PROJECT INITIATION DOCUMENT

for

BATTERY MANAGEMENT SYSTEM

By

ADEV - B2

Under the guidance of

Mrs. Nagma Tazeen

Mr. Kaushik B.K

Mr. Pavan Kumar E

RV - SKILLS CENTER FOR EMERGING TECHNOLOGIES

Automotive Electronics

INDEX

PROJECT TITLE

BATTERY MANAGEMENT SYSTEM

BRIEF DESCRIPTION:

Battery management systems (BMS) are electronic control circuits that monitor and regulate the charging and discharging of batteries. The battery characteristics to be monitored include the detection of battery voltages, temperature, capacity, state of charge, Health of battery, power consumption, remaining operating time, charging cycles, Battery protection and some more characteristics.

The task of battery management systems is to ensure the optimal use of the residual energy present in a battery. BMS protect the batteries from deep discharging and over charging, which are results of extreme fast charge and extreme high discharge current which will affect the battery life. In the case of multi-cell batteries, the battery management system also provides a cell balancing function, to manage that different cell of battery.

Hardware Description:

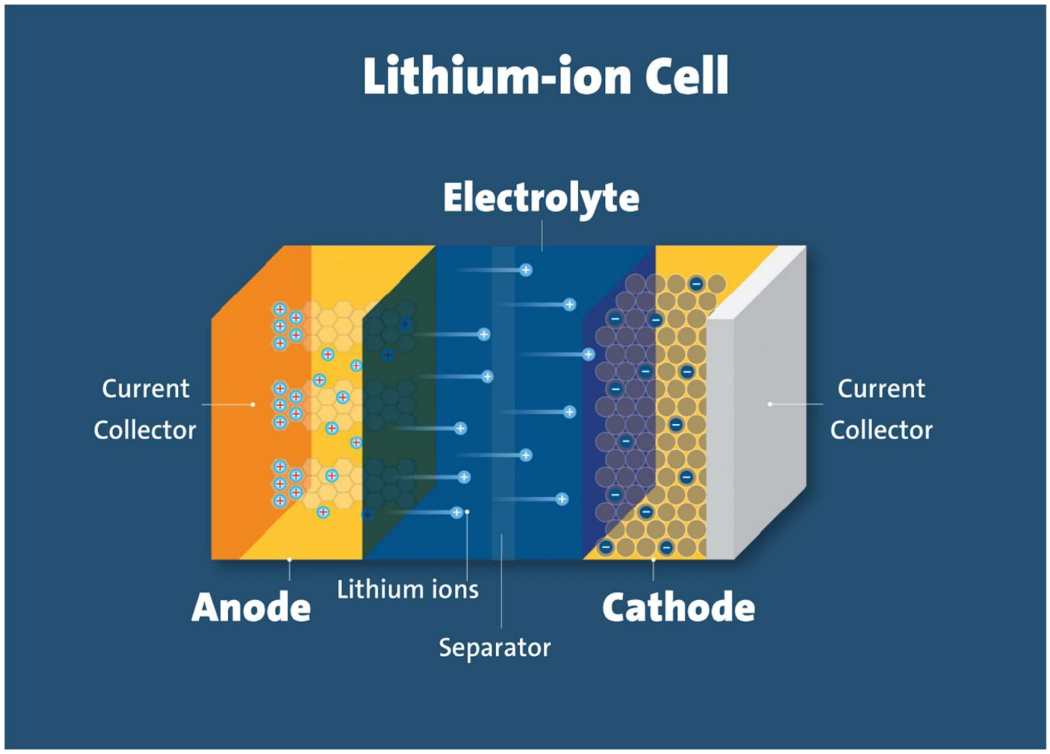
Battery cells [LI-ION 3.2V 6A]:

A lithium-ion battery is a family of rechargeable battery types in which lithium ions move from the negative electrode to the positive electrode during discharge and back when charging.

Lithium-ion is the most popular rechargeable battery chemistry used today. Lithium-ion batteries power the devices we use every day, like our mobile phones and electric vehicles.

Lithium-ion batteries consist of single or multiple lithium-ion cells, along with a protective circuit board. They are referred to as batteries once the cell, or cells, are installed inside a device with the protective circuit board.

Components of a lithium-ion cell:



Electrodes: The positively and negatively charged ends of a cell. Attached to the current collectors.

Anode: The negative electrode.

Cathode: The positive electrode.

Electrolyte: A liquid or gel that conducts electricity.

Current collectors: Conductive foils at each electrode of the battery that are connected. to the terminals of the cell. The cell terminals transmit the electric current between the battery, the device and the energy source that powers the battery.

Separator: A porous polymeric film that separates the electrodes while enabling the exchange of lithium ions from one side to the other.

Lithium-ion cell working principle:

In a lithium-ion battery, lithium ions (Li+) move between the cathode and anode internally. Electrons move in the opposite direction in the external circuit. This migration is the reason the battery powers the device—because it creates the electrical current.

While the battery is discharging, the anode releases lithium ions to the cathode, generating a flow of electrons that helps to power the relevant device.

When the battery is charging, the opposite occurs: lithium ions are released by the cathode and received by the anode.

Lithium-Ion cell

Voltage Sensors [EC-2173]:

Voltage Sensor is a precise low-cost sensor for measuring voltage. It is based on principle of resistive voltage divider design. It can make the red terminal connector input voltage to 5 times smaller. The voltage detection module input voltage not greater than 5Vx5=25V (if using 3.3V systems, input voltage not greater than 3.3Vx5=16.5V).

Features:

- Voltage input range: DC 0-25V
- Voltage detection range: DC 0.02445V-25V
- Voltage Analog Resolution: 0.00489V
- DC input connector: Terminal cathode connected to VCC, GND negative pole
- Output interface: "+" connect 5/3.3V, "-" connect GND, "s" connect the STM32F103 AD pins.

Voltage Sensors [EC-2173]

Current Sensor [ACS712]:

The ACS712 is a fully integrated, hall effect-based linear current sensor with 2.1kVRMS voltage isolation and a integrated low-resistance current conductor. Technical terms aside, it's simply put forth as a current sensor that uses its conductor to calculate and measure the amount of current applied.

Features:

- 80khz bandwidth.
- 66 to 185 mv/A output sensitivity.
- Low-noise analog signal path.
- Device bandwidth is set via the new FILTER pin.
- 1.2 mω internal conductor resistance.
- Total output error of 1.5% at TA = 25°C.
- Stable output offset voltage.
- Near zero magnetic hysteresis

ACS712 Current Sensor work:

Current sensors can work either be done through direct or indirect sensing. For the ACS712, it uses indirect sensing.

For current sensors that work by direct sensing, ohm's law is being applied to measure the drop in voltage when flowing current is detected.

Current flows through the onboard hall sensor circuit in its IC. The hall effect sensor detects the incoming current through its magnetic field generation. Once detected, the hall effect sensor generates a voltage proportional to its magnetic field that's then used to measure the amount of current.

Current Sensor [ACS712]

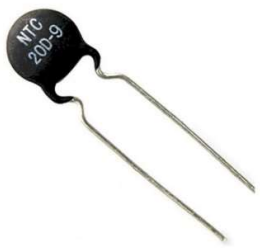
Temperature Sensor [20D9 NTC Thermistor]:

NTC stands for "Negative Temperature Coefficient". NTC thermistors are **resistors with a negative temperature coefficient**, which means that the resistance decreases with increasing temperature. They are primarily used as resistive temperature sensors and current-limiting devices.

20D9 NTC Thermistor (NTC-Negative Temperature Coefficient), these R-T curve (Resistance-Temperature) matched thermistors are small, high quality, epoxy encapsulated, precision devices. These epoxy-coated interchangeable chip thermistors offer true interchangeability over wide temperature ranges, permitting the circuit designer to standardize circuitry. This eliminates the need to individually adjust circuits and allows the thermistors to be easily replaced without the need for re-calibration.

Specifications:

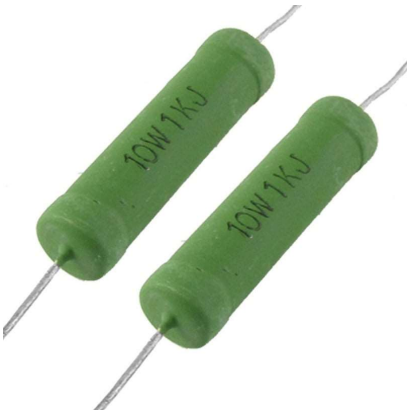
Resistance (Ω)	20
Tolerance (%)	1
Temperature Co-efficient	NTC
Operating Temperature Range (°C)	-55 to 200
Diameter (mm)	9
Weight (gm)	0.5 each
Shipment Weight	0.01 kg
Shipment Dimensions	2 × 1 × 1 cm



20D9 NTC Thermistor

Power Resistors:

Power resistors are **designed to withstand and dissipate large amounts of power**. The common trait of all power resistors is that they are built to dissipate as much power as possible, while keeping their size as small as possible. In general, they have a power rating of at least 5 W. Power resistors are made from materials with a high thermal conductivity, allowing efficient cooling. They are also often designed to be coupled with heat sinks to be able to dissipate the high amounts of power. Some power resistors even require forced air or liquid cooling while under maximum load to efficiently remove the heat generated.



Power Resistors

Power MOSFETS [IRF 540]:

IRF540 third generation power MOSFETs provide the designer with the best combination of fast switching, ruggedized device design, low on-resistance and cost-effectiveness. The TO-220AB package is universally preferred for all commercial-industrial applications at power dissipation levels to approximately 50 W.

Features:

• Dynamic dV/dt rating

• Repetitive avalanche rated

• Fast switching

• Ease of paralleling

• Simple drive requirements

• Compliant to RoHS directive 2002/95/EC

Detailed Specifications:

Number of Channels	1 Channel
Transistor Polarity	N-Channel
Drain-Source Breakdown Voltage (Vds)	100V
Continuous Drain Current (Id)	33A
Drain-Source Resistance (Rds On)	44mOhms
Gate-Source Voltage (Vgs)	20V
Gate Charge (Qg)	71 nC
Operating Temperature Range	-55 - 175°C
Power Dissipation (Pd)	130W

Power MOSFETS [IRF 540]

Voltage Regulator [LM7805]:

The LM7805 is a voltage regulator that outputs +5 volts.

Like most other regulators in the market, it is a three-pin IC; input pin for accepting incoming DC voltage, ground pin for establishing ground for the regulator, and output pin that supplies the positive 5 volts

Product Features:

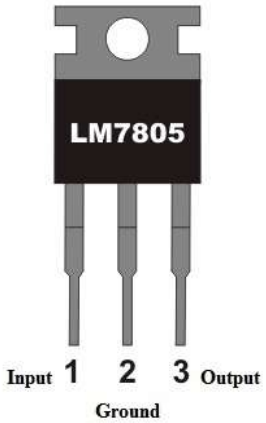
- 3-Terminal Regulators
- Output Current up to 1.5A
- Internal Thermal-Overload Protection
- High Power-Dissipation Capability
- Internal Short-Circuit Current Limiting
- Output Transistor SAFE-Area Compensation

Absolute Maximum Input Voltage

- 35V

Recommended Operating Conditions

- Input Voltage: Minimum 7V, Maximum 25V
- Output Current: 1.5A
- Operating Virtual Junction Temperature: Minimum 0, Maximum 125°C



LM7805 IC

ESP8266 NODE MCU:

The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability.

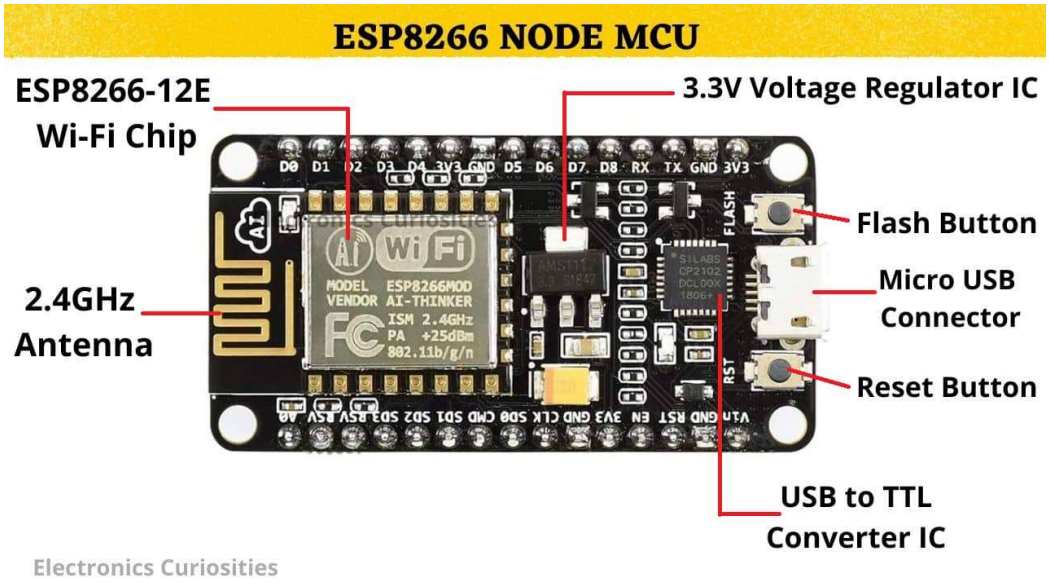
All ESP8266 variants have a ESP8266EX core processor and a Tensilica L106 32-bit micro controller unit. This is a low cost, high performance, low power consumption, easy to program, wireless SoC (System-On-Chip). It provides capabilities for 2.4 GHz Wi-Fi (802.11 b/g/n, supporting WPA/WPA2), general-purpose input/output (13 GPIO), Inter-Integrated Circuit (I^2C), analog-to-digital conversion (10-bit ADC), Serial Peripheral Interface (SPI), I^2S interfaces with DMA (sharing pins with GPIO), UART (on dedicated pins, plus a transmit-only UART can be enabled on GPIO2), and pulse-width modulation (PWM).

It has a build-in programmer and a voltage regulator, that allow flashing and powering the device via micro-USB. The system operates at 3.3V.

Specifications:

- Tensilica L106 32-bit micro controller unit at 80 MHz (or overclocked to 160 MHz)
- 32 kB instruction RAM
- 80 kB user data RAM
- 16 kB ETS system data RAM
- Flash Memory 4Mb
- USB – micro-USB port for power, programming and debugging
- 13 GPIO pins
- 802.11 b/g/n, supporting WPA/WPA2
- STA / AP modes support
- TCP / IP protocol stack, One socket
- TCP / UDP Server and Client
- Pin-compatible with Arduino UNO, Mega
- KEY button: modes configuration
- 32-bit hardware timer

- WiFi operation current: continuous transmission operation: ≈70mA (200mA MAX), deep sleep mode: <3mA
- Serial WiFi transmission rate: 110-460800bps
- Temperature: -40℃ ~ + 125 ℃
- Humidity: 10%-90% non-condensing
- Weight: about 20g (0.7oz)
- Pulse-Width Modulation (PWM)
- Interrupt capability
- 3.3V operating voltage, internal voltage regulator allows 5V on power input
- maximum current through GPIO pins: 12mA (source), 20mA (drain)
- available firmware for Arduino IDE
- WebSocket libraries available



ESP8266 NODE MCU:

Esp8266 V3 CH340:

CH340 ESP8266 NodeMcu ESP 8266 V3 CH340 Lua WIFI Internet Of Things Development Lolin Board Based ESP8266 CH 340 Wireless Module ESP-12E Electronics Circuitry & Parts. ESP8266MOD or ESP-12E or ESP-12F is New Lolin NodeMcu Lua V3 Wireless Module Wifi Internet of Things (IoT) Development Board Based on ESP8266 Micro USB To TTL

Specifications & Features:

- Wireless 802.11 b / g / n standard
- Support STA / AP / STA + AP three operating modes
- Built-in TCP / IP protocol stack to support multiple TCP Client connections (5 MAX)
- D0 ~ D8, SD1 ~ SD3: used as GPIO, PWM, IIC, etc., port driver capability 15mA
- AD0: 1 channel ADC
- Power input: 4.5V ~ 9V (10VMAX), USB-powered
- Current: continuous transmission: ≈70mA (200mA MAX), Standby: <200uA
- Transfer rate: 110-460800bps
- Support UART / GPIO data communication interface
- Remote firmware upgrade (OTA)
- Support Smart Link Smart Networking
- Working temperature: -40 ℃ ~ + 125 ℃
- Drive Type: Dual high-power H-bridge driver

Esp8266 V3 CH340

STM32F103RBT6 Microcontroller:

The STM32F103RBT6 medium density performance microcontroller incorporates the high-performance ARM Cortex™-M3 32-bit RISC core operating at a 72MHz frequency, high speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. This device offers two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN.

The NUCLEO-F103RB is a STM32 Nucleo development board with STM32F103RBT6 MCU allows user to build their own prototypes for embedded applications. The Arduino connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board series are based on ARM Cortex-M 32-bit RISC cores optimised for high performance and energy efficiency. ARM Cortex-M features a high-density instruction set and a Nested Vectored Interrupt Controller (NVIC) providing excellent interrupt handling abilities.

