IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676, p-ISSN: 2320-3331, Volume 16, Issue 2 Ser. I (Mar. – Apr. 2021), PP 39-68
www.iosrjournals.org

Analysis of Free RTOS Vs Bare Metal using ESP32

EE.Ahmad A A Alsaleh
Switching Department, Higher Institute of Telecommunication and Navigation,
The Public Authority for Applied Education and Training (PAAET), Kuwait

Abstract: *This paper presents a comparison between Free RTOS and Bare Metal tasks execution. Now such a comparison allows us to understand the significance of having two different open-source entities for micro-controllers and microprocessors. Such comparison plays an essential role to understand the performance and functionality of the controllers as per requirements. Such significance requires a setup having a micro-controller or microprocessor, to achieve this a prototype of a Fire-Fighting Car performing multiple-tasks using FreeRTOS and Bare Metal.*

Keywords: ESP32 DevKit, DC Motor, Stepper Motor, Servo Motor, Neo-Pixel LED strip, OLED 0.96 inch, IR Sensor, Arduino IDE.

Date of Submission: 07-03-2021 Date of Acceptance: 20-03-2021

I. Introduction

Many Prior works have focused on the FreeRTOS vs Bare Metal performance for different microcontrollers and microprocessors. Here, we only discuss the few works that are relevant to our goals. A work of prototype presenting a fire fighting car using ESP32 Development Board which supports rich sets of peripherals. Using multi-functional pins of ESP32 to operate different peripherals like motors, Neo-Pixel LED, Buzzer as output which requires PWM for their functionality and where are IR Sensors and BLE and Buttons used as Input to ESP32. ESP32 S is a dual-core MCU that can execute two tasks simultaneously.

Since ESP32 supports a rich set of peripherals the prototype designed to function as a fire bridged which has an alert message display, flag signal, water pump, LED siren. Arduino IDE is used for code compilation and uploading. Now the functionality of the prototype and its peripheral is whenever there is a fire emergency call will be made a vehicle equipped with all the necessary tool will reach the location.

There are two manually pushed buttons and an IR Sensor which sends input to ESP32 based on which there are certain task will perform, when button one will be pushed the vehicle will rotate stepper motor to move the ladder up and down and a Neo-pixel LED strip glows with green and blue color. When button two is pushed a servo motor turn to raise the flag and an OLED displays a danger alert with some warning message. When IR Sensor finds any obstacle in between it sends a single DC motor to turn the vehicle left or right. To perform all these tasks and multiple functionality ESP32 supports FreeRTOS and Bare-Metal both the sources.

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.

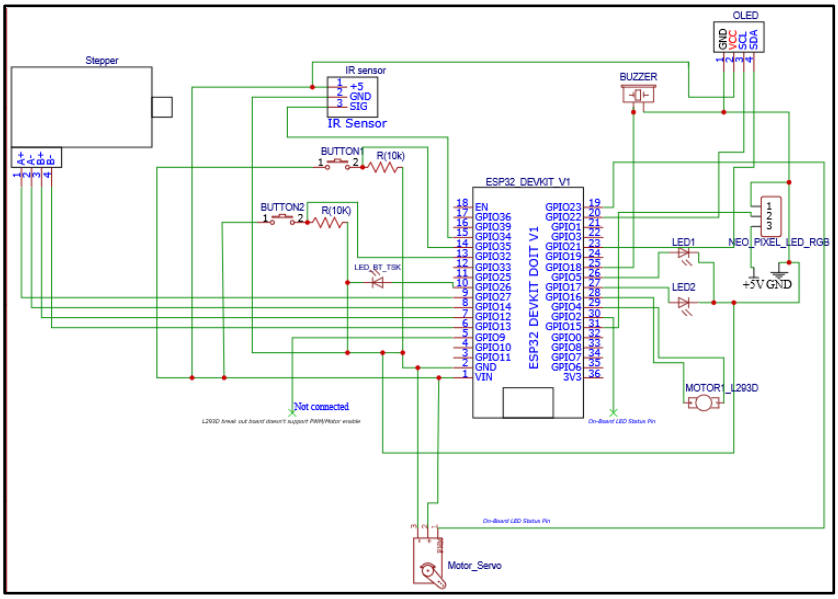

Figure 1. Equivalent Circuit diagram of Prototype

2.2 Free RTOS

Free RTOS is a real-time operating system (RTOS) for any microcontroller 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.

