**7. Firmware Code of Bike IOT Circuit.**

**1. Bike IOT code file :**

#include "batterystructs.h"

#include <ESP8266WiFi.h>

#include <ArduinoJson.h>

#include <EEPROM.h>

#include <mcp\_can.h>

#include <SoftwareSerial.h>

#include <ESP8266WebServer.h>

appdatastruct s;      //Global variable that contains all App's data

uint8\_t DEBUGLVL;

bool motorUartEnable, canEnable, appEnable;

bool canInitialized;

void setup() {

  Serial.begin(9600);

  delay(1000);

  setup\_CAN();

  setupUart();

  setupAP();

  setupEEPROM();

  loadEEPROMData();

  checkSwap();

  beginTrip();

  loadConfigs();

  Serial.print("DEBUG Level = ");

  Serial.println(DEBUGLVL);

  if (DEBUGLVL > 0) {Serial.println(WiFi.macAddress());}

  if ((Serial.available()) && (Serial.read() == 's'||'S')) {   //Enter Settings mode

    debugging();

  }

}

void loop() {

  handlemyClient();

  //CAN

 if(canEnable && canInitialized){

  bmsbasic t = get\_Basic\_BMS\_Data();

  //bmsData1 q =  get\_BMS\_Data1();

  if ((t.SOC > (s.current\_soc + 2)) || (t.SOC < (s.current\_soc - 2))) {

    if (DEBUGLVL > 1) Serial.print("SOC Updated:"); Serial.println(t.SOC);

    s.current\_soc = t.SOC;

    writeIntoEEPROM(CURRENT\_SOC, uint32\_t(s.current\_soc \* 10));

  }

  // delay(1000);

  //  uint8\_t \*bat\_Id = get\_Bat\_id();//Get Battery id: Untested

 }

  //UART Motor controller

  if(motorUartEnable){

  bikeDataStruct ud = getUartData();

  bmsData1 q =  get\_BMS\_Data1();

  if (ud.readSuccess) {

    // ud.bikeSpeed = 10;

    if (ud.bikeSpeed > 0) {

      updateAtSpeedFetch(ud.bikeSpeed);

    }

      printAppData(ud.bikeSpeed,q.max\_temp);

  }

  }

  //App data

//  if(appEnable){

  //delay(1000);

//  printAppData(ud.bikeSpeed);

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
2. App data File:**void setupEEPROM()

{

  EEPROM.begin(512);

  delay(1000);

}

void writeIntoEEPROM(int address, uint32\_t number)

{

  EEPROM.write(address, (number >> 24) & 0xFF);

  EEPROM.write(address + 1, (number >> 16) & 0xFF);

  EEPROM.write(address + 2, (number >> 8) & 0xFF);

  EEPROM.write(address + 3, number & 0xFF);

  EEPROM.commit();

}

uint32\_t readFromEEPROM(int address)

{

  return (EEPROM.read(address) << 24) + (EEPROM.read(address + 1) << 16) + (EEPROM.read(address + 2) << 8) + EEPROM.read(address + 3);

}

void resetEEPROMData() {

  EEPROM.write(DEBUG\_ADDRESS, 1);

  EEPROM.write(MOTOR\_UART\_ENABLE, 1);

  EEPROM.write(CAN\_ENABLE, 1);

  EEPROM.write(APP\_ENABLE, 1);

  EEPROM.commit();

  writeIntoEEPROM(NUM\_TRIPS, 0);

  writeIntoEEPROM(EEPROM\_DISTANCE, 0);

  writeIntoEEPROM(AVG\_SPEED\_THIS\_T, 0);

 // writeIntoEEPROM(NEW\_SPEED\_T, 0);        //

  writeIntoEEPROM(CURRENT\_SOC, 0);

   writeIntoEEPROM(VOLTAGE, 0);

   writeIntoEEPROM(MAX\_TEMP, 0);

  writeIntoEEPROM(THIS\_ST\_START\_SOC, 0);

  writeIntoEEPROM(LAST\_ST\_START\_SOC, 0);

  writeIntoEEPROM(LAST\_ST\_FINAL\_SOC, 0);

  writeIntoEEPROM(DIST\_START\_ST, 0);

 // writeIntoEEPROM(KMS\_THIS\_ST, 0);

  writeIntoEEPROM(KMS\_LAST\_ST, 0);

  writeIntoEEPROM(NUM\_SWAP\_TRIPS, 0);

  delay(1000);