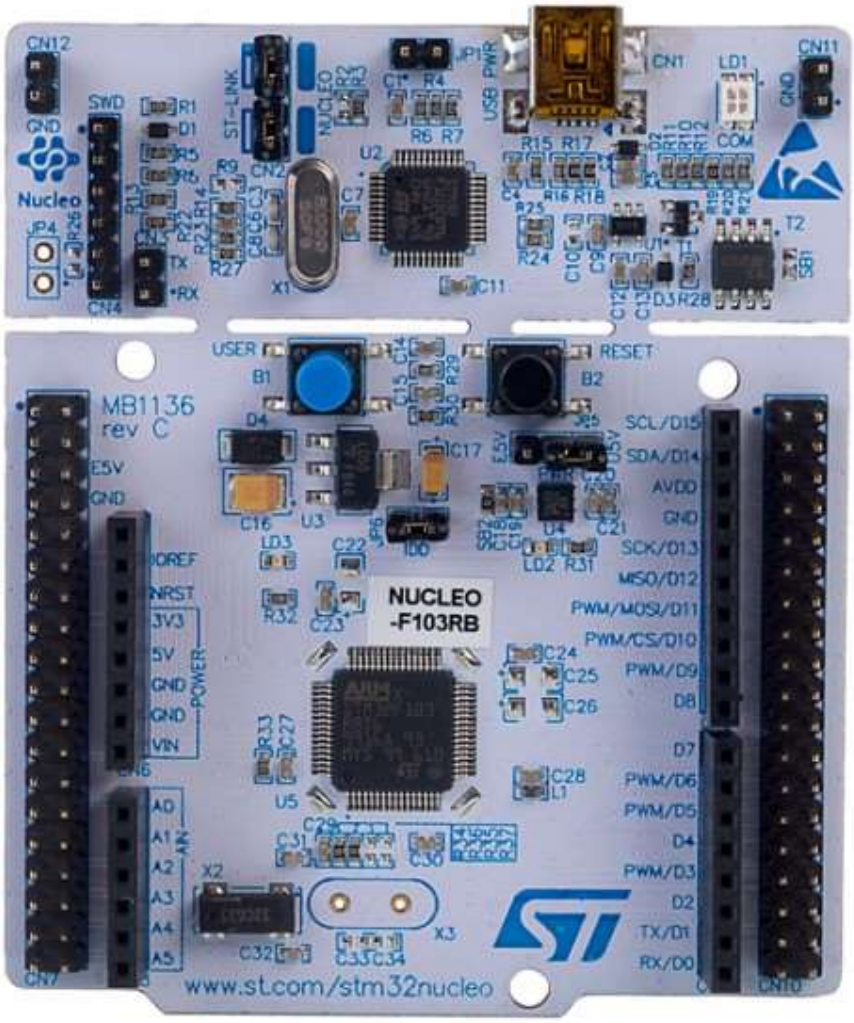
Features:

- Onboard STM32F103RBT6 microcontroller.
- Arduino Uno Revision 3 connectivity and STMicroelectronics Morpho extension resources supported.
- Onboard ST-LINK/V2-1 debugger/programmer with SWD connector
- USB VBUS or external source power supplies (3.3V, 5V, 7-12V), Power management access point.
- Three LED indications for USB communication, user LED, power LED.
- Two push buttons for user and reset

- Three interfaces supported on USB. Virtual Com port, Mass storage and Debug port
- STM32 comprehensive software HAL library
- Supported by Integrated Development Environments (IDEs) including IAR, Keil, GCC based IDEs

Specifications:

- 72MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
- Single cycle multiplication and hardware division
- 20 Kbytes of SRAM
- Clock, reset and supply management
- 2V to 3.6V application supply and I/Os
- POR, PDR, and programmable voltage detector (PVD)
- 4MHz to 16MHz crystal oscillator
- Internal 8MHz factory trimmed RC
- Internal 40kHz RC
- PLL for CPU clock
- 32kHz oscillator for RTC with calibration
- Sleep, stop and standby modes
- VBAT supply for RTC and backup registers
- 2 x 12 bit, 1µs A/D converters (up to 16 channels)
- Conversion range: 0V to 3.6V
- Dual sample and hold capability
- Temperature sensor
- 7 channel DMA controller



STM32F103RBT6 Microcontroller

IMPLEMENTATION STRATEGY:

Design 1: Developing a BMS system from component level.

Design 2: Interfacing of BMS module with Battery pack.

Phase1 for Design 1:

1. Designing a battery pack of 24V with 20Ah.
2. Measuring individual Cell voltages in battery pack.
3. Calculating total voltage and current of battery pack.
4. Monitoring the temperature of the battery pack.
5. Battery Low indication.
6. Battery Full indication.
7. Display the above parameters on GUI (mobile app).

Phase 2 Design 1:

1. Calculating SOC.
2. Calculating SOH.
3. Cell balancing of battery pack.

Design 2:

1. Designing a battery pack of 24V with 20Ah.
2. Interfacing of BMS module with battery pack.
3. Reading the SOC and SOH value from BMS and display on the system.
4. Monitoring the temperature of battery pack.
5. Battery Low indication.
6. Battery Full indication.
7. Display the above parameter on GUI (mobile app).

RESOURCE PLANNING:

Design1:

1. Li-ion cells battery pack.
2. STM32 controller which supports CAN protocol (STM32F103C6).
3. Voltage, Current and Temperature sensors.
4. Wi-Fi module.
5. Cell balancing and monitoring circuit.
6. Battery protection circuit.

Design2:

1. Li-ion cells battery pack.
2. BMS module.
3. STM32 controller which supports CAN protocol (STM32F103C6).
4. Wi-Fi module.

We are implementing design 1 for our project.

As part of our project, we have designed a battery pack of 25.6V 18 A.

It will give the power output of 460Watt (P=VI).

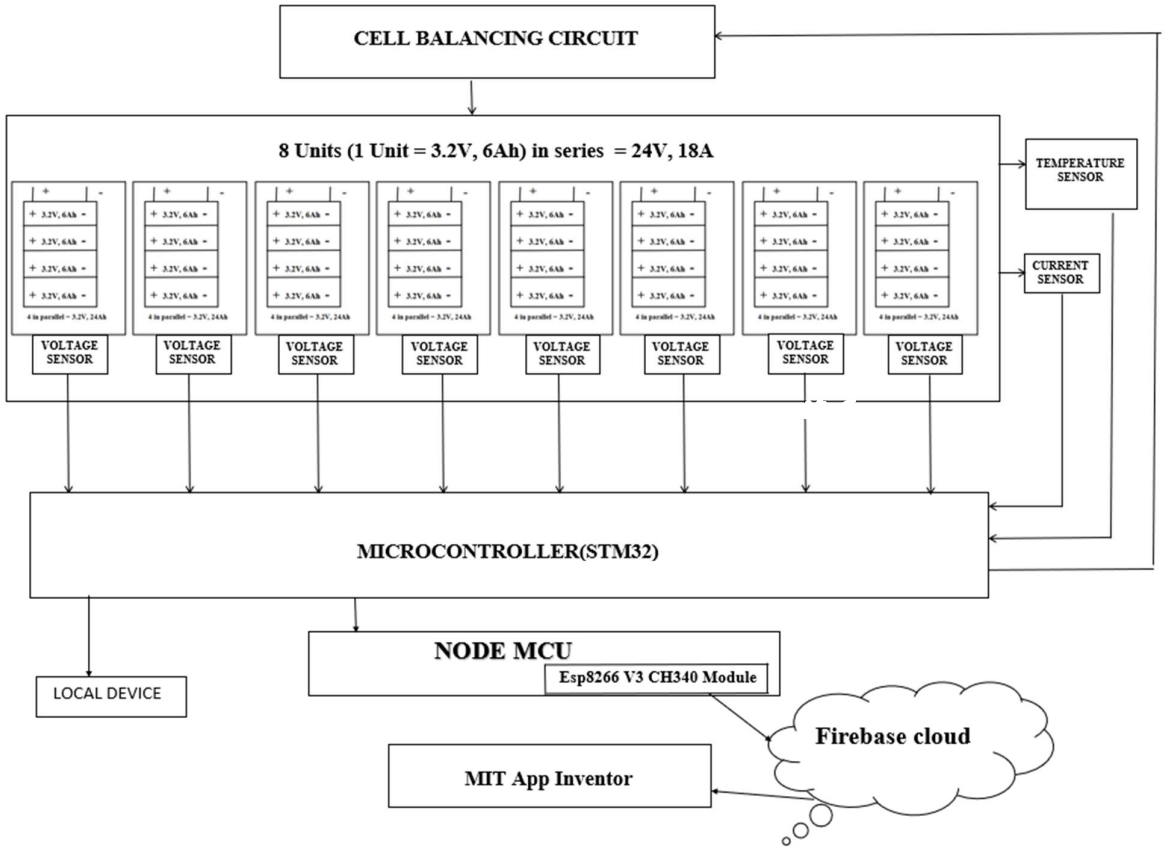
To design the above specification, we have used Li-ion battery cell (3.2v,6A) of 24 cells.

In this Battery pack we have arranged 3 cells in parallel(stacks) such that each stack will give us 3.2V 6A, 8 stacks are arranged in series.

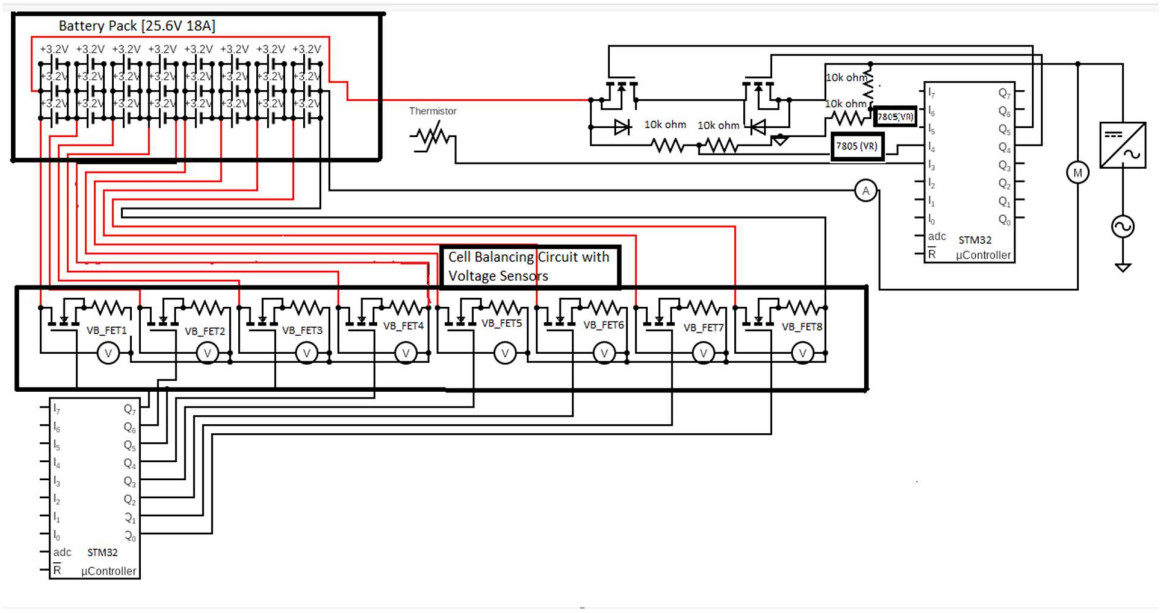
Overall, in this battery pack we are getting 25.6V 18A.

power output is 460W.

Initial BMS prototype:



Detailed design:



BMS Algorithm:

(Please refer the INDEX on the last page for elaboration of the abbreviations used).

Step 1: Start by displaying Hello message.

Step 2: Read individual voltages from all voltage sensors (Voltage monitoring function).

Step 3: While Charger is not connected, and Battery is connected.

Step 4: If any of the cell voltage is <=2.9 display low charge warning message, turn off D-Relay and exit.

Step 5: Else call the discharge function where power is supplied continuously to the load till the cells discharge up to 2.9V.

Step 6: Call voltage monitoring function.

Step 7: While charger is connected, and voltages of cells are <=3.65V.

Step 8: Call charger function where the C-Relay is turned ON and Cell balancing function is called.

Voltage Monitoring:

1. Continuously read the analogue output of Voltage sensors into the ADC of STM32.
2. Voltage V1=Vs1-Vs2 (Vs is Voltage read by the sensor) V2=Vs2-Vs3 V3=Vs3-Vs4 V4=Vs4 V5=Vs5-Vs6 V6=Vs6-Vs7 V7=Vs7-Vs8 V8=Vs8 Vt=Vs1 (Vt is the Total voltage of the battery pack).
3. Display individual cell voltages along with Vt.

Discharge Function:

Let's consider the pin connected at source of C- Relay as Dp.

1. Call voltage monitoring function ().
2. While Dp==1 && V1>2.9 && V2>2.9.....V8>2.9{
3. Set o/p pin connected to D- Relay.
4. Call voltage monitoring function ().
5. If (V1<=2.9 or V2<=2.9 or.....V8<=2.9) {
6. Display LOW CHARGE WARNING!
7. Clear output pin connected to D- Relay (Discharging Relay).
8. Exit.}}

Charger Function:

Let's consider the pin connected at source of D- Relay as Cp.

1. Set output pin connected to C- Relay (Charging Relay).
2. Call cell balancing function.
3. Clear output pin connected to C- Relay (Charging Relay).
4. Exit.

Cell Balancing Function:

Assign the gates in parallel to voltage sensors as G1, G2, …. G8 for V1, V2, …. V8 respectively.

1. While V1>=3.65 or V2>=3.65 or.....V8>=3.65.
2. Call voltage monitoring function ().
3. while(v1> all other cells) {Turn on gate associated with v1}
4. while(v2> all other cells) {Turn on gate associated with v2}
5. while(v3> all other cells) {Turn on gate associated with v3}

   !!!
6. while(v8> all other cells) {Turn on gate associated with v8}

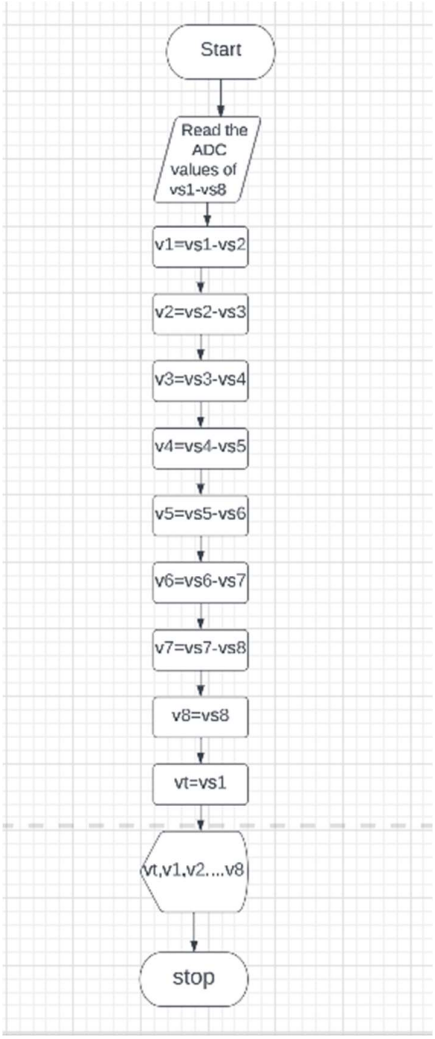
Main Function:

1. Display("Hello!!!").
2. While (1).
3. Call voltage monitoring function ().
4. While Cp! =1&& Dp==1. {
5. If (V1<=2.9 or V2<=2.9 or.....V8<=2.9). {
6. Display LOW CHARGE WARNING!
7. Clear output pin connected to D- Relay (Discharging Relay).
8. Exit.}
9. Call Discharge function ().}

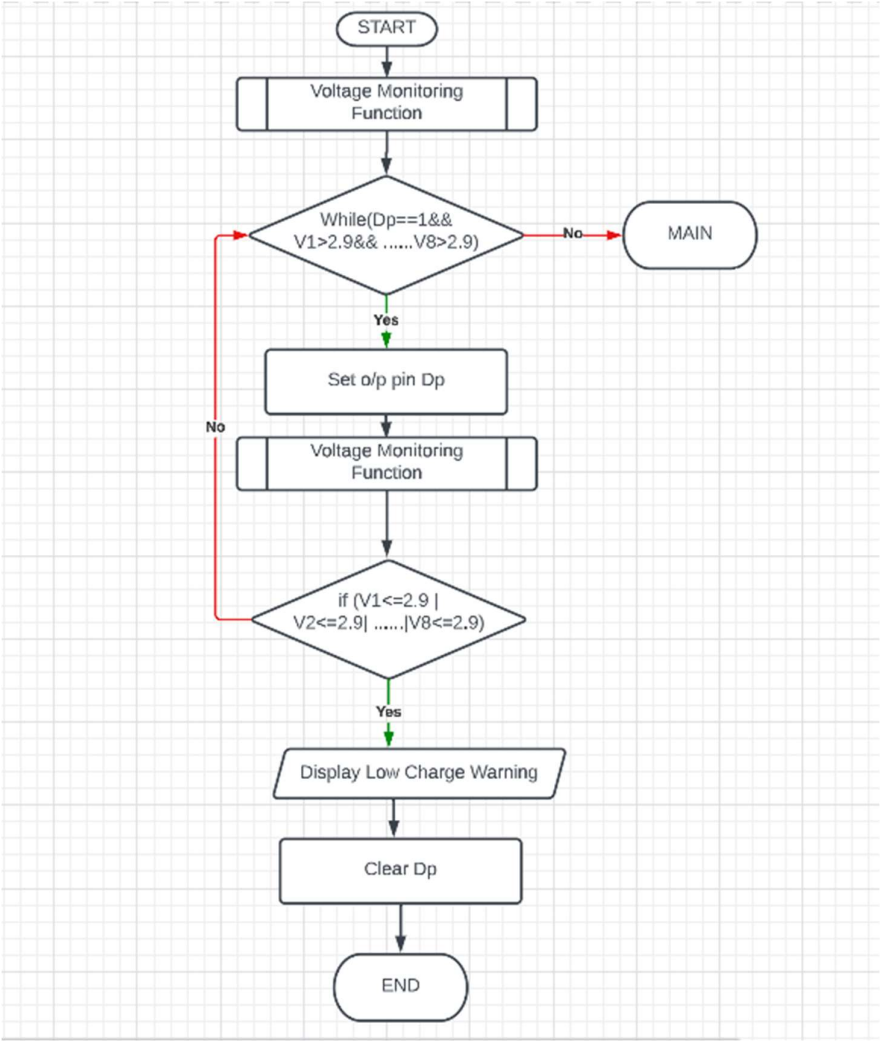
10. Clear output pin connected to D- Relay (Discharging Relay).
11. Call voltage monitoring function ().
12. while Cp==1 and V1<=3.65 and V2<=3.65 and.....V8<=3.65.
13. Call Charger function ().
14. After coming out of the while loop Clear o/p connected to C- Relay.
15. Call voltage monitoring function ().
16. If 3.65<V1<V2<v3<V8.

