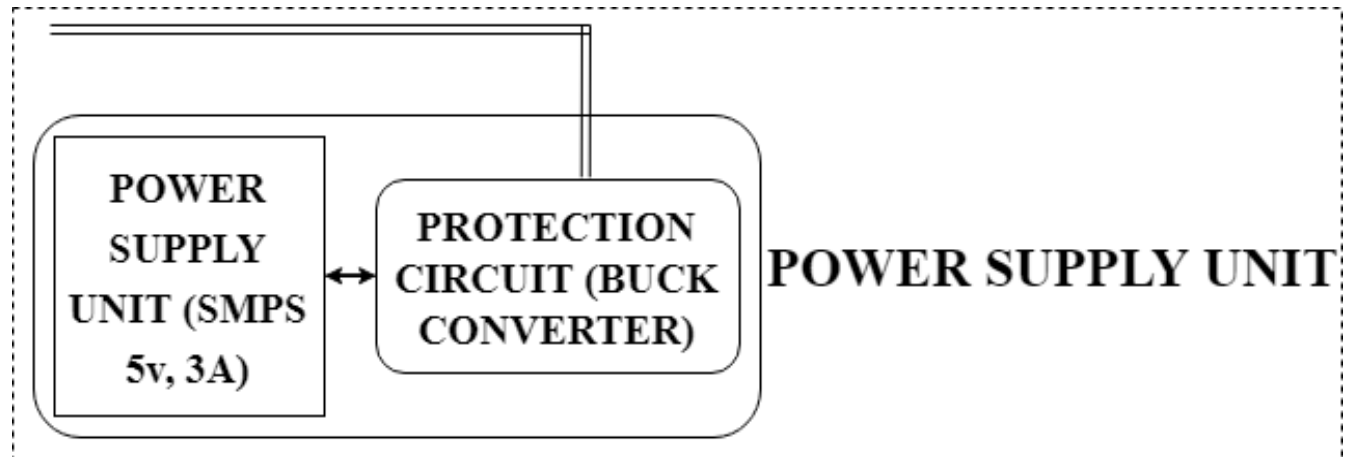
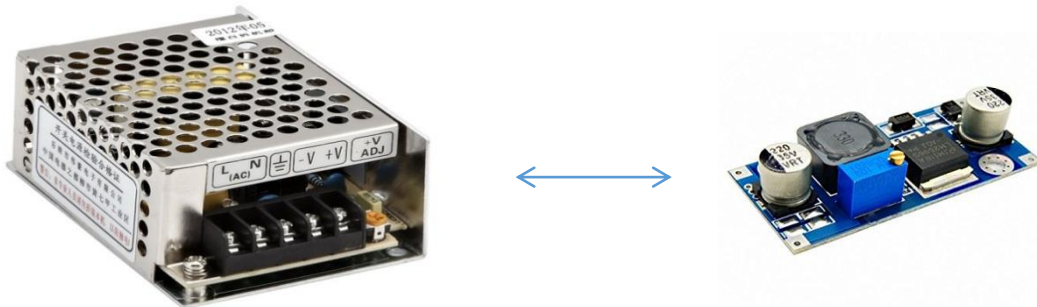


Figure 7-Power supply circuit



POWER SUPPLY SPECIFICATIONS:

DC VOLTAGE	5V
RATED CURRENT	3A
CURRENT RANGE	0-3A
RATED POWER	15W
RIPPLE AND NOISE(max)	80mVp-p
VOLTAGE ADJUSTMENT RANGE	4.75-5.5V
LINE REGULATION	+/- 0.5%
LOAD REGULATION	+/- 1.5%

3.1.5 PERIPHERAL COMPONENTS

Sl. No	COMPONENT
3.1.5.1	HC-05 Bluetooth Module (Wireless offline communication)
3.1.5.2	LED's
3.1.5.3	8 Channel Relay module 1
3.1.5.4	8 Channel Relay module 2

INTERFACING ATMEGA2560 WITH HC-05 BLUETOOTH MODULE, RELAY AND LED:

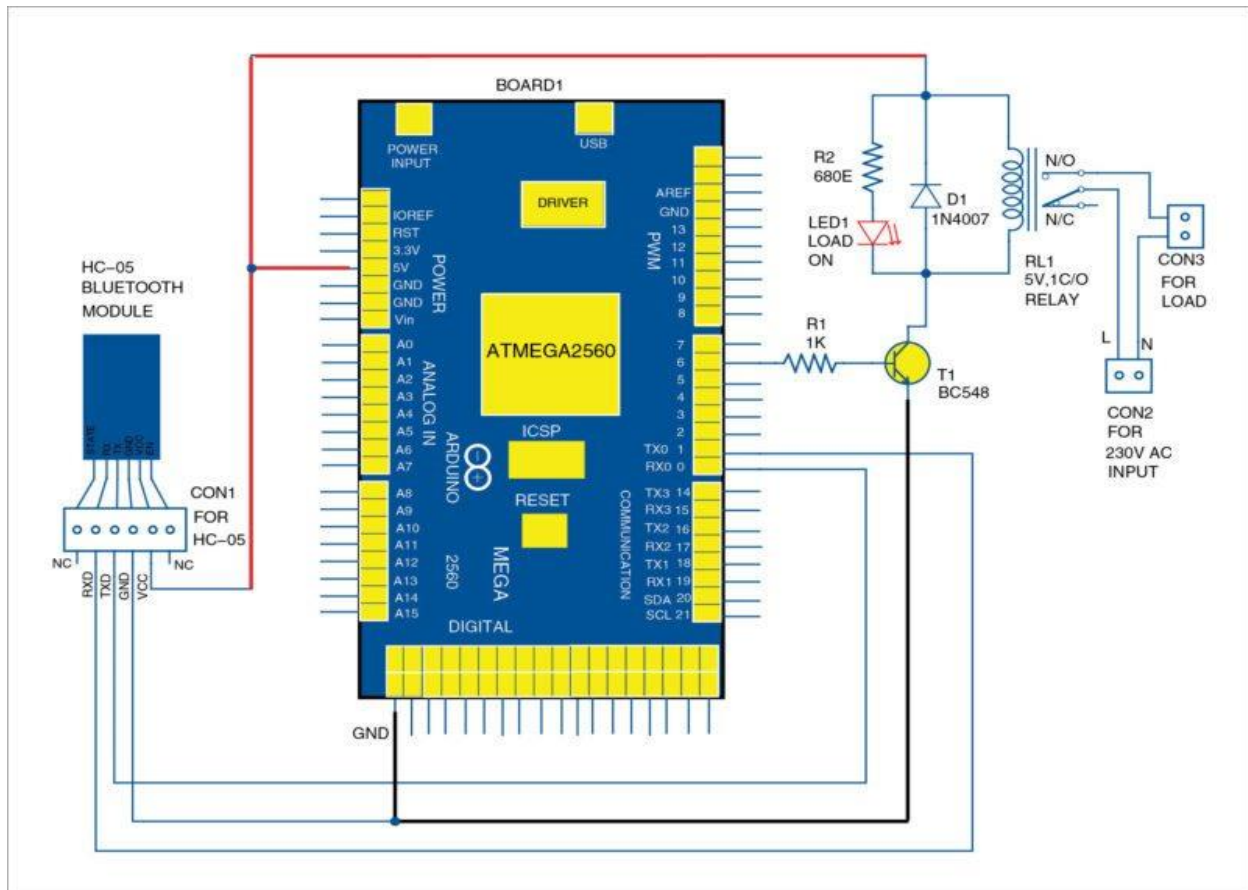


Figure 8-Atmega2560 interfacing

3.2 COMPONENT SPECIFICATIONS

The datasheet and specifications of various components used in the embedded system are mentioned below.

Sl. No	COMPONENT
3.2.1	Broadcom BCM2873 processor
3.2.2	ATMega 2560 controller
3.2.3	5” TFT HDMI display
3.2.4	DHT (Digital Humidity and Temperature) Sensor
3.2.5	Bluetooth module (HC-05)

3.2.1 Broadcom BCM2873 Processor (Raspberry pi 3)

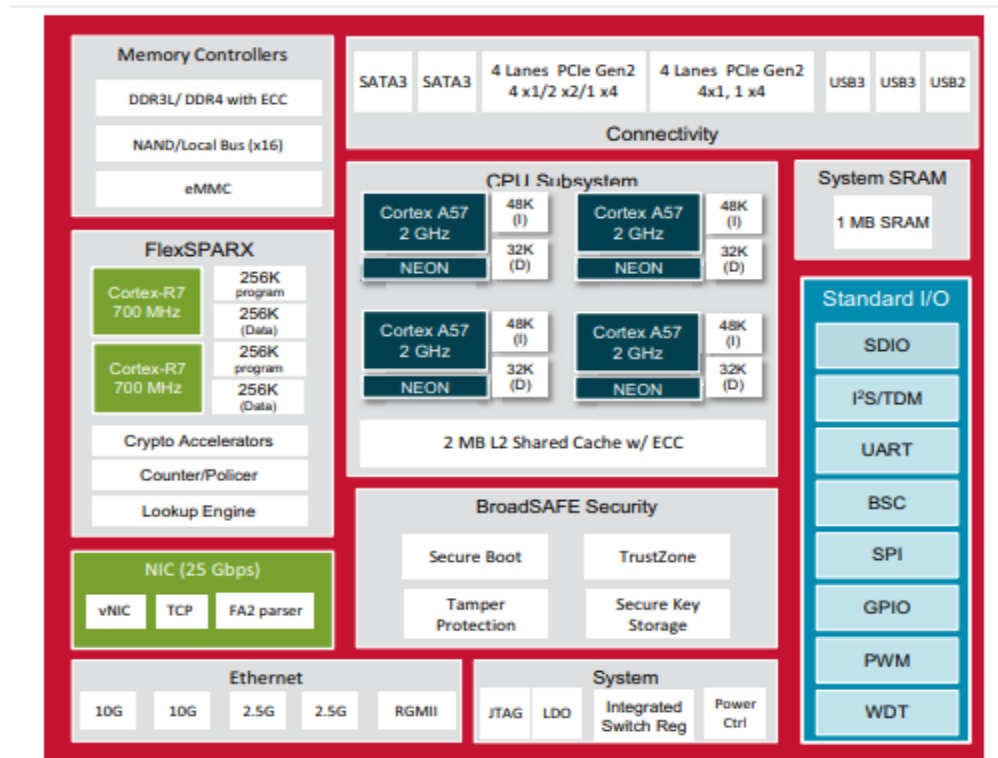


Figure 9-BCM2873 Block diagram

SPECIFICATIONS:

CPU: Quad-core Cortex-A57 64-bit ARMv8 up to 1.8GHz – Two-stage address translation for virtualization with I/O memory management unit (MMU) – 48 KB I-cache and 32 KB D-cache per core – 2 MB shared L2 cache.

Memory: 1MB of platform SRAM – 72-bit wide DDR3/3L/4 memory controller with error correction coding (ECC) (64b data, 8b ECC) up to DDR4-2400.

Flash interface: 4x UART, two BSC, two SPI, MDIO, GPIO (32), Local Bus, TDM.

3.2.2 ATmega2560 Controller

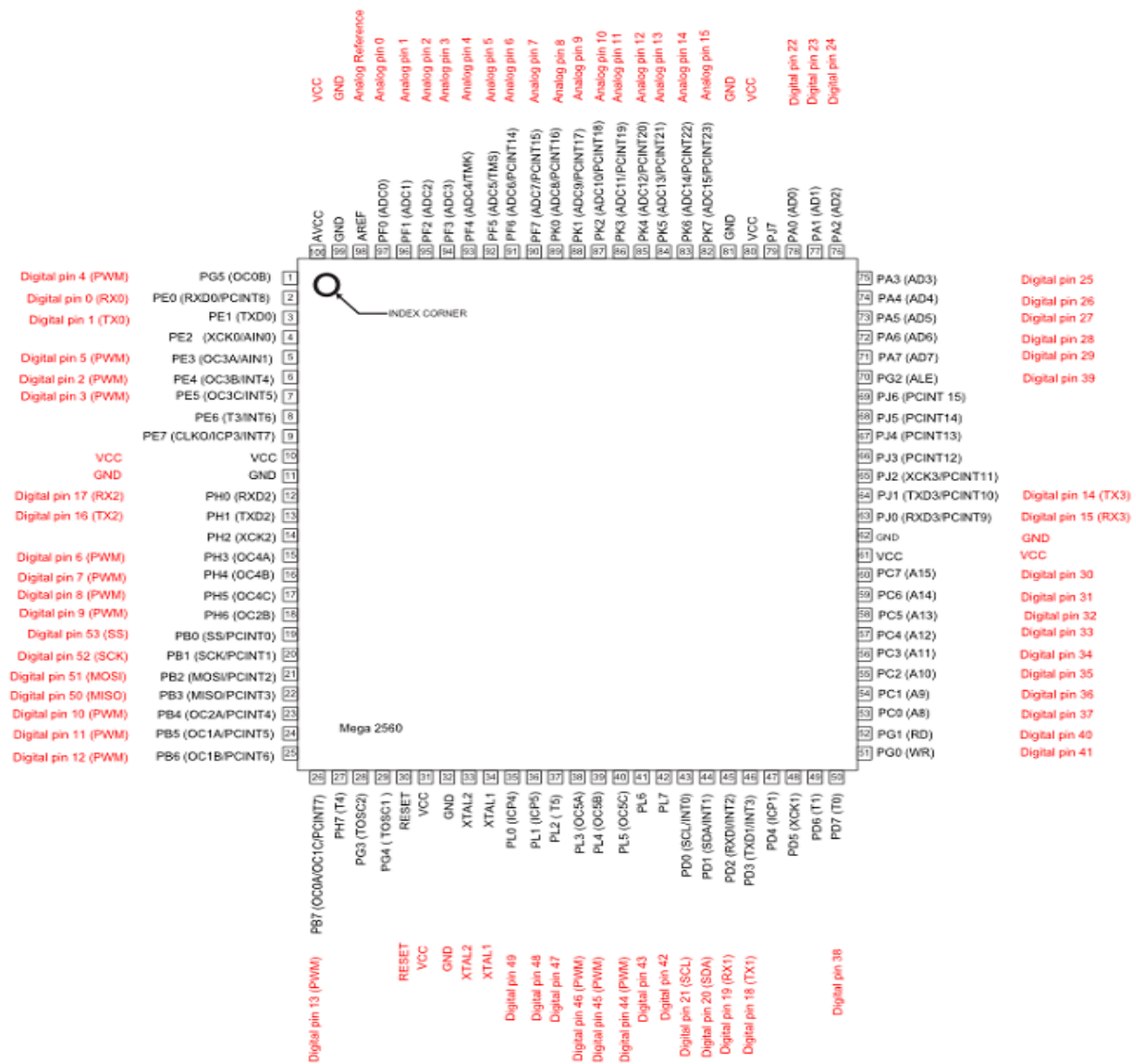


Figure 10-ATmega2560 Pin Layout

SPECIFICATIONS:

1. Advanced RISC Architecture
2. Up to 16 MIPS Throughput at 16 MHz
3. 256K Bytes of In-System Self-Programmable Flash
4. 8K Bytes RAM
5. 4K Byte Internal SRAM
6. 86 Programmable I/O Lines arranged in nine 8 bit ports
7. In-System Programming by On-chip Boot Program
8. JTAG
9. 16-channel, 10-bit ADC
10. Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
11. Four 16-bit Timer/Counter with Separate Prescalers, Compare Mode, and Capture
12. Real Time Counter with Separate Oscillator
13. Twelve 16 bit and Four 8 bit PWM Channels,
14. Four Programmable Serial USART
15. Eight external interrupts
16. Master/Slave SPI Serial Interface
17. Byte-oriented Two-wire Serial Interface
18. Programmable Watchdog Timer with Separate On-chip Oscillator
19. 100 pin TQFP package

3.2.3 5" TFT HDMI DISPLAY

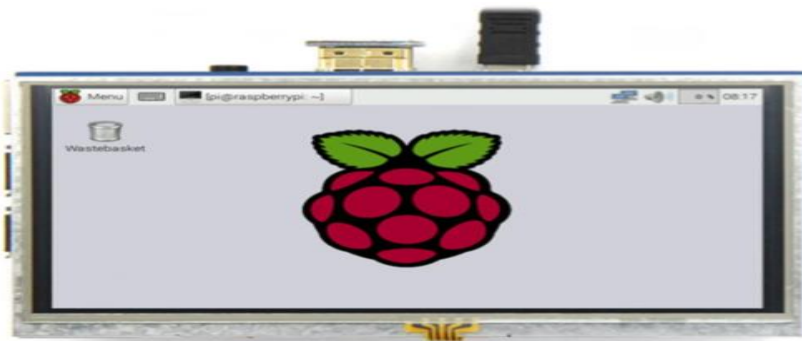


Figure 11-5" TFT display panel

The HDMI display is a resistive touchscreen panel that serves as both display and touch interface. It is a cheap and inexpensive device available in any standard electronics market for a price of 3000 Rs. It can be connected as a display interface to any Raspberry Pi or a development board with HDMI interface.