}

void loadEEPROMData() {

  s.num\_trips = readFromEEPROM(NUM\_TRIPS);

  s.eeprom\_distance = readFromEEPROM(EEPROM\_DISTANCE);

  s.distance = s.eeprom\_distance;

  s.avg\_speed\_this\_t = readFromEEPROM(AVG\_SPEED\_THIS\_T);

  //s.new\_speed\_t = readFromEEPROM(NEW\_SPEED\_T);

  s.Voltage = readFromEEPROM(VOLTAGE);

  s.max\_temp = readFromEEPROM(MAX\_TEMP);

  //s.avg\_speed = readFromEEPROM(AVG\_SPEED);

  s.num\_swap\_trips = readFromEEPROM(NUM\_SWAP\_TRIPS);

  s.current\_soc = readFromEEPROM(CURRENT\_SOC) / 10;

  s.this\_st\_start\_soc = readFromEEPROM(THIS\_ST\_START\_SOC);

  s.last\_st\_start\_soc = readFromEEPROM(LAST\_ST\_START\_SOC);

  s.last\_st\_final\_soc = readFromEEPROM(LAST\_ST\_FINAL\_SOC);

  s.dist\_start\_st = readFromEEPROM(DIST\_START\_ST);

  s.kms\_this\_st = s.eeprom\_distance - s.dist\_start\_st;

  s.kms\_last\_st = readFromEEPROM(KMS\_LAST\_ST);

  delay(1000);

}

void beginTrip() {

  s.num\_trips += 1;

  writeIntoEEPROM(NUM\_TRIPS, s.num\_trips);

  s.num\_speed\_in\_trip = 0;

  s.avg\_speed\_this\_t = 0;

}

unsigned long int distUpdateMillis = 0;

void updateAtSpeedFetch(int bikeSpeed) {

  if (millis() > distUpdateMillis + 1000) {

   // s.avg\_speed\_this\_t = (s.avg\_speed\_this\_t\*s.num\_speed\_in\_trip + bikeSpeed) / (s.num\_speed\_in\_trip + 1);      //avg speed

     s.avg\_speed\_this\_t = (s.avg\_speed\_this\_t\*s.num\_speed\_in\_trip + bikeSpeed) / (s.num\_speed\_in\_trip + 1);

    s.num\_speed\_in\_trip += 1;

    //s.new\_speed\_t =bikeSpeed;

    s.distance += uint32\_t(bikeSpeed \* ((millis() - distUpdateMillis) / (1000 \* 60 \* 60)));

    distUpdateMillis = millis();

    if (s.distance >= s.eeprom\_distance + 1) {

      s.eeprom\_distance = s.distance;

      s.kms\_this\_st = s.eeprom\_distance - s.dist\_start\_st;

      writeIntoEEPROM(EEPROM\_DISTANCE, s.eeprom\_distance);

    }

  }

}

void checkSwap() {

  bmsbasic t = get\_Basic\_BMS\_Data();

 // bmsData1 q =  get\_BMS\_Data1();

  if (t.SOC > s.current\_soc + 10) {   //Swap done

    if (DEBUGLVL > 0) Serial.println("Swap Detected");

    s.num\_swap\_trips = s.num\_swap\_trips + 1;

    writeIntoEEPROM(NUM\_SWAP\_TRIPS, s.num\_swap\_trips);

    s.last\_st\_start\_soc = s.this\_st\_start\_soc;

    s.last\_st\_final\_soc = s.current\_soc;

    s.Voltage = t.Voltage;

   // s.max\_temp = q.max\_temp;

    writeIntoEEPROM(LAST\_ST\_START\_SOC, s.last\_st\_start\_soc);

    writeIntoEEPROM(LAST\_ST\_FINAL\_SOC, s.last\_st\_final\_soc);

    writeIntoEEPROM(VOLTAGE, s.Voltage);

  writeIntoEEPROM(MAX\_TEMP, s.max\_temp);

    s.this\_st\_start\_soc = t.SOC;

    writeIntoEEPROM(THIS\_ST\_START\_SOC, s.this\_st\_start\_soc);

    s.current\_soc = t.SOC;

    writeIntoEEPROM(CURRENT\_SOC, s.current\_soc);

    s.kms\_last\_st = s.kms\_this\_st;

    writeIntoEEPROM(KMS\_LAST\_ST, s.kms\_last\_st);

    s.kms\_this\_st = 0;

    s.dist\_start\_st = s.eeprom\_distance;

    writeIntoEEPROM(DIST\_START\_ST, s.dist\_start\_st);

    //loadEEPROMData();