Flowcharts:

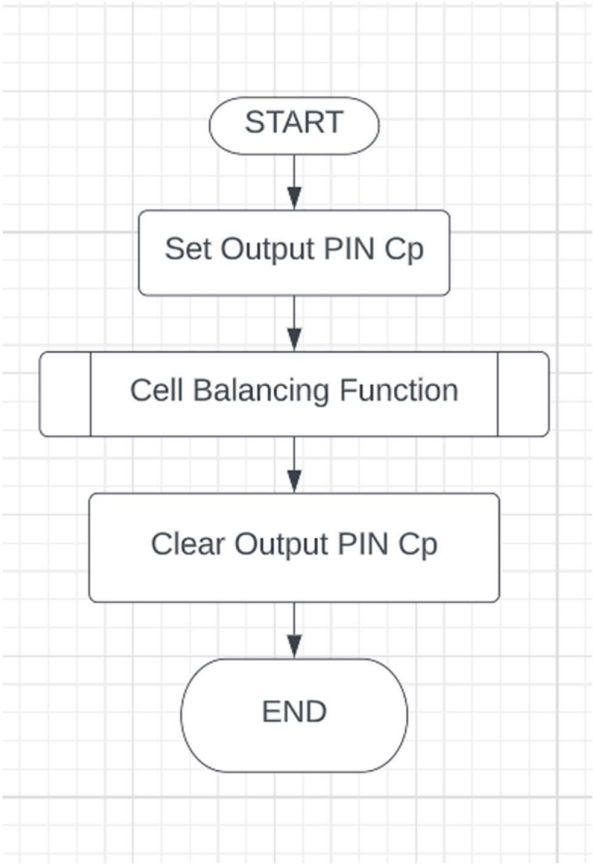
Voltage Monitoring Function:



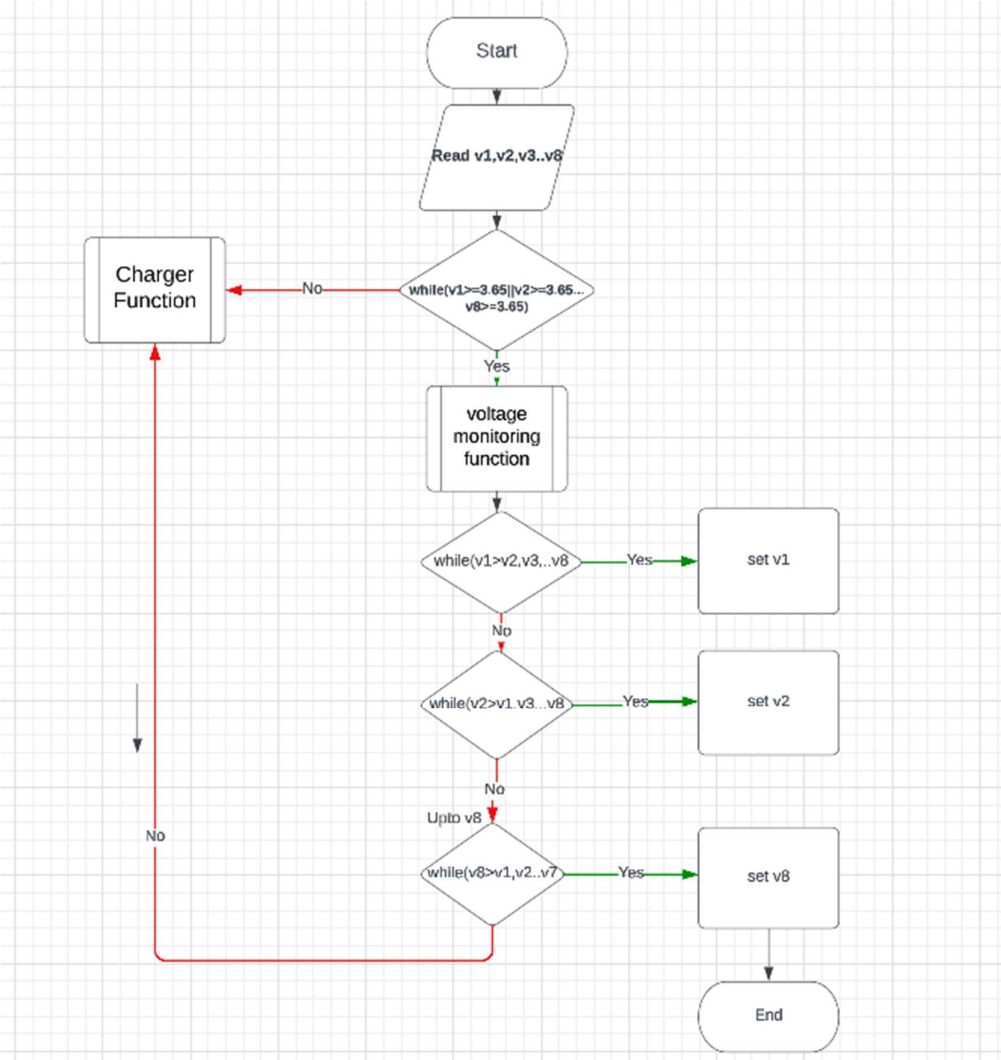
Discharging function:



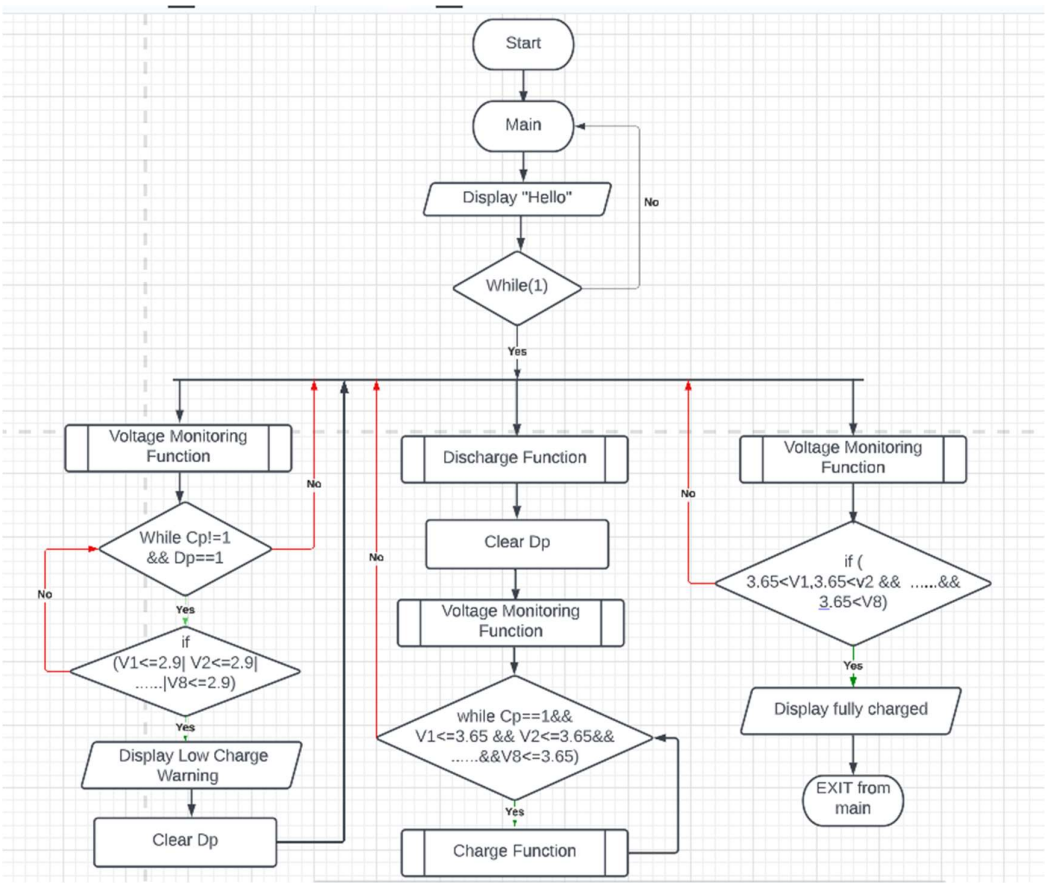
Charger Function:



Cell Balancing Function:



Main Function:



Challenges faced:

Reading the individual stack voltages: Voltage sensor can take a maximum input voltage of 25V but the voltage of the battery pack could go all the way till 30-32 volt.

Balancing: One of the main challenges in BMS is balancing the cells in a battery pack. Each cell can have a slightly different capacity or state of charge, which can cause some cells to discharge faster than others. BMS must ensure that all cells in the pack are balanced, which requires monitoring the voltage and current of each cell and redistributing the energy as needed.

Communication and data management: BMS must communicate with other systems and devices to ensure proper battery operation. This includes monitoring the battery's status, sending alerts and warnings, and controlling the battery's charging and discharging processes.

Data from database was not going to mobile app as the location of db was not default (us- central).

Initially when tested with stm32, we did not get proper data in UART due to mismatch of baudrate.

Setting up developing environment:

The STM32CubeIDE is a complete development system to develop code for almost all STM32-based microcontrollers from ST Microelectronics. As the name suggests, it is an Integrated Development Environment (IDE) that essentially includes the STMCubeMx GUI HW configuration tool, and a full compiler.

It can be used as a development platform for all STM32 MCU's, whether it is on a development board such as from one of ST's Nucleo or Discovery family, or a custom-designed board.

Configuring the Hardware:

Step1. The first thing to do is to install STM32CubeIDE.Note that registration is required.

Step2. After installation, you can start up the application. Select from File-New-STM32 as shown in Figure 1

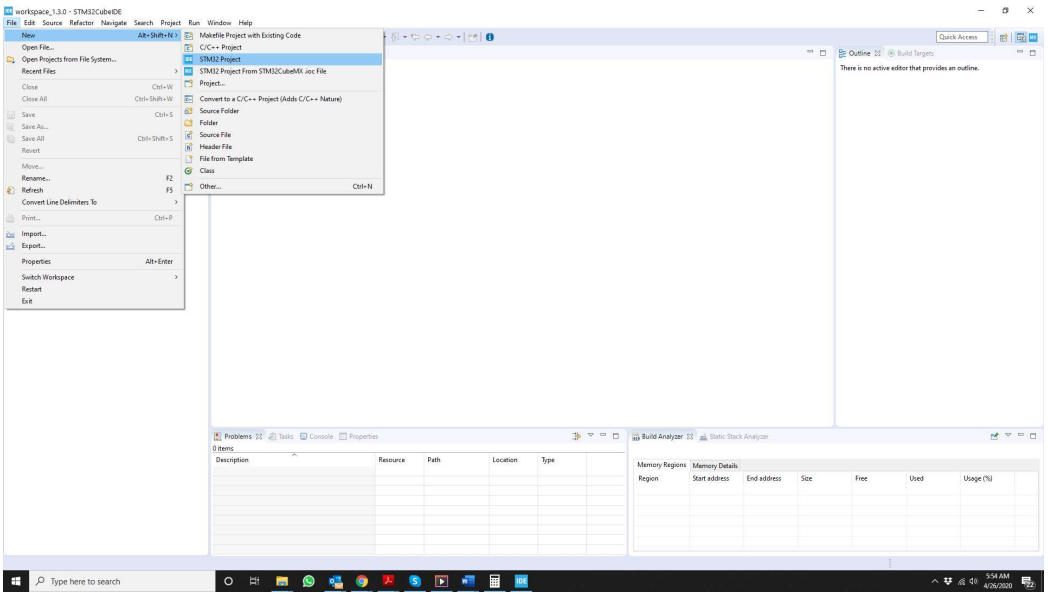


Figure 1 – initial screen of STM32Cube IDE showing selection drop-downs for a new project

Step3. After a little while, the device selection screen shown in Figure 2 will appear.

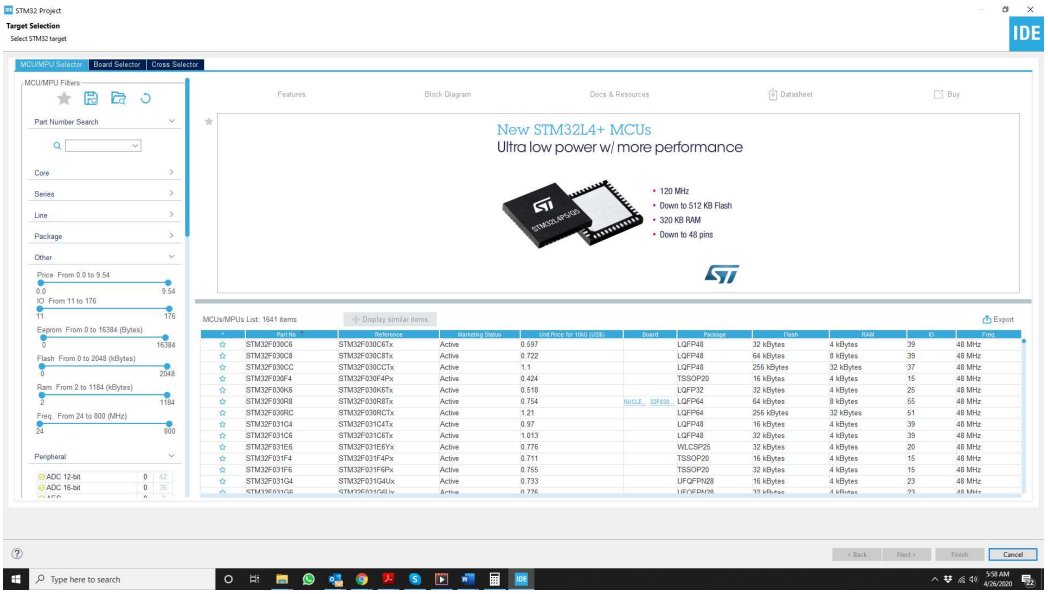


Figure 2 – Device selection screen

Step4. The next screen requests a name for the project, and also some information about the type of project. In this case, as shown in Figure 3. the project will be an executable called Blink, and it will be done using C.

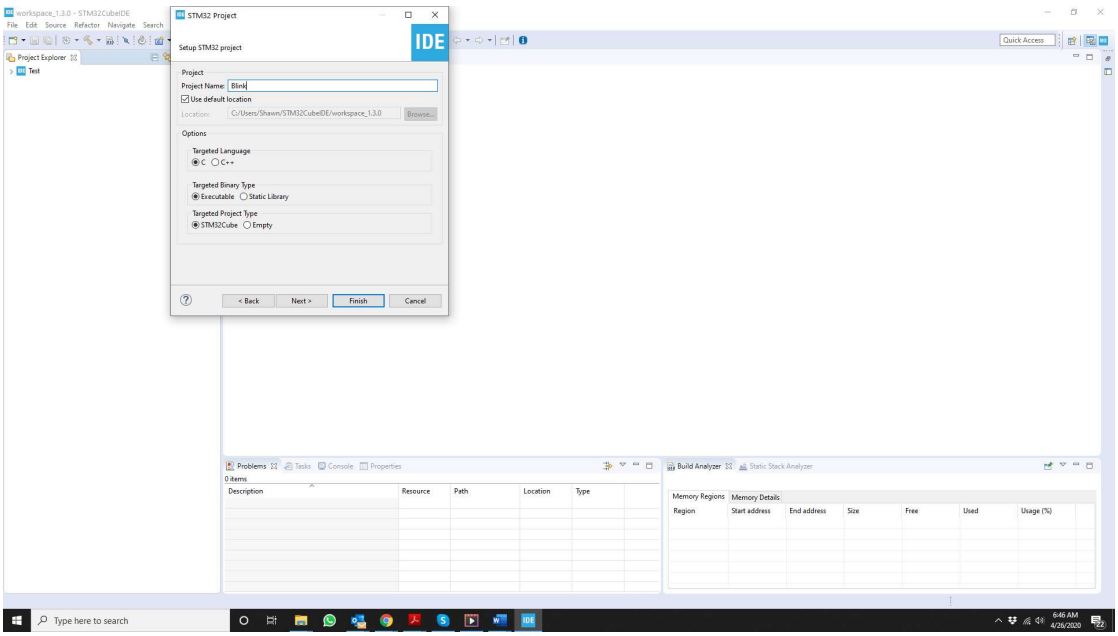


Figure 3 – Project information screen

Step5. After a while, and answering "yes" to the pop-up about STMCube, the screen shown in Figure 4 will appear. This is the device pinout view screen. It allows the user to choose the function of the MCU pins.

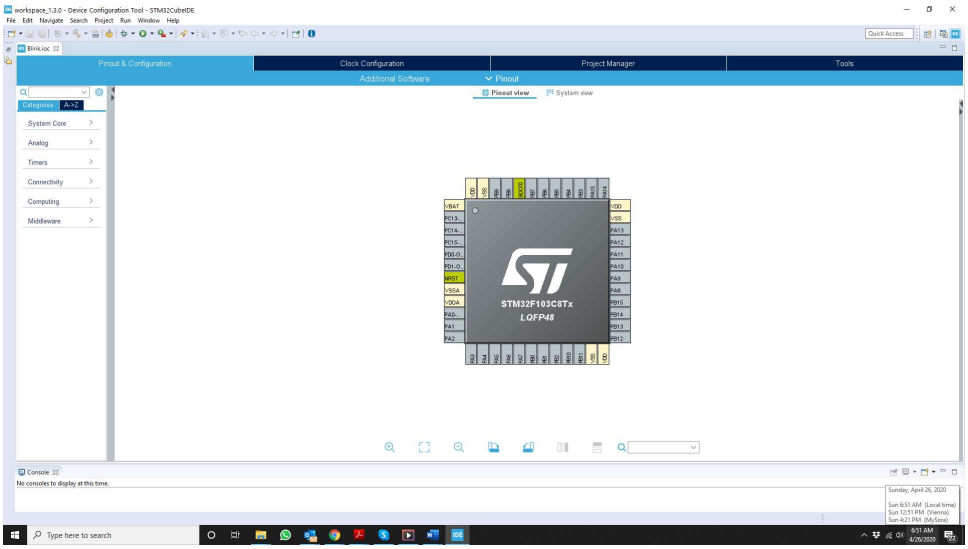


Figure 4 – Pinout view of the STM32F103C8Tx MCU

Step6. The other options allow the peripherals to be clocked at frequencies other than core clock. This can be used to reduce power consumption, for instance. However, in this case, the rest will be left as is. Figure 5 actually shows the proper values as just described

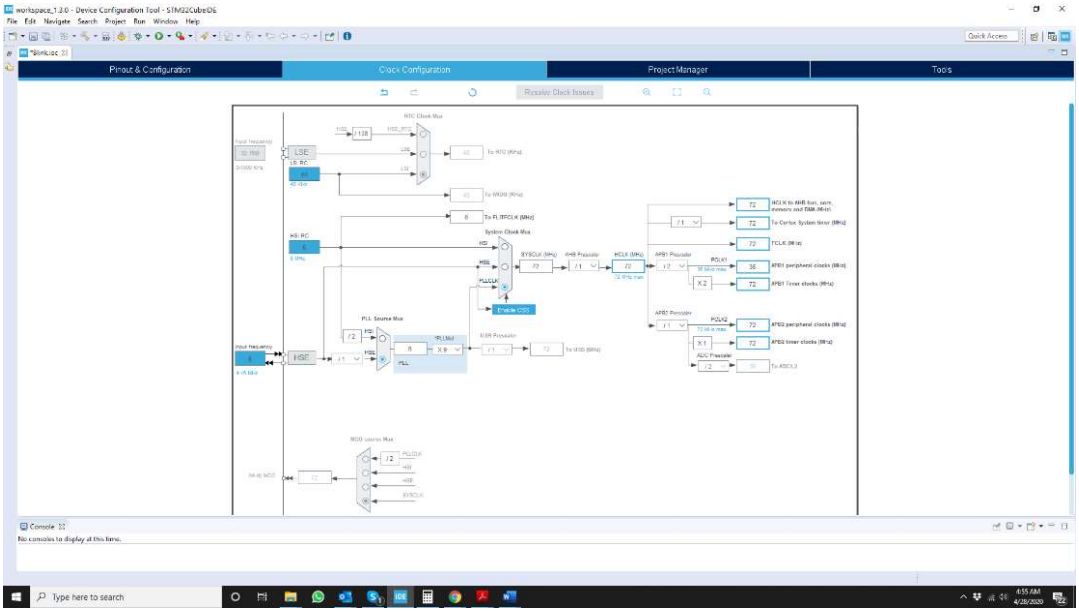


Figure 5 – Clock configuration screen

Step7. Now, select System View on the main window, and select GPIO. The screen will look like that shown in Figure 6. In the GPIO Mode and Configuration window, various settings for the GPIO pin can be selected.