The following image shows the display along with the pin layout:

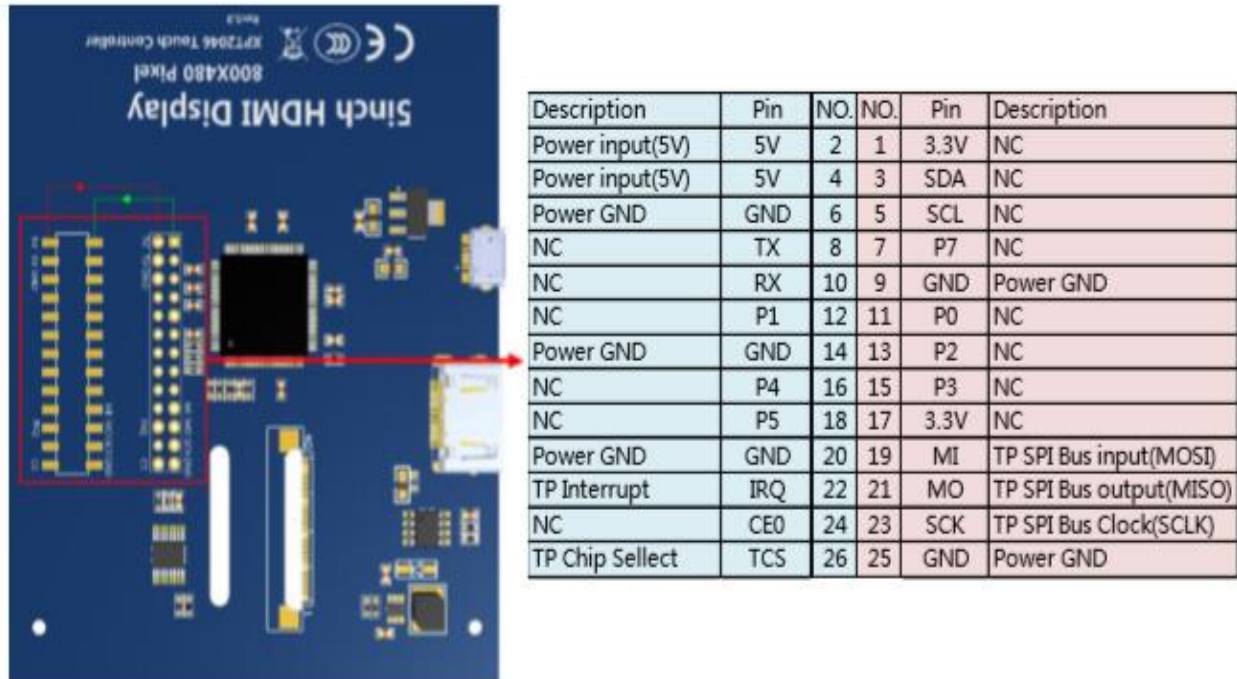


Figure 12-5" Display Pin layout

SPECIFICATIONS:

1. 5" standard display, 800 × 480 resolution
2. With a resistive touch screen, support touch control
3. Support backlight control alone, the backlight can be turned off to save power
4. Supports standard HDMI interface input, compatible with and can be directly inserted with Raspberry Pi (3rd, 2nd, and 1st generation)
5. Can be used as general-purpose-use HDMI monitor, for example: connect with a computer HDMI as the sub-display (resolution need to be able to force output for 800 x480)

6. Used as a raspberry pie display that supports Raspbian, Ubuntu, Kodi, win10 IOT(resistive touch)
7. Work as a PC monitor, support XP,win7, win8, win10 system(do not support touch)
8. CE, RoHS certification.

INTERFACING RASPBERRY PI AND DISPLAY:



Figure 13-Interfacing Display and Raspberry Pi

1. The 5" HDMI display connects to 13 pins of the Raspberry Pi GPIO pins and also connects to the HDMI port of the Raspberry Pi. It also has mounting holes which can be used to fix the Display to the Raspberry Pi using standard 2mm mounting screws.
2. USB Interface: Gets 5V Power from USB, If -13*2 Pin Socket has been connected, that this USB interface can be No Connect.
3. HDMI Interface : for HDMI transmission. Backlight Power switch Controls the backlight turned on and off to save power.
4. 13*2 Pin Socket : Gets 5V Power from raspberry Pi to LCD, at the same time transfers touch signal back to Raspberry Pi.
5. Extended interface : Extended 13*2 Pin Socket signal Pin-to-Pin.

3.2.4 DHT (DIGITAL HUMIDITY AND TEMPERATURE) SENSOR

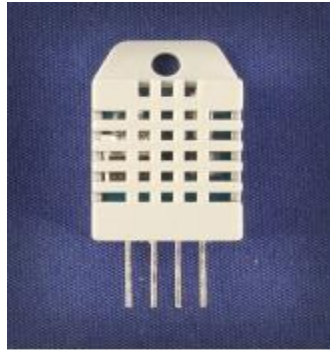


Figure 14-DHT Sensor

The DHT sensor is a cheap and inexpensive Digital humidity and temperature sensor which works on the principle of using a capacitive based dielectric for relative humidity and temperature measurement. The working of the sensor corresponds to the time period of sampling in which the sensor is calibrated to sense the relative humidity and temperature.

SENSOR WORKING DIAGRAM:

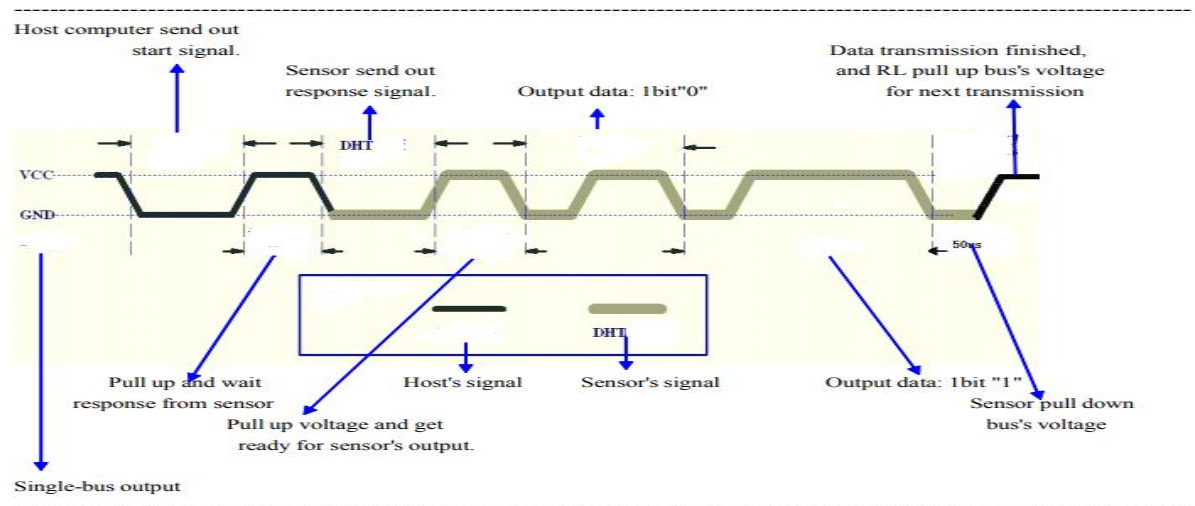


Figure 15-Sensor working diagram

As seen above, the DHT sensor sends a signal and awaits a response from the capacitive material, the time duration taken for a response from the sensor is the digital equivalent to the relative humidity and temperature.

TECHNICAL SPECIFICATIONS:

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of program code in OTP memory when the sensor is detecting, it will cite coefficient from memory.

Small size, low consumption and long transmission distance (20m) enable the DHT11 to be suited to all kinds of harsh application occasions. Single-row packaged with four pins, making the connection very convenient.

Model	DHT11 / DHT22
Power Supply	3.3-6V DC
Output Signal	Digital Signal via a single bus
Sensing element	Polymer humidity capacitor & DS18B20 for detecting temperature.
Accuracy	Humidity $\pm 2\%RH$(Max $\pm 5\% RH$); Temperature $\pm 0.2Celsius$
Resolution	Humidity $0.1\%RH$; Temperature $0.1Celsius$
Repeatability	Humidity $\pm 1\%RH$; temperature $\pm 0.2Celsius$
Humidity hysteresis	$\pm 0.3\%RH$
Long-term Stability	$\pm 0.5\%RH/year$
Sensing period	Average: 2s
Interchangeability	Fully interchangeable

This sensor is connected to the IN-board and the digital bits are samples and also stored in the EEPROM of the controller. The DHT sensors work as inputs in a typical Industrial IoT setup where Real-time data must be sampled and specific parameters must be monitored.

It can be customized and changed according to the required specifications and different sensors can also be used in place of the DHT sensor.

The main advantage of using the DHT sensor is the acquisition of 2 bits of information in one clock cycle of the microcontroller. So, more data can be sampled for the same number of clock cycles as that of other sensors.

3.2.5 Bluetooth module HC-05

HC-05 is an inexpensive Bluetooth module which can easily be interfaced with microcontrollers such as ATmega2560 and is easily available. The module runs a low power wireless BLE RF Transceiver which makes it an energy efficient solution for the embedded system design.

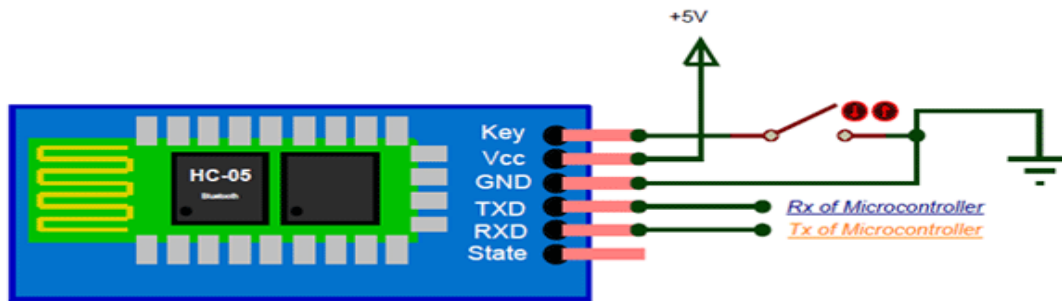


Figure 16-Bluetooth module HC-05

PIN NUMBER	PIN NAME	DESCRIPTION
1	Enable / Key	This pin is used to toggle between Data Mode (set low) and AT command mode (set high). By default, it is in Data mode
2	Vcc	Powers of the module. Connect to +5V Supply voltage
3	Ground	Ground pin of the module, connect to system ground.
4	TX Transmitter	Transmits Serial Data. Everything received via Bluetooth will be given out by this pin as serial data.

5	RX Receiver	–	Receive Serial Data. Every serial data given to this pin will be broadcasted via Bluetooth
6	State		The state pin is connected to onboard LED, it can be used as a feedback to check if Bluetooth is working properly. Indicates the status of Module
7	LED		<ul style="list-style-type: none">• Blink once in 2 sec: Module has entered Command Mode• Repeated Blinking: Waiting for connection in Data Mode• Blink twice in 1 sec: Connection successful in Data Mode
8	Button		Used to control the Key/Enable pin to toggle between Data and command Mode

3.3 HARDWARE DESIGN USING EAGLE

The In-Board and the Out-Board are designed in such a way that both the logical firmware's can be implemented on the same PCB. This PCB is designed using the EAGLE software. To design the PCB layout, the schematic is first made which is corresponding to the controller and peripheral component pinout. To design the schematic, first a working model must be made on a breadboard and the concept must be verified.

The use of designing and fabricating PCB's is so that the project can also be made commercial after customizing to the specific application such as home automation, where switching and display standalone controllers are required. So, only the Out-Board and the processor unit can be used. For an Industrial IoT application, the data acquisition, switching, and display controllers are required. So, the In-Board, Out-board and the processor boards can be used.

This brings about the customizability in the application at hand. The same firmware can be dumped onto the respective controller boards by altering the pinouts and the same application software's can be used by increasing or decreasing the parameters of switching and monitoring.

On creating the schematic circuit of the logic, the same schematic can be converted into a board file and here, the component layout and placement must be made. For doing so, we must first have an idea of the actual component dimensions that need to be taken into account. The Arduino Mega Board is used as a substitute for the ATmega2560 and a Raspberry Pi 3 is used as the main processor unit onto which the display unit fits perfectly as it is a Raspberry Pi specific module. The Arduino Mega pinouts are taken as a footprint file in the design of the controller card. A footprint is basically a vector representation of the number of pins, its placement and pitch distances (Pitch: Least distance between the pins of the controller).

After finalizing all the peripheral footprints, such as the HC-05 module footprint which is a 6 pin through-hole standard 2.54 mils pitch. For conversion, 1mils is equivalent to one-thousandth of an inch which is 0.00254cm. It must be taken care so as to choose the perfect pitch from the component datasheet if not, the component cannot be mounted on the board post-fabrication.