  }

}

void printAppData(int bikeSpeed,float max\_temp){

  if (DEBUGLVL == 2) {

    Serial.println();

    Serial.print("App Data:\n  No. of Trips= ");

    Serial.println(s.num\_trips);

    Serial.print("  No. of Speeds in Trip= ");

    Serial.println(s.num\_speed\_in\_trip);

    Serial.print("  EEPROM Distance= ");

    Serial.println(s.eeprom\_distance);

    Serial.print("  Distance= ");

    Serial.println(s.distance);

    Serial.print("  Avg. speed of current trip= ");

    Serial.println(s.avg\_speed\_this\_t);

 ///Serial.print("  Avg. speed = ");

//    Serial.println(s.avg\_speed);

   // Serial.print("Speed");

    //Serial.println(s.new\_speed\_t);

    Serial.print("  Current SOC = ");

    Serial.print(s.current\_soc);

    Serial.println();

  }

  if (DEBUGLVL == 1) {

    Serial.print("App Data: trip=");

    Serial.print(s.num\_trips);

    Serial.print(", num\_speed=");

    Serial.print(s.num\_speed\_in\_trip);

    Serial.print(", eeprom\_distance=");

    Serial.print(s.eeprom\_distance);

    Serial.print(", Distance=");

    Serial.print(s.distance);

    Serial.print(", avg\_v\_this\_t=");

    Serial.print(s.avg\_speed\_this\_t);

//    Serial.print(", avg\_v=");

//    Serial.print(s.avg\_speed);

    Serial.print(", current\_soc=");

    Serial.print(s.current\_soc);

    Serial.print(", 1st\_soc=");

    Serial.print(s.this\_st\_start\_soc);

    Serial.print(", 2st\_soc=");

    Serial.print(s.last\_st\_start\_soc);

    Serial.print(", 2end\_soc=");

    Serial.print(s.last\_st\_final\_soc);

    Serial.print(", dist\_start=");

    Serial.print(s.dist\_start\_st);

    Serial.print(", kms1=");

    Serial.print(s.kms\_this\_st);

    Serial.print(", kms2\_st=");

    Serial.println(s.kms\_last\_st);

  }

 if (DEBUGLVL == 3) {

 // bmsbasic mybms =get\_Basic\_BMS\_Data();

 // bmsData1 q =  get\_BMS\_Data1();

 // bikeDataStruct bikeSt =getUartData();

  Serial.print("App Data: trip=");

  Serial.print(s.num\_trips);

  Serial.print(", voltage=");

  Serial.print(s.Voltage);

  Serial.print(", current\_soc=");

  Serial.print(s.current\_soc);

  Serial.print(", 1st\_soc=");

  Serial.print(s.this\_st\_start\_soc);

  Serial.print(", 2st\_soc=");

  Serial.print(s.last\_st\_start\_soc);

  Serial.print(", 2end\_soc=");

  Serial.print(s.last\_st\_final\_soc);

  Serial.print(": max\_Temp=");

  //Serial.print(max\_Temp);

  //Serial.print("C at Cell No.=");

  //Serial.print(q.max\_temp\_cell\_no);

  //Serial.print(", min\_Temp=");

  //Serial.print(q.min\_temp);

  //Serial.print("C at Cell No.=");

 // Serial.print(q.min\_temp\_cell\_no);

  //Serial.print(" Gear=");

  //Serial.print(bikeSt.gear);

    Serial.print(" Speed =");

    //Serial.println(s.new\_speed\_t);

    Serial.println(bikeSpeed);

 }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**3. Debugging File:**   
  
  
  
void debugging() {

  char t='z';

  while (1) {

    if (t != '\n') {

      Serial.println("SETTINGS");

      Serial.print("a: Change Debug Level:");

      Serial.println(DEBUGLVL);

      Serial.println("b: Exit");

      Serial.println("c: Erase EEPROM");

      Serial.print("d: Change Motor Uart Enable:");

      Serial.println(motorUartEnable);

      Serial.print("e: Change CAN Enable:");

      Serial.println(canEnable);

      Serial.print("f: Change App Enable:");

      Serial.println(appEnable);

    }

    while (!Serial.available());

    t = Serial.read();

    if (t == 'a'||'A') {

      DEBUGLVL += 1;

      if (DEBUGLVL > MAXDEBUGLVL) DEBUGLVL = 0;

      EEPROM.write(DEBUG\_ADDRESS, DEBUGLVL);

    }

    else if (t == 'b'||'B') {

      EEPROM.commit();

      break;