As shown in Figure 6, the initial output is Low, the mode is push-pull and there is no pull-up or pull-down. That should be fine for this application, but just be aware that these settings can be changed to suit the application. This completes the required settings for this application.

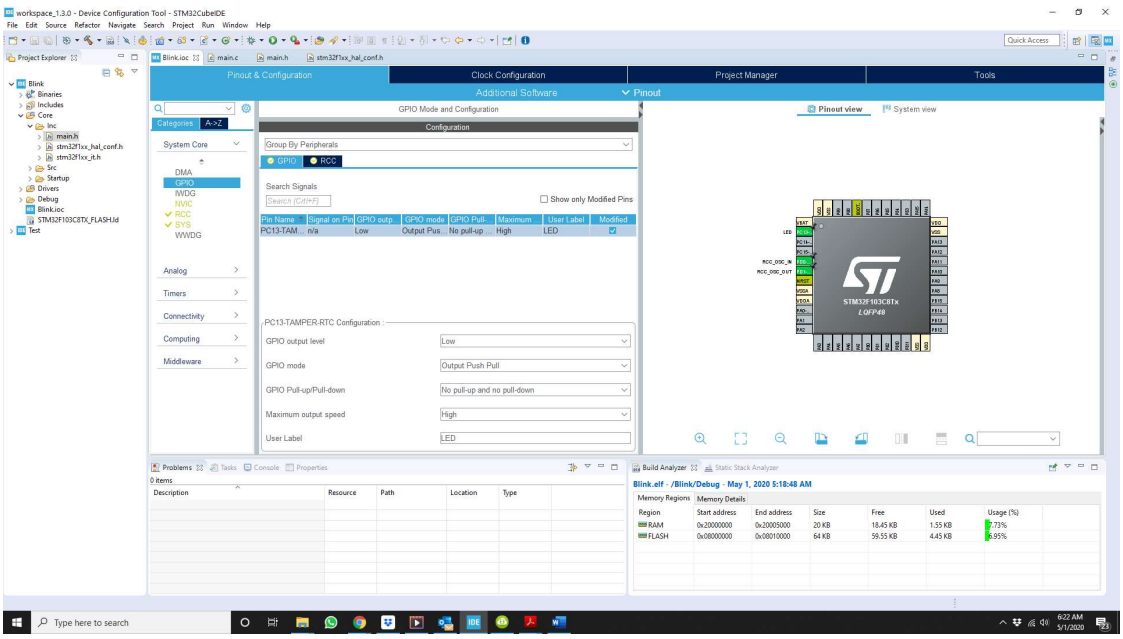


Figure 6 – System view screen

what does all the configuration in the previous section do? It allows the automatic generation of initialization code for setting up the clock and GPIO.

Step8. Just go the "Project" tab, and click on "Generate Code". A pop-up progress window will appear, and after it is done, navigate to the "Project Explorer" tab.

From there go to "core", then to "Src", and click on main.c. This will open up the main.c file in the main window. As can be seen in figure 7, lots of code has been generated already.

Scrolling down this window to the *int main(void)* function, it is seen that it already calls three functions: *HAL_Init()*, *SystemClock_Config()*, and *MX_GPIO_Init()*. The actual

functions are defined further down in main.c. These are the HW setup functions that were automatically generated based on the user configurations entered previously.

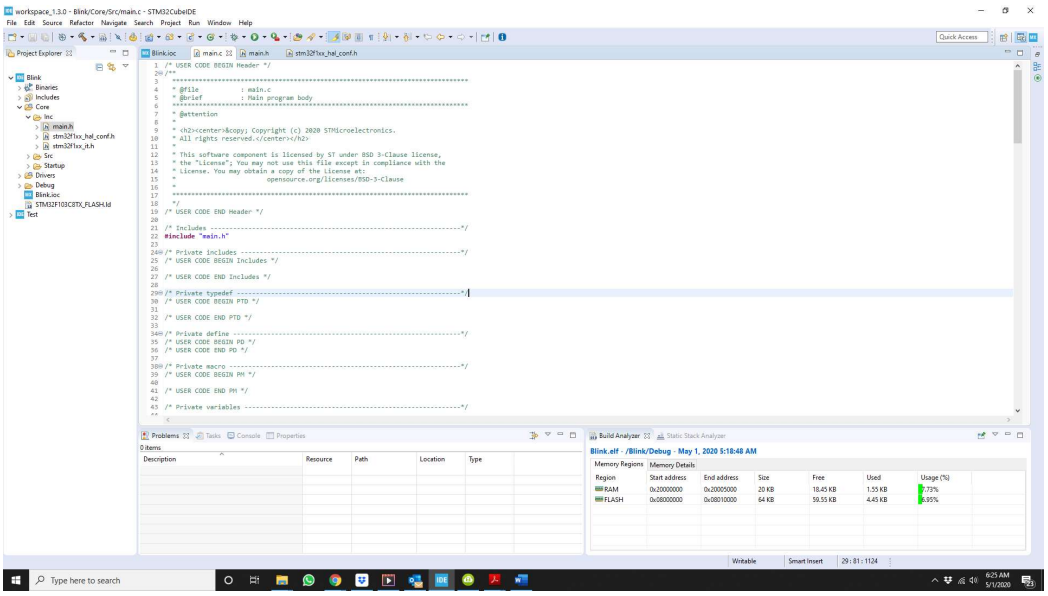


Figure 7 – main.c

Step9. Also note that there are various sections with comments that start with something like /* USER CODE BEGIN WHILE */, and /* USER CODE END WHILE */. The space between any comments that are bracketed by comments that start with "USER CODE …" is where the user is expected to enter the application-specific code. This is shown in figure 8.

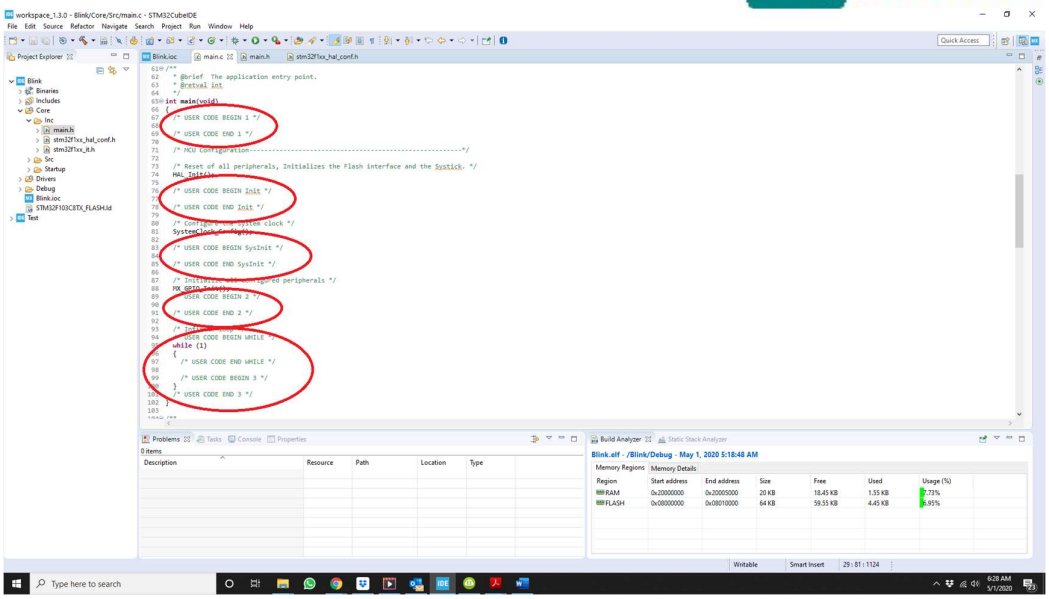


Figure 8 – main.c areas where the user can enter code

Step10. To compile the code, simply go to the Project tab, and select Build All. Assuming there are no compile errors move to the Debug folder and locate the .bin file. As shown in figure 13, this is Blink.bin. This is the file that has to be loaded to the actual STM32.

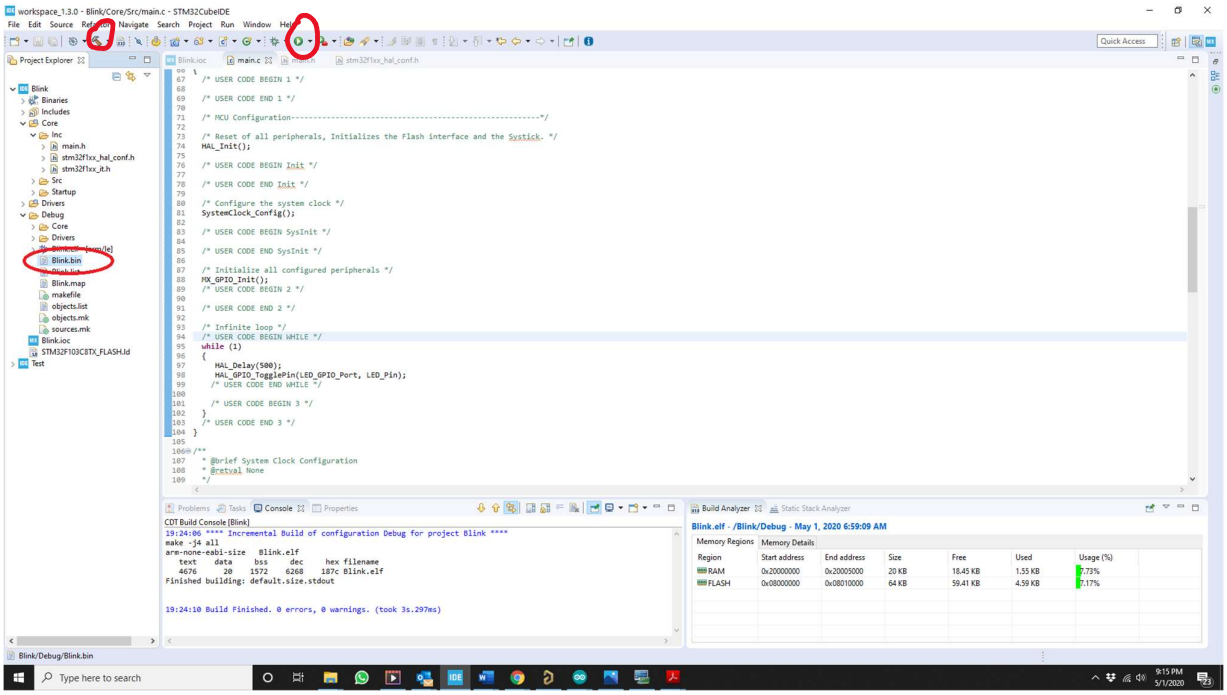
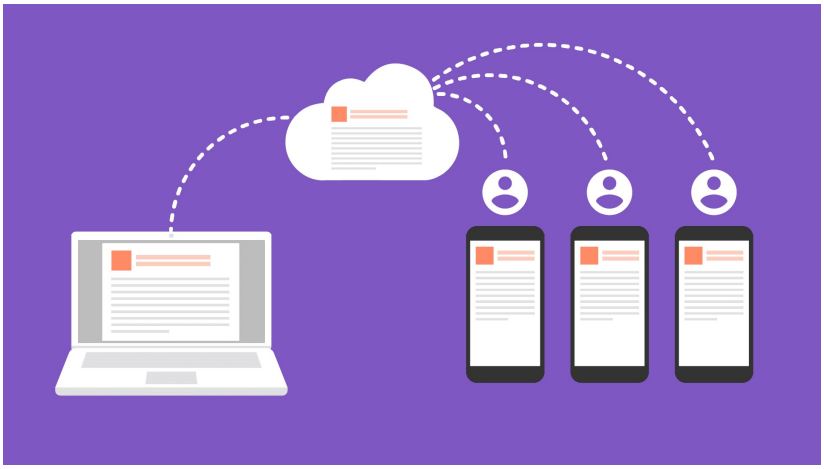


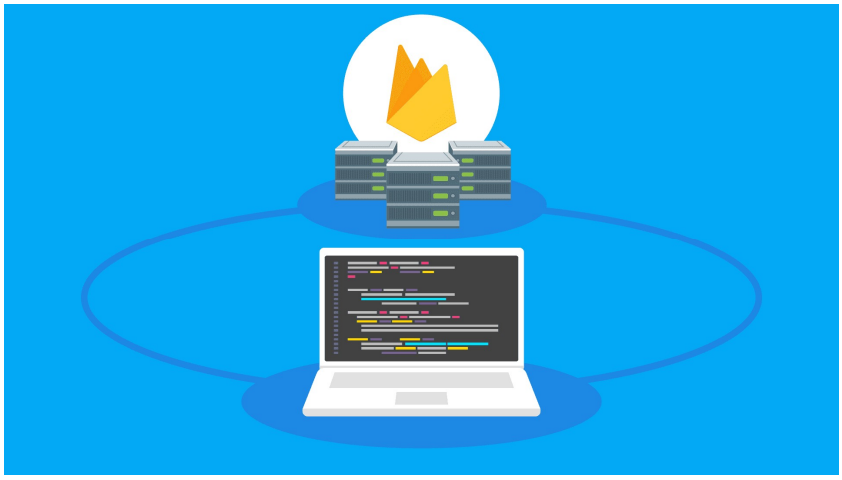
Figure 9 – Locating the .bin file

Google Firebase: The Firebase Realtime Database is **a cloud-hosted NoSQL database that lets you store and sync data between your users in realtime**. NEW: Cloud Firestore enables you to store, sync and query app data at global scale.

Collaborate across devices with ease: Realtime syncing makes it easy for your users to access their data from any device: web or mobile, and it helps your users collaborate with one another.

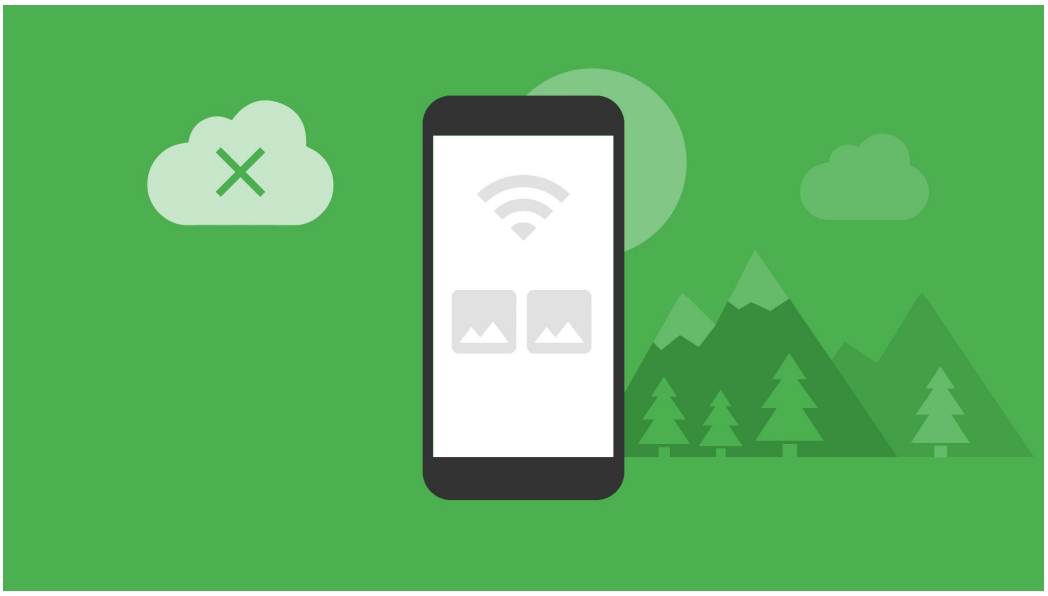


Build serverless apps: Realtime Database ships with mobile and web SDKs so you can build apps without the need of servers. You can also execute backend code that responds to events triggered by your database using Cloud Functions for Firebase.



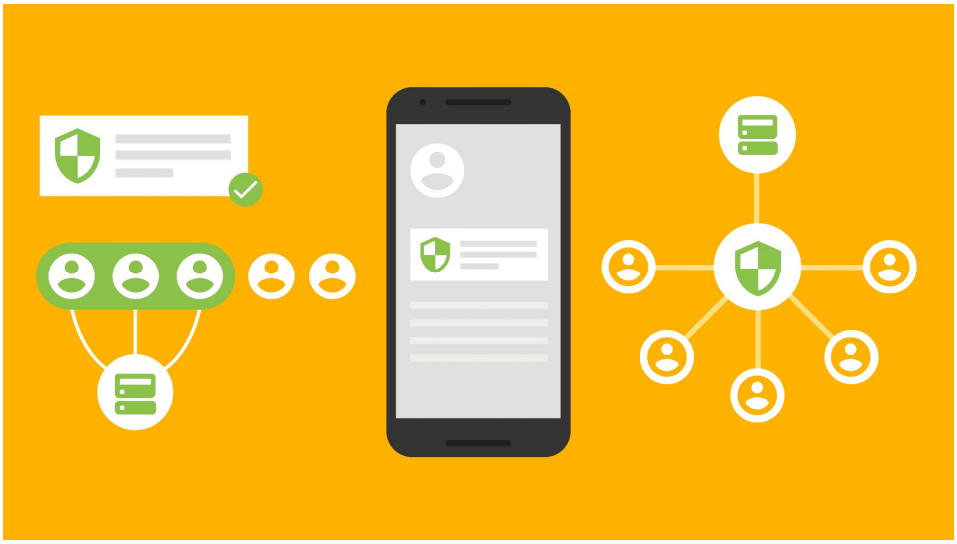
Optimized for offline use:

When your users go offline, the Realtime Database SDKs use local cache on the device to serve and store changes. When the device comes online, the local data is automatically synchronized.