3.3.1 BOARD LAYOUT AND FABRICATION

SCHEMATIC:

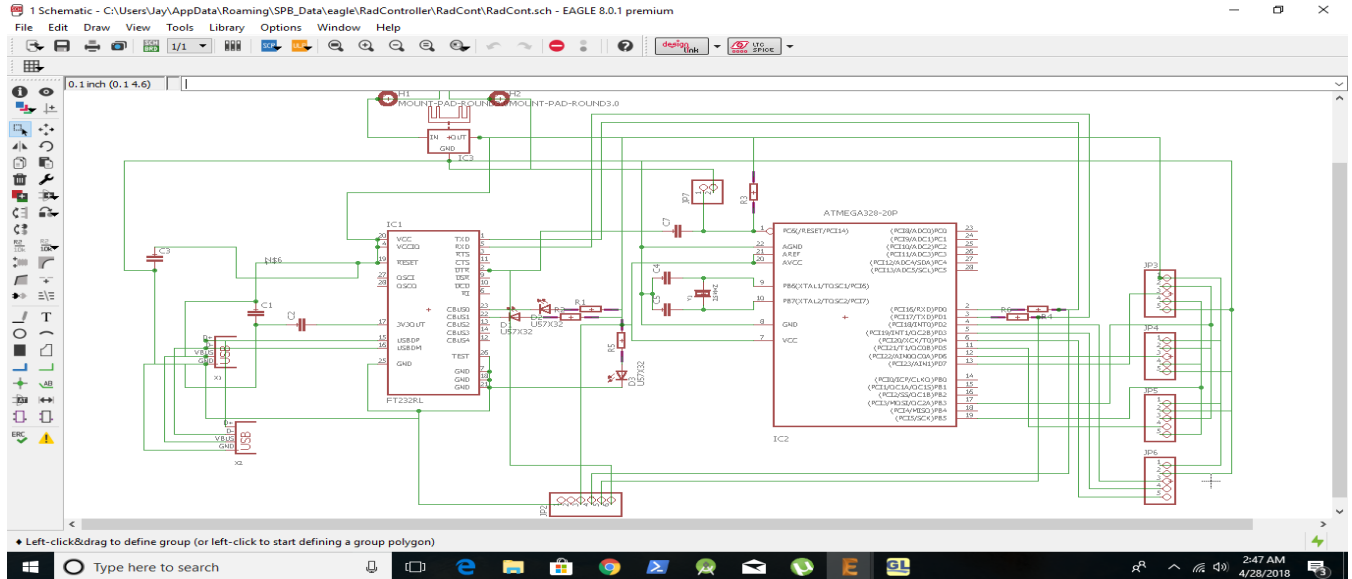


Figure 17-Board Schematic

COMPONENT PLACEMENT:

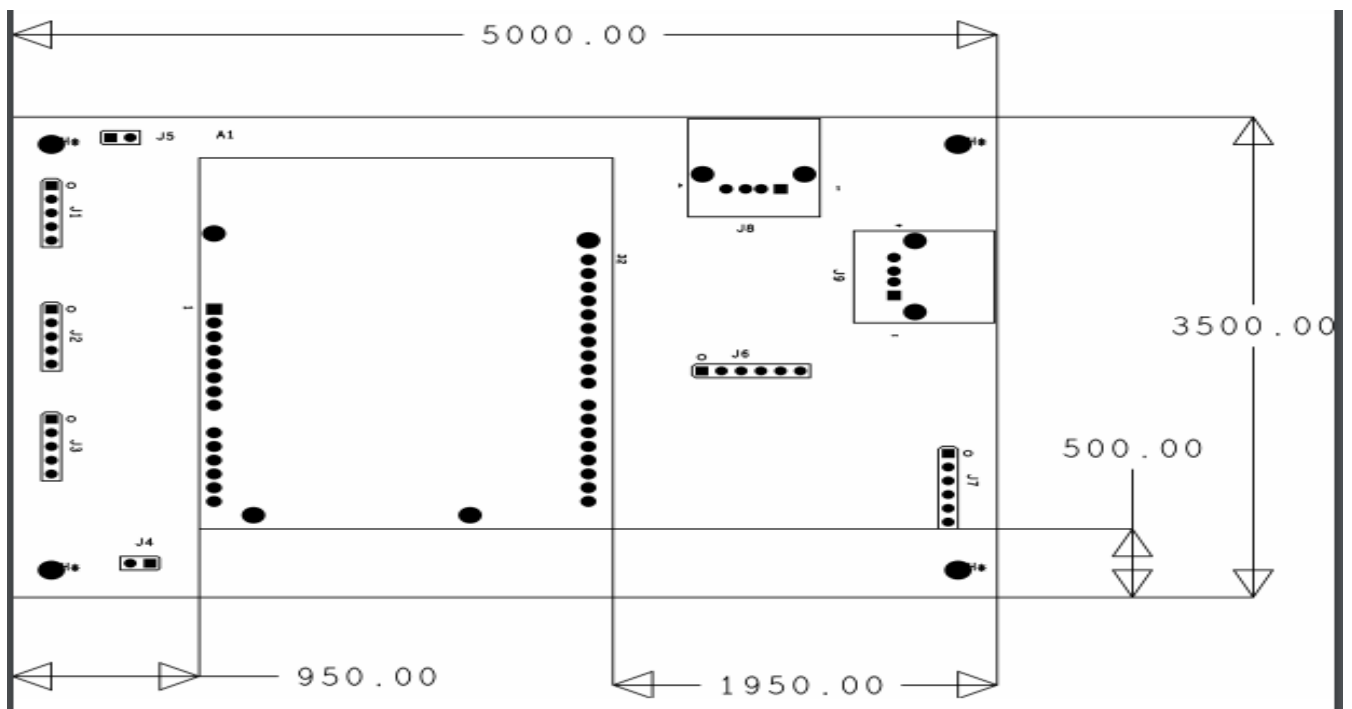
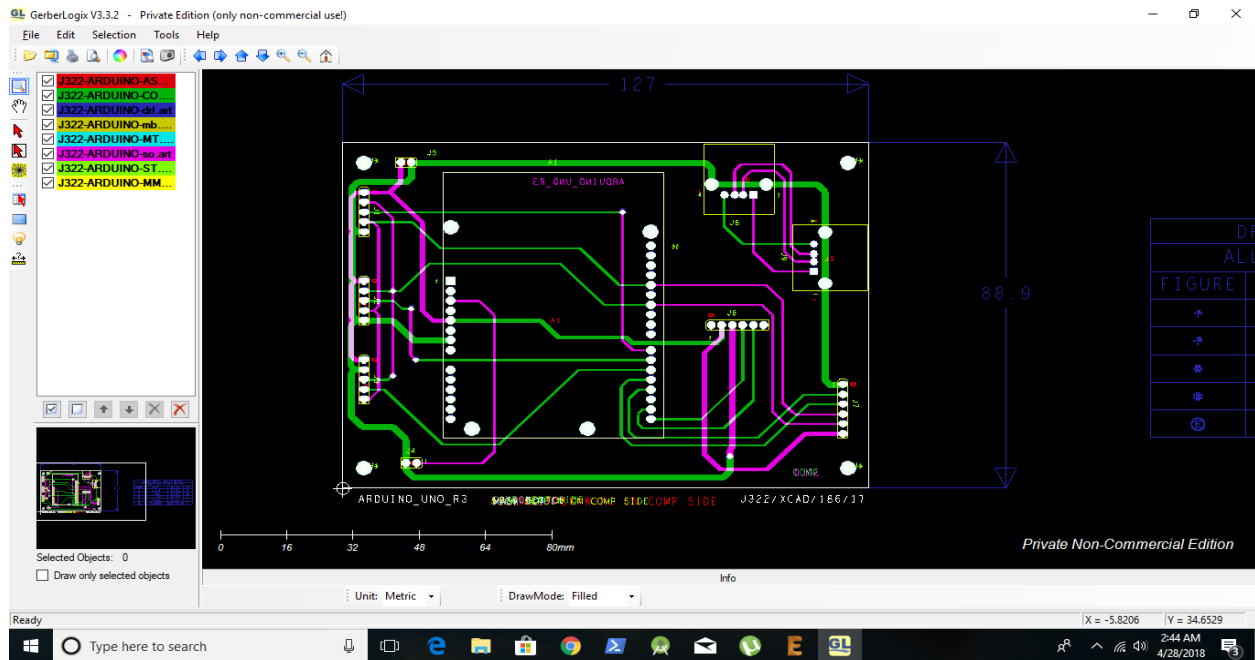
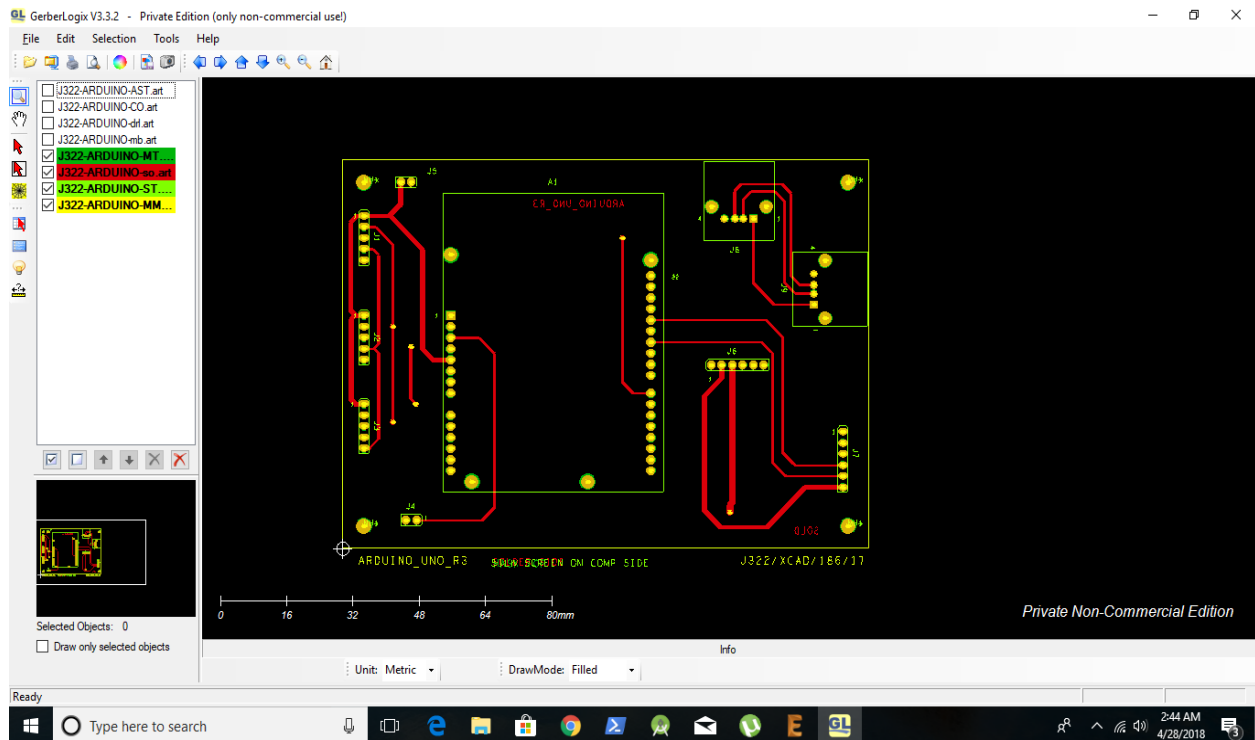


Figure 18-Component Placement

BOARD FILE FOR FABRICATION:



TOP LAYER:



DIMENSIONS IN MILLIMETERS FOR FABRICATION:

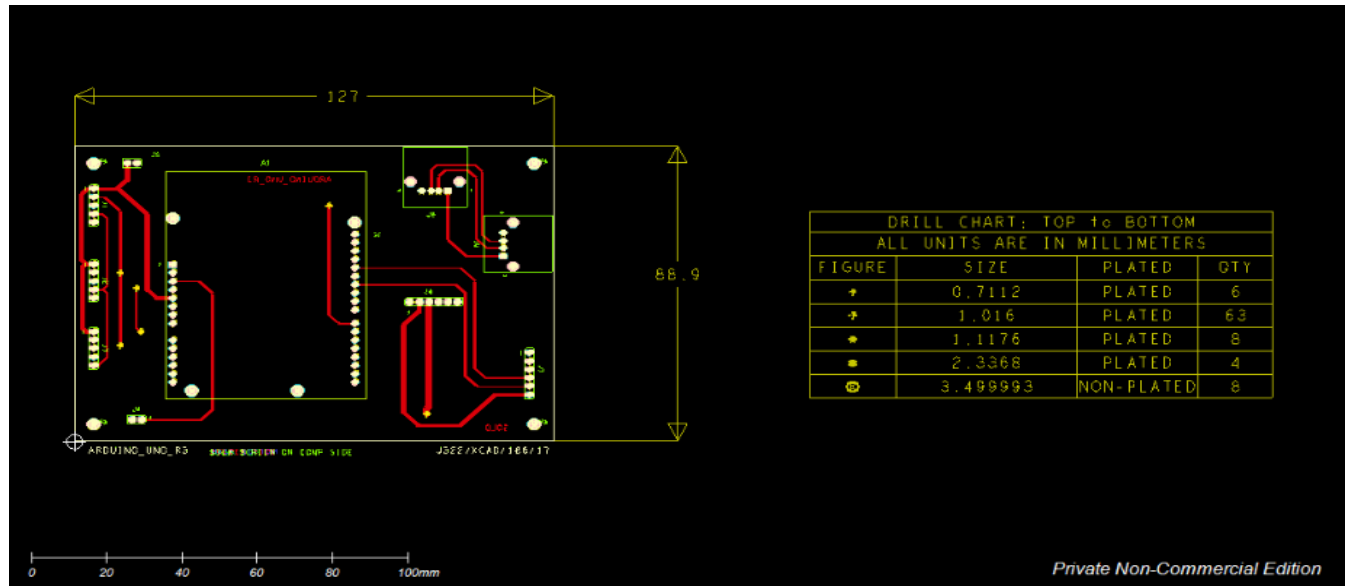


Figure 21-Dimensions

3.3.2 POST FABRICATION

The board file designed in EAGLE was converted to Gerber files which contain 8 layers required for fabrication, Namely Top layer, a bottom layer, Silkscreen, Pads and tracks, holes, and drills, etc.

These files are also called as the CAM files (Computer-aided manufacturing) files, which are required for the CNC machine to create holes, pads, tracks and other precise etching on the perforated copper board. The PCB after processing through the CNC is then dipped into a hot molten lead solution which gets accumulated on top of the board where the etching and CuS (Copper Sulphide) solution is applied to. The PCB is now sent to the green masking table, where the PCB is coated with a layer of green solution which forms a protective layer for the PCB. It prevents shorts, reduces EMI and EMF from external sources and also gives a recognizable color to the PCB. After green masking is done, the PCB is sent to the silkscreen print stage. Here, the PCB is printed with the required lines, dots and component locations. Any required important information can also be printed onto the PCB such as “Arduino UNO R3” which is printed on the In and Out boards.