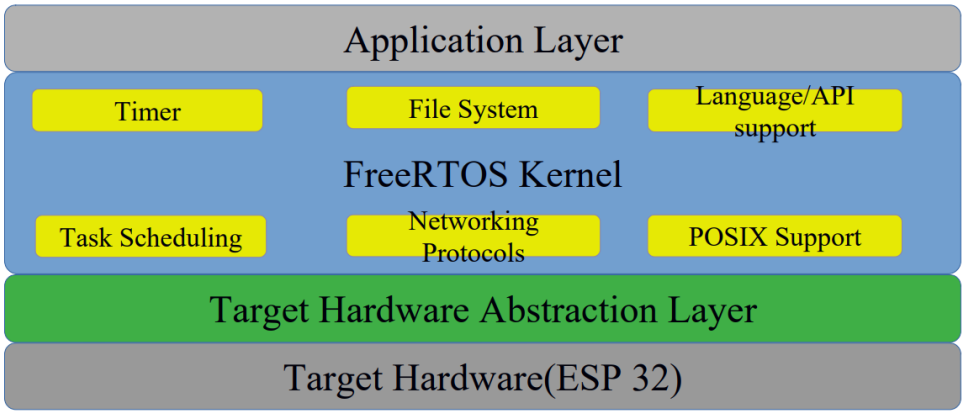

Figure 2. FreeRTOS Firmware

2.3. Bare Metal

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. As you'll see within the following diagram, an infinite loop is going to be running all the time and every one task are going to be executed sequentially. For instance, if the first Task1 will execute and, then Task2 followed by Task3 and this will continue, until an interrupt event occurs. A microcontroller will stop sequential execution-only there's an interrupt event.

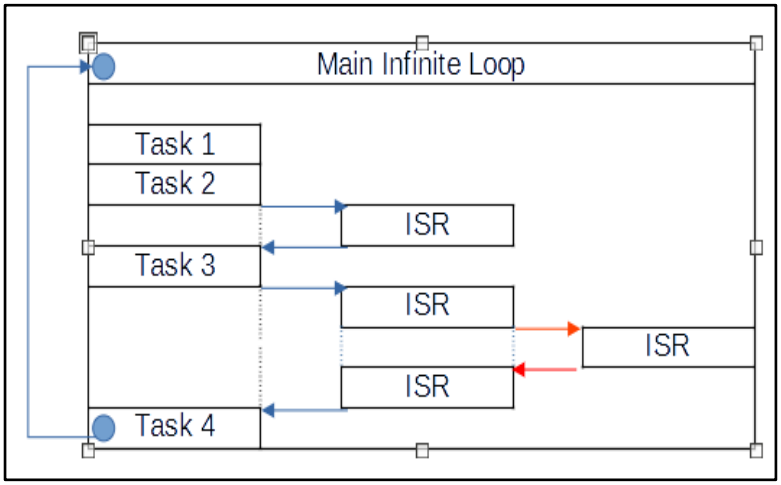

Figure3. Bare Metal Firmware

III. Analyzing the difference between FreeRTOS and Bare Metal

As we can see in Figure 2 and Figure 3 there is a difference in execution and scheduling mechanism between FreeRTOS and Bare Metal. To analysis this we executed the same setup code with FreeRTOS APIs and Bare Metal to calculate the execution time and schedule time of all the tasks in the Ideal case and when input is given to those peripherals.

3.1IR Sensor

IR sensor is a device, that emits sunshine to sense some object of the environment. An IR sensor can measure the heat of an object also as detects the motion. Usually, within the spectrum, all the objects radiate some sort of thermal radiation. These sorts of radiations are invisible to our eyes, but the infrared sensor can detect these radiations. the space sensor is often connected to any ADC (the analog-digital converter) channel of the ESP32 module. Here the analog input received by IR Sensor is shipped to the PIN set for it and the sent data is converted to digital input for the task to run the DC(Direct Current) motor as output and the functionality of the motor is to rotate the vehicle.