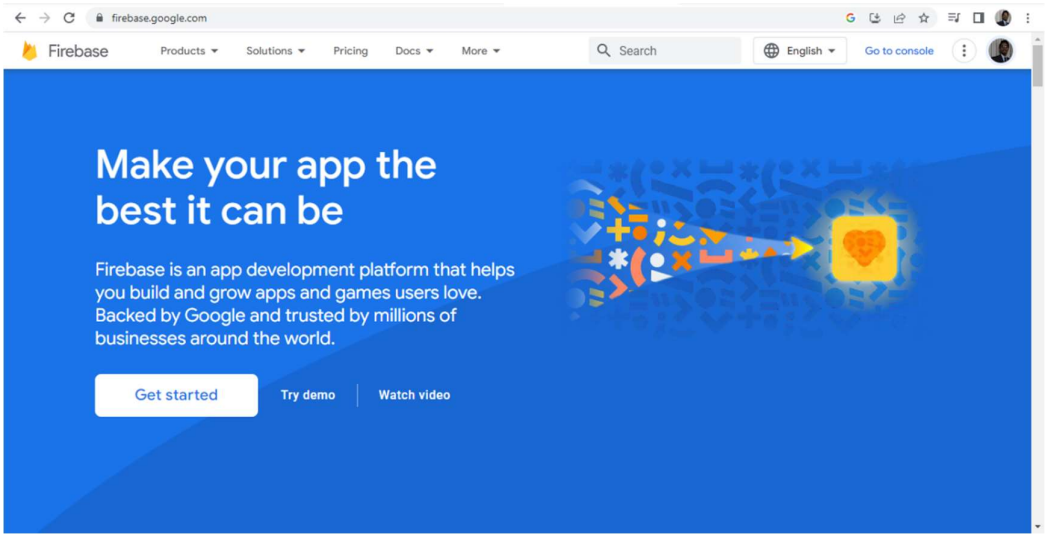
Strong user-based security:

The Realtime Database integrates with Firebase Authentication to provide simple and intuitive authentication for developers. You can use our declarative security model to allow access based on user identity or with pattern matching on your data.



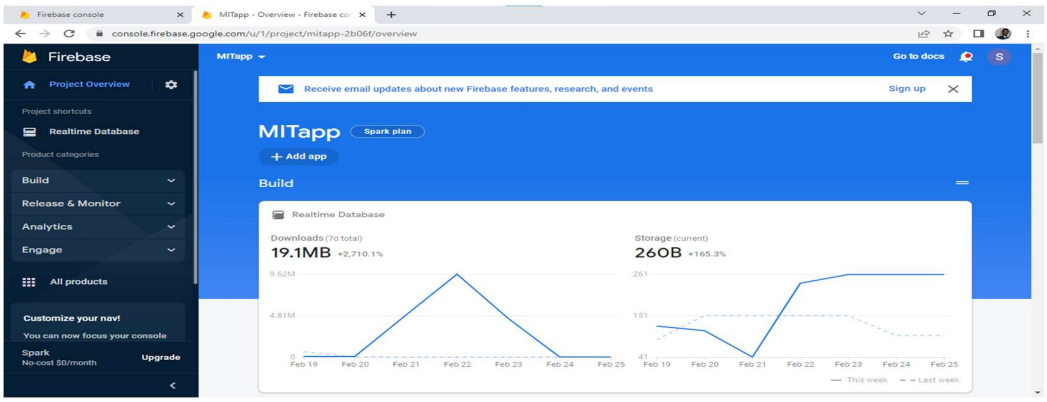
Creating a database in firebase

Step1. Go to firebase.google.com> Sign in using your G-mail account> go to console.

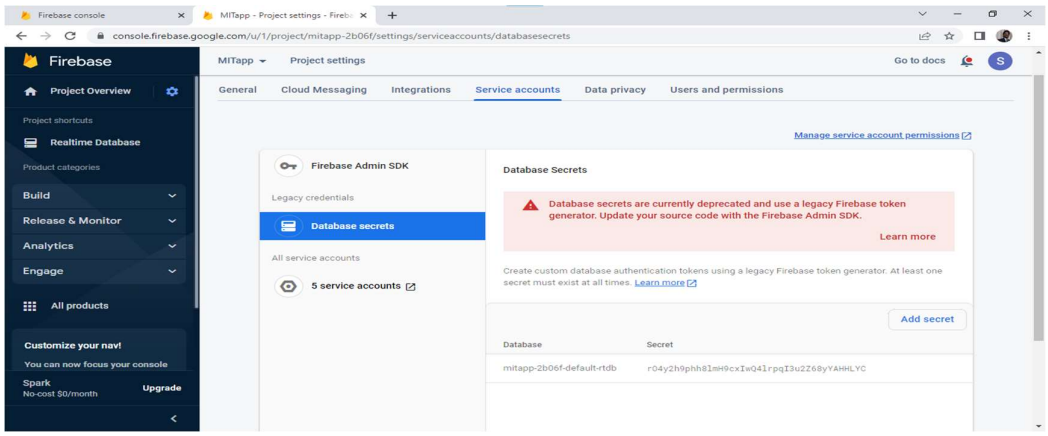


Step2. Add a project in the console> Enter the name for the project> continue> continue> select the google

analytics account> create project> click continue on successful creation of the project.

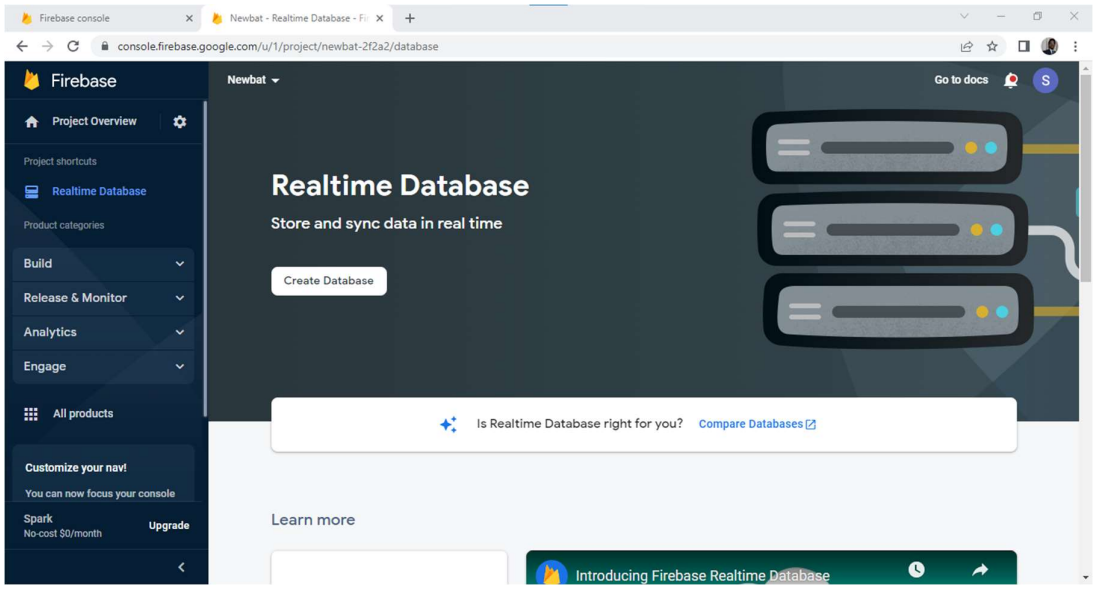


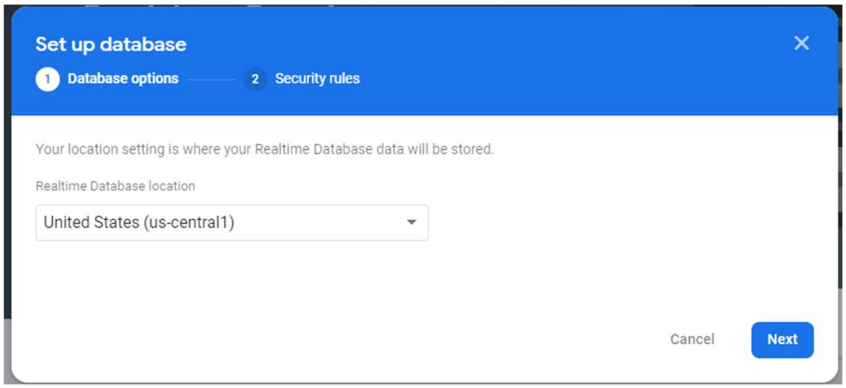
Step3. In the project overview> go to project settings in the top left> go to service accounts> database secrets> Here we get the API key for the particular database project. It is required in Arduino scripting.

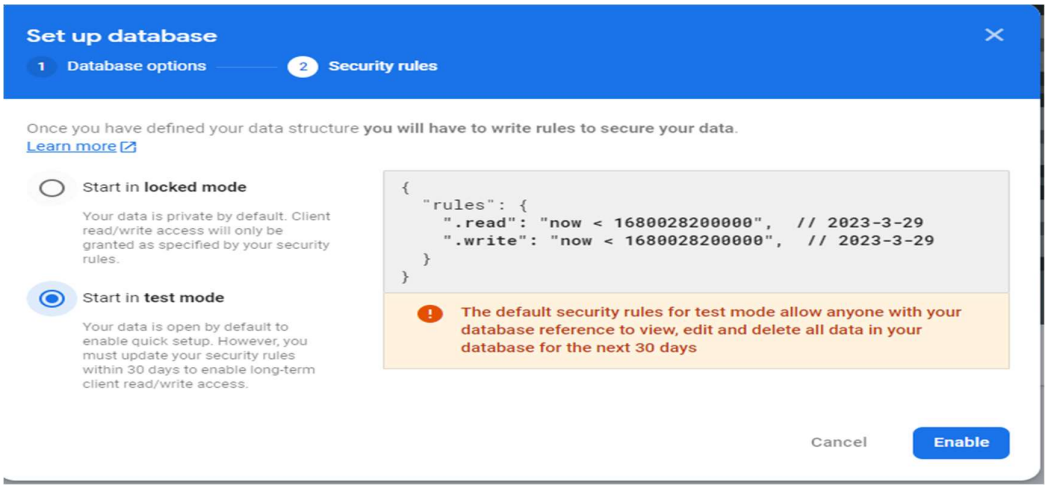


Step4. Click on realtime database on the left side> create database> start in test mode> next> Select Database location (default us-central)> done.

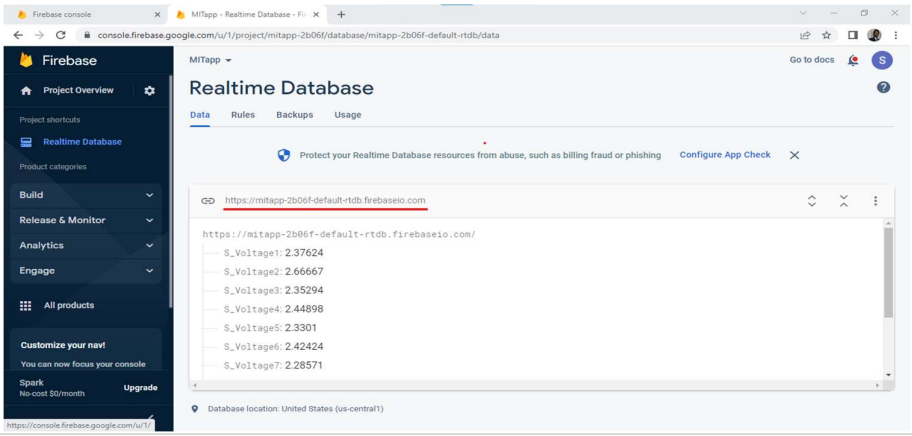
Security rules> start in test mode> enable



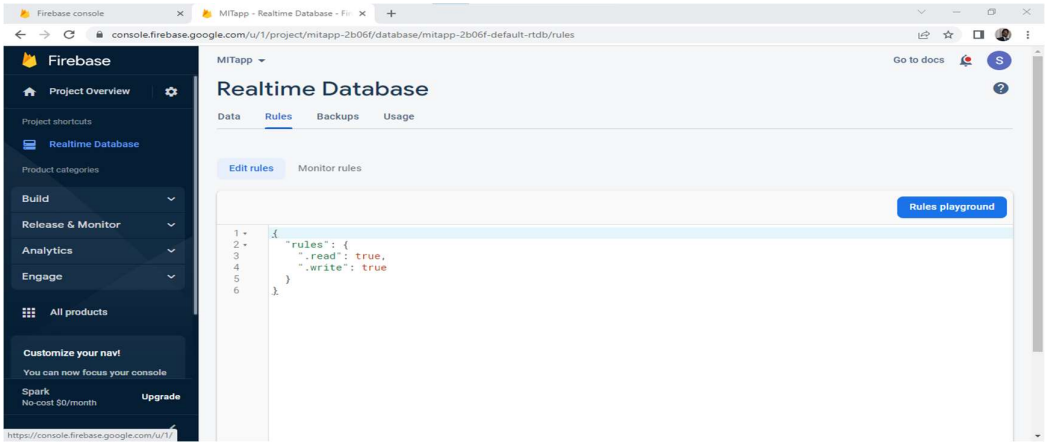




Step5. Go to build in left corner> real time database> copy the database URL for Arduino scripting.

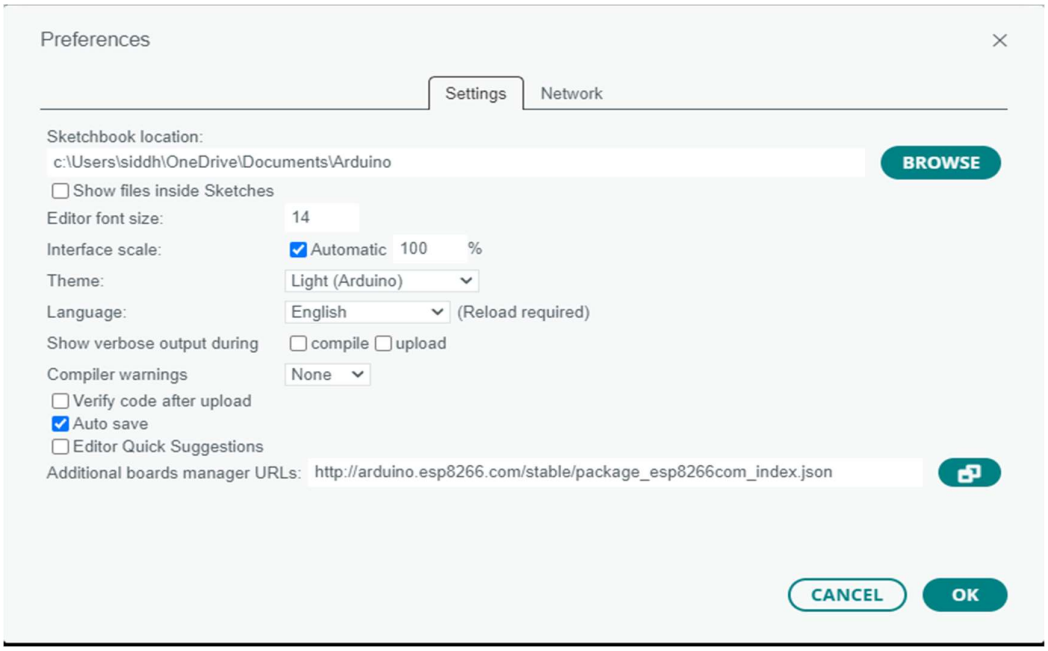


Step6. Need to change the rules for read and write as true> publish.

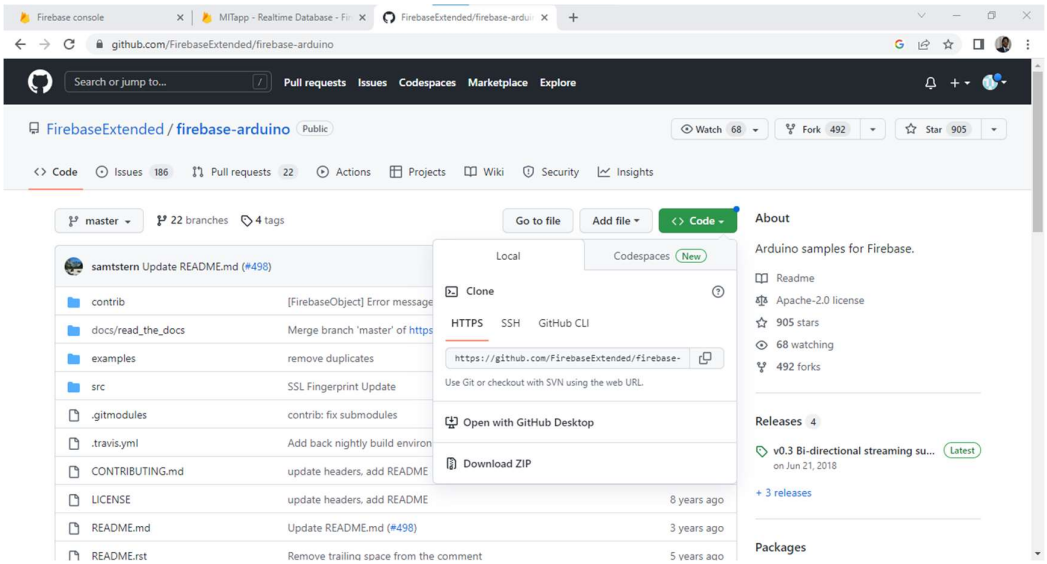


Node MCU (ESP8266) programming:

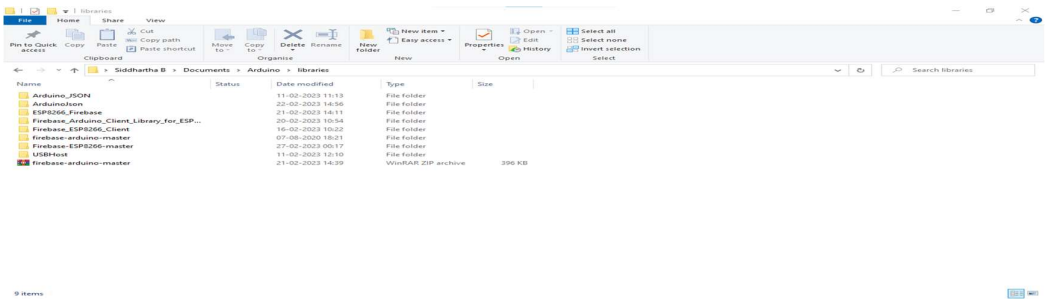
Step1. In the Arduino IDE> open new file> go to file> preferences> paste this URL http://arduino.esp8266.com/stable/package_esp8266com_index.json in the additional board manager> OK



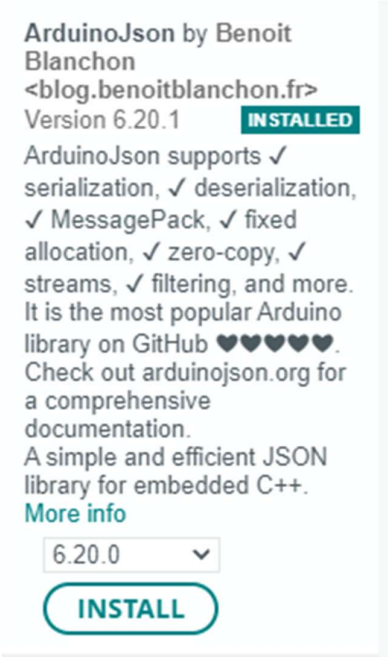
Step2. Go to Firebase Arduino library> download the zip file> go to this link https://github.com/FirebaseExtended/firebase-arduino> go to codes> download ZIP.