3.4 ASSEMBLING HARDWARE AND COMPONENT PLACEMENT

1. The DHT11 sensors are connected to the IN-BOARD and the USB-Serial cable is connected to the Raspberry Pi using a USB A type to USB B type cable.
2. The Relays are connected to the OUT-BOARD using a 10 pin RMC (Relay mate connectors) cable. The OUT-BOARD is connected to the Raspberry Pi using a USB A-USB B type serial cable.
3. The display is mounted on the Raspberry Pi and connected to the HDMI port via the 2 way male to male HDMI jack.
4. All the subsystems are mounted on a perforated metal base using metal studs and nuts.
5. The individual firmware's are now dumped onto the In and Out boards and the Processor is loaded with the Real-time batch processing application that initializes on commands from the terminal of the Linux based Debian OS, (a.k.a) Raspbian Jessie.

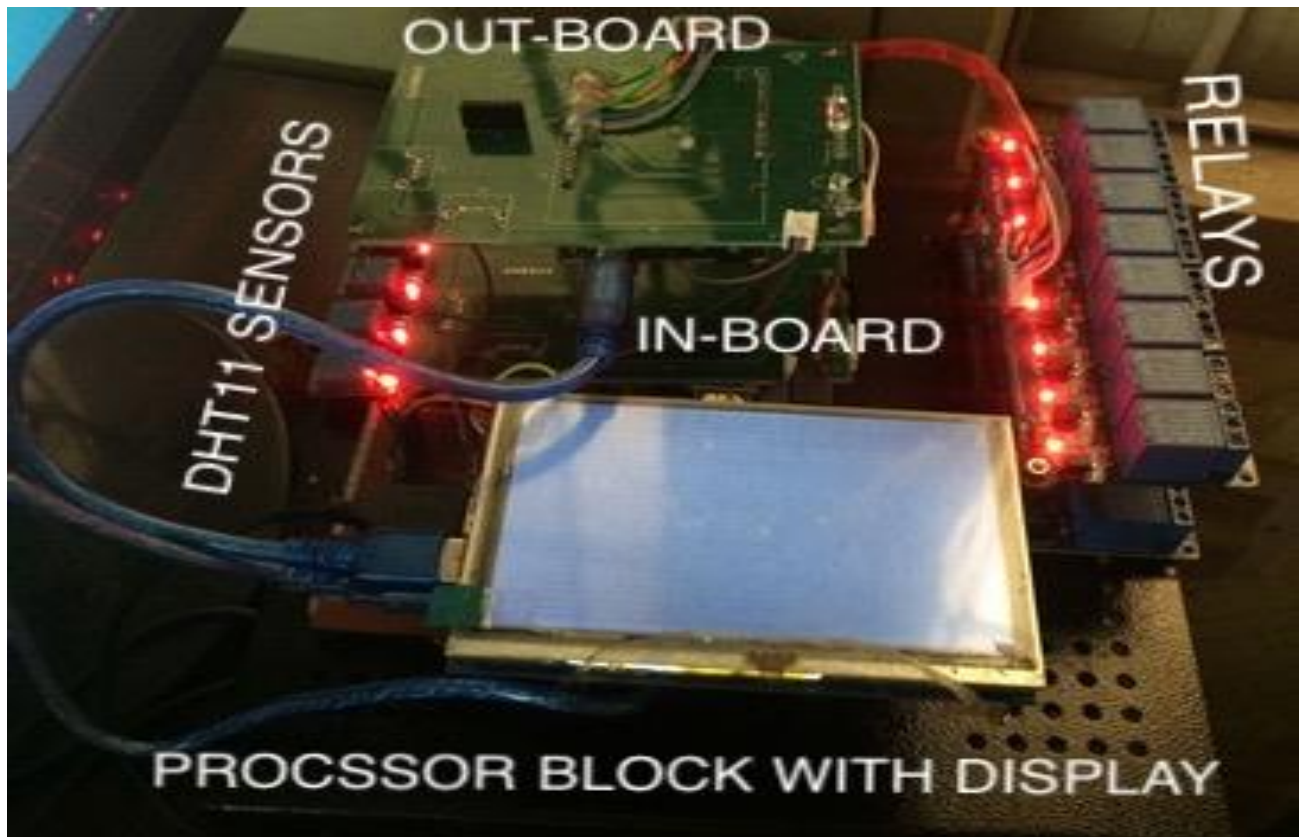


Figure 24-Assembling components

The final peripheral components are assembled in place and the Bluetooth module (HC-05) is also mounted. The Embedded system is now complete and is awaiting user commands for processing and monitoring tasks.

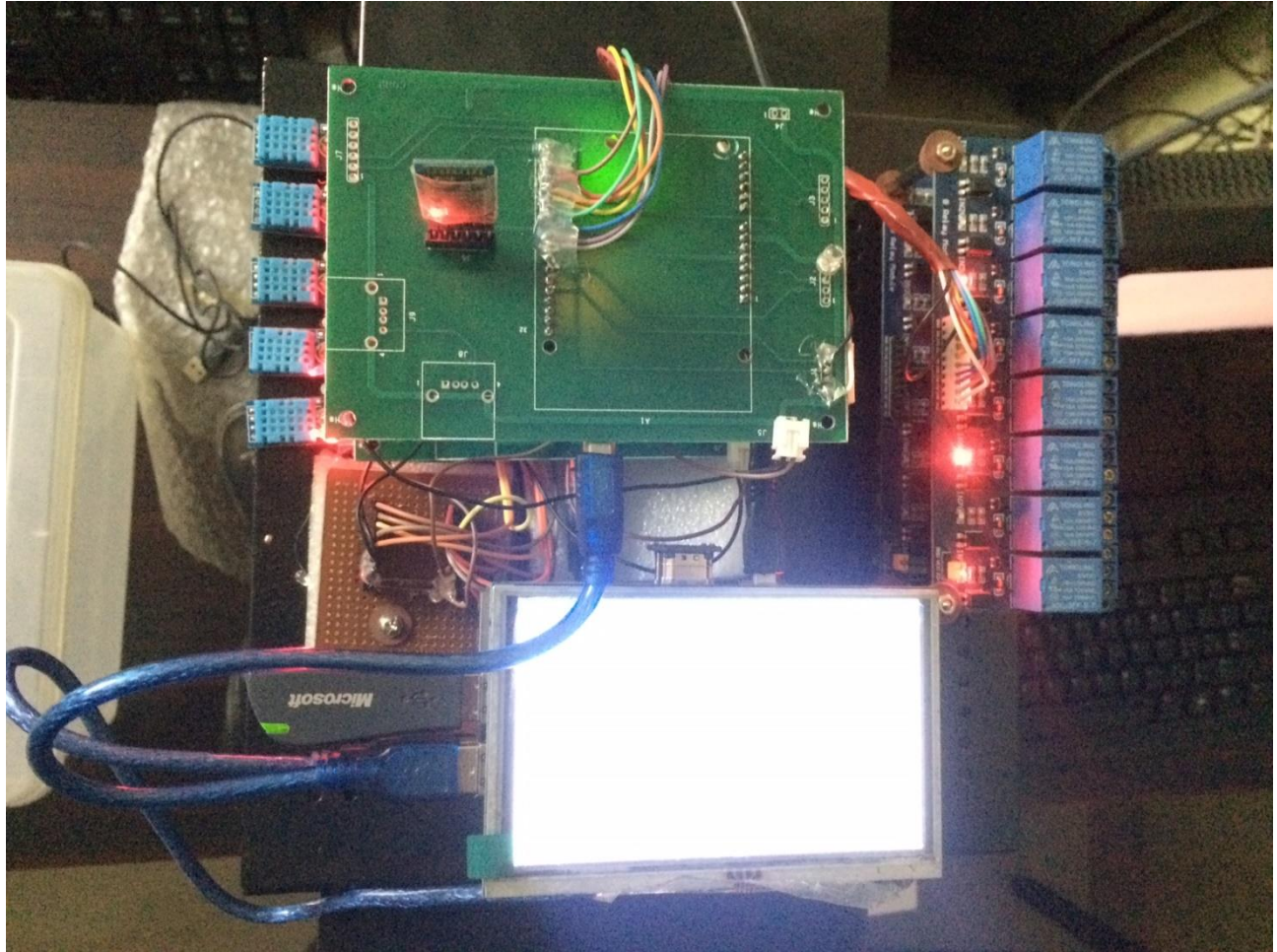


Figure 25-Final Assembly

3.5 SOFTWARE BLOCK

The Software block consists of the different types of software, code and application sources required to build the real-time embedded system.

The different software's are namely, the specific controller firmware's, the processor application kernel and the mobile application development source code (Android Studios).

Different software tools are employed in writing and compiling the code of the software.

3.5.1 TYPICAL CONTROL FLOW

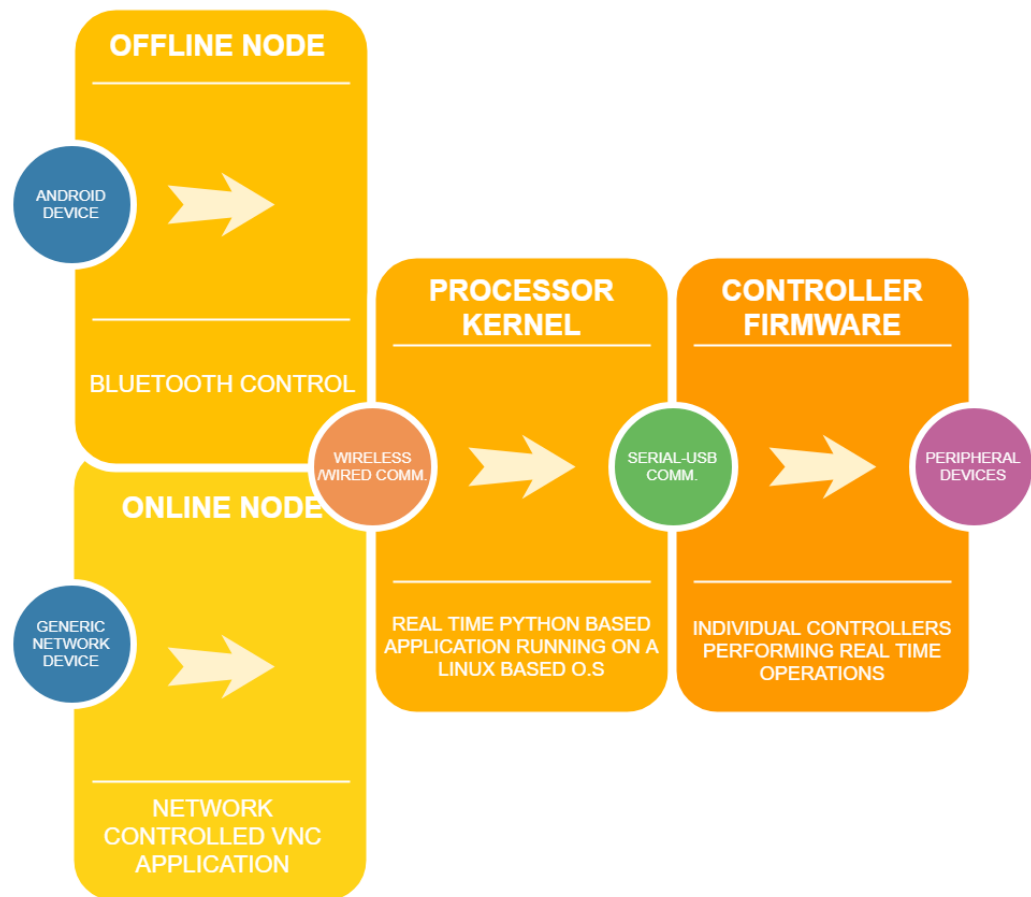


Figure 26-Control flow diagram

1. As shown in the above control flow diagram, the first event trigger can originate from either the offline node or the online node, the offline node being an android device which runs a mobile application that is an instance of the same features of the processor application. The code of the offline node is compiled and emulated in Java using the

Android Studios IDE. A signed apk is generated which can be installed on to any android mobile above the version 4.1 i.e. (Jelly Bean).

2. The online node is any device connected to the same network as that of the processor. By using VNC (Virtual Network Computing), which is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer, we can access the processor application from a remote location and monitor the data logging and switching of relays and LED's.
3. The processor application kernel is a simple Python-based code that runs the real-time data acquisition algorithm which acquires data from the IN-BOARD and it also runs a switching code that can be used to switch Relays and LED's. Both the tasks are accomplished by connecting the Raspberry Pi to the controllers using the USB-Serial connector cables. This application code is written in Python that is interpreted using PiP3 (Pip installs python) using which, the code can be deployed in any platform that supports PiP3.
4. The Controllers have specific firmware's to accept data from the serial port process it and perform the necessary task or to use the serial port and send acquired data. This is a C++ code written on Arduino IDE, an open source platform for compiling and dumping code onto the controllers.

3.5.2 SOFTWARE TOOLS USED

CODE	SOFTWARE USED
Controller Firmware	Arduino IDE
Processor Application	PiP3 (Pip Installs Python) Version 3
Online Application	Processor application running on VNC
Offline Application	Android Studios

3.5.3 CONTROLLER FIRMWARE

IN-BOARD FIRMWARE:

This code must only perform the task on continuously acquiring data from the 5 DHT (Digital Humidity and Temperature) sensors and sending it to the Processor via the USB-Serial interface.

The code must also be written keeping the baud rate (data transfer rate) in mind, so to obtain a baud rate of 2000 samples/ minute, there must be absolutely no delay in acquiring data and send it via the USB-COM port. The only delay that is allowed is the clocking delay of taking data from each sensor one after the other.

The different libraries used in the IN-BOARD are:

1. HCMAX7219.h: 7 segment LED display library
2. SPI.h: Serial Port Interface library for displaying to the 7 segment LED's serially
3. DHT.h: The Digital Humidity and Temperature library which contains the memory addresses of the capacitive data coefficients of the DHT sensors in order to acquire and sample data.

The data acquired is sampled in a way that can be understood during data transfer. The Bits must be sent in an array of Strings. The array arrSend[] is declared as the data transfer array in the code. The array has a dimension of 15(0-14), wherein the first bit represents the ON/OFF bit i.e. 1/0 of the device, the next 5 bits are the Temperature bits from the 5 DHT sensors, the next 5 bits are the Relative Humidity bits from the 5 DHT sensors, the next bit represents the Tavg bit, which is the average temperature of the 5 bits, the next bit is the Havg bit, which represents the average relative humidity of the 5 humidity bits. The next two bits are also On/OFF bits used to signify that both the Temperature and Humidity values are samples from all 5 sensors.

CODE SNIPPET (DATA ACQUISITION):

```
for(i=0;i<5;i++)
{
    h[i] = dht[i].readHumidity();
    t[i] = dht[i].readTemperature();
    arrSend[i+1]=t[i];
```

```
arrSend[i+6]=h[i];  
if (isnan(t[i]) || isnan(h[i]))  
{  
    t[i]=h[i]=0;  
    Serial.print("Failed to read from DHT at Sensor =");  
    Serial.print(i+1);  
}  
}
```

OUT-BOARD FIRMWARE:

This firmware must perform the task of receiving data from the processor for switching of relays and LED's and should be configurable for future applications. The firmware must also be equipped with ports for connecting the 7 segment LED panels and data transfer to other peripheral devices using SPI protocol.

