IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676, p-ISSN: 2320-3331, Volume 16, Issue 3 Ser. I (May – June 2021), PP 35-72
www.iosrjournals.org

Analysis of Free RTOS Vs Bare Metal using OTA over ESP32

Ahmad A A Alsaleh
Switching Department , Higher Institute of Communication and Navigation,
Public Authority of Applied Education and Training , Kuwait

Abstract: *This paper presents Over The Air (OTA) updates using ESP32, with increasing technology and understanding the importance of time and manual efforts that take place to update firmware through physical connections and providing authenticity to all the updates.*
Keywords: ESP32 DevKit, DC Motor, Stepper Motor, Servo Motor, Neo-Pixel LED strip, OLED 0.96 inch, IR Sensor, Arduino IDE, Mosquitto Server.

Date of Submission: 08-05-2021 Date of Acceptance: 23-05-2021

I. Introduction

IoT (Internet of Thing) is a kind of system that allows devices to communicate with each other through the Internet. Many Prior works have focused on firmware upgrades using wireless media to avoid physical connections and saving time. OTA is the best utilization of wireless technology to upgrade an entire firmware with required changes just with few clicks. There are many cloud service and with upcoming wireless technologies and to reduce the cable mess and save time and hardware access dependency IoT is what is the solution. There are many supporting languages and software tool to ease the access of devices. OTA is a most frequently used way to upgrade the firmware. Even authenticity and security are also provided in OTA to avoid missing of data and data encryption.

OTA even allows the user to provide authenticity while establishing a connection with hardware that needs to be upgraded. It provides point to point connection where sent data will reach the provided destination.

II. Circuit Design of Fire Fighting Car Prototype

2.1 Pin Configuration of ESP32

ESP32 is a dual-core MCU, capable of running at 240 MHz. Its further features have 512 KB of internal SRAM, it also comes with integrated 2.4 GHz, 802.11 b/g/n Wi-Fi, and BLE(Bluetooth-Low-Energy) 5.0 connectivity that provides long-range support. It has 32 programmable GPIOs. Supports both serial and wireless communication.

The below figure shows the prototype schematic circuit diagram with all the peripheral used to set up the prototype.

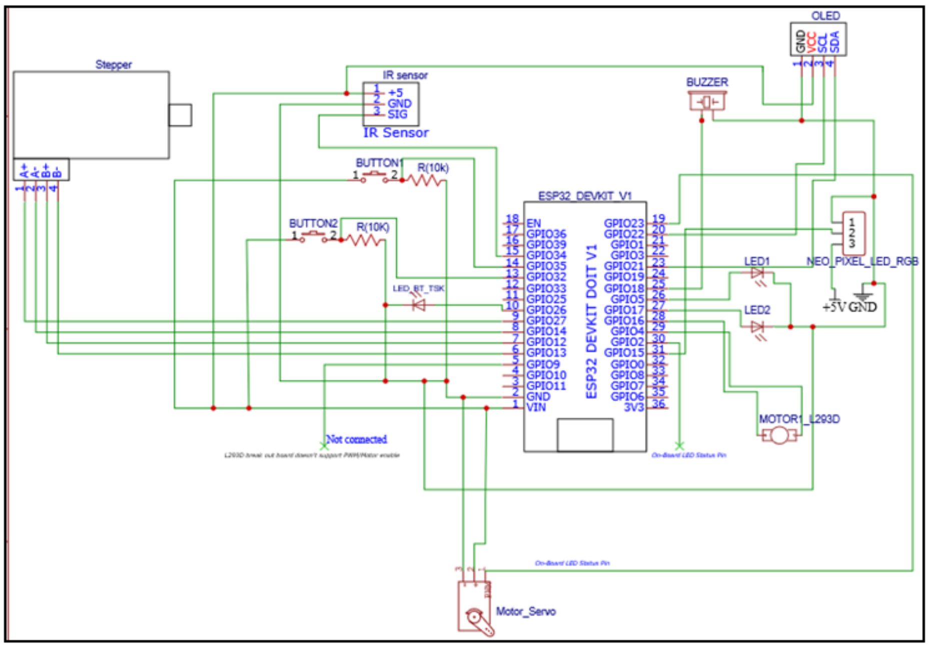

Figure 1. Equivalent Circuit diagram of Prototype

2.2 FreeRTOS Vs Bare Metal

FreeRTOS is a real-time operating system (RTOS) for any micro-controller or small microprocessors. FreeRTOS is designed to be small and simple. The kernel itself consists of only a few C files and supporting libraries. To make it portable and hardware supporting it are written mostly in C, but there are a few assembly functions included mostly in architecture-specific scheduler routines.

FreeRTOS like any other OS provides methods for multiple threads or tasks, mutexes, semaphore, and software timer. For low-power applications, a tick-less mode is provided to utilize the energy. Thread priorities are supported. FreeRTOS applications can be completely statically allocated.

Writing code for Bare metal is a firmware that can is flashed directly on the bare hardware without having any underlying abstraction layer like operating systems have. Bare-metal programs have a minimal boot loader that use to initiate the processor, clock, and memory and jump to the main program to start the board.A bare-metal embedded development approach is additionally referred to as super-loops or background systems.

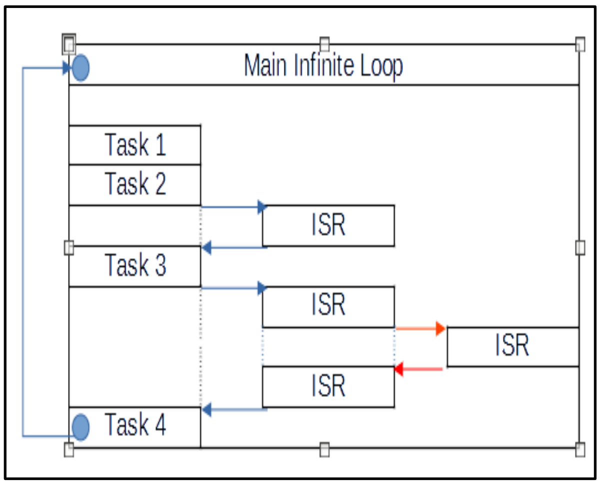

Figure 2. FreeRTOS Firmware
Figure3. Bare Metal Firmware

2.3. OTA

OTA called as Over-The-Air is a wireless service provider to remotely upgrade firmware for wireless service providing devices especially devices with low memory constraints. OTA is useful in the situation where you do not have physical access to the device you want to communicate with.

Like if we have a device kept at home and we are in some remote location and made some changes in Firmware code to get a different functionality from a particular sensor or piece of Hardware peripherals connected to ESP32 which is having a WiFi Module without connecting to USB we can push the code changes through Web Server to ESP32 and implement changes for peripherals functionality.
OTA does support further wireless protocol integration to enhance the performance of device connectivity and upgrading speed.

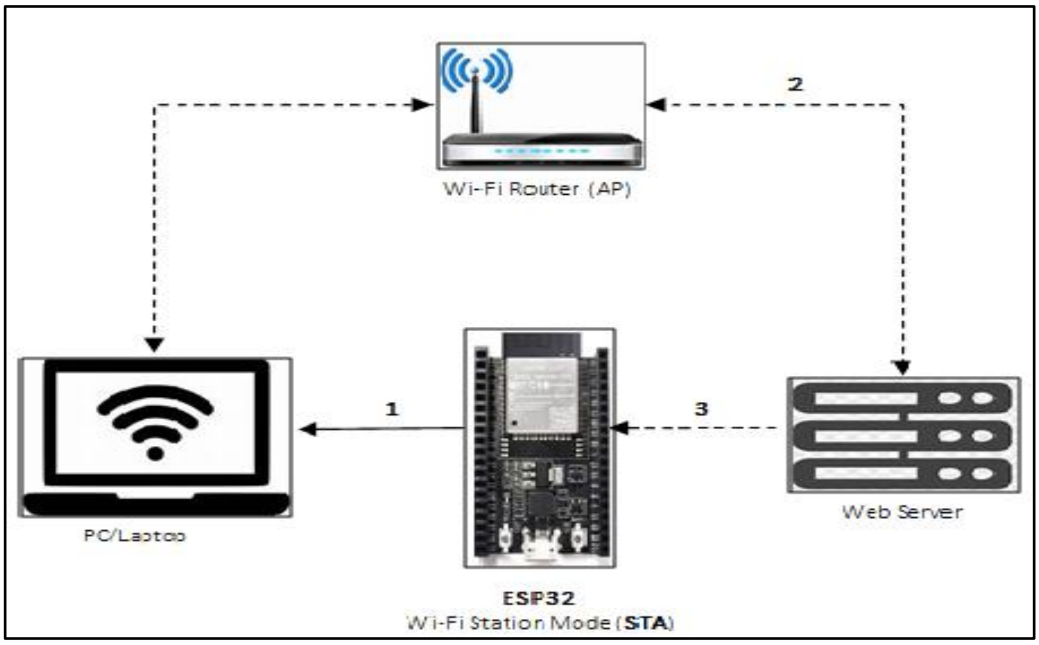

Figure 4. OTA Firmware Upgrade Process

2.4. MQTT

MQTT known as Message Queuing Telemetry Transport is one such most used IoT Protocol. It is a system where we can Publish and Subscribe Message as a client. It is developed in 1999 by Andy Stanford-Clark of IBM and Arlen Nipper of Cirrus Link. MQTT is faster than HTTP because MQTT messages can be as small as 2 Bytes, whereas HTTP includes headers that contain extra information which sometimes not used by that device. MQTT protocol is very feasible for a device with low memory where memory is a constrain and we need to exchange messages remotely for device operation.