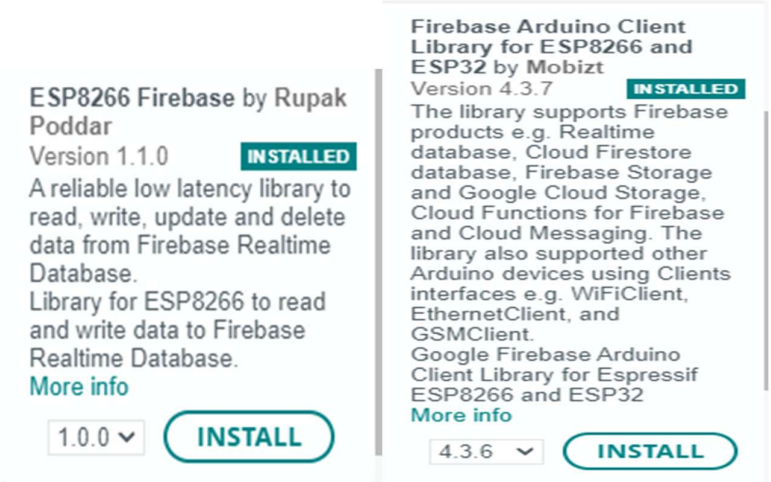
Step3. Go to documents> Arduino> libraries> copy this file in this folder and extract.



Step4. In the Arduino IDE> tools> manage libraries> search for arduino json> install the version above 5.13.1



Step4. Install the ESP8266 firebase v1.1.0 library, firebase arduino client library for ESP8266.



The following code is uploaded into node MCU:

```
#include <ESP8266WiFi.h> // library header file
#include <FirebaseESP8266.h> // library header file
#include <SoftwareSerial.h> //library file for UART communication
#define FIREBASE_HOST "mitapp-2b06f-default-rtdb.firebaseio.com" //Database host
#define FIREBASE_AUTH "rO4y2h9phh8lmH9cxIwQ4lrpqI3u2Z68yYAHHLYC" // Database web API for writing to the database
#define WIFI_SSID "RVAELP021 8857" // Wifi SSID to connect ESP8266
#define WIFI_PASSWORD "rvae@12345" // Wifi password to connect ESP8266
                                    FirebaseData firebaseData;
float total;
float out, out1, out2, out3;

 void setup()
        {
            Serial.begin(9600); // Baud rate for UART communication
            randomSeed(analogRead(0));

            WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
            Serial.print("connecting"); //print on the serial monitor
            while (WiFi.status() != WL_CONNECTED)
                    {
                        Serial.print(".");
                        delay(500);
                    }
            Serial.println();
            Serial.print("connected: ");
            Serial.println(WiFi.localIP());

            Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
        }
```

```
void loop()
    {
        if (Serial.available()) //if anything available in the serial monitor of the other microcontroller
    {
                float data= Serial.parseFloat();
                Serial.print("val= ");
                Serial.println(data);
                out=data;
 Firebase.setFloat(firebaseData, "/voltage0", out);  // to send the data to the firebase*/
 if (Serial.available()) //if anything available in the serial monitor of the other microcontroller
{
        float data1= Serial.parseFloat();
       Serial.print("val1= ");
       Serial.println(data1);
       out1=data1;
      Firebase.setFloat(firebaseData, "/voltage1", out1);
if (Serial.available()) //if anything available in the serial monitor of the other microcontroller
    {
        float data2= Serial.parseFloat();
        Serial.print("val2= ");
        Serial.println(data2);
        out2=data2;
       Firebase.setFloat(firebaseData, "/voltage2", out2);
if (Serial.available()) //if anything available in the serial monitor of the other microcontroller
     {
         float data3= Serial.parseFloat();
         Serial.print("val3= ");
         Serial.println(data3);
         out3=data3;
```

```
            Firebase.setFloat(firebaseData, "/voltage3", out3);
            total= out+out1+out2+out3;
            Firebase.setFloat(firebaseData, "/total_volt", total);
                        }
                    }
            }
        }
}
```

Connection of micro controller with node MCU:

Tx of micro controller -> Rx of Node MCU

Rx of micro controller -> Tx of Node MCU

Gnd -> Gnd

The values will trigger to the arduino serial monitor and later gets triggered to the database provided with the credentials in the code.

MIT App Inventor

MIT App Inventor is an online platform designed to teach computational thinking concepts through development of mobile applications. Students create applications by dragging and dropping components into a design view and using a visual blocks language to program application behavior.

MIT App Inventor Overview:

The MIT App Inventor user interface includes two main editors: the design editor and the blocks editor. The design editor, or designer (see Fig. 1), is a drag and drop interface to lay out the elements of the application's user interface (UI). The blocks editor (see Fig. 2) is an environment in which app inventors can visually lay out the logic of their apps using color-coded blocks that snap together like puzzle pieces to describe the program. To aid in development and testing, App Inventor provides a mobile app called the App Inventor Companion (or just "the Companion") that developers can use to test and adjust the behavior of their apps in real time. In this way, anyone can quickly build a mobile app and immediately begin to iterate and test.

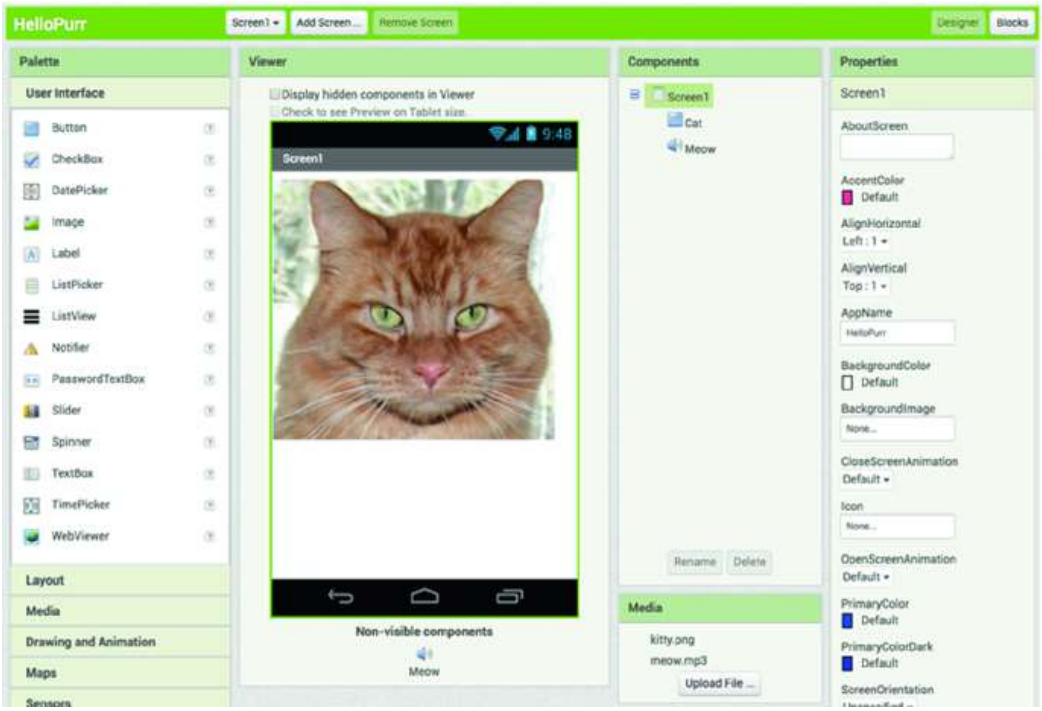


Fig.1

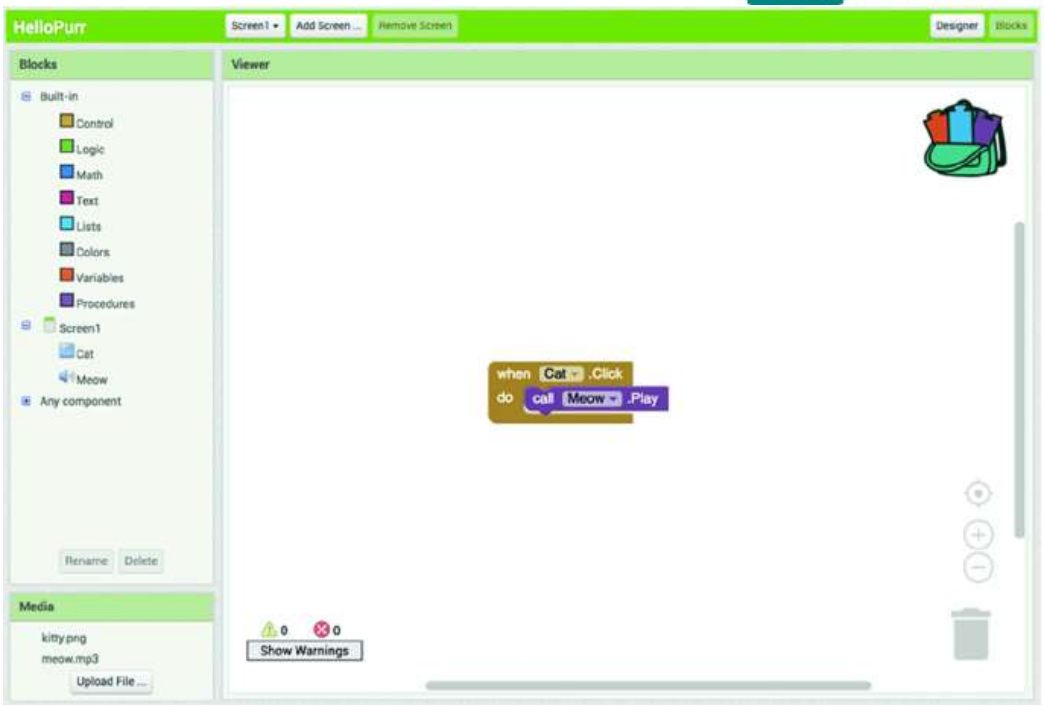
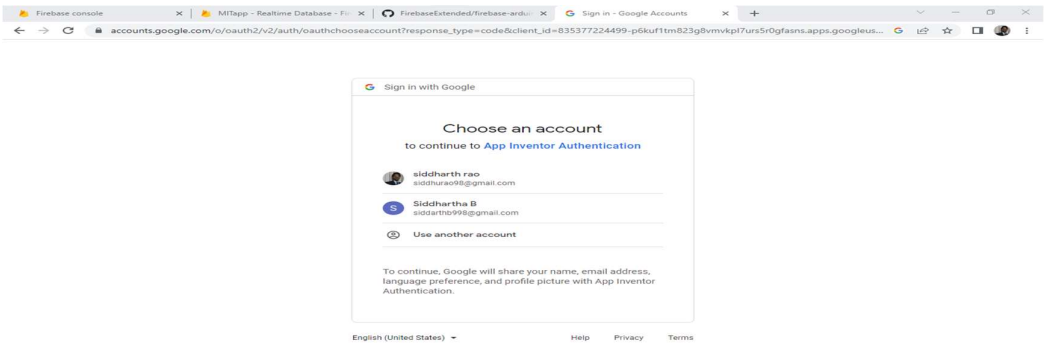


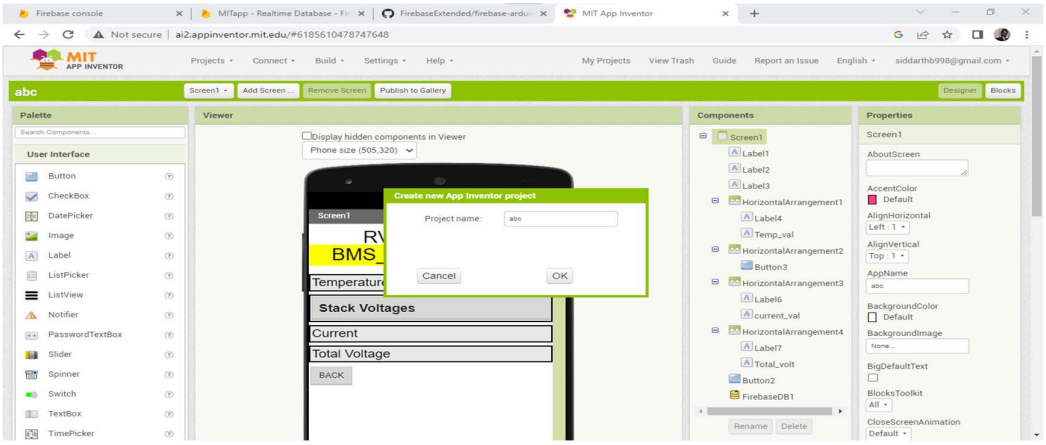
Fig.2

This is an interface to create mobile app for any particular application.

Step1. For creating a application in MIT app inventor, go to link http://ai2.appinventor.mit.edu/ > sign in with your Gmail account



Step2. Go to projects> start new project> give a name to the project> Select OK

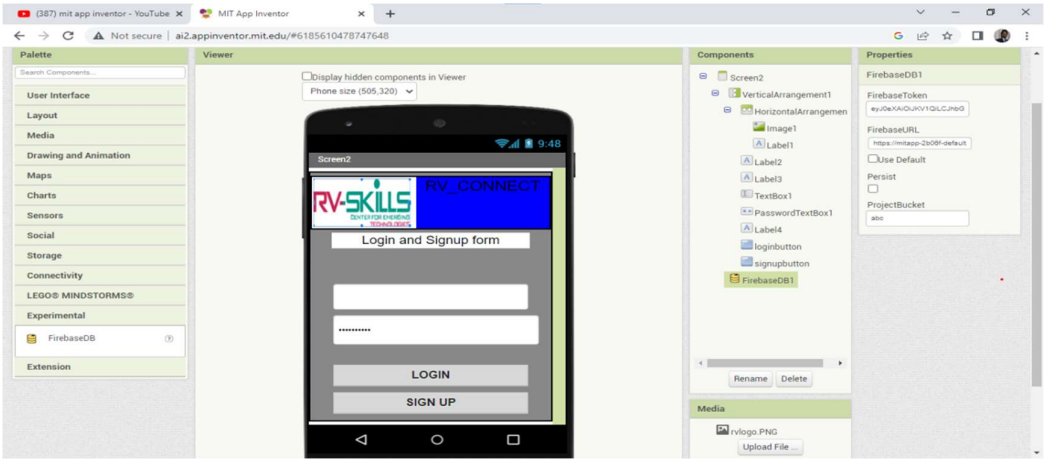


Step3. Designing the login screen.

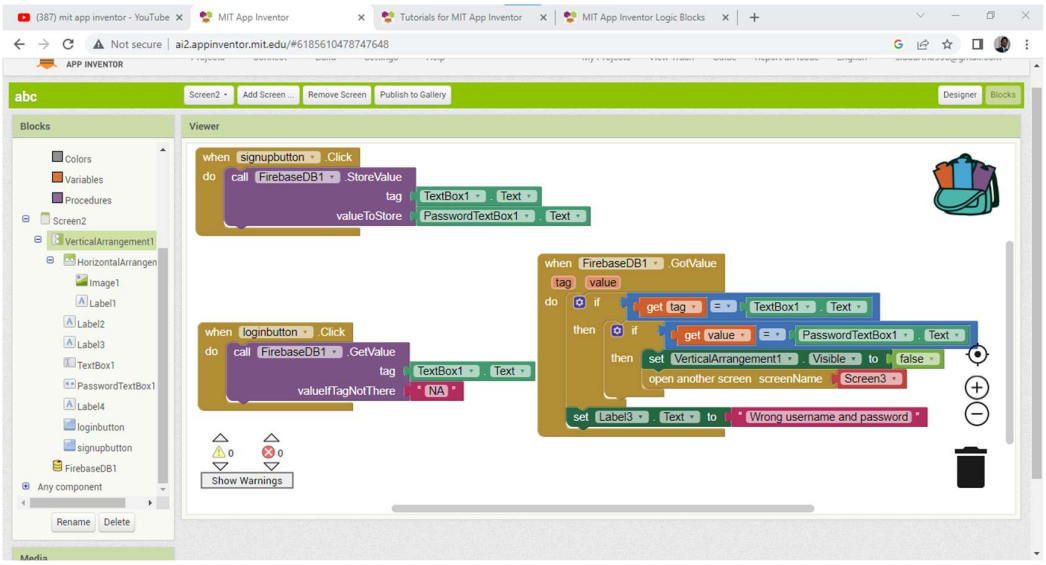
Initially we have to add the firebase database to the components in the right side, Go to experimental on the left side of the interface> under this we can find the firebase DB> add this to the component.

Once we get the select the database > we get the properties of the DB> we have to add the database URL in this.