CODE SNIPPET (Serial Port request handling):

```
void loop()  
{  
    Serial.flush();  
if (Serial.available() > 0)//If serial port is open  
{string = "";}//input string=0  
while(Serial.available() > 0)  
{  
    command = ((byte)Serial.read());//read a chunk of bytes from serial port  
if(command == ':')//If serial read() gives a ":" symbol, then stop reading  
{  
    break;  
}  
else  
{  
    string += command;//Keep accumulating the String "string" with next bits.  
}  
    delay(1);//Delay reading by 1 millisecond.  
}  
}
```

This code shows how data is read and stored from the serial port.

The respective tasks to be performed, such as LED On/OFF, Relay ON/OFF is done in this way, by using a chunk of bytes sent via Serial port.

3.5.4 PROCESSOR APPLICATION AND KERNEL

The processor must run the real-time data acquisition and batch processing system what can communicate with both the IN-BOARD and OUT-BOARD and also connect to the Online Node of the IoT Gateway.

APPLICATION FRONT END:

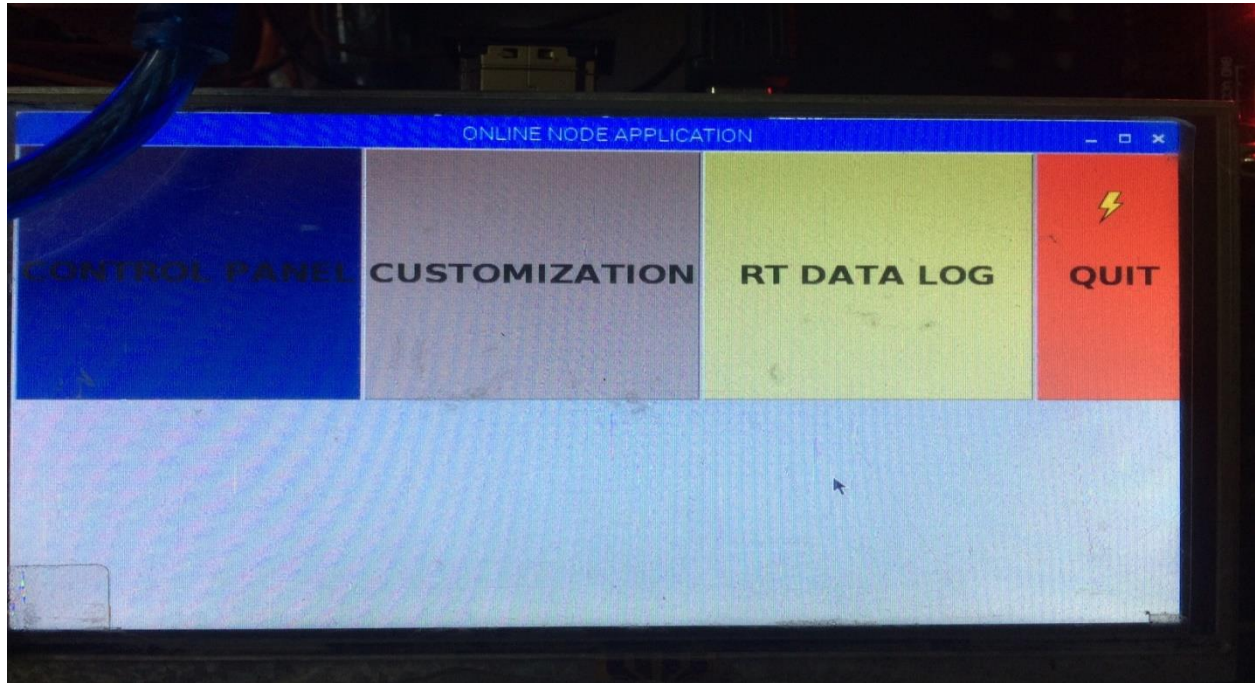


Figure 27-Online application front end

1. The Control Panel is the window which can be used in switching ON/OFF all devices such as LED's and Relays.
2. The Customization window can be used to customize parameters such as input/output checking, for example, if the temperature of T1 bit is >35 deg Celcius, switch off Relay 1 and 2. This kind of customizability is used for Industrial IoT applications.
3. The RT data log window acquires data from the IN-BOARD, logs it in the Micro SD card present in the device, and displays it to the user for monitoring based applications. A futuristic dashboard can also be made using the same real-time log.
4. The quit option quits the Online Node application kernel.

3.5.5 ONLINE APPLICATION

The VNC client is set up in any device connected to the same network as that of the processor and the IP address and port number of the processor is obtained by typing the instruction “ifconfig” in the terminal of the processor Operating System. These details are taken and fed into the VNC client of any system and made to connect automatically.

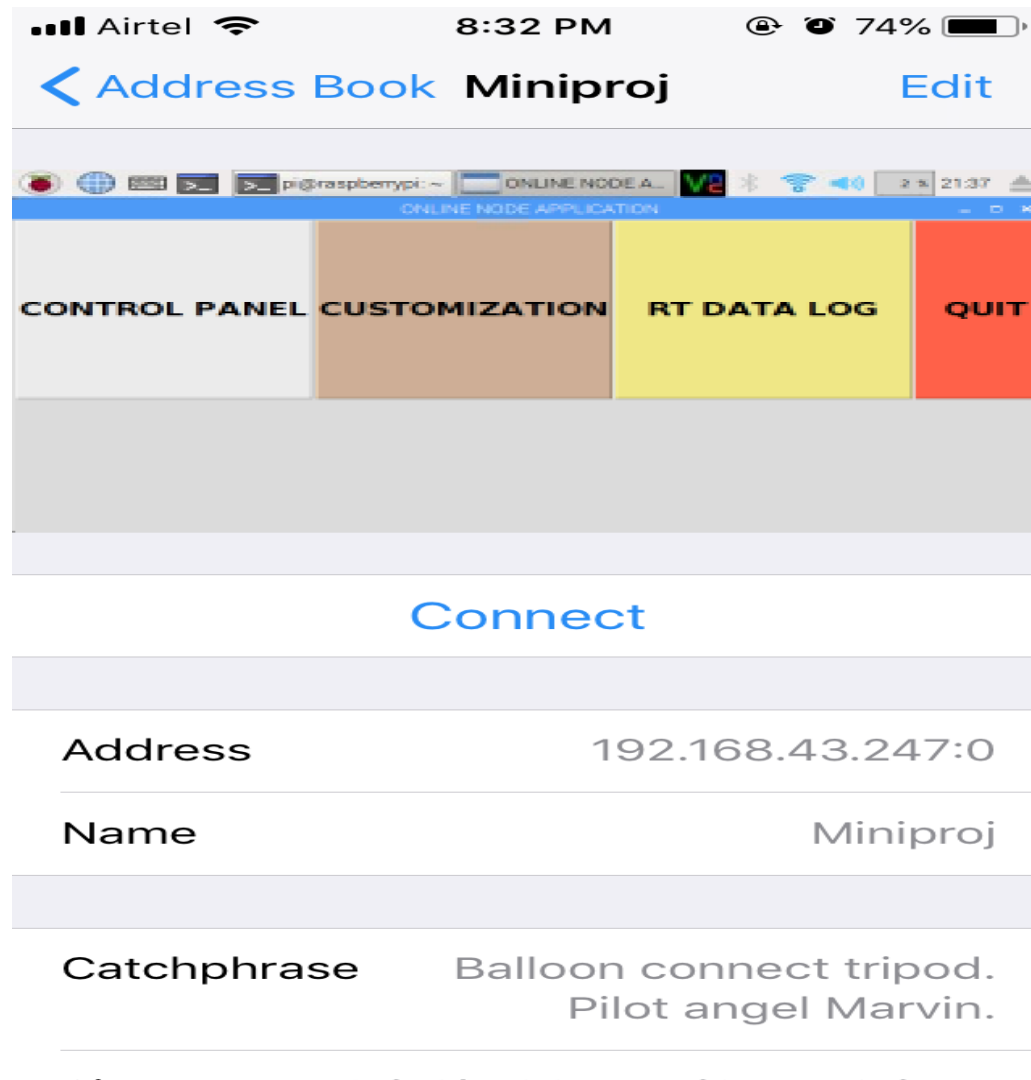


Figure 28-VNC Terminal application implementation

As seen above, the same processor application is accessible from any device connected to the same network. If the device is a remote node, then a global IP address is required for accessing the application. As seen above the VNC client recognizes the application in the local IP address of 192.168.43.247 and in Port 0.

3.5.6 OFFLINE MOBILE APPLICATION

The Offline Application is a mobile application which runs on any generic android device with version >4.1 (Jellybean). This application can be deployed to any android device and can communicate with the Bluetooth module of the OUT board and directly switch the LED's and the Relays. As seen below, the opening page of the application is designed and emulated.

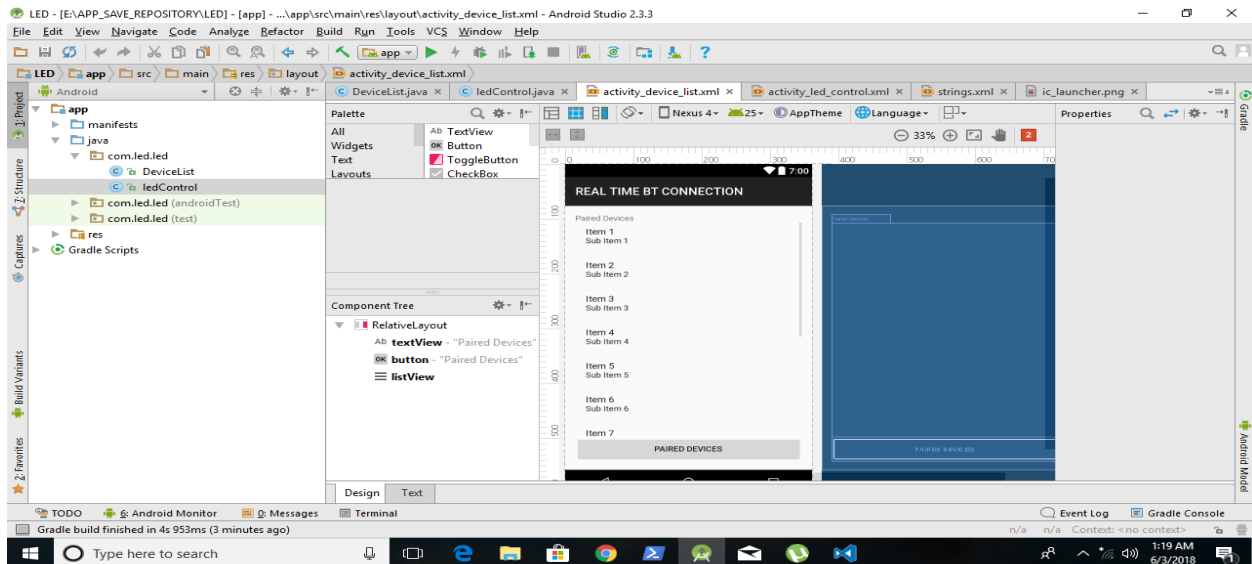


Figure 29-XML Emulated design

The Main page of the application is designed with Text boxes and Buttons.

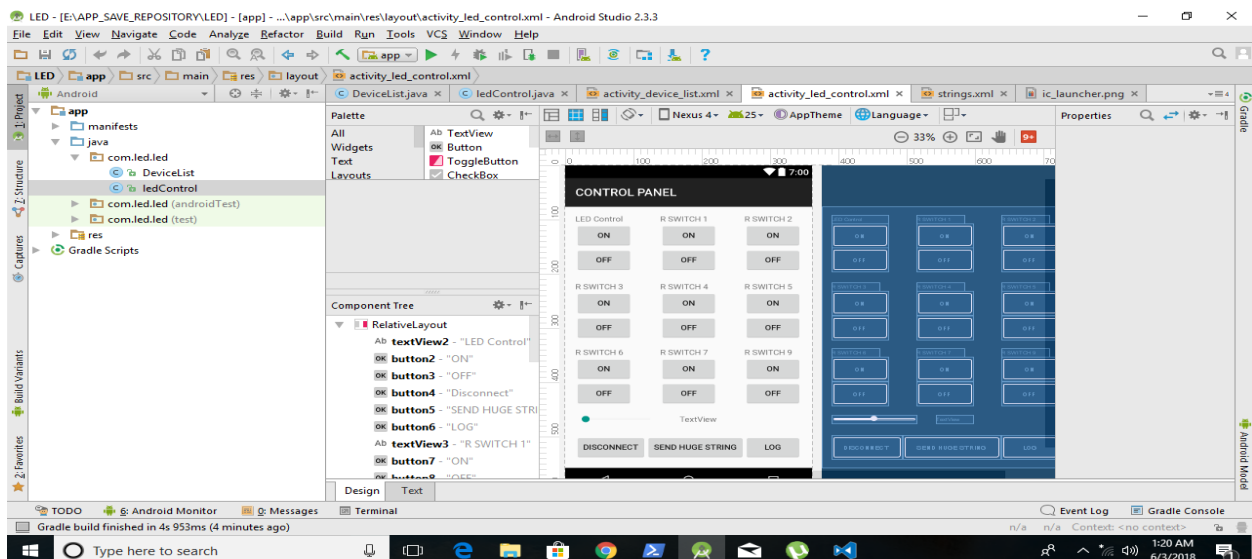
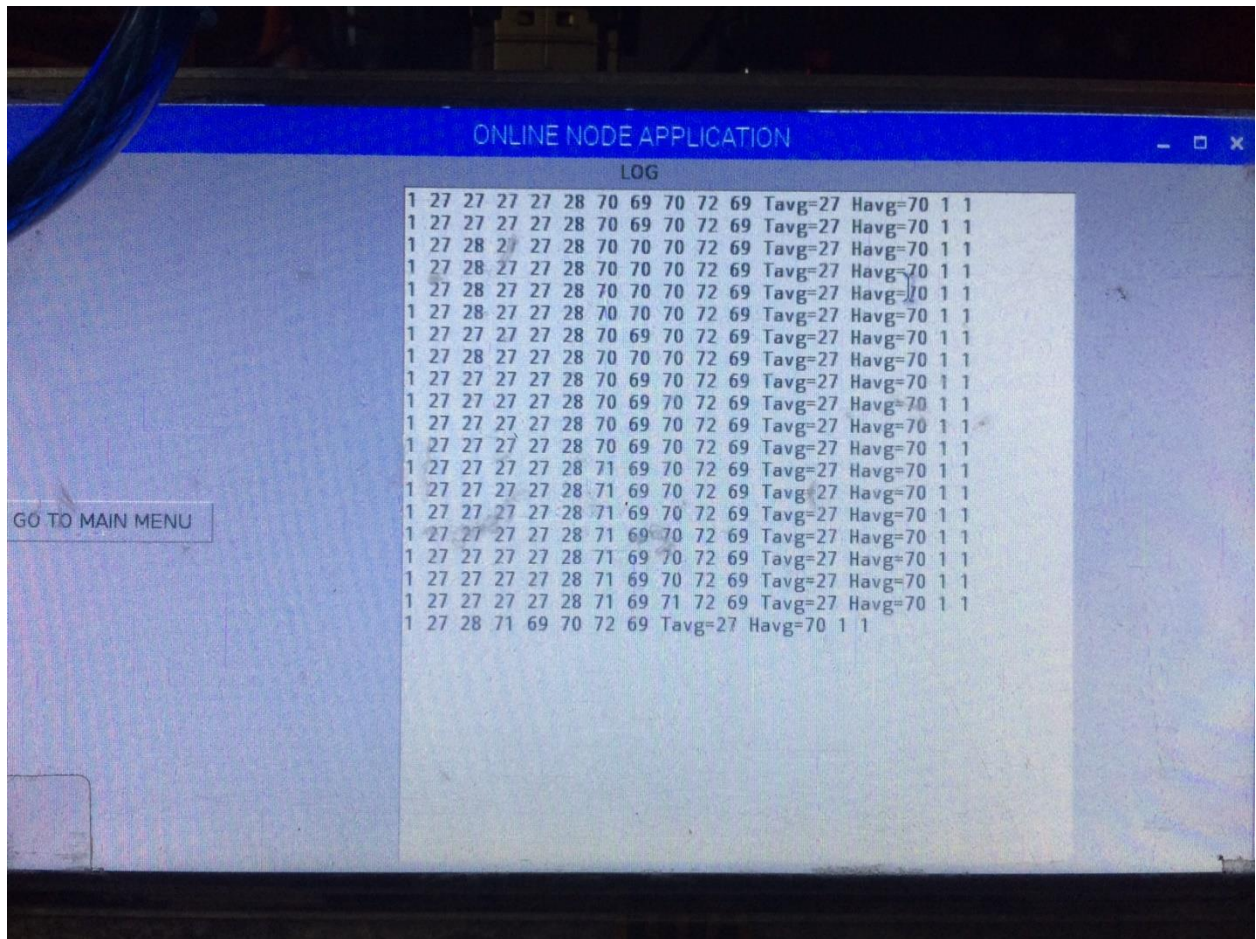


Figure 30-Main page emulation

4 CHAPTER 4: RESULTS AND OUTPUTS



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1



4.2 OUT-BOARD RESULTS

The OUT-BOARD is connected to an LED and 8 channel relays. So, the output from these peripherals can be viewed as a result of event triggers from online and offline IoT nodes.