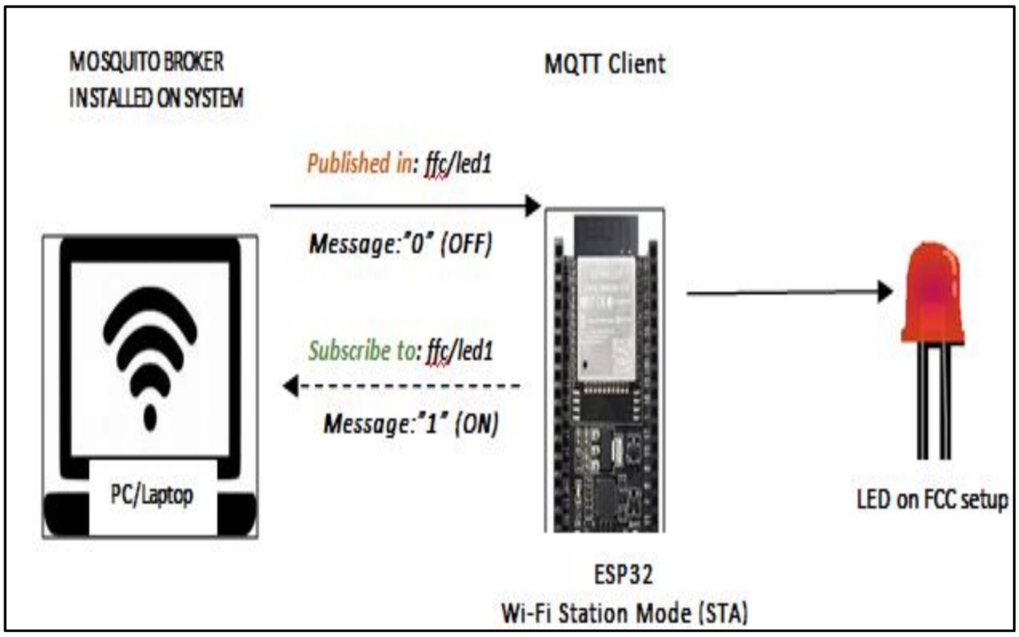

Figure 5. MQTT Publish and Subscribe

III. Analyzing the operation of OTA Upgrade and MQTT Messaging

Figure 2 shows the firmware upgrade through OTA while Figure 3 shows MQTT protocol sending a message to client ESP32. We can say that the above figure shows a huge success in wireless technology. Using the above two methods we can easily beat the thoughts where memory size is a limit. With FreeRTOS OTA update and MQTT messaging are scheduled task and doesn't need to wait for another task to complete, where as when similar tasks are part of Bare Metal application they need to wait for other task to complete their execution. Here we are comparing the behavior of OTA when similar task of OTA Web updater and MQTT are executed over FreeRTOS and Bare-Metal. When the initial sketch with WiFi connection(IP address setup) is uploaded via USB then other Firmware and messages can be sent without any physical access to the device i.e. remotely we can upgrade the device. Below we are discussing steps for connections and upgrades.

3.1. OTA Web Updater

So there is an OTA basic update and the other is OTA Web Updater, In basic OTA update the firmware needs up to be uploaded using Arduino IDE by including an OTA sketch in every code update. Once the Sketch is ready to connect the Board with USB and upload the first sketch to add the IP address after adding an IP address to Arduino ID you can connect ESP32 WiFi and upload other updates via OTA and does not require a USB connection.

Whereas in OTA web update you have a web page to upload the updated changes on the firmware. Here as well the first sketch with SSID(Service Set Identifier) and Password needs to be added to the code to give power of OTA to ESP32. We need to provide these credentials to connect to the router so that we can receive updates. Access the web browser or web-page created by ESP32.

OTA on ESP32 requires some understanding regarding the size available on ESP32, how the memory is divided since upgrading a huge file from the internet needs to settle somewhere in memory to upgrade the entire firmware with required changes on booting the hardware. ESP32 flash may contain multiple applications and different kinds of data(file system, calibration data, parameters storage, etc.) So for that, we have to make changes in the project configuration menu, and since the file system is less difficult to manage configurations and bit mapping. So by keeping flash size lesser and try to provide much memory on the application side as flash is just to keep the file and flash the new firmware.

3.2. OLED 0.96 INCH Message DISPLAY with OTA

Once the connection got established with the WiFi router the device is ready for OTA upgrades. The received connection status is displayed to the OLED 0.96 Inch display connected to ESP32 in the Fire fighting car setup showing authentication messages and IP address. With the OTA update, some display messages are integrated into the OTA upgrade code which is part of the fire fighting car setup.

3.3. MQTT Messaging Protocol

As discussed earlier that MQTT is a publish and subscribe wireless communication protocol so to establish this connection we need a server and a client to request the service for that we installed Mosquitto which is a full functional MQTT broker (server) and we can even diagnose MQTT client issues using it. Mosquitto can be accessed through Terminal on your system using the command *"mosquito"*. Here ESP32 is the MQTT client, publishing and Subscription is done on the same topic for us it "fcc/led1" to display the MQTT connection between client and broker as shown in figure 5. There are some essential commands to publish, subscribe, and to the connection between client and broker. *"-t" is to state the topic (*fcc/led1) and "-m" is to send the message(LED:0(OFF)/LED:1(ON)). Send messages are to On/Off the LED's connected to one to of the PIN on ESP32.

To create authentication while establishing the connection.

```
./mosquitto_passwd -U <passwordfile>
```

Start Mosquitto server using the below command, the mosquitto configuration file(mosquitto.conf) has all the details to initalize mosquitto server

```
./mosquitto.exe -c ./mosquitto.conf
```

Publish to this mosquitto server from windows terminal by below command

```
./mosquitto_pub -t <topic> -h <IP address> -u esp --pw 1234 -p <port number>  -m '<message>'
```

Subscribe on terminal for debugging

```
./mosquitto_sub -t <topic> -h <IP address> -u esp --pw 1234 -p <port number>
```

Below is the data of multiple iteration and the time taken in second to upload a file from Web Browser for the application running on Bare-Metal and application running over FreeRTOS.

Table 1. Bare-Metal File Upload Case

Iteration	**Time Taken(Sec)**
1	67
2	67
3	67
4	67
5	67
6	65
7	67
8	60
9	63
10	70

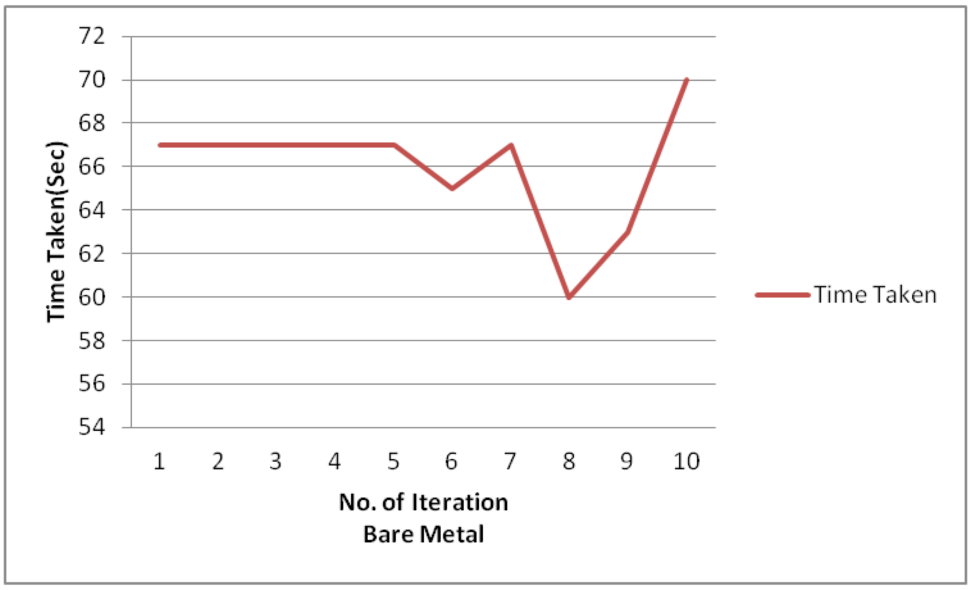

Figure 6. Bare Metal File Upload chart

Table 2. FreeRTOS File Upload Case

Iteration	**Time Taken(Sec)**
1	64
2	66
3	60
4	63
5	64
6	60
7	60
8	60
9	60
10	60

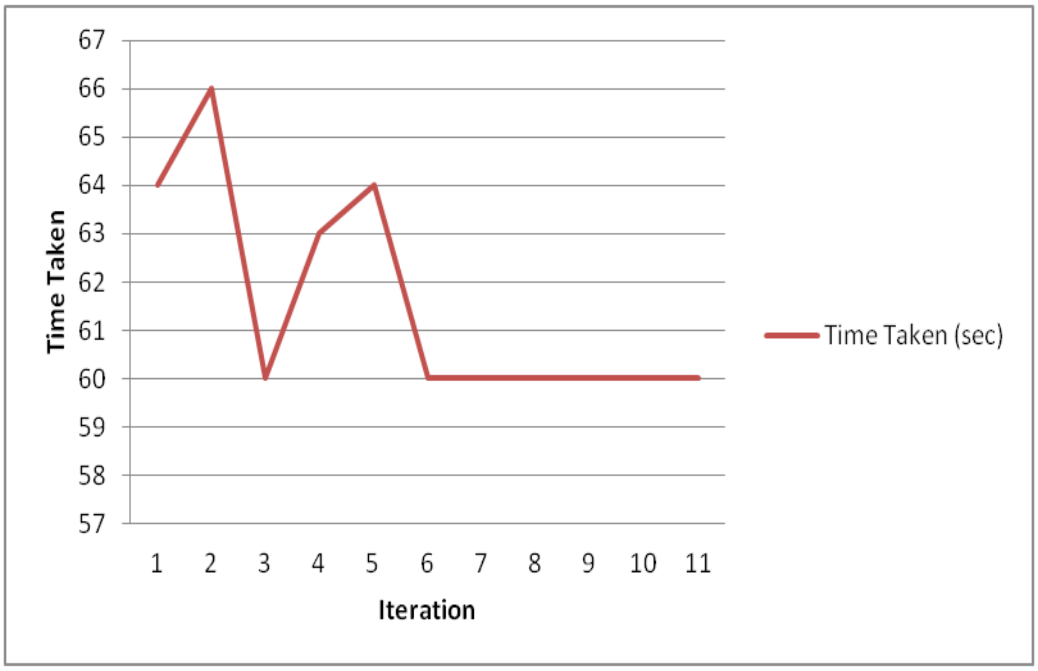

Figure 7. FreeRTOS File Upload Chart

IV. Programming and project analysis

4.1 Embedded System Programming

To achieve this task most of the coding is done using C and C++ using Arduino IDE environment to develop. The purpose of having these task is to get a comparison between to have an application running on OS and a code running over bare metal