Attached is the thing file of knowledge collected for the IR Sensor task where input is taken as either LOW(0) or High(1).

Below is the chart plotted showing Schedule Time and Execution Time difference in FreeRTOS and Bare Metal code:

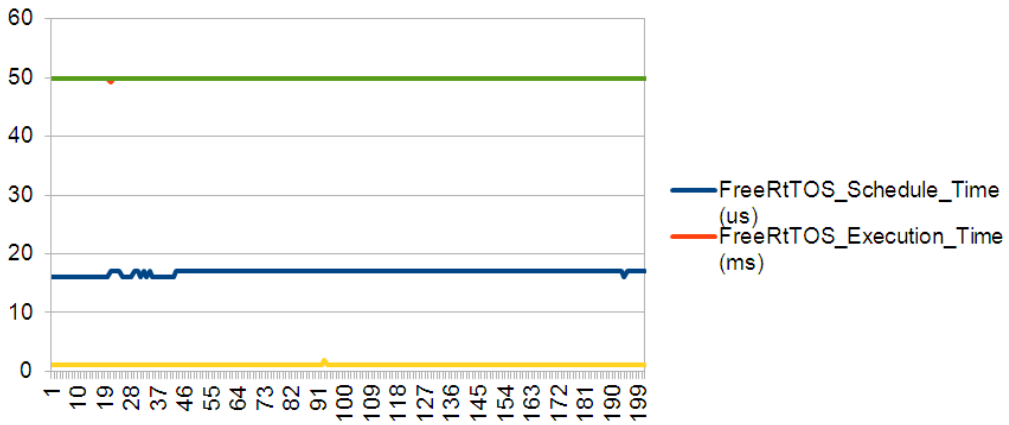

Figure 4. Execution and Schedule Time in Ideal Task

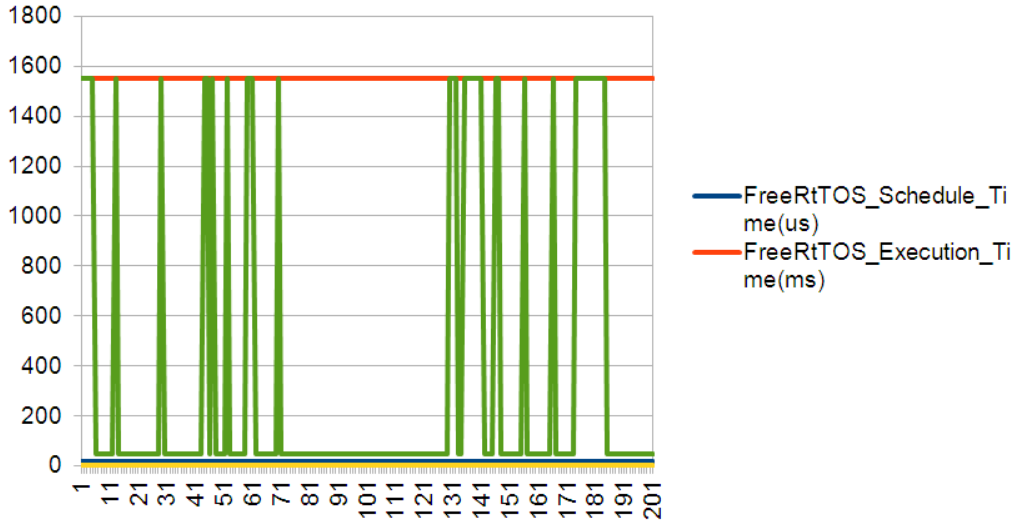

Figure 5. Execution and Schedule Time with input

3.2. OLED 0.96 INCH And Servo Motor

An organic LED (OLED or organic LED), also mentioned as organic electromagnetism (organic EL) diode, could also be a LED (LED) during which the emissive elector-lumination layer could also be a movie of the compound that emits light in response to an electrical current. It can communicate with the microcontroller using the I2C serial communication protocol. While I2C requires only two pins and should be shared with other I2C peripherals. Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors. Typical voltages used are +5V or +3.3 V, although systems with other voltages are permitted.