    }

    else if (t == 'c'||'C') {

      resetEEPROMData();

      Serial.println("EEPROM erased");

      loadEEPROMData();

    }

    else if (t == 'd'||'D') {

      motorUartEnable = !motorUartEnable;

      EEPROM.write(MOTOR\_UART\_ENABLE, motorUartEnable);

    }

    else if (t == 'e'||'E') {

      canEnable = !canEnable;

      EEPROM.write(CAN\_ENABLE, canEnable);

    }

    else if (t == 'f'||'F') {

      appEnable = !appEnable;

      EEPROM.write(APP\_ENABLE, appEnable);

    }

  }

  EEPROM.commit();

}

void loadConfigs(){

    DEBUGLVL = EEPROM.read(DEBUG\_ADDRESS);

  motorUartEnable = (EEPROM.read(MOTOR\_UART\_ENABLE)>0);

  canEnable = (EEPROM.read(CAN\_ENABLE)>0);

  appEnable = (EEPROM.read(appEnable)>0);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
  
**4.UART DATA Read file:**

//SoftwareSerial SerialPort(5, 4); //Define hardware connections     before

SoftwareSerial SerialPort(D1, D2); //Define hardware connections          change

//SoftwareSerial SerialPort(4, 5, false, 256); //Define hardware connections

void setupUart() {

  SerialPort.begin(9600);

  delay(1000);

}

bikeDataStruct getUartData() {

  bikeDataStruct bikeSt;

  //Send request

  byte sendbytes[] = {0xC9, 0x14, 0x2, 0x53, 0x48, 0x4F, 0x57, 0x0, 0x0, 0x0, 0x0, 0x0, 0xAA, 0x0, 0x0, 0x0, 0x25, 0xAA, 0x5, 0xB5, 0x0, 0x41, 0x8, 0xD};

  for (int i = 0; i <= 23; i++) {

    SerialPort.write(sendbytes[i]);

  }

  bikeSt.readSuccess =0;

  //Wait for receiving data

  if (SerialPort.available()) {

    byte mybytes[24];

    int bytecount = SerialPort.readBytes(mybytes, 24);

    if (mybytes[0] == 0xC0 && mybytes[1] == 0x14) {         //Check if data is valid and correct

      bikeSt.readSuccess = 1;

      //Find Gear

      if (mybytes[20] & 0b00000100) bikeSt.gear = -1;

      else if ((mybytes[20] & 0b00000010) && (mybytes[20] & 0b00000001)) bikeSt.gear = 4;

      else if (mybytes[20] & 0b00000010) bikeSt.gear = 3;

      else if (mybytes[20] & 0b00000001) bikeSt.gear = 2;

      else bikeSt.gear = 1;

      //Find Brake status

      if ((mybytes[20] & 0b10000000) || (mybytes[20] & 0b00010000)) bikeSt.brake = 1;

      else bikeSt.brake = 0;

      //Find Speed

      float xs = mybytes[14] \* 256 + mybytes[15];

      int Speed = round((-0.778295) + (0.0442666) \* (xs));

      if (Speed<0 and Speed>-5) Speed=0;

      if (Speed>=0 and Speed<140) bikeSt.bikeSpeed = Speed;

    }

    else {

      if(DEBUGLVL>0){Serial.println("Error");}

      while (SerialPort.available() > 0) {SerialPort.read();}

    }

  if(DEBUGLVL>0){

  Serial.print("Motor Data: Gear=");

  Serial.print(bikeSt.gear);

  Serial.print(",Brake=");

  Serial.print(bikeSt.brake);

  Serial.print(",Speed=");

  Serial.println(bikeSt.bikeSpeed);}

  }

  else{

    if(DEBUGLVL>0){

      Serial.println("Motor data not received");

      }

    }

  return bikeSt;

}

/\*

void setup()

{

  setupUart();

}

void loop()

{

  getUartData();

  delay(100);

}\*/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**wifi Code file:**   
  
  
ESP8266WebServer httpServer(80);

void setupAP(){

  Serial.print("  Starting AP: ");

  WiFi.mode(WIFI\_AP);

  WiFi.softAP(getssid(), "Nayel@123",0, 0);

  IPAddress IP = WiFi.softAPIP();

  Serial.print("  AP: ");

  Serial.println(getssid());

  Serial.println();

  Serial.print("  AP IP address: ");

  Serial.println(IP);

  delay(1000);

  httpServer.on("/read", handleJson);

//    httpServer.on("/exit", handleExit);

    httpServer.begin();