FreeRTOS

ESP32_OTA_FreeRtos.c

```
/*****************************************************************************************/
/*AUTHOR : Ahmad Alsaleh*/
/*PROGRAM: FIRE FIGHTING CAR USING FREERTOS                              */
/*VERSION: V01                                                  */
/*****************************************************************************************/

// in this example we simulate a fire fighting car

/*****************************************************************************************/
/*                       INCLUDES                                 */
/*****************************************************************************************/

#include "FFC_PERFConfig.h"
#include "OTASupp.h"

/*****************************************************************************************/
/*                    INTERNAL DEFINES                                    */
/*****************************************************************************************/

/*****************************************************************************************/

/*****************************************************************************************/
/*                          OBJECTS                               */
/*****************************************************************************************/

// create SerialBT object to control Bluetooth connection
```

```
//create serial Bluetooth terminal to send and receive commands
BluetoothSerial SerialBT;
// create servo object to control a servo
// 16 servo objects can be created on most boards
Servo myservo;
// initialize the stepper library on pins :
Stepper myStepper(Steps_Per_Revolution, Stepper_Pin1, Stepper_Pin3, Stepper_Pin2, Stepper_Pin4);

// create the motor object to control a DC MOTOR
Robojax_L298N_DC_motor motor(Motor1_Pin1, Motor1_Pin2, Motor1_Enable_Pin, Motor1_Debug_Logs);
// create a strip object to control a strip of RGBs
Adafruit_NeoPixel strip(Number_Of_Pixels, NEO_STRIP_RGB_PIN, NEO_GRB + NEO_KHZ800);
// create a display object to control the OLED screen
SSD1306Wire  display(OLED_I2C_ADR, OLED_SDA, OLED_SCL);
WebServer server(80);
WiFiClient espClient;
PubSubClient mqttClient(espClient);
bool updateStartedDoNoYeild = false;

/*EspMQTTClient mqttClient(
  ssid,
  password,
  ipAddress,  // MQTT Broker server ip
  "esp",   // Can be omitted if not needed
  "1234",   // Can be omitted if not needed
  "FFC",     // Client name that uniquely identify your device
  1883             // The MQTT port, default to 1883. this line can be omitted
);*/
/******************************************************************************************/
/*                    FUNCTIONS  PROTOTYPE                              */
/******************************************************************************************/

void Init_Task            (void);
void Init_Buttons         (void);
void Init_Bluetooth        (void);
void Init_LED             (void);
void Init_OLED            (void);
void Init_BUZZER          (void);
void Init_RGB             (void);
void Init_Stepper         (void);
void Init_Servo           (void);
void Init_IR_Sensor        (void);
void Init_DC_Motors        (void);
void Init_OTA_Setup        (void);
void RGB_Change_Color      (int Red,int Green, int Blue);
void Bluetooth_Task        (void *parameter);
void Get_Buttons_State_Task (void *parameter);
void Task1                (void *parameter);
void Task2                (void *parameter);
void Task3                (void *parameter);
void Task4                (void *parameter);
void Task5                (void *parameter);
void otaWebUpdate          (void *parameter);
void mqttReceiver          (void *parameter);
void displayString         (String str, float displaytime, int fontSize = 24);
void taskSuspendAll        (void);
void mqttReceiverCb        (char* topic, byte* message, unsigned int length);

/******************************************************************************************/
```

```
/****************************************************************************************/
/*                        TASK HANDLERS                                    */
/****************************************************************************************/

TaskHandle_t   Task_BLUET_Handle;
//TaskHandle_t   Task_Handle_0;
TaskHandle_t   Task_Handle_1;
TaskHandle_t   Task_Handle_2;
TaskHandle_t   Task_Handle_3;
TaskHandle_t   Task_Handle_4;
TaskHandle_t   Task_Handle_WebUpdate;
TaskHandle_t   Task_Handle_Mqtt;

void setup()
{
  Serial.begin(115200);

/****************************************************************************************/
 /*                        CONFIGURATIONS                                    */

/****************************************************************************************/
 //call the initialization function
 StatusIndicator_init();

 Init_Task();
 //tasks creation
 xTaskCreate(Bluetooth_Task       ,"BLUE_TASK",2524,NULL,1,&Task_BLUET_Handle);
 xTaskCreate(Task1                ,"TASK1",1024,NULL,1,&Task_Handle_1);
 xTaskCreate(Task2                ,"TASK2",1536,NULL,1,&Task_Handle_2);
 xTaskCreate(Task3                ,"TASK3",1024,NULL,1,&Task_Handle_3);
 xTaskCreate(Task4                ,"TASK4",1024,NULL,1,&Task_Handle_4);
 xTaskCreate(otaWebUpdate         ,"WebUpdate",3524,NULL,3,&Task_Handle_WebUpdate);
 xTaskCreate(mqttReceiver         ,"mqttReceiver",2524,NULL,1,&Task_Handle_Mqtt);
 Serial.printf("setup done %d",configMINIMAL_STACK_SIZE);

}


void loop()
{
 //DO NOT WRITE ANY CODE HERE
}

void StatusIndicator_init(){
 pinMode(LED1_Status_Pin, OUTPUT);
 //Serial.println("Status Runner");
}
 int ledState = LOW;
void RunIndicator()
{
 if (ledState == LOW) {
     ledState = HIGH;
   } else {
     ledState = LOW;
   }
 digitalWrite(LED1_Status_Pin, ledState);
 //Serial.println("Status Runner");
}
```

```
/***************************************************************************************************/
/*                        FUNCTIONS DECLERATIONS                              */
/***************************************************************************************************/

/***************************************************************************************************/
/*NOTES    : this function initialize all of the pripherals and GPIOs                  */
/*INPUTS   : No INPUTS                                          */
/*RETURNES : NO OUTPUTS                                          */
/***************************************************************************************************/

void IRAM_ATTR Init_Task (void)
{
  //serial monitor
  RunIndicator();
  //this function interferes with the other code part Init_OLED()
  //initializing bluetooth
  Init_Bluetooth();
  RunIndicator();
  //initializing buttons
  Init_Buttons();
  //initialize led
  Init_LED();
  //initialize RGB strip
  Init_RGB();
  //initialize Buzzer
  Init_BUZZER();
  RunIndicator();
  //initialize stepper
  Init_Stepper();
  RunIndicator();
  //initialize servo
  Init_Servo();
  RunIndicator();
  //initialize DC motor;
  Init_DC_Motors();
  RunIndicator();
  //initialize IR sensor
  Init_IR_Sensor();
  RunIndicator();
  //initialize OLED screen
  Init_OLED();
  //Initialize OTA, this will print IP on screen
  Init_OTA_Setup();
  RunIndicator();
}

/***************************************************************************************************/
/*NOTES    : this function initialize the bluetooth                          */
/*INPUTS   : No INPUTS                                          */
/*RETURNES : NO OUTPUTS                                          */
/***************************************************************************************************/

void IRAM_ATTR Init_Bluetooth(void)
{
  //Bluetooth device name, you can give it any name
  SerialBT.begin("ESP32_Blutooth");
}
```

```
/******************************************************************************************/
/*NOTES    : this function initialize Push Buttons pin as input pull up              */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Init_Buttons (void)
{
  //configure push buttons as input pull up
  pinMode(Push_Button1, INPUT);
  pinMode(Push_Button2, INPUT);
}

/******************************************************************************************/
/*NOTES    : this function initialize LED pins as outputs                         */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Init_LED (void)
{
  //configure pins as output
  pinMode(LED1_Pin, OUTPUT);
  pinMode(LED2_Pin, OUTPUT);
  pinMode(BT_TSK_LED, OUTPUT);
}

/******************************************************************************************/
/*NOTES    : this function initialize OLED screen                             */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Init_OLED (void)
{
  // initialize OLED with I2C addr 0x3C or the address provided with your LCD
  display.init();
  display.setFont(ArialMT_Plain_16);
  displayString("Booting Up",1,24);
  vTaskDelay(1000/portTICK_PERIOD_MS);
  displayclear();
  //vTaskDelay(1000/portTICK_PERIOD_MS);
}

/******************************************************************************************/
/*NOTES    : this function initialize the buzzer                            */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Init_BUZZER (void)
{
  // Configure BUZZER functionalities. ledcSetup(ledChannel, freq, resolution);
  ledcSetup(BUZZER_CHANNEL, BUZZER_Frequency, BUZZER_Resoluation);
  // Attach BUZZER pin.   ledcAttachPin(GPIO,Channel)
  ledcAttachPin(BUZZER_Pin, BUZZER_CHANNEL);
}

/******************************************************************************************/
```

```
/*NOTES    : this function initialize te RGB strip                    */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                            */
/*******************************************************************************************/

void IRAM_ATTR Init_RGB (void)
{
  //INITIALIZE NeoPixel strip object (REQUIRED)
  strip.begin();
  RGB_Change_Color(0, 0, 0);
}

/*******************************************************************************************/
/*NOTES    : this function initialize stepper motors                  */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                            */
/*******************************************************************************************/

void IRAM_ATTR Init_Stepper (void)
{
  // set the speed at rpm:
  myStepper.setSpeed(Stepper_Speed);

}

/*******************************************************************************************/
/*NOTES    : this function initialize se servo motor                  */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                            */
/*******************************************************************************************/

void IRAM_ATTR Init_Servo (void)
{
  //initialize servo for esp32
  // Allow allocation of all timers
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  // Standard 50hz servo
  myservo.setPeriodHertz(50);
  // using default min/max of 1000us and 2000us
  // different servos may require different min/max settings
  // for an accurate 0 to 180 sweep
  //attach servo pin to the obect
  myservo.attach(Servo1_Pin, 100, 2000);
}

/*******************************************************************************************/
/*NOTES    : this function initialize the DC motor                    */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                            */
/*******************************************************************************************/

void IRAM_ATTR Init_DC_Motors(void)
{
  // INITIALIZE motor object (REQUIRED)
  motor.begin();
}
```

```
/*****************************************************************************************/
/*NOTES    : this function initialize IR Sensor and set it in receiving mode          */
/*INPUTS   : No INPUTS                                          */
/*RETURNES : NO OUTPUTS                                          */
/*****************************************************************************************/

void IRAM_ATTR Init_IR_Sensor(void)
{
  // Start the receiver
  //IR_Receiver.enableIRIn();
  pinMode(IR_Receive_Pin,INPUT);
}

/*****************************************************************************************/
/*NOTES    : this function initialize OTA related setup          */
/*INPUTS   : No INPUTS                                          */
/*RETURNES : NO OUTPUTS                                          */
/*****************************************************************************************/
bool once_displ =false;
void IRAM_ATTR  Init_OTA_Setup() {
     //setup function
    FEED_WATCH_DOG
    WiFi.begin(ssid, password);
    FEED_WATCH_DOG
    Serial.println("");
    // Wait for connection