A servo motor could also be a kind of motor that can rotate with great precision. This type of motor consists of a circuit that provides feedback on the position of the motor shaft, this feedback allows the servo motors to rotate with great precision. An input to servos to achieve the desired result is done by sending a pulse waveform with variable width also known as pulse width modulated (PWM) wave. A PWM wave is defined by the repetition rate, high time followed by the low time. The percentage of high time/ low time decides the angle of rotation usually the range of servo is 0 to 180 degrees depending upon form factor and manufacturer.

The functionality of OLED and Servo motor is based on the sense that when button1 is getting to be pressed there will be an alert message display that is getting to be shown and a danger flag is getting to be raised. The task is that the mixture of the device working on PWM and I2C communication protocol connected to ESP32 with a set of Pins configured for PWM where all PINS in ESP32 is PWM supporting whereas PIN22 and PIN21 are to support I2C communication.

Attached are the object files of data collected for PWM and I2C communication protocol support by ESP32 using OLED and Servo motor.

Below is the chart plotted showing Schedule Time and Execution Time difference in FreeRTOS and Bare Metal code:

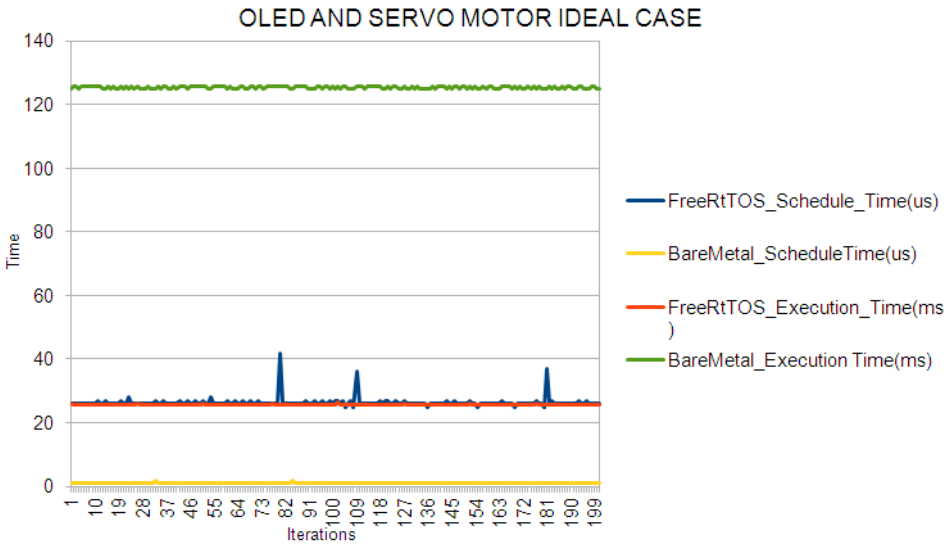

Figure 6. Execution and Schedule Time in Ideal Task

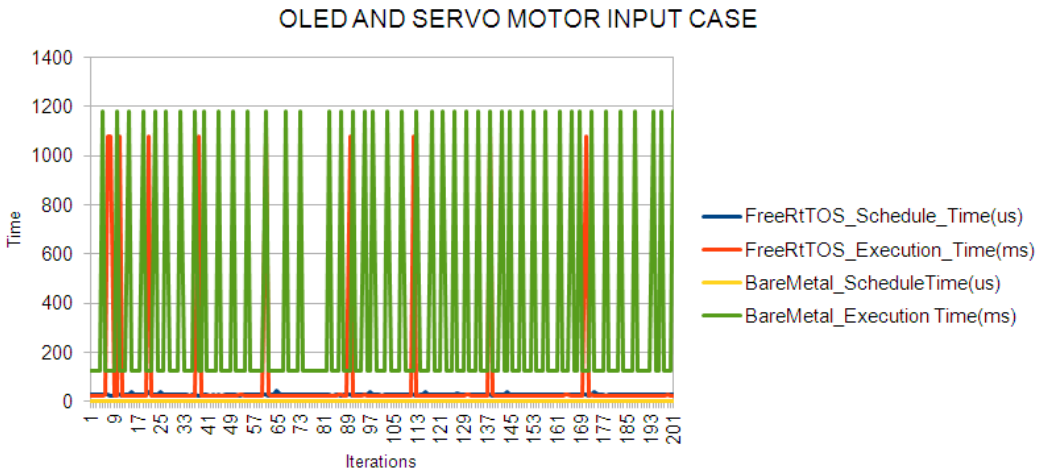

Figure 7. Execution and Schedule Time with Input

3.3 Neo-pixel LED(RGB) and Stepper Motor

Neo-Pixel is individually addressable LEDs all housed on a string which will be controlled from one pin on a microcontroller. This suggests one pin can control all of the LEDs' colors and which LEDs are on at any given time. many companies sell strips of addressable LEDs however the foremost popular is that the Neo-pixel by Adafruit. They require 3 PIN connections GND,3.3 Vcc, and a knowledge (Din) PIN which is connected to ESP32. Neo-Pixel LED works with NZR (Non-Return-to-Zero)Communicationmode.

A stepper motor also referred to as a step motor or stepper, maybe a brush-less DC motor that divides a full rotation into a variety of equal steps. The functionality of a stepper motor is that it converts a pulsing electrical current, controlled by a stepper motor driver, into precise one-step movements of this gear-like toothed component around a central shaft. Each of those stepper motor pulses moves the rotor through one precise and glued increment of a full turn. Here stepper motor is controlled using digital output from ESP32. there's a TTL (Transistor-Transistor Logic) signals produced by the digital I/O system are amplified by a Texas Instruments ULN2003 (high voltage, high current Darlington transistor array) IC board to connect Stepper thereto.

The significance of getting these two peripheral in the prototype is to offer a light-weight indication of when the vehicle is going to be presenting things, with green and blue color. And stepper motor is to boost the ladder to succeed in top floor for eviction and water pumping.

Attached are the thing files of knowledge collected for Digital IO and NZR communication mode supported by ESP32 using Stepper Motor and Neo-Pixel LED strip.

Below is the chart plotted showing Schedule Time and Execution Time difference in FreeRTOS and Bare Metal code:

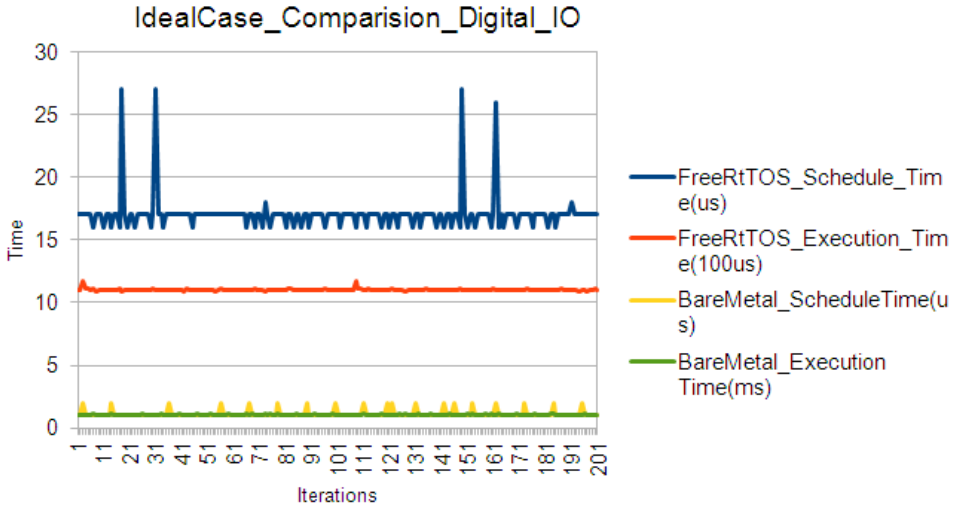

Figure 8. Execution and Schedule Time for Ideal Task

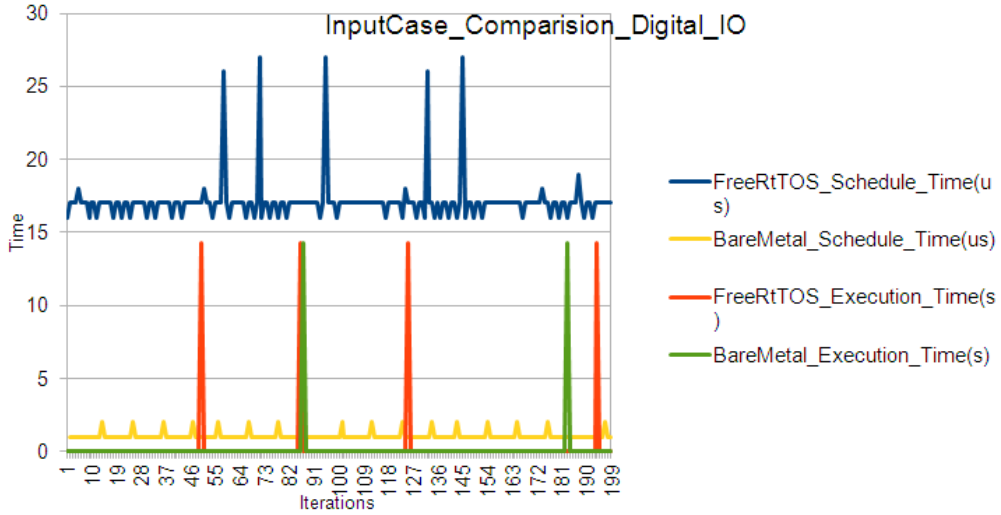

Figure 9. Execution and Schedule Time for Input Case

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 and flash the code to ESP32. It allows us to add supporting libraries for the peripherals added to ESP32.

ESP32 is a series of low-cost, low-power systems-on-chip microcontrollers with integrated Wi-Fiand dual-mode Bluetooth. The ESP32 series employs a TensilicaXtensa LX6 microprocessor in both dual-core and single-corevariations and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules. ESP32 is created and developed by Espressif Systems, a Shanghai-based Chinese company, and is manufactured by TSMCusing their 40 nm process.

I. FreeRTOS

FirstESP32.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 <sys/time.h>
/*****************************************************************************************/
/*                        INTERNAL DEFINES                                               */
/*****************************************************************************************/
struct timevalstart_time, curT;
/*****************************************************************************************/
/*****************************************************************************************/
/*                        OBJECTS                                                        */
/*****************************************************************************************/
// create SerialBT object to control bluetooth connection
//create serial blutooth terminal to send and receive commands
BluetoothSerialSerialBT;
// 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_NeoPixelstrip(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);


/*****************************************************************************************/
/*                        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 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 displayString   (String str, int displaytime);
void feedTheDog   (void);
void displayclear   (void);
/*****************************************************************************************/
/*****************************************************************************************/
/*                          TASK HANDLERS                                                */
/*****************************************************************************************/