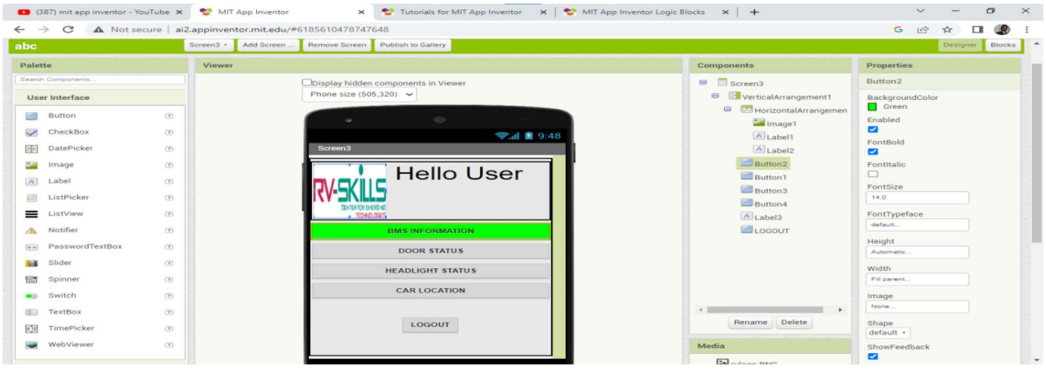
Design of the first page looks like this



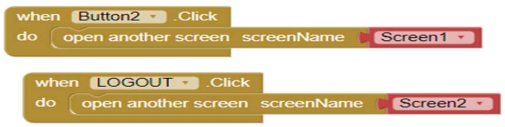
Logic for this:



Screen 2 design:

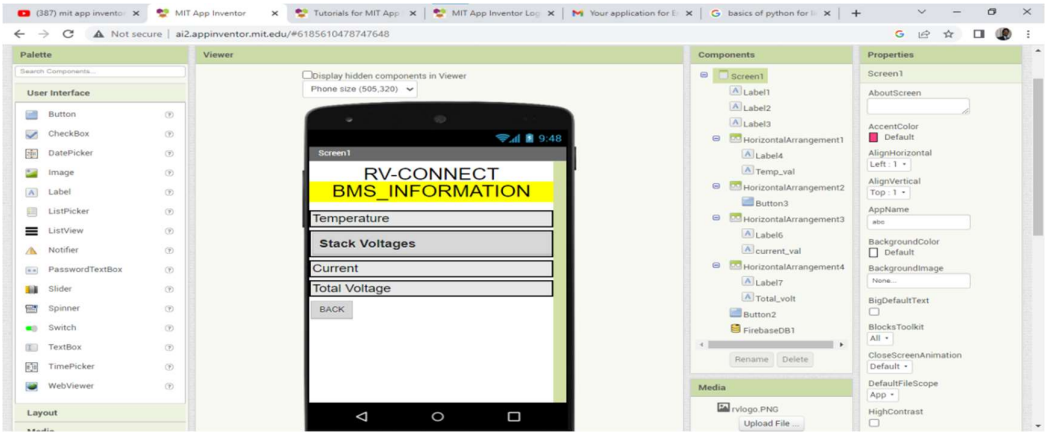


Screen 3 design:

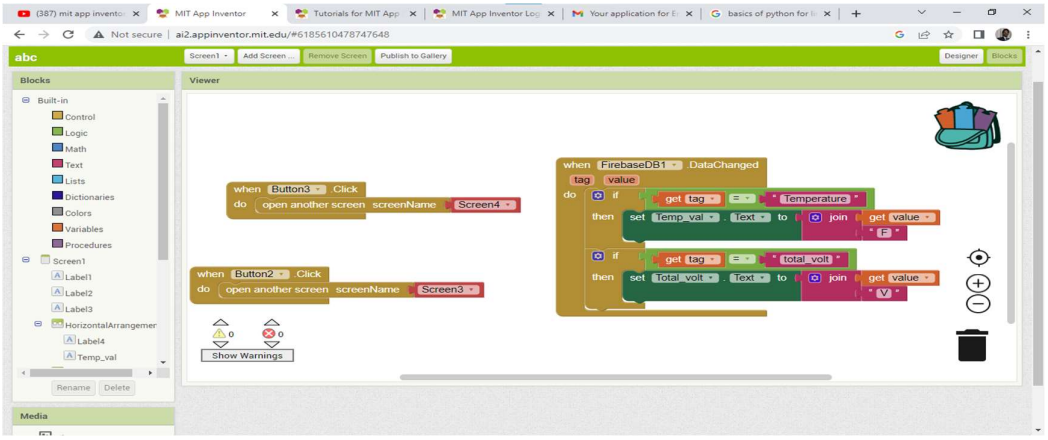


Screen 4 design:

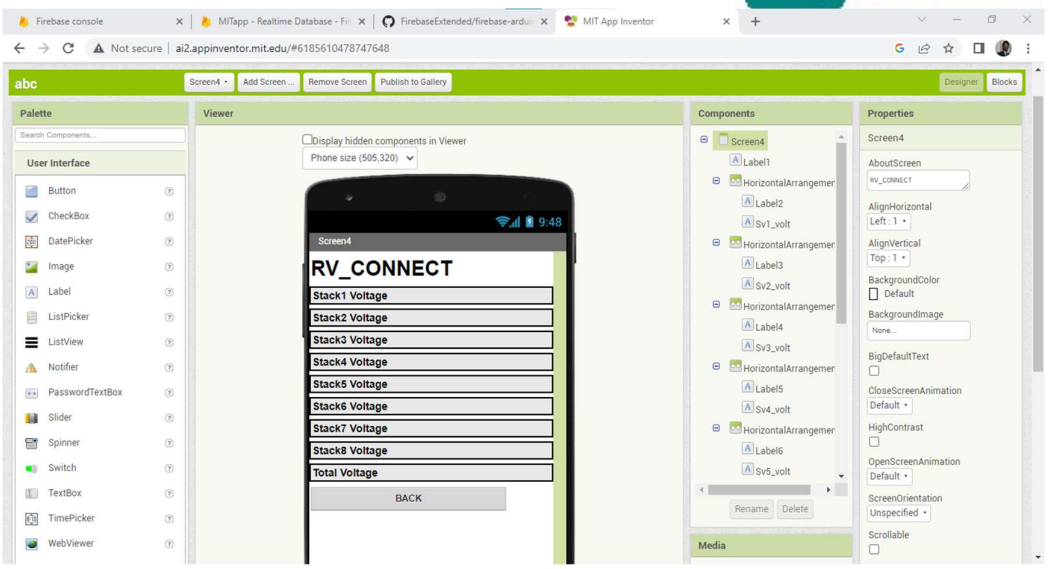
Step4. Need to add the database in the screen as we are taking the values from the database.



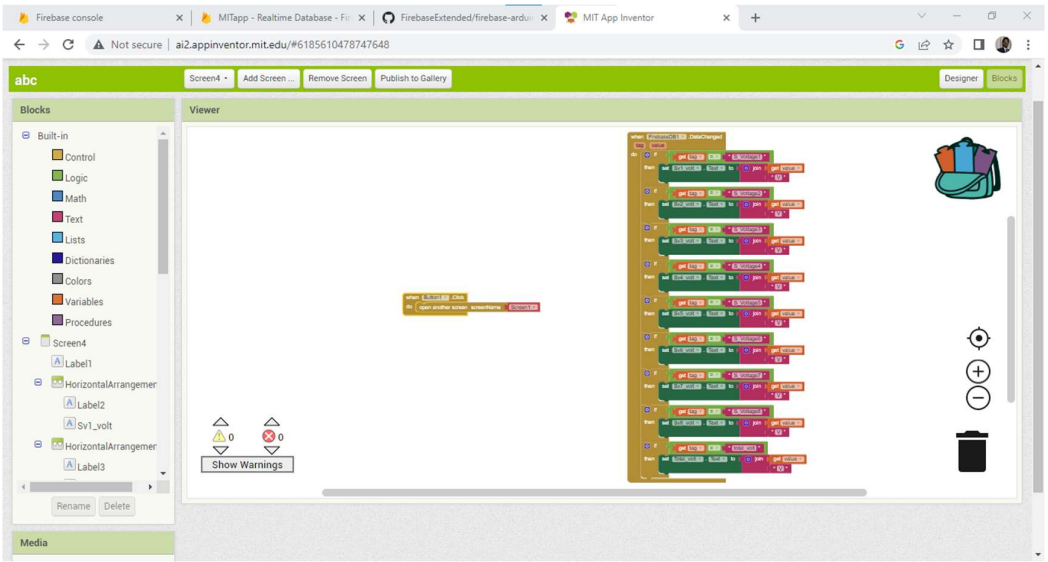
Screen 5 design:



Screen 6 design:



Screen 7 design:



Coding Standards followed:

Code:

1. Code should be indented.
2. No constants are allowed in the code.
3. All the constants should be #defined.
4. Code should be modular.
5. Code should be well commented.

Functions/identifies:

1. All functions should have declarations.
2. Declarations should be in header files.
3. If function declarations are in source file they should appear after #defines.
4. Function names should match with the names in the design document.
5. Lengthy function names should be combined with _.
   Ex: battery_pack_voltage ()
6. Every function should be preceded with a header comment describing the task the function performs.

Ex:

```
/*
  Display battery pack voltage
*/
battery_pack_voltage (){
}
```

Variables/identifies

1. All the local variables should be declared in the beginning of the functions.
2. Long variable names should be combined with _.
   Ex: voltage_battery_pack
3. The names of the variables should match to the names in the design document.
4. Unless needed variables should have local scope.

The following code is uploaded into STM32F103RBT6:

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define VB_FET_0 GPIO_PIN_14   //PC
#define VB_FET_1 GPIO_PIN_14   //PA
#define VB_FET_2 GPIO_PIN_3            //PB
#define VB_FET_3 GPIO_PIN_15   //PC
#define VB_FET_4 GPIO_PIN_5            //PB
#define VB_FET_5 GPIO_PIN_9            //PB
#define VB_FET_6 GPIO_PIN_11   //PB
#define VB_FET_7 GPIO_PIN_12   //PB
#define C_FET    GPIO_PIN_7            //PC
#define D_FET    GPIO_PIN_8            //PC
/*#define BMOS9 GPIO_PIN_13
#define BMOS10 GPIO_PIN_14
#define BMOS11 GPIO_PIN_15
#define CMOS12 GPIO_PIN_6
#define CMOS15 GPIO_PIN_12*/
#define C_PIN  GPIO_PIN_12
```

```
#define D_PIN  GPIO_PIN_2
#define LOW  0
#define HIGH 1
#define VL_SAFE 2.9                    //Lower safe operating voltage
#define VU_SAFE 3.65          //Upper safe operating voltage
#define Vref 3.5                    //Reference voltage for ADC
#define CB_MIN_DIFF 0.15 //Minimum difference between the max and min voltage
desired for cell balancing
#define VL_RANGE 3.5
#define VU_RANGE 3.7
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;

/* USER CODE BEGIN PV */
int ADC_val0,ADC_val1,ADC_val4;
int ADC_val5,ADC_val6,ADC_val7;
int ADC_val8,ADC_val9;
int Temp,Current,CVAL,DVAL,check;

float Voltage0,V0,Voltage1,V1,Voltage4,V4;
float Voltage5,V5,Voltage6,V6,Voltage7,V7;
float Voltage8,V8,Voltage9,V9,volt[8];
float v1,v2,v3,v4,v5,v6,v7,v0,V_total;
float Temp_Val,T1, Current_Val,C1;

char msg[50],disp[50];
int v[]={1,2,3,4,5,6,7,8};
/* USER CODE END PV */
```

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);
/* USER CODE BEGIN PFP */
void Charge(void);
void Discharge(void);
void Cell_Balancing(void);
void ADC_Select_CH0(void);
void ADC_Select_CH1(void);
void ADC_Select_CH4(void);
void ADC_Select_CH5(void);
void ADC_Select_CH6(void);
void ADC_Select_CH7(void);
void ADC_Select_CH8(void);
void ADC_Select_CH9(void);
void ADC_Select_CH10(void);
void ADC_Select_CH12(void);
void ADC_Select_CH13(void);
void Battery_voltage(void);
void SerialPrint(float,int);
void voltage_monitoring(void);
void reset_FET(void);
void display(float ,int );
void CB_FET(int);
void Voltage_sorting(void);

float getVoltage(int);
int Check_Voltage(void);
/* USER CODE END PFP */
```

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/*
 Selecting ADC channel 0 for measurement of Voltage sensor0 value of stack 0
 */
void ADC_Select_CH0(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};//Structure definition of ADC channel for regular group
	sConfig.Channel = ADC_CHANNEL_0;//Specifies the channel to configure into ADC regular group.
	sConfig.Rank = ADC_REGULAR_RANK_1;//Specifies the rank in the regular group sequencer
	sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;//Sampling time value to be set for the selected channel.
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)//checks and Configures the the selected channel to be linked to the regular group.
	{
		Error_Handler();//This function is executed in case of error occurrence.
	}

}

/*
 Selecting ADC channel 1 for measurement of Voltage sensor1 value of stack 1
 */
void ADC_Select_CH1(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_1;
	sConfig.Rank = ADC_REGULAR_RANK_1;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
```

```
            Error_Handler();
        }
}

/*
 Selecting ADC channel 4 for measurement of Voltage sensor2 value of stack 2
 */
void ADC_Select_CH4(void)
{
        ADC_ChannelConfTypeDef sConfig = {0};
        sConfig.Channel = ADC_CHANNEL_4;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
            Error_Handler();
        }
}

/*
 Selecting ADC channel 5 for measurement of Voltage sensor3 value of stack 3
 */
void ADC_Select_CH5(void)
{
        ADC_ChannelConfTypeDef sConfig = {0};
        sConfig.Channel = ADC_CHANNEL_5;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
            Error_Handler();
        }
}

/*
 Selecting ADC channel 6 for measurement of Voltage sensor4 value of stack 4
```

```
 */
void ADC_Select_CH6(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_6;
	sConfig.Rank = ADC_REGULAR_RANK_1;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 7 for measurement of Voltage sensor5 value of stack 5
 */
void ADC_Select_CH7(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_7;
	sConfig.Rank = ADC_REGULAR_RANK_1;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 8 for measurement of Voltage sensor6 value of stack 6
 */
void ADC_Select_CH8(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_8;
	sConfig.Rank = ADC_REGULAR_RANK_1;
```

```
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 9 for measurement of Voltage sensor7 value of stack 7
 */
void ADC_Select_CH9(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_9;
	sConfig.Rank = ADC_REGULAR_RANK_1;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 10 for measurement of Current sensor value
 */
void ADC_Select_CH10(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_10;
	sConfig.Rank = ADC_REGULAR_RANK_1;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}

}
```

```
/*
 Selecting ADC channel 11 for measurement of Temperature sensor value
 */
void ADC_Select_CH11(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_11;
	sConfig.Rank = ADC_REGULAR_RANK_10;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 12 for measurement of Voltage sensor10 value
 */
void ADC_Select_CH12(void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_12;
	sConfig.Rank = ADC_REGULAR_RANK_11;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Selecting ADC channel 13 for measurement of Voltage sensor11 value
 */
void ADC_Select_CH13(void)
{
```

```
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_13;
	sConfig.Rank = ADC_REGULAR_RANK_12;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
		Error_Handler();
	}
}

/*
 Function to assign converted value from ADC to user defined variable
 */
float getVoltage(int channel)
{
	switch(channel)
	{
	case 0:
		ADC_Select_CH0();//Selecting ADC channel 0 for measurement of Voltage sensor0 value
		HAL_ADC_Start(&hadc1);//Enables ADC, starts conversion of regular group.
		HAL_ADC_PollForConversion(&hadc1,10000);//Wait for regular group conversion to be completed.
		ADC_val0=HAL_ADC_GetValue(&hadc1);//Get ADC regular group conversion result and assign to ADC_val0
		V0=(float)ADC_val0*(Vref/4096.0);//to get exact value of voltage measured
		HAL_ADC_Stop(&hadc1);//Stop ADC conversion of regular group, disable ADC peripheral.
		return V0;
		break;
	case 1:
		ADC_Select_CH1();
		HAL_ADC_Start(&hadc1);
```

```
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val1=HAL_ADC_GetValue(&hadc1);
        V1=(float)ADC_val1*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V1;
        break;
case 4:
        ADC_Select_CH4();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val4=HAL_ADC_GetValue(&hadc1);
        V4=(float)ADC_val4*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V4;
        break;
case 5:
        ADC_Select_CH5();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val5=HAL_ADC_GetValue(&hadc1);
        V5=(float)ADC_val5*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V5;
        break;
case 6:
        ADC_Select_CH6();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val6=HAL_ADC_GetValue(&hadc1);
        V6=(float)ADC_val6*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V6;
        break;
case 7:
```

```
        ADC_Select_CH7();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val7=HAL_ADC_GetValue(&hadc1);
        V7=(float)ADC_val7*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V7;
        break;
case 8:
        ADC_Select_CH8();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val8=HAL_ADC_GetValue(&hadc1);
        V8=(float)ADC_val8*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V8;
        break;
case 9:
        ADC_Select_CH9();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        ADC_val9=HAL_ADC_GetValue(&hadc1);
        V9=(float)ADC_val9*(Vref/4095.0);
        HAL_ADC_Stop(&hadc1);
        return V9;
        break;
case 10:
        ADC_Select_CH10();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,10000);
        Temp=HAL_ADC_GetValue(&hadc1);
        T1=(float)Temp*(3.3/4095.0);
        HAL_ADC_Stop(&hadc1);
        return T1;
```

```
                break;
        case 11:
                ADC_Select_CH11();
                HAL_ADC_Start(&hadc1);
                HAL_ADC_PollForConversion(&hadc1,10000);
                Current=HAL_ADC_GetValue(&hadc1);
                C1=(float)Current*(3.3/4095.0);
                HAL_ADC_Stop(&hadc1);
                return C1;
                break;
        default:
                sprintf(msg,"Error %d\r\n",404);
                HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg),
10000);//Sends an amount of data in blocking mode.
                return 0;
                break;
        }
}

/*
 To display voltage values in Realterm tool
 */
void SerialPrint(float print,int i)
{
        sprintf(msg,"Stack%d Voltage-> %0.2f\r\n",i,print);//to display string message
        HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);//Sends
an amount of data in blocking mode
}

/*
 Function to check stack voltages and not to let them go below safe operating points
 */
int Check_Voltage(void)
{
```

```
	voltage_monitoring();
	if(v0<=VL_SAFE||v1<=VL_SAFE||v2<=VL_SAFE||v3<=VL_SAFE||v4<=V
L_SAFE||v5<=VL_SAFE||v6<=VL_SAFE||v7<=VL_SAFE)//if any one of stack
voltage goes below 2.6V
	{
		sprintf(msg,"Oops, Battery Low!!! Please plug in the Charger"
				" %f\r\n");//display low battery!!!
		HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg),
10000);//Sends an amount of data in blocking mode.
		return 0;
	}
	else
		return 1;

}

/*
 Charging function
 */
void Charge(void)
{
	sprintf(msg,"Charging function\r\n");
	HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
	check=Check_Voltage();//calling voltage monitoring function to get voltage
values
	while((CVAL==HIGH)&&(DVAL==HIGH)&&(v0<VU_SAFE)&&(v1<VU
_SAFE)&&(v2<VU_SAFE)&&(v3<VU_SAFE)&&(v4<VU_SAFE)&&(v5<VU_SA
FE)&&(v6<VU_SAFE)&&(v7<VU_SAFE))//if all stacks are charging below 4V
	{

	sprintf(msg,"****************************************\r\n\r\n");
			HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg),
10000);
```