    while (WiFi.status() != WL_CONNECTED) {
      FEED_WATCH_DOG
            vTaskDelay(10/portTICK_PERIOD_MS);
      FEED_WATCH_DOG
            Serial.printf(". %d",WiFi.status());
      FEED_WATCH_DOG
    }
    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");

    FEED_WATCH_DOG
      String str = WiFi.localIP().toString();
      Serial.println(str);
    FEED_WATCH_DOG
      displayString(str,5,10);
    FEED_WATCH_DOG

    mqttClient.setServer(ipAddress, 1883);
    mqttClient.setCallback(mqttReceiverCb);
    mqttClient.connect(mqttClientID, mqttUser, mqttpwd);

    /*use mdns for host name resolution*/
    if (!MDNS.begin(host)) { //http://esp32.local
      Serial.println("Error setting up MDNS responder!");
      while (1) {
        vTaskDelay(10/portTICK_PERIOD_MS);
      }
    }
    Serial.println("mDNS responder started");
    /*return index page which is stored in serverIndex*/
  server.on("/", HTTP_GET, []() {
    //suspend once connection established- reboot can only restablish all
      taskSuspendAll();
    FEED_WATCH_DOG
      server.sendHeader("Connection", "close");
    FEED_WATCH_DOG
      server.send(200, "text/html", loginIndex);
      displayString("Connection Established",2,10);
      updateStartedDoNoYeild = true;
    FEED_WATCH_DOG
  });
  server.on("/serverIndex", HTTP_GET, []() {
      server.sendHeader("Connection", "close");
    FEED_WATCH_DOG
      server.send(200, "text/html", serverIndex);
      displayString("Authentication Success\nStarting Update",1,10);
    FEED_WATCH_DOG
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
      server.sendHeader("Connection", "close");
    FEED_WATCH_DOG
      server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    FEED_WATCH_DOG
      ESP.restart();
    FEED_WATCH_DOG
        }, []() {
        FEED_WATCH_DOG
        HTTPUpload& upload = server.upload();
          Serial.printf("UpdateStatus: %lu\n", upload.status);
          if (upload.status == UPLOAD_FILE_START) {
            Serial.printf("Update: %s Max Size %d\n", upload.filename.c_str(),UPDATE_SIZE_UNKNOWN);
            if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
                Update.printError(Serial);
            }
          } else if (upload.status == UPLOAD_FILE_WRITE) {
              if(!once_displ)
                displayString("Writing File",1,10);
              once_displ = true;
              Serial.printf("writing: %lu %lu \n", upload.currentSize, upload.totalSize);
              /* flashing firmware to ESP*/
              if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
                Update.printError(Serial);
              }
          } else if (upload.status == UPLOAD_FILE_END) {
            if (Update.end(true)) { //true to set the size to the current progress
              Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
              displayString("Update Success \n nRebooting",1,10);
            }
          }
          else if (upload.status == UPLOAD_FILE_ABORTED) {
             Serial.printf("UnExpected Error Rebooting");
             ESP.restart();
          }
        FEED_WATCH_DOG
        });
        server.begin();
    FEED_WATCH_DOG
}
```

```
/*****************************************************************************************/
/*NOTES    : this function Sets color to the RGB strip to activat any color set 255 or   */
/* any desired brightness level to deactivate a color put o inits place                */
/*INPUTS   : RED , Green , Blue. respectevily                                   */
/*RETURNES : NO OUTPUTS                                                         */
/*****************************************************************************************/

void IRAM_ATTR RGB_Change_Color (int Red, int Green, int Blue)
{
  RunIndicator();
  // Set all pixel colors to 'off'
  strip.clear();
  for(int i=Number_Of_Pixels; i>=0; i--)
  {
    strip.setPixelColor(i, strip.Color(Red, Green, Blue));
  }
  strip.show();

}

/*****************************************************************************************/
/*NOTES    : task connect to a bluetooth and get data                           */
/*INPUTS   : No INPUTS                                                    */
/*RETURNES : NO OUTPUTS                                                         */
/*****************************************************************************************/

void IRAM_ATTR Bluetooth_Task(void *parameter)
{

  while(1)
  {
    RunIndicator();
    //cheack the blutooth is connected
    if (SerialBT.available())
    {
      Bluetooth_data = SerialBT.read();
      //start sending data to the mobile
      SerialBT.write(Bluetooth_data);
      //update the bluetooth data with the received data
    }
    void *vt = NULL;
    Task5(vt);
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}


/*****************************************************************************************/
/*NOTES    : this task checks if the button is pressed or not if presed it updates its   */
/* related flag. flag1 for button1, flag2 for button2                          */
/*INPUTS   : No INPUTS                                                    */
/*RETURNES : NO OUTPUTS                                                         */
/*****************************************************************************************/

void IRAM_ATTR Get_Buttons_State_Task  (void *parameter){}


/*****************************************************************************************/
```

```
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Task1  (void *parameter)
{
  while(1)
 {
   Serial.println("Task1");
   RunIndicator();
   //turn on LEDs
   digitalWrite(LED1_Pin,HIGH);
   digitalWrite(LED2_Pin,LOW);
   ////set the  buzzer tone to DO. ledcWriteTone(channel,freq)
   ledcWriteTone(BUZZER_CHANNEL, BUZZER_Frequency_Tone1);
   //delay for 0.5 s
   vTaskDelay(500/portTICK_PERIOD_MS);
   //toggle LEDs
   digitalWrite(LED1_Pin,LOW);
   digitalWrite(LED2_Pin,HIGH);
   //change buzzer tone  to MI
   ledcWriteTone(BUZZER_CHANNEL, BUZZER_Frequency_Tone2);
   //delay for 0.5 s
   vTaskDelay(500/portTICK_PERIOD_MS);
  }

}

/******************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Task2  (void *parameter)
{
  while(1){
   Serial.println("Task2");
   bool ButtonPressed = false;
   RunIndicator();
   //check if the button1 is pressed or not
   if(digitalRead(Push_Button2) == 1)
   {
     // to handel debouncing
     //Aman:while(digitalRead(Push_Button2) == 0);
     vTaskDelay(50/portTICK_PERIOD_MS);
     if(digitalRead(Push_Button2) == 1)
     {
       //update flage state to indicate its pressed
       ButtonPressed = true;
     }
   }
   if (ButtonPressed == true )
   {
     //set the angle to the servo
     myservo.write(Servo_High_Angle);
     displayString("Danger",1);
   }
```

```
    else
    {
      // display nothing on the LCD
      displayclear();
      //set the servo in its Low position
      myservo.write(Servo_Low_Angle);
    }
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}

/*******************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                        */
/*******************************************************************************************/

void IRAM_ATTR Task3  (void *parameter)
{
  while(1){
    Serial.println("Task3");
    RunIndicator();
    bool ButtonPressed = false;
    RunIndicator();
    //check if the button1 is pressed or not
    if(digitalRead(Push_Button1) == 1)
    {
      // to handel debouncing
      vTaskDelay(10/portTICK_PERIOD_MS);
      if(digitalRead(Push_Button1) == 1)
      {
        //update flage state to indicate its pressed
        ButtonPressed = true;
      }
    }

    if(ButtonPressed){
      //RGB (Green)
      RGB_Change_Color (0, 255, 0);
      // Stepper motor rotate CW
      myStepper.step(Steps_Per_Revolution);

      //delay for 5 s
      vTaskDelay(5000/portTICK_PERIOD_MS);
      //Before Rotating Stepper change colour to blue
      RGB_Change_Color(0, 0, 255);
      // Rotate Stepper Back
      myStepper.step(-Steps_Per_Revolution);
      }
      else{
        RGB_Change_Color(0, 0, 0);
      }
  }
}
/*******************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                        */
/*******************************************************************************************/
```

```
void IRAM_ATTR Task4  (void *parameter)
{
  while(1){
    Serial.println("Task4");
    RunIndicator();
    //Check if received data is available and if yes, try to decode it.
    //Decoded result is in the IrReceiver.decodedIRData structure.
    int inputVal = digitalRead(IR_Receive_Pin);
      //check the recived data between 0 and 500
    if(!inputVal)
    {
      //DC Motor rotate left
      motor.rotate(motor1, DC_Motor_Speed, CCW );//run motor1 at 10% speed in CCW direction
      vTaskDelay(1000/portTICK_PERIOD_MS);
      motor.brake(motor1);
      vTaskDelay(500/portTICK_PERIOD_MS);
    }
    vTaskDelay(50/portTICK_PERIOD_MS);
  }
}

/*****************************************************************************************************/
/*NOTES    : task controls DC motor based on the received data from bluetooth            */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                  */
/*****************************************************************************************************/

void IRAM_ATTR Task5  (void *parameter)
{
    RunIndicator();
    Serial.println("Task5");
    if((Bluetooth_data == 'A') || (Bluetooth_data == 'a'))
    {
      int N = 2;
      //Blink N times BT TASK LED
      for(int i = 0; i < N; i++){
        digitalWrite(BT_TSK_LED,HIGH);
        vTaskDelay(500/portTICK_PERIOD_MS);
        digitalWrite(BT_TSK_LED,LOW);
        vTaskDelay(500/portTICK_PERIOD_MS);
      }
     }
    else if ((Bluetooth_data == 'B') || (Bluetooth_data == 'b'))
    {
      int N = 5;
      //Blink N times BT TASK LED
      for(int i = 0; i < N; i++){
        digitalWrite(BT_TSK_LED,HIGH);
        vTaskDelay(100/portTICK_PERIOD_MS);
        digitalWrite(BT_TSK_LED,LOW);
        vTaskDelay(100/portTICK_PERIOD_MS);
      }

    }
}

void IRAM_ATTR otaWebUpdate (void *parameter)
{
  while(1){
     bool ButtonPressed = false;
      //for(int i = 0; i < INT_MAX; i++);
              server.handleClient();
      Serial.printf("Update: %s(0_2)\n",__func__);
      if(once_displ)
      vTaskDelay(10/portTICK_PERIOD_MS);
      else
      vTaskDelay(1000/portTICK_PERIOD_MS);

   }//end while 1
}

/******************************************************************************************/
/*NOTES    : Function to print string on display                                   */
/*INPUTS   : Display Strting                                              */
/*RETURNES : NO OUTPUTS                                                    */
/******************************************************************************************/
//Supported Font Size 10,16,24
void setFontSize(int fontSize) {
  switch (fontSize) {
    case 10:
      display.setFont(ArialMT_Plain_10);
      break;
    case 16:
      display.setFont(ArialMT_Plain_16);
      break;
    case 24:
    default:
      display.setFont(ArialMT_Plain_24);
      break;
    }

}

void IRAM_ATTR mqttReceiver (void *parameter)
{
  while(1){
    Serial.println("mqttReceiver__update");
    mqttClient.subscribe("ffc/led1");
    mqttClient.loop();
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}


void mqttReceiverCb(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

    if (String(topic) == "ffc/led1") {
```

```
    digitalWrite(BT_TSK_LED,HIGH);
    vTaskDelay(500/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,LOW);
    vTaskDelay(500/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,HIGH);
    vTaskDelay(500/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,LOW );
    vTaskDelay(500/portTICK_PERIOD_MS);


   if ((messageTemp == "LED:0")) {
     digitalWrite(BT_TSK_LED,LOW);

   }
    else if((messageTemp == "LED:1"))
     digitalWrite(BT_TSK_LED,HIGH);
   }

}

void displayString(String str,float displaytime, int fontSize){
    displayclear();
    setFontSize(fontSize);
    if(str.length() > 0) {
      display.setTextAlignment(TEXT_ALIGN_LEFT);
      display.drawString(0, 10, str);
      // Display it on the screen
      display.display();
      vTaskDelay((((int)displaytime)*1000)/portTICK_PERIOD_MS);
    }
};

void displayclear()
{
  for(int i=0;i<5000000;i++);
  display.init();
  display.display();
 }

void taskSuspendAll() {
  vTaskSuspend( Task_BLUET_Handle);
  vTaskSuspend( Task_Handle_1);
  vTaskSuspend( Task_Handle_2);
  vTaskSuspend( Task_Handle_3);
  vTaskSuspend( Task_Handle_4);
  vTaskSuspend( Task_Handle_Mqtt);
}
void taskResumeAll() {
  vTaskResume( Task_BLUET_Handle);
  vTaskResume( Task_Handle_1);
  vTaskResume( Task_Handle_2);
  vTaskResume( Task_Handle_3);
  vTaskResume( Task_Handle_4);
  vTaskResume( Task_Handle_Mqtt);
}
```