}

void handleExit(){

  httpServer.sendHeader("Cache-Control", "no-cache");

  httpServer.close();

}

String getssid(){ //last 4 bytes of mac based ssid

  uint8\_t mac[6];

  WiFi.macAddress(mac);

  char baseMacChr[18] = {0};//%02X:%02X:%02X:%02X:

  sprintf(baseMacChr, "%02X%02X", mac[4], mac[5]);   //mac[0], mac[1], mac[2], mac[3],

  String ssid = "NAYEL\_" + String(baseMacChr);

 // String ssid = "NAYEL\_1C 11";

  return ssid;

}

void handleJson(){

bikeDataStruct bikeSt =getUartData();

 //updateAtSpeedFetch(bikeSt.bikeSpeed);

//bmsbasic mybms =get\_Basic\_BMS\_Data();

bmsData1 q =  get\_BMS\_Data1();

 StaticJsonDocument<256> doc;

  doc["mac\_address"] = String(WiFi.macAddress());

  doc["millis"] = String(millis());

  doc["trips\_count"] = s.num\_trips;

  doc["Speed"]= bikeSt.bikeSpeed;

  doc["swap\_trips\_count"] = s.num\_swap\_trips;

  doc["voltage"] = s.Voltage;

  doc["max\_temp"] = q.max\_temp;

  doc["current\_SOC"] = s.current\_soc;

  doc["current\_ST\_start\_SOC"] = s.this\_st\_start\_soc;

  doc["last\_ST\_start\_SOC"] = s.last\_st\_start\_soc;

  doc["last\_ST\_final\_SOC"] = s.last\_st\_final\_soc;

  JsonObject errors = doc.createNestedObject("errors");

//  // Add some errors

  errors["error\_1"] = "Invalid password";

  errors["error\_2"] = "MAC address not found";

//

//  // Serialize the JSON object to a string

  String jsonString;

  serializeJson(doc, jsonString);

 //httpServer.sendHeader("Content-Length", String(jsonString.length()));

 httpServer.send(200, F("application/json"), jsonString);

//  httpServer.send(200, F("text/html"), doc);

//========================================================================================================previous jason=============

 /\* StaticJsonDocument<256> doc;

  doc["password"] = 123456;

  doc["mac\_address"] = String(WiFi.macAddress());

   //doc["mac\_address"] = String("34:94:54:8E:1C:11");

  doc["millis"] = String(millis());

  doc["trips\_count"] = s.num\_trips;

  doc["distance"] = s.distance;

  doc["current\_trip\_speed\_avg"] = s.avg\_speed\_this\_t;

  doc["speed\_avg"] = s.avg\_speed\_this\_t;

  //doc["speed\_avg"] =30;

  doc["swap\_trips\_count"] = s.num\_swap\_trips;

  doc["current\_SOC"] = s.current\_soc;

  doc["current\_ST\_start\_SOC"] = s.this\_st\_start\_soc;

  doc["last\_ST\_start\_SOC"] = s.last\_st\_start\_soc;

  doc["last\_ST\_final\_SOC"] = s.last\_st\_final\_soc;

  doc["current\_ST\_kms"] = s.kms\_this\_st;

  doc["last\_ST\_kms"] = s.kms\_last\_st;

//  doc["last\_ST\_range"] = 100;//100 \*s.kms\_last\_st/(s.last\_st\_start\_soc - s.last\_st\_final\_soc) ; >causing reset on 17/11/23

  //doc["range\_avg"] = ;

  JsonObject errors = doc.createNestedObject("errors");

//

//  // Add some errors

  errors["error\_1"] = "Invalid password";

  errors["error\_2"] = "MAC address not found";

//

//  // Serialize the JSON object to a string

  String jsonString;

  serializeJson(doc, jsonString);

 //httpServer.sendHeader("Content-Length", String(jsonString.length()));

 httpServer.send(200, F("application/json"), jsonString);

//  httpServer.send(200, F("text/html"), doc);   \*/

}

void handlemyClient(){

//  if (WiFi.softAPgetStationNum() == 0) {

//    // No clients connected, put ESP32 into low-power mode or idle state

//    // esp\_sleep\_enable\_timer\_wakeup(1000000LL \* 60); // 1 minute sleep

//    // esp\_deep\_sleep\_start();

//    Serial.println("No clients connected, ESP32 is in idle state");

//  }

//  else{

//    httpServer.on("/read", handleJson);

////    httpServer.on("/exit", handleExit);

//    httpServer.begin();

//  }

    httpServer.handleClient();

  }