The trigger parameters are predefined in the code as follows:

LED ON/LED OFF	“TO”/”TOFF”
RELAY1 ON/OFF	“RSON1”/”RSOFF1”
RELAY2 ON/OFF	“RSON2”/”RSOFF2”
RELAY3 ON/OFF	“RSON3”/”RSOFF3”
RELAY4 ON/OFF	“RSON4”/”RSOFF4”
RELAY5 ON/OFF	“RSON5”/”RSOFF5”
RELAY6 ON/OFF	“RSON6”/”RSOFF6”
RELAY7 ON/OFF	“RSON7”/”RSOFF7”
RELAY8 ON/OFF	“RSON8”/”RSOFF8”

4.21 RELAY AND LED OUTPUTS

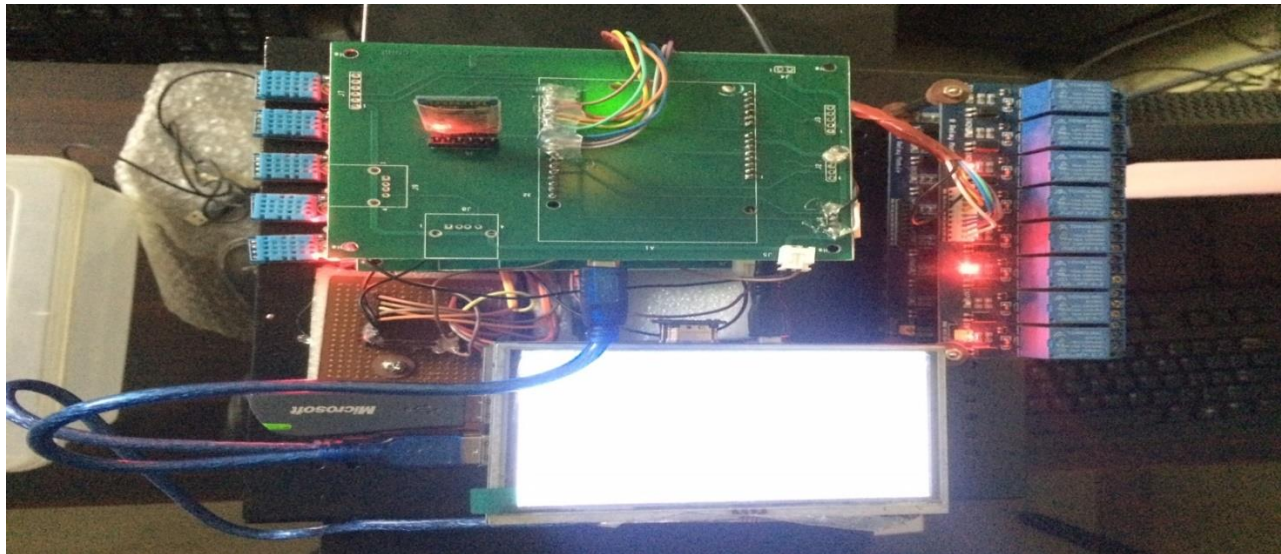


Figure 33-Relay 1,2,6-OFF , LED -OFF

Here, relays 1,2 and 6 are made to be OFF and LED is also OFF

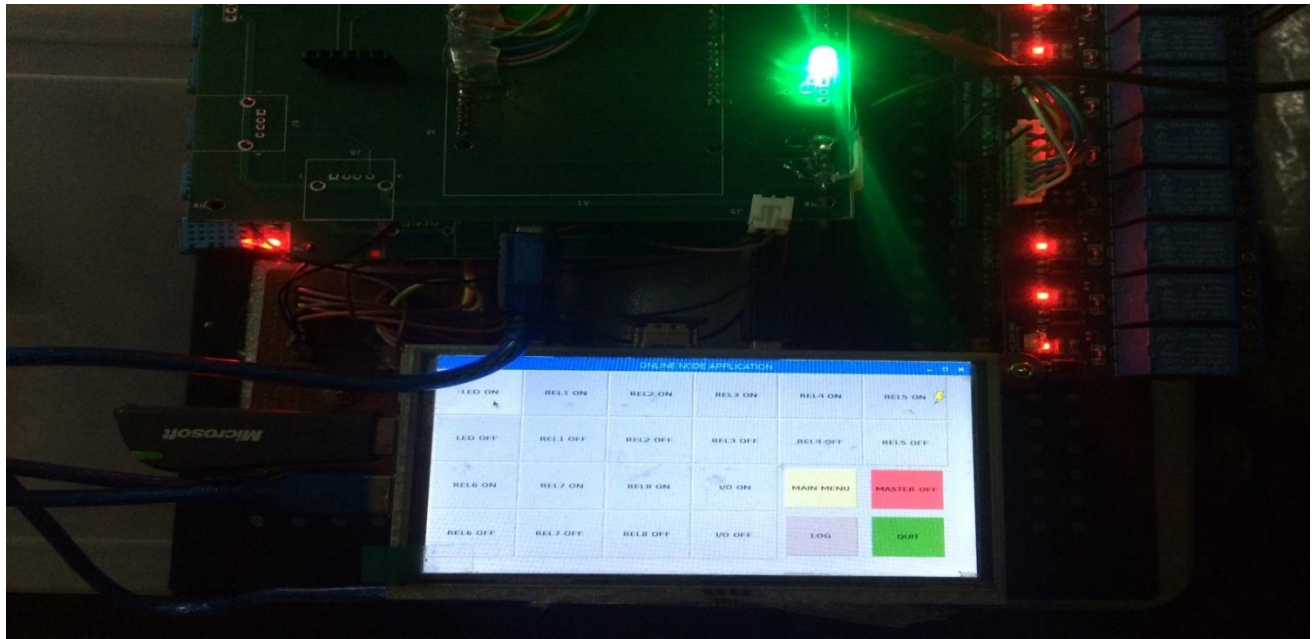


Figure 34-Relay1-8 ON,LED-ON

Here, all the relays from 1-8 are made to be on and LED is also ON. This is an event trigger from the processor.

4.22 EVENT TRIGGERS FROM PROCESSOR (ONLINE) AND OFFLINE NODES

As seen in the results, the Relays and LED must be controlled either by the processor (Online) or a mobile device (Offline) node.

PROCESSOR (ONLINE) TRIGGER:

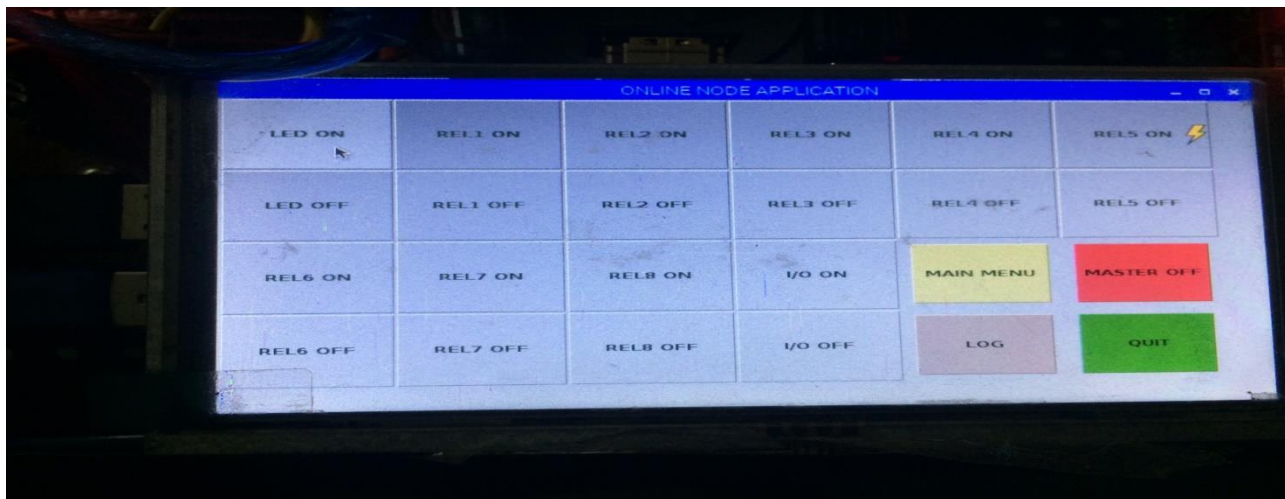


Figure 35-Online trigger

As seen in the picture, the LED ON button is clicked and the processor sends a signal via, the Serial port and the LED on the OUT-BOARD turns on.

OFFLINE TRIGGER:

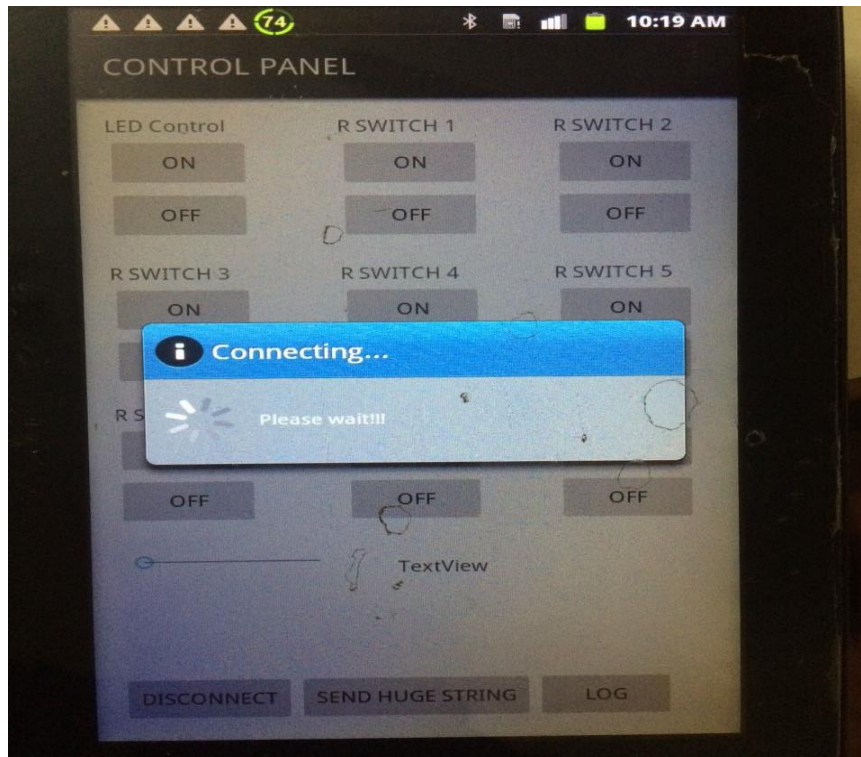


Figure 36-Offline trigger

As seen above, the mobile application is used for sending the signals via Bluetooth to the embedded system. The OUT-BOARD Bluetooth module connects directly to the Mobile device once it is initialized.

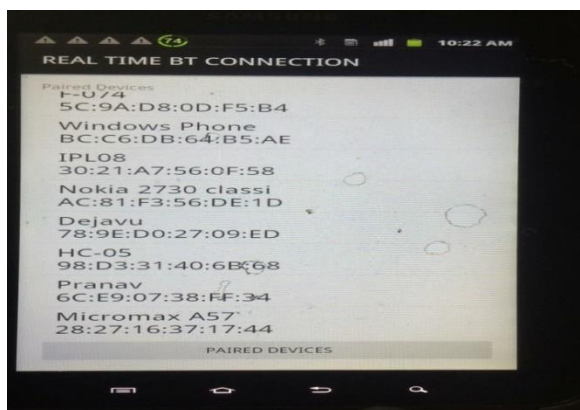


Figure 37-Android app

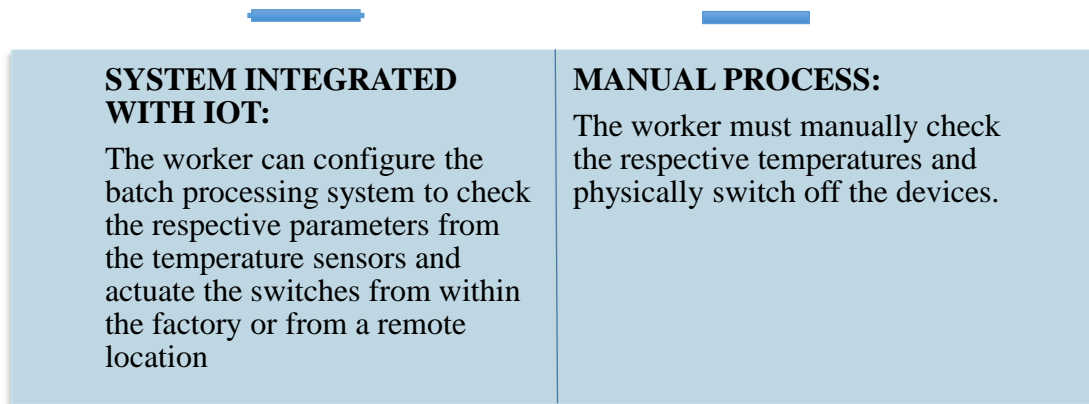
As seen in the picture, the device HC-05 is connected to the application running on the Android device.

Once, it is connected to the device, it connects automatically the next time the Bluetooth module is plugged into the OUT-BOARD.