Bare-Metal

ESP32_OTA_BareMetal.c

```
/*****************************************************************************************/
/*AUTHOR : Ahmad Alsaleh*/
/*PROGRAM: FIRE FIGHTING CAR USING FREERTOS                                              */
/*VERSION: V01                                                  */
/*****************************************************************************************/

// in this example we simulating a fire fighting car

/*****************************************************************************************/
/*                      INCLUDES                                 */
/*****************************************************************************************/

#include "FFC_PERFConfig.h"
#include "OTASupp.h"
#define vTaskDelay(x) delay(x*portTICK_PERIOD_MS)

/*****************************************************************************************/
/*                    INTERNAL DEFINES                                  */
/*****************************************************************************************/

/*****************************************************************************************/

/*****************************************************************************************/
/*                        OBJECTS                                */
/*****************************************************************************************/

// create SerialBT object to control bluetooth connection
//create serial blutooth terminal to send and receive commands
BluetoothSerial SerialBT;
// create servo object to control a servo
// 16 servo objects can be created on most boards
Servo myservo;
// initialize the stepper library on pins :
Stepper myStepper(Steps_Per_Revolution, Stepper_Pin1, Stepper_Pin3, Stepper_Pin2, Stepper_Pin4);

// create motor object to control a DC MOTOR
Robojax_L298N_DC_motor motor(Motor1_Pin1, Motor1_Pin2, Motor1_Enable_Pin, Motor1_Debug_Logs);
// create a strip object to control a strip of RGBs
Adafruit_NeoPixel strip(Number_Of_Pixels, NEO_STRIP_RGB_PIN, NEO_GRB + NEO_KHZ800);
// create a display object to control the OLED screen
SSD1306Wire  display(OLED_I2C_ADR, OLED_SDA, OLED_SCL);
WebServer server(80);
WiFiClient espClient;
PubSubClient mqttClient(espClient);
bool updateStartedDoNoYeild = false;

/*EspMQTTClient mqttClient(
  ssid,
  password,
  ipAddress,  // MQTT Broker server ip
  "esp",   // Can be omitted if not needed
  "1234",   // Can be omitted if not needed
  "FFC",     // Client name that uniquely identify your device
  1883            // The MQTT port, default to 1883. this line can be omitted
);*/
/*****************************************************************************************/
```

```
/*                    FUNCTIONS  PROTOTYPE                              */
/*************************************************************************************/

void Init_Task              (void);
void Init_Buttons           (void);
void Init_Bluetooth         (void);
void Init_LED               (void);
void Init_OLED              (void);
void Init_BUZZER            (void);
void Init_RGB               (void);
void Init_Stepper           (void);
void Init_Servo             (void);
void Init_IR_Sensor         (void);
void Init_DC_Motors         (void);
void Init_OTA_Setup         (void);
void RGB_Change_Color       (int Red,int Green, int Blue);
void Bluetooth_Task         (void *parameter);
void Get_Buttons_State_Task (void *parameter);
void Task1                  (void *parameter);
void Task2                  (void *parameter);
void Task3                  (void *parameter);
void Task4                  (void *parameter);
void Task5                  (void *parameter);
void otaWebUpdate           (void *parameter);
void mqttReceiver           (void *parameter);
void displayString          (String str, float displaytime, int fontSize = 24);
void taskSuspendAll         (void);
void mqttReceiverCb         (char* topic, byte* message, unsigned int length);
struct timeval start_time, curT;
bool markStart = false;
/*************************************************************************************/

/*************************************************************************************/
/*                        TASK HANDLERS                              */
/*************************************************************************************/

TaskHandle_t   Task_BLUET_Handle;
//TaskHandle_t   Task_Handle_0;
TaskHandle_t   Task_Handle_1;
TaskHandle_t   Task_Handle_2;
TaskHandle_t   Task_Handle_3;
TaskHandle_t   Task_Handle_4;
TaskHandle_t   Task_Handle_WebUpdate;
TaskHandle_t   Task_Handle_Mqtt;

void setup()
{
  Serial.begin(115200);

/*************************************************************************************/
 /*                       CONFIGURATIONS                              */

/*************************************************************************************/
 //call the initialization function
 StatusIndicator_init();

 Init_Task();
 //tasks creation
 /*
```

```
  xTaskCreate(Bluetooth_Task       ,"BLUE_TASK",2524,NULL,1,&Task_BLUET_Handle);
  xTaskCreate(Task1               ,"TASK1",1024,NULL,1,&Task_Handle_1);
  xTaskCreate(Task2               ,"TASK2",1536,NULL,1,&Task_Handle_2);
  xTaskCreate(Task3               ,"TASK3",1024,NULL,1,&Task_Handle_3);
  xTaskCreate(Task4               ,"TASK4",1024,NULL,1,&Task_Handle_4);
  xTaskCreate(otaWebUpdate         ,"WebUpdate",3524,NULL,3,&Task_Handle_WebUpdate);
  xTaskCreate(mqttReceiver         ,"mqttReceiver",2524,NULL,1,&Task_Handle_Mqtt);
  Serial.printf("setup done %d",configMINIMAL_STACK_SIZE);*/

}


void loop()
{
  Bluetooth_Task (NULL);
Task1         (NULL);
Task2         (NULL);
Task3         (NULL);
Task4         (NULL);
otaWebUpdate   (NULL);
mqttReceiver   (NULL);


  //DO NOT WRITE ANY CODE HERE
}

void StatusIndicator_init(){
  pinMode(LED1_Status_Pin, OUTPUT);
  //Serial.println("Status Runner");
}
  int ledState = LOW;
void RunIndicator()
{
  if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }
  digitalWrite(LED1_Status_Pin, ledState);
  //Serial.println("Status Runner");
}

/*****************************************************************************************/
/*                    FUNCTIONS DECLERATIONS                              */
/*****************************************************************************************/

/*****************************************************************************************/
/*NOTES    : this function initialize all of the pripherals and GPIOs                */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                              */
/*****************************************************************************************/

void IRAM_ATTR Init_Task (void)
{
  //serial monitor
  RunIndicator();
  //this function interferes with the other code part Init_OLED()
  //initializing bluetooth
  Init_Bluetooth();
  RunIndicator();
  //initializing buttons
  Init_Buttons();
  //initialize led
  Init_LED();
  //initialize RGB strip
  Init_RGB();
  //initialize Buzzer
  Init_BUZZER();
  RunIndicator();
  //initialize stepper
  Init_Stepper();
  RunIndicator();
  //initialize servo
  Init_Servo();
  RunIndicator();
  //initialize DC motor;
  Init_DC_Motors();
  RunIndicator();
  //initialize IR sensor
  Init_IR_Sensor();
  RunIndicator();
  //initialize OLED screen
  Init_OLED();
  //Initialize OTA, this will print IP on screen
  Init_OTA_Setup();
  RunIndicator();
}

/*******************************************************************************************/
/*NOTES    : this function initialize the bluetooth                          */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                                  */
/*******************************************************************************************/

void IRAM_ATTR Init_Bluetooth(void)
{
  //Bluetooth device name, you can give it any name
  SerialBT.begin("ESP32_Blutooth");
}

/*******************************************************************************************/
/*NOTES    : this function initialize Push Buttons pin as input pull up               */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                                  */
/*******************************************************************************************/

void IRAM_ATTR Init_Buttons (void)
{
  //configure push buttons as input pull up
  pinMode(Push_Button1, INPUT);
  pinMode(Push_Button2, INPUT);
}

/*******************************************************************************************/
/*NOTES    : this function initialize LED pins as outputs                        */
/*INPUTS   : No INPUTS                                            */
/*RETURNES : NO OUTPUTS                                                  */
/*******************************************************************************************/
```

```
void IRAM_ATTR Init_LED (void)
{
  //configure pins as output
  pinMode(LED1_Pin, OUTPUT);
  pinMode(LED2_Pin, OUTPUT);
  pinMode(BT_TSK_LED, OUTPUT);
}

/**********************************************************************************************/
/*NOTES    : this function initialize OLED screen                                 */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                   */
/**********************************************************************************************/

void IRAM_ATTR Init_OLED (void)
{
  // initialize OLED with I2C addr 0x3C or the address provided with your LCD
  display.init();
  display.setFont(ArialMT_Plain_16);
  displayString("Booting Up",1,24);
  vTaskDelay(1000/portTICK_PERIOD_MS);
  displayclear();
  //vTaskDelay(1000/portTICK_PERIOD_MS);
}

/**********************************************************************************************/
/*NOTES    : this function initialize the buzzer                               */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                   */
/**********************************************************************************************/

void IRAM_ATTR Init_BUZZER (void)
{
  // Configure BUZZER functionalities. ledcSetup(ledChannel, freq, resolution);
  ledcSetup(BUZZER_CHANNEL, BUZZER_Frequency, BUZZER_Resoluation);
  // Attach BUZZER pin.   ledcAttachPin(GPIO,Channel)
  ledcAttachPin(BUZZER_Pin, BUZZER_CHANNEL);
}

/**********************************************************************************************/
/*NOTES    : this function initialize te RGB strip                              */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                   */
/**********************************************************************************************/

void IRAM_ATTR Init_RGB (void)
{
  //INITIALIZE NeoPixel strip object (REQUIRED)
  strip.begin();
  RGB_Change_Color(0, 0, 0);
}

/**********************************************************************************************/
/*NOTES    : this function initialize stepper motors                             */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                   */
/**********************************************************************************************/
void IRAM_ATTR Init_Stepper (void)
```

```
{
  // set the speed at rpm:
  myStepper.setSpeed(Stepper_Speed);

}

/*****************************************************************************************/
/*NOTES    : this function initialize se servo motor                          */
/*INPUTS   : No INPUTS                                              */
/*RETURNES : NO OUTPUTS                                                */
/*****************************************************************************************/

void IRAM_ATTR Init_Servo (void)
{
  //initialize servo for esp32
  // Allow allocation of all timers
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  // Standard 50hz servo
  myservo.setPeriodHertz(50);
  // using default min/max of 1000us and 2000us
  // different servos may require different min/max settings
  // for an accurate 0 to 180 sweep
  //attach servo pin to the obect
  myservo.attach(Servo1_Pin, 100, 2000);
}

/*****************************************************************************************/
/*NOTES    : this function initialize the DC motor                            */
/*INPUTS   : No INPUTS                                              */
/*RETURNES : NO OUTPUTS                                                */
/*****************************************************************************************/

void IRAM_ATTR Init_DC_Motors(void)
{
  // INITIALIZE motor object (REQUIRED)
  motor.begin();
}

/*****************************************************************************************/
/*NOTES    : this function initialize IR Sensor and set it in receiving mode           */
/*INPUTS   : No INPUTS                                              */
/*RETURNES : NO OUTPUTS                                                */
/*****************************************************************************************/

void IRAM_ATTR Init_IR_Sensor(void)
{
  // Start the receiver
  //IR_Receiver.enableIRIn();
  pinMode(IR_Receive_Pin,INPUT);
}

/*****************************************************************************************/
/*NOTES    : this function initialize OTA related setup          */
/*INPUTS   : No INPUTS                                              */
/*RETURNES : NO OUTPUTS                                                */
/*****************************************************************************************/
```

```
bool once_displ =false;
void IRAM_ATTR  Init_OTA_Setup() {
    //setup function
   FEED_WATCH_DOG
   WiFi.begin(ssid, password);
   FEED_WATCH_DOG
   Serial.println("");
   // Wait for connection
   while (WiFi.status() != WL_CONNECTED) {
      FEED_WATCH_DOG