TaskHandle_tTask_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_5;
TaskHandle_tTask_Idle;
TaskHandle_t    Task_Idle2;


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",2536,NULL,1,&Task_Handle_2);
xTaskCreate(Task3                ,"TASK3",2048,NULL,1,&Task_Handle_3);
xTaskCreate(Task4                ,"TASK4",2024,NULL,1,&Task_Handle_4);
xTaskCreate(TaskIdle               ,"TaskIdle",1536,NULL,1,&Task_Idle);
xTaskCreate(TaskIdle               ,"TaskIdle",1536,NULL,1,&Task_Idle2);
Serial.println("setup done");
}
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 peripherals 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();
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",0.5);
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 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
setting for an accurate 0 to 180 sweep
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 Sets the color to the RGB strip to activate any color set 255 or   */
/* any desired brightness level to deactivate a color put o inits place     */
/*INPUTS: RED, Green, Blue. respectively                            */
/*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 connects to a Bluetooth and gets data                                      */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void IRAM_ATTR Bluetooth_Task(void *parameter)
{

while(1)
  {
Serial.println(__func__);
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 pressed 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 requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void IRAM_ATTR Task1  (void *parameter)
{
while(1)
  {
Serial.println(__func__);
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 requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNS : NO OUTPUTS                                                                   */
/*****************************************************************************************/
void IRAM_ATTR Task2  (void *parameter)
{
 int SchedTime;

while(1){
gettimeofday(&start_time, NULL);
   //Serial.println(__func__);
   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);
   }
SchedTime = (start_time.tv_sec - curT.tv_sec)*1000+ start_time.tv_usec - curT.tv_usec;
gettimeofday(&curT, NULL);
Serial.printf("Task 2 Start %lu  Stop %lu  Time %d SchedTime %d\r\n", start_time.tv_sec*1000000  +
start_time.tv_usec ,curT.tv_sec*1000000  + curT.tv_usec ,(curT.tv_sec - start_time.tv_sec)*1000000
+ (curT.tv_usec - start_time.tv_usec),SchedTime);
gettimeofday(&curT, NULL);
vTaskDelay(0/portTICK_PERIOD_MS);