```
		HAL_GPIO_WritePin(GPIOC, C_FET, LOW);//continue to turn on the gate of charging MOSFET
		HAL_Delay(100);
		check=Check_Voltage();
	}
	if((v0>=VU_SAFE)||(v1>=VU_SAFE)||(v2>=VU_SAFE)||(v3>=VU_SAFE)||(v4>=VU_SAFE)||(v5>=VU_SAFE)||(v6>=VU_SAFE)||(v7>=VU_SAFE))
	{
		sprintf(msg,"\r\n CELL BALANCING  \r\n");
			HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
		reset_FET();
		Cell_Balancing();
	}
	if(CVAL==LOW){
		HAL_GPIO_WritePin(GPIOC, C_FET, HIGH);//turn off the gate of charging MOSFET
		reset_FET();
	}
}

/*
 Discharging function
 */
void Discharge(void)
{
	sprintf(msg,"Discharging function\r\n");
	HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
	check=Check_Voltage();
		if(check==0){
			reset_FET();
			exit (0);
		}
```

```
	while(CVAL==LOW&&DVAL==HIGH&&v0>VL_SAFE&&v1>VL_SAFE
&&v2>VL_SAFE&&v3>VL_SAFE&&v4>VL_SAFE&&v5>VL_SAFE&&v6>VL_
SAFE&&v7>VL_SAFE){

		sprintf(msg,"_____\r\n\r\n");
				HAL_UART_Transmit(&huart2, (uint8_t *)msg,
strlen(msg), 10000);
			HAL_GPIO_WritePin(GPIOC, D_FET, LOW);//continue to
turn on the gate of dis-charging MOSFET
			HAL_Delay(100);
			check=Check_Voltage();
				if(check==0){
					reset_FET();
					exit(0);
				}
		}reset_FET();
}

/*
 function to stop discharging
 */
void NO_Discharge(void)
{
	HAL_GPIO_WritePin(GPIOC, D_FET, LOW);//turn off the gate of dis-
charging MOSFET
	sprintf(msg,"Low Battery!!! Volt = %f\r\n",V_total);////display low battery
	HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);//Sends
an amount of data in blocking mode.
	HAL_Delay(100);
}

/*
 * To display individual stack voltages
```

```
 */
void display(float v,int i){
	sprintf(msg,"Stack_V%d= %f\r\n",i,v);////display low battery
	HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);//Sends an amount of data in blocking mode.
	HAL_Delay(100);
}
/*
 voltage monitoring function
 */
void voltage_monitoring(void)
{
	Battery_voltage();//function call to get ADC converted voltage values
		v0=Voltage0-Voltage1;
		v1=Voltage1-Voltage4;
		v2=Voltage4-Voltage5;
		v3=Voltage5-Voltage6;
		v4=Voltage6-Voltage7;
		v5=Voltage7-Voltage8;
		v6=Voltage8-Voltage9;
		v7=Voltage9;
		V_total=Voltage0;;
		float volt_disp[] ={v0,v1,v2,v3,v4,v5,v6,v7};
		CVAL = HAL_GPIO_ReadPin(GPIOC, C_PIN);//Charger is connected or not(0 or 1)
		DVAL = HAL_GPIO_ReadPin(GPIOD, D_PIN);//Battery is connected or not(0 or 1)
		for(int i=0;i<=7;i++)
			display(volt_disp[i],i);
		sprintf(msg,"Total_V= %f\r\n",V_total);////display low battery
		HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
}

/*
```

```
 Function to assign ADC converted values to user defined variables
 */
void Battery_voltage(void)
{
        Voltage0=getVoltage(0)*5*2.25;//voltage of stack0,1,2 and 3
        Voltage1=getVoltage(1)*5*2.355;//voltage of stack1,2 and 3
        Voltage4=getVoltage(4)*5*1.183;//voltage of stack2 and 3
        Voltage5=getVoltage(5)*5*2.155;//voltage of stack3
        Voltage6=getVoltage(6)*5;//voltage of stack4,5,6 and 7
        Voltage7=getVoltage(7)*5;//voltage of stack5,6 and 7
        Voltage8=getVoltage(8)*5;//voltage of stack6 and 7
        Voltage9=getVoltage(9)*5;//voltage of stack7
}
/*
 Function to sort the volatges
 */
/*void Voltage_sorting(void)
{
float temp;
int t;
voltage_monitoring();
volt[] ={v0,v1,v2,v3,v4,v5,v6,v7};
for(int i=0;i<8;i++)
        for(int j=i+1;j<8;j++)
                if(volt[i] < volt[j])
                {
                        temp = volt[i];
                        volt[i] = volt[j];
                        volt[j] = temp;
                        t=v[i];
                v[i]=v[j];
                v[j]=t;
                }
}
```

```
*/
/*
 Function to turn on cell balancing(CB) FET's
 */
void CB_FET(int a){
        switch (a){
        case 0:
                        HAL_GPIO_WritePin(GPIOC, VB_FET_0, HIGH);
                        break;
        case 1:
                        HAL_GPIO_WritePin(GPIOA, VB_FET_1, HIGH);
                        break;
        case 2:
                        HAL_GPIO_WritePin(GPIOB, VB_FET_2, HIGH);
                        break;
       case 3:
                        HAL_GPIO_WritePin(GPIOC, VB_FET_3, HIGH);
                        break;
       case 4:
                        HAL_GPIO_WritePin(GPIOB, VB_FET_4, HIGH);
                        break;
       case 5:
                        HAL_GPIO_WritePin(GPIOB, VB_FET_5, HIGH);
                        break;
       case 6:
                        HAL_GPIO_WritePin(GPIOB, VB_FET_6, HIGH);
                        break;
       case 7:
                        HAL_GPIO_WritePin(GPIOB, VB_FET_7, HIGH);
                        break;
        }
}

/*
```

```
 cell balancing function
 */

void Cell_Balancing(void){
	sprintf(msg,"\r\nCell balancing function\r\n");
	HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
	if((v0>v1)&&(v0>v2)&&(v0>v3)&&(v0>v4)&&(v0>v5)&&(v0>v6)&&(v0>
v7))
	{

	while((v0>(v1+CB_MIN_DIFF))||(v0>(v2+CB_MIN_DIFF))||(v0>(v3+CB_
MIN_DIFF))||(v0>(v4+CB_MIN_DIFF))||(v0>(v5+CB_MIN_DIFF))||(v0>(v6+CB_
MIN_DIFF))||(v0>(v7+CB_MIN_DIFF)))
		{
			CB_FET(0);
			voltage_monitoring();
		}
	}
	reset_FET();
	if((v1>v0)&&(v1>v2)&&(v1>v3)&&(v1>v4)&&(v1>v5)&&(v1>v6)&&(v1>
v7))
	{

	while((v1>(v0+CB_MIN_DIFF))||(v1>(v2+CB_MIN_DIFF))||(v1>(v3+CB_
MIN_DIFF))||(v1>(v4+CB_MIN_DIFF))||(v1>(v5+CB_MIN_DIFF))||(v1>(v6+CB_
MIN_DIFF))||(v1>(v7+CB_MIN_DIFF)))
		{
			CB_FET(1);
			voltage_monitoring();
		}
	}
	reset_FET();
	if((v2>v0)&&(v2>v1)&&(v2>v3)&&(v2>v4)&&(v2>v5)&&(v2>v6)&&(v2>
v7))
```

```
        {

        while((v2>(v0+CB_MIN_DIFF))||(v2>(v1+CB_MIN_DIFF))||(v2>(v3+CB_
MIN_DIFF))||(v2>(v4+CB_MIN_DIFF))||(v2>(v5+CB_MIN_DIFF))||(v2>(v6+CB_
MIN_DIFF))||(v2>(v7+CB_MIN_DIFF)))
                {
                        CB_FET(2);
                        voltage_monitoring();
                }
        }
        reset_FET();
        if((v3>v0)&&(v3>v1)&&(v3>v2)&&(v3>v4)&&(v3>v5)&&(v3>v6)&&(v3>
v7))
        {

        while((v3>(v0+CB_MIN_DIFF))||(v3>(v1+CB_MIN_DIFF))||(v3>(v2+CB_
MIN_DIFF))||(v3>(v4+CB_MIN_DIFF))||(v3>(v5+CB_MIN_DIFF))||(v3>(v6+CB_
MIN_DIFF))||(v3>(v7+CB_MIN_DIFF)))
                {
                        CB_FET(3);
                        voltage_monitoring();
                }
        }
        reset_FET();
        if((v4>v0)&&(v4>v1)&&(v4>v2)&&(v4>v3)&&(v4>v5)&&(v4>v6)&&(v4>
v7))
        {

        while((v4>(v0+CB_MIN_DIFF))||(v4>(v1+CB_MIN_DIFF))||(v4>(v2+CB_
MIN_DIFF))||(v4>(v3+CB_MIN_DIFF))||(v4>(v5+CB_MIN_DIFF))||(v4>(v6+CB_
MIN_DIFF))||(v4>(v7+CB_MIN_DIFF)))
                {
                        CB_FET(4);
                        voltage_monitoring();
```

```
            }
        }
        reset_FET();
        if((v5>v0)&&(v5>v1)&&(v5>v2)&&(v5>v3)&&(v5>v4)&&(v5>v6)&&(v5>
v7))
        {

        while((v5>(v0+CB_MIN_DIFF))||(v5>(v1+CB_MIN_DIFF))||(v5>(v2+CB_
MIN_DIFF))||(v5>(v3+CB_MIN_DIFF))||(v5>(v4+CB_MIN_DIFF))||(v5>(v6+CB_
MIN_DIFF))||(v5>(v7+CB_MIN_DIFF)))
            {
                CB_FET(5);
                voltage_monitoring();
            }
        }
        reset_FET();
        if((v6>v0)&&(v6>v1)&&(v6>v2)&&(v6>v3)&&(v6>v4)&&(v6>v5)&&(v6>
v7))
        {

        while((v6>(v0+CB_MIN_DIFF))||(v6>(v1+CB_MIN_DIFF))||(v6>(v2+CB_
MIN_DIFF))||(v6>(v3+CB_MIN_DIFF))||(v6>(v4+CB_MIN_DIFF))||(v6>(v5+CB_
MIN_DIFF))||(v6>(v7+CB_MIN_DIFF)))
            {
                CB_FET(6);
                voltage_monitoring();
            }
        }
        reset_FET();
        if((v7>v0)&&(v7>v1)&&(v7>v2)&&(v7>v3)&&(v7>v4)&&(v7>v5)&&(v7>
v6))
        {

        while((v7>(v0+CB_MIN_DIFF))||(v7>(v1+CB_MIN_DIFF))||(v7>(v2+CB_
```

```
MIN_DIFF))||(v7>(v3+CB_MIN_DIFF))||(v7>(v4+CB_MIN_DIFF))||(v7>(v5+CB_
MIN_DIFF))||(v7>(v6+CB_MIN_DIFF)))
                {
                        CB_FET(7);
                        voltage_monitoring();
                }
        }
        reset_FET();

}
/*
 Function to reset all the FET's
 */
void reset_FET(void){
        HAL_GPIO_WritePin(GPIOC, VB_FET_0, LOW);
        HAL_GPIO_WritePin(GPIOA, VB_FET_1, LOW);
        HAL_GPIO_WritePin(GPIOB, VB_FET_2, LOW);
        HAL_GPIO_WritePin(GPIOC, VB_FET_3, LOW);
        HAL_GPIO_WritePin(GPIOB, VB_FET_4, LOW);
        HAL_GPIO_WritePin(GPIOB, VB_FET_5, LOW);
        HAL_GPIO_WritePin(GPIOB, VB_FET_6, LOW);
        HAL_GPIO_WritePin(GPIOB, VB_FET_7, LOW);
        HAL_GPIO_WritePin(GPIOC, C_FET, HIGH);
        HAL_GPIO_WritePin(GPIOC, D_FET, HIGH);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
```

```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration--------------------------------------------------------*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_ADC1_Init();
MX_USART1_UART_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
sprintf(msg,"Hello user, have a safe operarion with the BMS\r\n");
        HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg), 10000);
        HAL_Delay(500);
```

```
 reset_FET();
 int i=1;
        while (i)
        {
                //i--;
   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
                /*Read=HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_2);
if(Read==1)
{
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, SET);
        HAL_Delay(1000);
}
else
{
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, RESET);
}*/

                //voltage_monitoring();
                check=Check_Voltage();
                if(check==LOW){
                        reset_FET();
                        exit(0);
                }
                sprintf(msg,"Hi Siddharth and Vishwas\r\n");
                HAL_UART_Transmit(&huart3, (uint8_t *)msg, strlen(msg), 10000);
                reset_FET();
                CVAL = HAL_GPIO_ReadPin(GPIOC, C_PIN);//Charger is connected
or not(0 or 1)
                DVAL = HAL_GPIO_ReadPin(GPIOD, D_PIN);//Battery is connected
or not(0 or 1)
                while(CVAL==LOW&&DVAL==HIGH){
                        sprintf(msg,"Entered discharging while in main\r\n");
```

```
                    HAL_UART_Transmit(&huart2, (uint8_t *)msg,
strlen(msg), 10000);

                    check=Check_Voltage();
                    if(check==LOW){
                        reset_FET();
                        exit(0);
                    }
                    Discharge();
                }
                reset_FET();
                check=Check_Voltage();

        while(CVAL==HIGH&&DVAL==HIGH&&v0<VU_SAFE&&v1<VU_SAF
E&&v2<VU_SAFE&&v3<VU_SAFE&&v4<VU_SAFE&&v5<VU_SAFE&&v6<V
U_SAFE&&v7<VU_SAFE){
                    Charge();
                    check=Check_Voltage();

        if(((v0>=(VU_SAFE))&&(v1>=(VU_SAFE))&&(v2>=(VU_SAFE))&&(v3>
=(VU_SAFE))&&(v4>=(VU_SAFE))&&(v5>=(VU_SAFE))&&(v6>=(VU_SAFE))
&&(v7>=(VU_SAFE))))
                    {
                    reset_FET();
                        HAL_GPIO_WritePin(GPIOC, C_FET,
LOW);//turn off the gate of charging MOSFET
                        sprintf(msg,"Battery Full!!! Volt =
%f\r\n",V_total);//display battery full
                        HAL_UART_Transmit(&huart2, (uint8_t *)msg,
strlen(msg), 10000);//Sends an amount of data in blocking mode.
                        break;
                    }
                }
```

```
        if((CVAL==HIGH&&DVAL==HIGH&&((VL_RANGE<v0)&&(v0<VU_R
ANGE))&&((VL_RANGE<v1)&&(v1<VU_RANGE))&&((VL_RANGE<v2)&&(v
2<VU_RANGE))&&((VL_RANGE<v3)&&(v3<VU_RANGE))&&((VL_RANGE<v
4)&&(v4<VU_RANGE))&&((VL_RANGE<v5)&&(v5<VU_RANGE))&&((VL_R
ANGE<v6)&&(v6<VU_RANGE))&&((VL_RANGE<v7)&&(v7<VU_RANGE)))){
                        reset_FET();
                        HAL_GPIO_WritePin(GPIOC, C_FET, HIGH);//turn off the
gate of charging MOSFET
                        sprintf(msg,"Battery Full!!! Please remove the
charger\r\n");//display battery full
                        HAL_UART_Transmit(&huart2, (uint8_t *)msg, strlen(msg),
10000);//Sends an amount of data in blocking mode.
                        while(CVAL==HIGH){
                                CVAL = HAL_GPIO_ReadPin(GPIOC, C_PIN);
                        }
                }
        }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
```

```
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType                                      =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if    (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,    FLASH_LATENCY_0)    !=
HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
```

```
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  /*sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
```

```
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
  */
 /* sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = ADC_REGULAR_RANK_2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }*/

  /** Configure Regular Channel
  */
 /* sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
  */
 /* sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_REGULAR_RANK_4;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
```

```
  */
 /* sConfig.Channel = ADC_CHANNEL_6;
  sConfig.Rank = ADC_REGULAR_RANK_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
  */
 /* sConfig.Channel = ADC_CHANNEL_7;
  sConfig.Rank = ADC_REGULAR_RANK_6;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
  */
 /* sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = ADC_REGULAR_RANK_7;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
*/
  /** Configure Regular Channel
  */
/*  sConfig.Channel = ADC_CHANNEL_9;
  sConfig.Rank = ADC_REGULAR_RANK_8;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }*/
```

```
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */
```

```
  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */
```

```
  /* USER CODE END USART2_Init 2 */

}
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 9600;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
```

```
 */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14
                    |GPIO_PIN_15|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                    |GPIO_PIN_9, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9
                    |GPIO_PIN_12|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC13 PC10 PC11 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_10|GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PB11 PB12 PB3 PB4
                         PB5 PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_3|GPIO_PIN_4
```

```
                    |GPIO_PIN_5|GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : PB13 PB14 PB15 */
GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/Configure GPIO pins : PC6 PC9 PC14 PC15/
GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_9|GPIO_PIN_14|GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : PC7 PC8 */
GPIO_InitStruct.Pin = GPIO_PIN_7|GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : PA13 PA14 */
GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  /*Configure GPIO pin : PD2 */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pin : PC12 */
    GPIO_InitStruct.Pin = GPIO_PIN_12;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
          /* User can add his own implementation to report the HAL error return state */
          __disable_irq();
          while (1)
          {
          }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
```

```
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
          /* User can add his own implementation to report the file name and line number,
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Further enhancement:

1. Changing the current sequential code into event driven code using interrupts.
2. State of charge (SOC) implementation.
3. State of health (SOH) implementation.
4. Power modes.
5. Geofencing.
6. Geotargeting.
7. Regenerative braking.

Bill of material:

Materials purchased	Quantity	Price/unit	Total price
Cells, 3.2V, 6A	24	265	7505
MOSFET(IRF540)	10	45	550
STM32F103RBT6 board	1	2950	2950
Voltage Sensors	8	36	350
Wattage resistors (10 W)	20	10	439
Current sensors (ACS712)	2	107	260
Temperature sensors (NTC)	2	20	20
10K ohm, 10W resistors	4	27	130
Wires	2	82	199
Soldering Iron	1	246	299
Soldering flux + solder flux	1	82	99
Voltage regulators (IC7805)	5	17	100
Node MCU	1	340	419
			100
		Total bill	13420

Abbreviation used:

C_ Relay: (Charging Relay).

D_ Relay: (Discharging Relay).

V1, V2...V8: Individual Voltages of the cells.

Vs1, Vs2.....Vs8: Voltages read by individual voltage sensors.

Dp: Discharging pin connected at source of C- Relay (refer Detailed Design).

Cp: Charging pin connected at source of D- Relay (refer Detailed Design).

G1, G2.....G8: Output pins connected to the gates of MOSFET's for cell charge. Dissipation in the cell balancing circuit (refer Detailed Design).