            vTaskDelay(10/portTICK_PERIOD_MS);
      FEED_WATCH_DOG
            Serial.printf(". %d",WiFi.status());
      FEED_WATCH_DOG
   }
   Serial.println("");
   Serial.print("Connected to ");
   Serial.println(ssid);
   Serial.print("IP address: ");

   FEED_WATCH_DOG
     String str = WiFi.localIP().toString();
     Serial.println(str);
   FEED_WATCH_DOG
     displayString(str,5,10);
   FEED_WATCH_DOG

   mqttClient.setServer(ipAddress, 1883);
   mqttClient.setCallback(mqttReceiverCb);
   mqttClient.connect(mqttClientID, mqttUser, mqttpwd);

   /*use mdns for host name resolution*/
   if (!MDNS.begin(host)) { //http://esp32.local
     Serial.println("Error setting up MDNS responder!");
     while (1) {
       vTaskDelay(10/portTICK_PERIOD_MS);
     }
   }
   Serial.println("mDNS responder started");
   /*return index page which is stored in serverIndex*/
   server.on("/", HTTP_GET, []() {
      //suspend once connection established- reboot can only restablish all
        taskSuspendAll();
      FEED_WATCH_DOG
        server.sendHeader("Connection", "close");
      FEED_WATCH_DOG
        server.send(200, "text/html", loginIndex);
        displayString("Connection Established",2,10);
        updateStartedDoNoYeild = true;
      FEED_WATCH_DOG
   });
   server.on("/serverIndex", HTTP_GET, []() {
        server.sendHeader("Connection", "close");
      FEED_WATCH_DOG
        server.send(200, "text/html", serverIndex);
        displayString("Authentication Success\nStarting Update",1,10);
      FEED_WATCH_DOG
   });
   /*handling uploading firmware file */
```

```
  server.on("/update", HTTP_POST, []() {
      server.sendHeader("Connection", "close");
     FEED_WATCH_DOG
      server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
     FEED_WATCH_DOG
      ESP.restart();
     FEED_WATCH_DOG
         }, []() {
         FEED_WATCH_DOG
         HTTPUpload& upload = server.upload();
           Serial.printf("UpdateStatus: %lu\n", upload.status);
           if (upload.status == UPLOAD_FILE_START) {
             Serial.printf("Update: %s Max Size %d\n", upload.filename.c_str(),UPDATE_SIZE_UNKNOWN);
             if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
                 Update.printError(Serial);
             }
           } else if (upload.status == UPLOAD_FILE_WRITE) {
               if(!once_displ)
                 displayString("Writing File",1,10);
               once_displ = true;
               if(!markStart) {
                 markStart = true;
                 gettimeofday(&start_time, NULL);
                 Serial.printf("Update: start time %lu\n",start_time.tv_sec*1000000  + start_time.tv_usec);
               }
               Serial.printf("writing: %lu %lu \n", upload.currentSize, upload.totalSize);
               /* flashing firmware to ESP*/
               if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
                  Update.printError(Serial);
               }
           } else if (upload.status == UPLOAD_FILE_END) {
             if (Update.end(true)) { //true to set the size to the current progress
               Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
               gettimeofday(&curT, NULL);
               Serial.printf("Update: endt time %lu TimeTaken for update %d\n",curT.tv_sec,(curT.tv_sec -
start_time.tv_sec));
               displayString("Update Success \n nRebooting",1,10);
             }
           }
           else if (upload.status == UPLOAD_FILE_ABORTED) {
              Serial.printf("UnExpected Error Rebooting");
              ESP.restart();
           }
         FEED_WATCH_DOG
         });
         server.begin();
     FEED_WATCH_DOG
}

/*****************************************************************************************/
/*NOTES    : this function Sets color to the RGB strip to activat any color set 255 or   */
/* any desired brightness level to deactivate a color put o inits place                   */
/*INPUTS   : RED , Green , Blue. respectevily                                            */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/

void IRAM_ATTR RGB_Change_Color (int Red,int Green, int Blue)
{
  RunIndicator();
  // Set all pixel colors to 'off'
  strip.clear();
  for(int i=Number_Of_Pixels; i>=0; i--)
  {
    strip.setPixelColor(i, strip.Color(Red, Green, Blue));
  }
  strip.show();

}

/***************************************************************************************/
/*NOTES    : task connect to a bluetooth and get data                              */
/*INPUTS   : No INPUTS                                                    */
/*RETURNES : NO OUTPUTS                                                         */
/***************************************************************************************/

void IRAM_ATTR Bluetooth_Task(void *parameter)
{

  //while(1)
  {
    RunIndicator();
    //cheack the blutooth is connected
    if (SerialBT.available())
    {
      Bluetooth_data = SerialBT.read();
      //start sending data to the mobile
      SerialBT.write(Bluetooth_data);
      //update the bluetooth data with the received data
    }
    void *vt = NULL;
    Task5(vt);
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}


/***************************************************************************************/
/*NOTES    : this task checks if the button is pressed or not if presed it updates its   */
/* related flag. flag1 for button1, flag2 for button2                          */
/*INPUTS   : No INPUTS                                                    */
/*RETURNES : NO OUTPUTS                                                         */
/***************************************************************************************/

void IRAM_ATTR Get_Buttons_State_Task  (void *parameter){}


/***************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                    */
/*RETURNES : NO OUTPUTS                                                         */
/***************************************************************************************/

void IRAM_ATTR Task1  (void *parameter)
{
  //while(1)
  {
    Serial.println("Task1");
    RunIndicator();
    //turn on LEDs
    digitalWrite(LED1_Pin,HIGH);
    digitalWrite(LED2_Pin,LOW);
    ////set the  buzzer tone to DO. ledcWriteTone(channel,freq)
    ledcWriteTone(BUZZER_CHANNEL, BUZZER_Frequency_Tone1);
    //delay for 0.5 s
    vTaskDelay(500/portTICK_PERIOD_MS);
    //toggle LEDs
    digitalWrite(LED1_Pin,LOW);
    digitalWrite(LED2_Pin,HIGH);
    //change buzzer tone  to MI
    ledcWriteTone(BUZZER_CHANNEL, BUZZER_Frequency_Tone2);
    //delay for 0.5 s
    vTaskDelay(500/portTICK_PERIOD_MS);
  }

}

/*****************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                   */
/*****************************************************************************************/

void IRAM_ATTR Task2  (void *parameter)
{
 //while(1)
  {
    Serial.println("Task2");
    bool ButtonPressed = false;
    RunIndicator();
    //check if the button1 is pressed or not
    if(digitalRead(Push_Button2) == 1)
    {
      // to handel debouncing
      //Aman:while(digitalRead(Push_Button2) == 0);
      vTaskDelay(50/portTICK_PERIOD_MS);
      if(digitalRead(Push_Button2) == 1)
      {
        //update flage state to indicate its pressed
        ButtonPressed = true;
      }
    }
    if (ButtonPressed == true )
    {
      //set the angle to the servo
      myservo.write(Servo_High_Angle);
      displayString("Danger",1);
    }
    else
    {
      // display nothing on the LCD
      displayclear();
      //set the servo in its Low position
      myservo.write(Servo_Low_Angle);
    }
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}
```

```
/*****************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                         */
/*****************************************************************************************/

void IRAM_ATTR Task3  (void *parameter)
{
 //while(1)
  {
    Serial.println("Task3");
    RunIndicator();
    bool ButtonPressed = false;
    RunIndicator();
    //check if the button1 is pressed or not
    if(digitalRead(Push_Button1) == 1)
    {
      // to handel debouncing
      vTaskDelay(10/portTICK_PERIOD_MS);
      if(digitalRead(Push_Button1) == 1)
      {
        //update flage state to indicate its pressed
        ButtonPressed = true;
      }
    }

    if(ButtonPressed){
      //RGB (Green)
      RGB_Change_Color (0, 255, 0);
      // Stepper motor rotate CW
      myStepper.step(Steps_Per_Revolution);

      //delay for 5 s
      vTaskDelay(5000/portTICK_PERIOD_MS);
      //Before Rotating Stepper change colour to blue
      RGB_Change_Color(0, 0, 255);
      // Rotate Stepper Back
      myStepper.step(-Steps_Per_Revolution);
      }
      else{
        RGB_Change_Color(0, 0, 0);
      }
  }
}
/*****************************************************************************************/
/*NOTES    : task actions are given in requirments                              */
/*INPUTS   : No INPUTS                                                   */
/*RETURNES : NO OUTPUTS                                                         */
/*****************************************************************************************/

void IRAM_ATTR Task4  (void *parameter)
{
 //while(1)
  {
    Serial.println("Task4");
    RunIndicator();
    //Check if received data is available and if yes, try to decode it.
    //Decoded result is in the IrReceiver.decodedIRData structure.
```

```
   int inputVal = digitalRead(IR_Receive_Pin);
    //check the recived data between 0 and 500
   if(!inputVal)
   {
    //DC Motor rotate left
    motor.rotate(motor1, DC_Motor_Speed, CCW );//run motor1 at 10% speed in CCW direction
    vTaskDelay(1000/portTICK_PERIOD_MS);
    motor.brake(motor1);
    vTaskDelay(500/portTICK_PERIOD_MS);
   }
   vTaskDelay(50/portTICK_PERIOD_MS);
  }
}

/******************************************************************************************/
/*NOTES    : task controls DC motor based on the received data from bluetooth            */
/*INPUTS   : No INPUTS                                                  */
/*RETURNES : NO OUTPUTS                                                  */
/******************************************************************************************/

void IRAM_ATTR Task5  (void *parameter)
{
   RunIndicator();
   Serial.println("Task5");
   if((Bluetooth_data == 'A') || (Bluetooth_data == 'a'))
   {
    int N = 2;
    //Blink N times BT TASK LED
    for(int i = 0; i < N; i++){
      digitalWrite(BT_TSK_LED,HIGH);
      vTaskDelay(500/portTICK_PERIOD_MS);
      digitalWrite(BT_TSK_LED,LOW);
      vTaskDelay(500/portTICK_PERIOD_MS);
    }
    }
   else if ((Bluetooth_data == 'B') || (Bluetooth_data == 'b'))
   {
    int N = 5;
    //Blink N times BT TASK LED
    for(int i = 0; i < N; i++){
      digitalWrite(BT_TSK_LED,HIGH);
      vTaskDelay(100/portTICK_PERIOD_MS);
      digitalWrite(BT_TSK_LED,LOW);
      vTaskDelay(100/portTICK_PERIOD_MS);
    }

   }
}

void IRAM_ATTR otaWebUpdate (void *parameter)
{
 //while(1)
 {
    bool ButtonPressed = false;
      //for(int i = 0; i < INT_MAX; i++);

               server.handleClient();
      Serial.printf("Update: %s(0_2)\n",__func__);
      if(once_displ)
```

```
        vTaskDelay(10/portTICK_PERIOD_MS);
        else
        vTaskDelay(1000/portTICK_PERIOD_MS);

    }//end while 1
}

/*****************************************************************************************/
/*NOTES    : Function to print string on display                              */
/*INPUTS   : Display Strting                                            */
/*RETURNES : NO OUTPUTS                                                  */
/*****************************************************************************************/
//Supported Font Size 10,16,24
void setFontSize(int fontSize) {
  switch (fontSize) {
    case 10:
        display.setFont(ArialMT_Plain_10);
        break;
    case 16:
        display.setFont(ArialMT_Plain_16);
        break;
    case 24:
    default:
        display.setFont(ArialMT_Plain_24);
        break;
    }

}

void IRAM_ATTR mqttReceiver (void *parameter)
{
  //while(1)
  {
    Serial.println("mqttReceiver__update");
    mqttClient.subscribe("ffc/led1");
    mqttClient.loop();
    vTaskDelay(100/portTICK_PERIOD_MS);
  }
}


void mqttReceiverCb(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();



  if (String(topic) == "ffc/led1") {
    digitalWrite(BT_TSK_LED,HIGH);
    vTaskDelay(100/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,LOW);
    vTaskDelay(100/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,HIGH);
    vTaskDelay(100/portTICK_PERIOD_MS);
    digitalWrite(BT_TSK_LED,LOW );
    vTaskDelay(100/portTICK_PERIOD_MS);


   if ((messageTemp == "LED:0")) {
      digitalWrite(BT_TSK_LED,LOW);

   }
    else if((messageTemp == "LED:1"))
      digitalWrite(BT_TSK_LED,HIGH);
  }

}

void displayString(String str,float displaytime, int fontSize){
    displayclear();
    setFontSize(fontSize);
    if(str.length() > 0) {
      display.setTextAlignment(TEXT_ALIGN_LEFT);
      display.drawString(0, 10, str);
      // Display it on the screen
      display.display();
      vTaskDelay((((int)displaytime)*1000)/portTICK_PERIOD_MS);
    }
};

void displayclear()
{
  for(int i=0;i<5000000;i++);
  display.init();
  display.display();
 }

void taskSuspendAll() {
  /*vTaskSuspend( Task_BLUET_Handle);
  vTaskSuspend( Task_Handle_1);
  vTaskSuspend( Task_Handle_2);
  vTaskSuspend( Task_Handle_3);
  vTaskSuspend( Task_Handle_4);
  vTaskSuspend( Task_Handle_Mqtt);*/
}
void taskResumeAll() {
 /* vTaskResume( Task_BLUET_Handle);
  vTaskResume( Task_Handle_1);
  vTaskResume( Task_Handle_2);
  vTaskResume( Task_Handle_3);
  vTaskResume( Task_Handle_4);
  vTaskResume( Task_Handle_Mqtt);*/
}
```