5 CHAPTER 5: USE-CASE APPLICATIONS

1. INDUSTRIAL IoT

A Real-time industrial application of the project would be in an industry which requires a batch processing system that acquires data from multiple locations, processes it in the form of a batch and performs the required tasks. Example: A manufacturing industry worker needs to turn off 3 devices such as a hydraulic press, CNC drill, and a grinder once the core temperatures of the devices reach 32°C, 45°C and 80°C respectively.



SYSTEM INTEGRATED WITH IOT: The worker can configure the batch processing system to check the respective parameters from the temperature sensors and actuate the switches from within the factory or from a remote location	MANUAL PROCESS: The worker must manually check the respective temperatures and physically switch off the devices.
---	---

Figure 38-Industrial IoT vs Manual process

2. HOME AUTOMATION

Smart homes are a growing trend in the present day and age. A home can be completely retrofit with the embedded hardware present in the project. A home can be made completely automated and also, the user must be able to monitor and control his/her home from a remote location.

A few of the main applications in the home automation sector is automatic switching, automatic dimming, and dipping of lighting and smart metering. Also, to bring about energy efficient power consumption solutions by using lights with proximity sensors at home. So, only when there is a person in a particular room, the lights and electrical devices will turn on, else they will turn off automatically.

3. SMART FARMING

Smart irrigation systems based on sensor data input. DHT (Digital Humidity and temperature) sensors can be used to predict and determine the water content present in the soil.

4. WEARABLES

Real-time monitoring of body parameters can be done using the same hardware. The parameters such as heart rate, body temperature, and step count can be measured on a wearable device implemented using the same system.

GPS connectivity for an on-board wearable device greatly benefits the visually challenged.

5. ADAS

(Advance driver-assistance systems)

Very commonly used technology in vehicles in today's world to assist the driver on road and provide multiple safety features.

6 CHAPTER 6: CONCLUSION AND SCOPE OF THE PROJECT

CONCLUSION:

The project is designed and implemented as described and, all test cases for IN-BOARD and OUT-BOARD are checked and verified for applications such as Industrial IoT, Home Automation, and Smart monitoring applications.

The project has also resulted in the design and fabrication of generic PCB's which can be used for different applications. A generic application is deployed on the processor which can be tailored for various use-case applications of the project.

A mobile application is also implemented which has a signed apk, that can be installed on any generic mobile device >Android 4.1 (Jelly Bean). This can be used to generate event triggers and communicate with the controller boards independently.

FUTURE SCOPE:

The data acquired from the controller can be sent to a cloud-based platform for storage and for creating a real-time dashboard. Using the real-time data on the cloud, a Machine learning algorithm can be developed and implemented on the cloud for performing even more intuitive tasks such as predictive modeling, speech-text conversion, and Image recognition and so on.

The processor application can be implemented with a machine learning algorithm that is specific to each application and can be tailored for a particular user who is not connected to the Internet. The Advanced driver assistance system can be modeled using this perspective.

The IoT devices can be connected to various display devices such as 7 segment displays and dot matrix display, for which the libraries and hardware is pre-designed in the embedded system.

7 BIBLIOGRAPHY

BIBLIOGRAPHY

IoT Application links:

- [1] <https://www.elprocus.com/building-the-internet-of-things-using-raspberry-pi/>
- [2] <https://www.silvan.co.in/projects-products.php>
- [3] <https://iot-analytics.com/10-internet-of-things-applications/>

Papers referred:

- [4] <http://www.engpaper.com/iot-2016.htm> (IEEE Journal)
- [5] <http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?reload=true&punumber=6488907>

Hardware (PCB) design using EAGLE:

- [6] <https://learn.sparkfun.com/tutorials/using-eagle-schematic>
- [7] <https://www.youtube.com/watch?v=1AXwjZoyNno>

Mobile Application Development:

- [8] <https://developer.android.com/training/basics/firstapp/> (Android app development tutorials and guide to designing a basic first application).
- [9] <https://developer.android.com/guide/topics/connectivity/bluetooth> (Android app development guide for Bluetooth connectivity and related features).

APPENDIX

A1 APPENDIX A: IN-BOARD FIRMWARE (C++ CODE)

```
#include <HCMAX7219.h>
#include "SPI.h"
#include "DHT.h"
#define DHTTYPE DHT11
const int sensorMin = 0;
const int sensorMax = 1024;
#define LOAD 7

DHT dht[5]{{2, DHTTYPE},{3, DHTTYPE},{4, DHTTYPE},{5, DHTTYPE},{6,
DHTTYPE}};
HCMAX7219 HCMAX7219(LOAD);

void setup() {

  Serial.begin(9600);
  dht[1].begin();dht[2].begin();dht[3].begin();dht[4].begin();dht[5].begin();
  // delay(1000);

}

void loop() {
  HCMAX7219.Clear();

  int i;int h[5],t[5],havg,tavg,arrSend[15];
  int ta,ha,remt,remh;
  int sensorReading1 = analogRead(A0);
  int sensorReading2 = analogRead(A1);
  int range1 = map(sensorReading1, sensorMin, sensorMax, 0, 3);
  int range2 = map(sensorReading2, sensorMin, sensorMax, 0, 3);
  arrSend[0]=1;
```

```
if ((range1==0) || (range1==1))
{

    arrSend[13]=1;
}
else
{
    arrSend[13]=0;
}
if ((range2==0) || (range2==1))
{

    arrSend[14]=1;
}
else
{
    arrSend[14]=0;
}

for(i=0;i<5;i++)
{
    h[i] = dht[i].readHumidity();
    t[i] = dht[i].readTemperature();
    arrSend[i+1]=t[i];
    arrSend[i+6]=h[i];
    if (isnan(t[i]) || isnan(h[i]))
    {
        t[i]=h[i]=0;
        //Serial.print("Failed to read from DHT at Sensor =");
        //Serial.print(i+1);
```

```
    }  
}  
  
tavg=(t[0]+t[1]+t[2]+t[3]+t[4])/5;  
  
havg=(h[0]+h[1]+h[2]+h[3]+h[4])/5;  
  
arrSend[11]=tavg;  
  
arrSend[12]=havg;  
  
//Serial.println("The Combined array is = ");  
for(i=0;i<15;i++)  
{  
    Serial.print(arrSend[i]);Serial.print(" ");  
    if(i==10){Serial.print("Tavg=");}  
    if(i==11){Serial.print("Havg=");}  
}  
Serial.println();  
ta=0;ha=0;  
while(arrSend[11]!=0)  
{  
  
    remt = arrSend[11]%10;  
    ta = ta*10 + remt;  
    arrSend[11] /= 10;  
  
}  
while(arrSend[12]!=0)  
{
```

```
    remh = arrSend[12]%10;
    ha = ha*10 + remh;
    arrSend[12] /= 10;

}

// Serial.println(ta);
// Serial.println(ha);
HCMAX7219.Refresh();
    HCMAX7219.print7Seg("guah",4);HCMAX7219.print7Seg(ha,8);
    HCMAX7219.print7Seg("gua7",12);HCMAX7219.print7Seg(ta,16);
    delay(1000);
    HCMAX7219.Refresh();
}
```

A2 APPENDIX B: OUT-BOARD FIRMWARE (C++ CODE)

```
char command;
String string;
boolean ledon = false;
#define led 6
#define relay1 2
#define relay2 3
#define relay3 4
#define relay4 5
#define relay5 6
#define relay6 7
#define relay7 8
#define relay8 9
```

```
void setup()
{
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  pinMode(relay1, OUTPUT);
  pinMode(relay2, OUTPUT);
  pinMode(relay3, OUTPUT);
  pinMode(relay4, OUTPUT);
  pinMode(relay5, OUTPUT);
  pinMode(relay6, OUTPUT);
  pinMode(relay7, OUTPUT);
  pinMode(relay8, OUTPUT);
}

void loop()
{
  Serial.flush();
  if (Serial.available() > 0)
  { string = "";}
  while(Serial.available() > 0)
  {
    command = ((byte)Serial.read());
    if(command == ':')
    {
      break;
    }
    else
    {
      string += command;
    }
    delay(1);
  }
}
```



```
if(string == "TO")
{
    ledOn();
    ledon = true;
}
if(string == "TF")
{
    ledOff();
    ledon = false;
}

if(string == "RSON1")
{
    digitalWrite(relay1, LOW);
    delay(500);
}
if(string == "RSOFF1")
{
    digitalWrite(relay1, HIGH);
    delay(500);
}

if(string == "RSON2")
{
    digitalWrite(relay2, LOW);
    delay(500);
}
if(string == "RSOFF2")
{
    digitalWrite(relay2, HIGH);
    delay(500);
}
```

```
}

if(string == "RSON3")
{
    digitalWrite(relay3,LOW );
    delay(500);
}

if(string == "RSOFF3")
{
    digitalWrite(relay3, HIGH);
    delay(500);
}

if(string == "RSON4")
{
    digitalWrite(relay4, LOW);
    delay(500);
}

if(string == "RSOFF4")
{
    digitalWrite(relay4, HIGH);
    delay(500);
}

if(string == "RSON5")
{
    digitalWrite(relay5, LOW);
    delay(500);
}
```

```
if(string=="RSOFF5")
{
    digitalWrite(relay5, HIGH);
    delay(500);
}

if(string=="RSON6")
{
    digitalWrite(relay6, LOW);
    delay(500);
}

if(string=="RSOFF6")
{
    digitalWrite(relay6, HIGH);
    delay(500);
}

if(string=="RSON7")
{
    digitalWrite(relay7, LOW);
    delay(500);
}

if(string=="RSOFF7")
{
    digitalWrite(relay7, HIGH);
    delay(500);
}
```

```
}

if(string == "RSON8")
{
    digitalWrite(relay8, LOW);
    delay(500);
}

if(string == "RSOFF8")
{
    digitalWrite(relay8, HIGH);
    delay(500);
}

//if ((string.toInt())>=0)&&(string.toInt())<=HIGH)
//{
// if (ledon==true)
// {
//     digitalWrite(led, string.toInt());
//     Serial.println(string); //debug
//     delay(500);
// }
//}

void ledOn()
{
    analogWrite(led, 255);
    delay(10);
}

void ledOff()
```

```
{  
    analogWrite(led, 0);  
    delay(10);  
}
```

A3 APPENDIX C: PROCESSOR APPLICATION SOURCE CODE (KERNEL)

```
from Tkinter import *  
import tkFont  
import RPi.GPIO as GPIO  
import serial  
  
ser=serial.Serial('/dev/ttyACM0',9600,timeout=0, writeTimeout=0)  
serin=serial.Serial('/dev/ttyACM1',9600,timeout=0, writeTimeout=0)  
  
def raise_frame(frame):  
    frame.tkraise()  
  
def exitProgram():  
    print("Exit Button pressed")  
    GPIO.cleanup()  
    root.quit()  
def TO():  
    print("LED ON")  
    ser.write("TO")  
def TF():
```

```
print("LED OFF")
ser.write('TF')
def rson1():
    print("RELAY SWITCH 1 ON")
    ser.write('RSON1')
def rsoff1():
    print("RELAY SWITCH 1 OFF")
    ser.write('RSOFF1')
def rson2():
    print("RELAY SWITCH 2 ON")
    ser.write('RSON2')
def rsoff2():
    print("RELAY SWITCH 2 OFF")
    ser.write('RSOFF2')
def rson3():
    print("RELAY SWITCH 3 ON")
    ser.write('RSON3')
def rsoff3():
    print("RELAY SWITCH 3 OFF")
    ser.write('RSOFF3')
def rson4():
    print("RELAY SWITCH 4 ON")
    ser.write('RSON4')
def rsoff4():
    print("RELAY SWITCH 4 OFF")
    ser.write('RSOFF4')
def rson5():
    print("RELAY SWITCH 5 ON")
    ser.write('RSON5')
def rsoff5():
    print("RELAY SWITCH 5 OFF")
```

```
    ser.write('RSOFF5')
def rson6():
    print("RELAY SWITCH 6 ON")
    ser.write('RSON6')
def rsoff6():
    print("RELAY SWITCH 6 OFF")
    ser.write('RSOFF6')
def rson7():
    print("RELAY SWITCH 7 ON")
    ser.write('RSON7')
def rsoff7():
    print("RELAY SWITCH 7 OFF")
    ser.write('RSOFF7')
def rson8():
    print("RELAY SWITCH 8 ON")
    ser.write('RSON8')
def rsoff8():
    print("RELAY SWITCH 8 OFF")
    ser.write('RSOFF8')
def rion1():
    print("RASPBERRY PI I/O ON")
    ser.write('RASPBERRY PI I/O ON')
def rioff1():
    print("RASPBERRY PI I/O OFF")
    ser.write('RASPBERRY PI I/O OFF')
def allOFF():
    print("MASTER OFF")
    ser.write('TF')
    ser.write('rsoff1')
    ser.write('rsoff2')
    ser.write('rsoff3')
```

```
ser.write('rsff4')
ser.write('rsff5')
ser.write('rsff6')
ser.write('rsff7')
ser.write('rsff8')

root = Tk()
myFont = tkFont.Font(family = 'Helvetica', size = 18, weight = 'bold')
myFonts = tkFont.Font(family = 'Helvetica', size = 10, weight = 'bold')

root.geometry('800x480')
root.title("ONLINE NODE APPLICATION")

f1 = Frame(root)
f2 = Frame(root)
f3 = Frame(root)
f4 = Frame(root)

for frame in (f1, f2, f3, f4):
    frame.grid(row=0, column=0, sticky='news')
Button(f1, text='CONTROL PANEL',bg="DodgerBlue4",font = myFont,
command=lambda:raise_frame(f2),height = 8 , width = 12).grid(row=0,column=0)
Button(f1, text='CUSTOMIZATION',bg="PeachPuff3",font = myFont,
command=lambda:raise_frame(f3),height = 8 , width = 12).grid(row=0,column=1)
Button(f1, text='RT DATA LOG',bg="khaki2",font = myFont,
command=lambda:raise_frame(f4),height = 8 , width = 12).grid(row=0,column=2)
Button(f1, text='QUIT',font = myFont,bg="tomato", command=lambda:exitProgram(),height = 8
, width = 5).grid(row=0,column=3)
```



```

Button(f2, text='LED ON',font = myFonts, command=lambda:TO(),height =6 , width =
11).grid(row=0,column=0)
Button(f2, text='LED OFF',font = myFonts, command=lambda:TF(),height =6 , width =
11).grid(row=1,column=0)
Button(f2, text='REL1 ON',font = myFonts, command=lambda:rson1(),height =6 , width =
11).grid(row=0,column=1)
Button(f2, text='REL1 OFF',font = myFonts, command=lambda:rsoff1(),height =6 , width =
11).grid(row=1,column=1)
Button(f2, text='REL2 ON',font = myFonts, command=lambda:rson2(),height =6 , width =
11).grid(row=0,column=2)
Button(f2, text='REL2 OFF',font = myFonts, command=lambda:rsoff2(),height =6 , width =
11).grid(row=1,column=2)
Button(f2, text='REL3 ON',font = myFonts, command=lambda:rson3(),height =6 , width =
11).grid(row=0,column=3)
Button(f2, text='REL3 OFF',font = myFonts, command=lambda:rsoff3(),height =6 , width =
11).grid(row=1,column=3)
Button(f2, text='REL4 ON',font = myFonts, command=lambda:rson4(),height =6 , width =
11).grid(row=0,column=4)
Button(f2, text='REL4 OFF',font = myFonts, command=lambda:rsoff4(),height =6 , width =
11).grid(row=1,column=4)
Button(f2, text='REL5 ON',font = myFonts, command=lambda:rson5(),height =6 , width =
11).grid(row=0,column=5)
Button(f2, text='REL5 OFF',font = myFonts, command=lambda:rsoff5(),height =6 , width =
11).grid(row=1,column=5)
Button(f2, text='REL6 ON',font = myFonts, command=lambda:rson6(),height =6 , width =
11).grid(row=3,column=0)
Button(f2, text='REL6 OFF',font = myFonts, command=lambda:rsoff6(),height =6 , width =
11).grid(row=4,column=0)
Button(f2, text='REL7 ON',font = myFonts, command=lambda:rson7(),height =6 , width =
11).grid(row=3,column=1)