FFC_PERPH_CONFIG.h

```
#include "FFC_PINConfig.h"
/*****************************************************************************************/
/*                          INCLUDES                                       */
/*****************************************************************************************/
```

```
#ifndef _FFC_PERF_CONFIG
#define _FFC_PERF_CONFIG

#include "FFC_PINConfig.h"

//macro to check if the bluetooth is enabled or not
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

/*****************************************************************************************/
/*                    VARIABLES DEFINETIONS                                             */
/*****************************************************************************************/

char  Bluetooth_data;

//** IR SENSOR **
#define   IR_Receive_Pin         A2

//** BUTTONS **
#define Push_Button1            A3
#define Push_Button2            A4




//** LED **
#define LED1_Status_Pin          2
#define LED1_Pin               O2
#define LED2_Pin               O3
#define BT_TSK_LED               A7 //O10


//** Stepper **
#define Stepper_Pin1            A8//O6
#define Stepper_Pin2            A9//O7
#define Stepper_Pin3            A10//O8
#define Stepper_Pin4            A11//O9
// change this to fit the number of steps per revolution to make 360 degree
#define Steps_Per_Revolution      500
#define Stepper_Speed            13 //set the stepper speed in rpm


//** Servo **
// Recommended PWM GPIO pins on the ESP32 include 2,4,12-19,21-23,25-27,32-33
#define Servo1_Pin              PWM3
#define Servo_High_Angle         180
#define Servo_Low_Angle          0


//** DC MOTOR **
// motor1 settings
#define Motor1_Debug_Logs         0
#define Motor1_Enable_Pin         PWM4 //channel that will be used to generate PWM internally
#define Motor1_Pin1              O4
#define Motor1_Pin2              O5
#define motor1                  1 // do not change
#define CW                     1 // do not change
#define CCW                     2 // do not change
```

```
#define DC_Motor_Speed           40


//** Buzzer **
#define BUZZER_CHANNEL           3   //channel that will be used to generate PWM internally
#define BUZZER_Pin               O1  //the GPIO pin that we will use to get the output to the buzzer
#define BUZZER_Frequency_Tone1    262 //the frequency for DO
#define BUZZER_Frequency_Tone2    330 //the frequency for MI
#define BUZZER_Resoluation        12  //PWM resolution
#define BUZZER_Frequency          1000//pwm frequency


//** RGB strip **
//Please note DO NOT connect the VCC and the GND of the strip with the 5v and the GND of the ESP32.
//use an external power supply.
// Which pin on the ESP is connected to the NeoPixels?
#define NEO_STRIP_RGB_PIN        O7 //O11
#define Number_Of_Pixels         30


//** OLED 0.96INCH**
// in case you dont have a reset in your OLED weite -1 instead of pin number
//note connect SDA with the D21 in ESP32 and SCL D22
#define OLED_RESET               -1
#define SCREEN_WIDTH              128 // OLED display width, in pixels
#define SCREEN_HEIGHT              64  // OLED display height, in pixels
#define OLED_I2C_ADR              0x3c // OLED I2C ADDERESS
#define OLED_SDA                 SDA1
#define OLED_SCL                 SCL1
#endif
```

FFC_PIN_CONF.h

```
/****************************************************************************************/
/*                        INCLUDES                                              */
/****************************************************************************************/
#ifndef _FFC_PINCONFIG_H
#define _FFC_PINCONFIG_H

#include <SPI.h>
#include <Wire.h>
#include <Stepper.h>
#include <ESP32Servo.h>
//#include <Adafruit_GFX.h>
#include <BluetoothSerial.h>
#include <Wire.h>
#include "SSD1306Wire.h"
#include <Adafruit_NeoPixel.h>
#include <Robojax_L298N_DC_motor.h>

//INPUTS
#define  A0   36  //**[Do Not Use-Crash on device]
#define  A1   39
#define  A2   34  // IR_Receive_Pin
#define  A3   35  // Push button1
#define  A4   32  // Push button2
#define  A5   33
#define  A6   25
```

```
#define  A7   26  // BT_TSK_LED
#define  A8   27  // Stepper_Pin1//**[Re-Confed As O/P]
#define  A9   14  // Stepper_Pin2//**[Re-Confed As O/P]
#define  A10  12  // Stepper_Pin3//**[Re-Confed As O/P]
#define  A11  13  // Stepper_Pin4//**[Re-Confed As O/P]
//OUTPUTS
#define  O0   19  // [Occupied as PWM4 initialized in PWN section]
#define  O1   18  // BUZZER_Pin
#define  O2   5   // LED1_Pin
#define  O3   17  // LED2_Pin
#define  O4   16  // Motor1_Pin1
#define  O5   4   // Motor1_Pin2
#define  O6   (int)(2)   //**[On Board LED - Not Usable]
#define  O7   15  // NEO_STRIP_RGB_PIN
#define  O8   0   //**[not usaable- unavailable pinout on breakout board ]
#define  O9   8   //**[not usaable- used by Bootloader ]
#define  O10  7   //**[not usaable- used by Bootloader ]
#define  O11  6   //**[not usaable- used by Bootloader ]
//PWM
#define  PWM0   9  //**[not usaable- used by Bootloader ]
#define  PWM1   10 //**[not usaable- used by Bootloader ]
#define  PWM2   11 //**[not usaable- used by Bootloader ]
#define  PWM3   23 // Servo1_Pin/ pwm channel 1
#define  PWM4   O0
 //I2C
#define  SDA1   21 //SDA OLED Pin
#define  SCL1   22 //SCL OLED Pin

// UART

#define  UART0_TX 1
#define  UART0_RX 3

#endif
```

OTASupp.h

```
#ifndef _OTA_SUPP_CONFIG
#define _OTA_SUPP_CONFIG

#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

#define FEED_WATCH_DOG vTaskDelay(10/portTICK_PERIOD_MS);
const char* host = "esp32";
const char* ssid = "the den";
const char* password = "popcorn123";
const char* ipAddress = "192.168.1.46";

const char* mqttClientID = "FFC";
const char* mqttUser = "esp";
const char* mqttpwd = "1234";
const char* loginIndex =
 "<form name='loginForm'>"
   "<table width='20%' bgcolor='A09F9F' align='center'>"
      "<tr>"
```

```
          "<td colspan=2>"
             "<center><font size=4><b>ESP32 Login Page</b></font></center>"
             "<br>"
          "</td>"
          "<br>"
          "<br>"
       "</tr>"
       "<td>Username:</td>"
       "<td><input type='text' size=25 name='userid'><br></td>"
       "</tr>"
       "<br>"
       "<br>"
       "<tr>"
          "<td>Password:</td>"
          "<td><input type='Password' size=25 name='pwd'><br></td>"
          "<br>"
          "<br>"
       "</tr>"
       "<tr>"
          "<td><input type='submit' onclick='check(this.form)' value='Login'></td>"
       "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
    "if(form.userid.value=='admin' && form.pwd.value=='admin')"
    "{"
    "window.open('/serverIndex')"
    "}"
    "else"
    "{"
    " alert('Error Password or Username')/*displays error message*/"
    "}"
    "}"
"</script>";

/*
     * Server Index Page
     */

const char* serverIndex =
"<script src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
   "<input type='file' name='update'>"
       "<input type='submit' value='Update'>"
    "</form>"
 "<div id='prg'>progress: 0%</div>"
 "<script>"
  "$('form').submit(function(e){"
  "e.preventDefault();"
  "var form = $('#upload_form')[0];"
  "var data = new FormData(form);"
  " $.ajax({"
  "url: '/update',"
  "type: 'POST',"
  "data: data,"
  "contentType: false,"
  "processData:false,"
  "xhr: function() {"
  "var xhr = new window.XMLHttpRequest();"
  "xhr.upload.addEventListener('progress', function(evt) {"
  "if (evt.lengthComputable) {"
  "var per = evt.loaded / evt.total;"
  "$('#prg').html('progress: ' + Math.round(per*100) + '%');"
  "}"
  "}, false);"
  "return xhr;"
  "},"
  "success:function(d, s) {"
  "console.log('success!')"
 "},"
 "error: function (a, b, c) {"
 "}"
 "});"
 "});"
 "</script>";

 #endif
```

V. Conclusion

Working with dependency over physical access to the hardware is not only time taking but a tedious process making static workspace. With the increasing demand for IoT, multiple wireless protocols have come up with a different solution to support different devices, based on your project requirement you can select any protocol and establish a wireless remote connection and communicate to the device. Here since ESP32 is a low memory size device which is a challenge when come to utilizing its integrated WiFi and other modules together for project requirement. One such application we made with OTA and MQTT to understand the challenges and we came with few optimized application that supports OTA Web update for remote firmware upgrade using a web-browser and MQTT which is a messaging protocol that can be used not only for one to one communication but to broadcast message among other clients that are connected to MQTT server to publish and subscribe the topic of their interest. Figure 4 and Figure 5 itself conclude the agenda to execute the similar application over Bare Metal and FreeRTOS that shows a clear comparison between the file uploading time difference from Web Browse and the message time difference. The time takes to upload a file from Bare Metal is more than the time takes to upload a file from a Web browser over a FreeRTOS in ESP32. Since Scheduling of tasks and memory management plays a major role to reduce the execution time it's easier and less time taking for FreeRTOS to utilize the wireless upcoming technology.

References

[1]. https://en.wikipedia.org/wiki/BareMetal
[2]. https://freertos.org
[3]. http://esp32.net
[4]. https://www.arduino.cc/
[5]. http://espressif.com/en/media_overview/news/espressif-announces-launch-esp32-cloud-chip-and-funding-fosun-group
[6]. With reference to previous paper *Analysis of FreeRTOS Vs Bare Metal using ESP32*
[7]. https://randomnerdtutorials.com/esp32-over-the-air-ota-programming/
[8]. https://github.com/jnsdbr/esp32-ota-update-mqtt
[9]. https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
[10]. https://www.instructables.com/Secure-Mosquitto-MQTT-Server-for-IoT-Devices-ESP32/

Ahmad A A Alsaleh. "Analysis of Free RTOS Vs Bare Metal using OTA over ESP32." *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, 16(3), (2021): pp. 35-72.