```

```
Button(f2, text='REL7 OFF',font = myFonts, command=lambda:rsoff7(),height =6 , width =
11).grid(row=4,column=1)
Button(f2, text='REL8 ON',font = myFonts, command=lambda:rson8(),height =6 , width =
11).grid(row=3,column=2)
Button(f2, text='REL8 OFF',font = myFonts, command=lambda:rsoff8(),height =6 , width =
11).grid(row=4,column=2)
Button(f2, text='I/O ON',font = myFonts, command=lambda:rion1(),height =6 , width =
11).grid(row=3,column=3)
Button(f2, text='I/O OFF',font = myFonts, command=lambda:rioff1(),height =6 , width =
11).grid(row=4,column=3)
Button(f2, text='MAIN MENU',font = myFonts,bg="khaki2",
command=lambda:raise_frame(f1),height =5 , width = 9).grid(row=3,column=4)
Button(f2, text='LOG',font = myFonts,bg="PeachPuff3",
command=lambda:raise_frame(f4),height =5 , width = 9).grid(row=4,column=4)
Button(f2, text='MASTER OFF',font = myFonts,bg="brown1 ",
command=lambda:allOFF(),height =5 , width = 9).grid(row=3,column=5)
Button(f2, text='QUIT',font = myFonts,bg="green4", command=lambda:exitProgram(),height =5
, width = 9).grid(row=4,column=5)

Label(f4, text='LOG').pack()
Button(f4, text='GO TO MAIN MENU', command=lambda:raise_frame(f1)).pack(side='left')

Label(f3, text='Dashboard').pack()
Button(f3, text='GO TO MAIN MENU', command=lambda:raise_frame(f1)).pack(side='left')
scrollbar = Scrollbar(f4)
scrollbar.pack(side=RIGHT, fill=Y)
log = Text ( f4, width=60, height=30, takefocus=0)
log.pack()
log.config(yscrollcommand=scrollbar.set)
```

```
scrollbar.config(command=log.yview)

serBuffer = ""
def readSerial():
    while True:
        c = serin.read() # attempt to read a character from Serial
        global serBuffer
        serBuffer += c # add to the buffer

    f4.after(10, readSerial) # check serial again soon

f4.after(100, readSerial)
raise_frame(f1)
root.mainloop()
```

A4 APPENDIX D: MOBILE APPLICATION DESIGN OFFLINE NODE (ANDROID STUDIOS)

LAYOUT 1 (FIRST PAGE): XML CODE:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:paddingLeft="@dimen/activity_horizontal_margin"

    android:paddingRight="@dimen/activity_horizontal_margin"

    android:paddingTop="@dimen/activity_vertical_margin"

    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".DeviceList">
```

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Paired Devices"  
    android:id="@+id/textView"  
    android:layout_alignParentTop="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true" />
```

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Paired Devices"  
    android:id="@+id/button"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentLeft="true"  
    android:layout_alignParentStart="true"  
    android:layout_alignParentRight="true"  
    android:layout_alignParentEnd="true" />
```

```
<ListView  
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"

        android:id="@+id/listView"

        android:layout_centerHorizontal="true"

        android:layout_above="@+id/button"

        android:layout_below="@+id/textView" />

</RelativeLayout>
```

A5 APPENDIX E: MOBILE APPLICATION SOURCE CODE (ANDROID STUDIOS)

BLUETOOTH HANDSHAKE CLASS:

```
package com.led.led;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
```

```
import java.util.Set;

public class DeviceList extends ActionBarActivity
{
    //widgets
    Button btnPaired;
    ListView devicelist;
    //Bluetooth
    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    public static String EXTRA_ADDRESS = "device_address";

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_device_list);

        //Calling widgets
        btnPaired = (Button)findViewById(R.id.button);
        devicelist = (ListView)findViewById(R.id.listView);

        //if the device has bluetooth
        myBluetooth = BluetoothAdapter.getDefaultAdapter();

        if(myBluetooth == null)
        {
            //Show a mensag. that the device has no bluetooth adapter
            Toast.makeText(getApplicationContext(), "Bluetooth Device Not Available",
            Toast.LENGTH_LONG).show();
        }
    }
}
```

```
//finish apk
finish();
}
else if(!myBluetooth.isEnabled())
{
    //Ask to the user turn the bluetooth on
    Intent turnBTon = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(turnBTon,1);
}

btnPaired.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        pairedDevicesList();
    }
});

}

private void pairedDevicesList()
{
    pairedDevices = myBluetooth.getBondedDevices();
    ArrayList list = new ArrayList();

    if (pairedDevices.size()>0)
    {
        for(BluetoothDevice bt : pairedDevices)
        {
            list.add(bt.getName() + "\n" + bt.getAddress()); //Get the device's name and the
address

```

```
    }  
    }  
    else  
    {  
        Toast.makeText(getApplicationContext(), "No Paired Bluetooth Devices Found.",  
Toast.LENGTH_LONG).show();  
    }  
  
    final ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,  
list);  
    devicelist.setAdapter(adapter);  
    devicelist.setOnItemClickListener(myItemClickListener); //Method called when the device  
from the list is clicked  
  
    }  
  
    private AdapterView.OnItemClickListener myItemClickListener = new  
AdapterView.OnItemClickListener()  
    {  
        public void onItemClick (AdapterView<?> av, View v, int arg2, long arg3)  
        {  
            // Get the device MAC address, the last 17 chars in the View  
            String info = ((TextView) v).getText().toString();  
            String address = info.substring(info.length() - 17);  
  
            // Make an intent to start next activity.  
            Intent i = new Intent(DeviceList.this, ledControl.class);  
  
            //Change the activity.  
            i.putExtra(EXTRA_ADDRESS, address); //this will be received at ledControl (class)  
Activity
```



```
        startActivity(i);
    }
};

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_device_list, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
}
```

MAIN CODE:

```
package com.led.led;
```

```
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.AsyncTask;

import java.io.IOException;
import java.util.UUID;

public class ledControl extends ActionBarActivity {

    Button btnOn, btnOff, btnDis,
    btnsendhuge,rson1,rsoff1,rson2,rsoff2,rson3,rsoff3,rson4,rsoff4,rson5,rsoff5,rson6,rsoff6,rson7,r
    soff7,rson8,rsoff8;

    SeekBar brightness;
    TextView lumn;
    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
```

```
BluetoothSocket btSocket = null;

private boolean isBtConnected = false;

//SPP UUID. Look for it
static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    Intent newint = getIntent();
    address = newint.getStringExtra(DeviceList.EXTRA_ADDRESS); //receive the address of
the bluetooth device

    //view of the ledControl
    setContentView(R.layout.activity_led_control);

    //call the widgtes
    btnOn = (Button)findViewById(R.id.button2);
    btnOff = (Button)findViewById(R.id.button3);

    rson1 = (Button)findViewById(R.id.button7);
    rsoff1 = (Button)findViewById(R.id.button8);
    rson2 = (Button)findViewById(R.id.button9);
    rsoff2 = (Button)findViewById(R.id.button10);
    rson3 = (Button)findViewById(R.id.button11);
    rsoff3 = (Button)findViewById(R.id.button12);
    rson4 = (Button)findViewById(R.id.button13);
    rsoff4 = (Button)findViewById(R.id.button14);
    rson5 = (Button)findViewById(R.id.button15);
```

```
rsoff5 = (Button)findViewById(R.id.button16);
rson6 = (Button)findViewById(R.id.button17);
rsoff6 = (Button)findViewById(R.id.button18);
rson7 = (Button)findViewById(R.id.button19);
rsoff7 = (Button)findViewById(R.id.button20);
rson8 = (Button)findViewById(R.id.button21);
rsoff8 = (Button)findViewById(R.id.button22);

btnDis = (Button)findViewById(R.id.button4);
btnsendhuge = (Button)findViewById(R.id.button5);
brightness = (SeekBar)findViewById(R.id.seekBar);
lumn = (TextView)findViewById(R.id.lumn);

new ConnectBT().execute(); //Call the class to connect

//commands to be sent to bluetooth
btnOn.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        turnOnLed(); //method to turn on
    }
});

btnOff.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v)
    {
        turnOffLed(); //method to turn off
    }
})
```

```
});

rson1.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON1".getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        } //method to turn on
    }
});

rsoff1.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF1".getBytes());
            }
        }
    }
});
```

```
        catch (IOException e)
        {
            msg("Error");
        }
    }    //method to turn on
}

});
rson2.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON2".getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }    //method to turn on
    }
});
rsoff2.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
```

```
{
    try
    {
        btSocket.getOutputStream().write("RSOFF2".toString().getBytes());
    }
    catch (IOException e)
    {
        msg("Error");
    }
} //method to turn on
}
});

rson3.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON3".toString().getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        } //method to turn on
    }
});
```

```
rsOff3.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF3".getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        } //method to turn on
    }
}); rson4.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON4".getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }
});
```



```
    }
    } //method to turn on
}
});
rsOff4.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF4".getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        } //method to turn on
    }
});
rson5.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON5".getBytes());
```

```
    }  
    catch (IOException e)  
    {  
        msg("Error");  
    }  
} //method to turn on  
}  
});  
rsoff5.setOnClickListener(new View.OnClickListener()  
{  
    @Override  
    public void onClick(View v)  
    {  
        if (btSocket!=null)  
        {  
            try  
            {  
                btSocket.getOutputStream().write("RSOFF5".getBytes());  
            }  
            catch (IOException e)  
            {  
                msg("Error");  
            }  
        } //method to turn on  
    }  
}); rson6.setOnClickListener(new View.OnClickListener()  
{  
    @Override  
    public void onClick(View v)  
    {  
        if (btSocket!=null)
```

```
{
    try
    {
        btSocket.getOutputStream().write("RSON6".toString().getBytes());
    }
    catch (IOException e)
    {
        msg("Error");
    }
} //method to turn on
}
});
rsoff6.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF6".toString().getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        } //method to turn on
    }
});
rson7.setOnClickListener(new View.OnClickListener()
{
```

```
@Override
public void onClick(View v)
{
    if (btSocket!=null)
    {
        try
        {
            btSocket.getOutputStream().write("RSON7".toString().getBytes());
        }
        catch (IOException e)
        {
            msg("Error");
        }
    } //method to turn on
}
});

rsoff7.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF7".toString().getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }
    }
});
```

```
        }    //method to turn on
    }
});    rson8.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSON8".toString().getBytes());
            }
            catch (IOException e)
            {
                msg("Error");
            }
        }    //method to turn on
    }
});
rsoff8.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if (btSocket!=null)
        {
            try
            {
                btSocket.getOutputStream().write("RSOFF8".toString().getBytes());
            }
        }
    }
});
```

```
        catch (IOException e)
        {
            msg("Error");
        }
    } //method to turn on
}
});
```

```
btnDis.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        Disconnect(); //close connection
    }
});

btnsendhuge.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        sendhuge(); //close connection
    }
});
```

```
brightness.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {  
    @Override  
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {  
        if (fromUser==true)  
        {  
            lumn.setText(String.valueOf(progress));  
            try  
            {  
                btSocket.getOutputStream().write(String.valueOf(progress).getBytes());  
            }  
            catch (IOException e)  
            {  
                  
            }  
        }  
    }  
});  
  
@Override  
public void onStartTrackingTouch(SeekBar seekBar) {  
  
}  
  
@Override  
public void onStopTrackingTouch(SeekBar seekBar) {  
  
}  
});  
}
```

```
private void Disconnect()
{
    if (btSocket!=null) //If the btSocket is busy
    {
        try
        {
            btSocket.close(); //close connection
        }
        catch (IOException e)
        { msg("Error");}
    }
    finish(); //return to the first layout
}

private void turnOffLed()
{
    if (btSocket!=null)
    {
        try
        {
            btSocket.getOutputStream().write("TF".getBytes());
        }
        catch (IOException e)
        {
            msg("Error");
        }
    }
}

private void turnOnLed()
```



```
{
    if (btSocket!=null)
    {
        try
        {
            btSocket.getOutputStream().write("TO".toString().getBytes());
        }
        catch (IOException e)
        {
            msg("Error");
        }
    }
}

private void sendhuge()
{
    if (btSocket!=null)
    {
        try
        {
            btSocket.getOutputStream().write("MY NAME IS PRANAV IM TESTING THE
BLUETOOTH APP WITH A HUGE STRING<IF YOU ARE ABLE TO READ THIS STRING
IT MEANS THAT IT WORKS>>>>>THANK YOU!!!:-)".toString().getBytes());
        }
        catch (IOException e)
        {
            msg("Error");
        }
    }
}

// fast way to call Toast
```

```
private void msg(String s)
{
    Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_led_control, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}

private class ConnectBT extends AsyncTask<Void, Void, Void> // UI thread
{
    private boolean ConnectSuccess = true; //if it's here, it's almost connected
```

```
@Override
protected void onPreExecute()
{
    progress = ProgressDialog.show(lcdControl.this, "Connecting...", "Please wait!!!");
//show a progress dialog
}

@Override
protected Void doInBackground(Void... devices) //while the progress dialog is shown, the
connection is done in background
{
    try
    {
        if (btSocket == null || !isBtConnected)
        {
            myBluetooth = BluetoothAdapter.getDefaultAdapter();//get the mobile bluetooth
device
            BluetoothDevice dispositivo = myBluetooth.getRemoteDevice(address);//connects to
the device's address and checks if it's available
            btSocket =
dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);//create a RFCOMM
(SPP) connection
            BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
            btSocket.connect();//start connection
        }
    }
    catch (IOException e)
    {
        ConnectSuccess = false;//if the try failed, you can check the exception here
    }
    return null;
}
```

```
}  
@Override  
protected void onPostExecute(Void result) //after the doInBackground, it checks if  
everything went fine  
{  
    super.onPostExecute(result);  
  
    if (!ConnectSuccess)  
    {  
        msg("Connection Failed. Is it a SPP Bluetooth? Try again.");  
        finish();  
    }  
    else  
    {  
        msg("Connected.");  
        isBtConnected = true;  
    }  
    progress.dismiss();  
}  
}
```