}
}
```

```
/*****************************************************************************************/
/*NOTES: task actions are given in requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void IRAM_ATTR Task3  (void *parameter)
{
  int SchedTime;
while(1){
    //struct timevalstart_time, curT;
gettimeofday(&start_time, NULL);
    //Serial.println(__func__);
    //disableCore0WDT();
   // disableCore1WDT();
    //disableLoopWDT();
feedTheDog();
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;
      }
    }

feedTheDog();
if(ButtonPressed){
feedTheDog();
      //RGB (Green)
RGB_Change_Color (0, 255, 0);
feedTheDog();
      // Stepper motor rotate CW
myStepper.step(Steps_Per_Revolution);
feedTheDog();
      //delay for 5 s
vTaskDelay(5000/portTICK_PERIOD_MS);
      //Before Rotating Stepper change colour to blue
RGB_Change_Color(0, 0, 255);
feedTheDog();
      // Rotate Stepper Back
myStepper.step(-Steps_Per_Revolution);
feedTheDog();
      }
else{
RGB_Change_Color(0, 0, 0);
      }
feedTheDog();
SchedTime = start_time.tv_sec - curT.tv_sec + start_time.tv_usec - curT.tv_usec;
gettimeofday(&curT, NULL);
Serial.printf("Task 3 Start %lu  Stop %lu  Time %d SchedTime %d\r\n", start_time.tv_sec*1000000  +
start_time.tv_usec ,curT.tv_sec*1000000  + curT.tv_usec ,(curT.tv_sec - start_time.tv_sec)*1000000
+ (curT.tv_usec - start_time.tv_usec),SchedTime);
gettimeofday(&curT, NULL);
```

```
  }
}
/*****************************************************************************************/
/*NOTES: task actions are given in requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
/*****************************************************************************************/
/*NOTES: task actions are given in requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void IRAM_ATTR Task4  (void *parameter)
{
  int SchedTime;
while(1){
    //struct timevalstart_time, curT;
gettimeofday(&start_time, NULL);

RunIndicator();
    //Check if received data is available and if yes, try to decode it.
    //Decoded result is in the receiver. decoded data 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);
SchedTime = start_time.tv_sec - curT.tv_sec + start_time.tv_usec - curT.tv_usec;
gettimeofday(&curT, NULL);
Serial.printf("Task 4 Start %lu  Stop %lu  Time %d SchedTime %d\r\n", start_time.tv_sec*1000000  +
start_time.tv_usec ,curT.tv_sec*1000000  + curT.tv_usec ,(curT.tv_sec - start_time.tv_sec)*1000000
+ (curT.tv_usec - start_time.tv_usec),SchedTime);
gettimeofday(&curT, NULL);
  }
}
/*****************************************************************************************/
/*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();
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);
     }
        }
}
/*****************************************************************************************/
/*NOTES: Function to print a string on display                                           */
/*INPUTS   : Display Strting                                                             */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void displayString(String str,intdisplaytime){
displayclear();
  if(str.length() > 0) {
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 10, str);
   // Display it on the screen
display.display();
vTaskDelay(displaytime*1000/portTICK_PERIOD_MS);
  }
};

void displayclear()
{
for(int i=0;i<5000000;i++);
display.init();
display.display();
 }
void feedTheDog(){
 /* // feed dog 0
  TIMERG0.wdt_wprotect=TIMG_WDT_WKEY_VALUE; // write enable
  TIMERG0.wdt_feed=1;                       // feed dog
  TIMERG0.wdt_wprotect=0;                   // write protect
  // feed dog 1
  TIMERG1.wdt_wprotect=TIMG_WDT_WKEY_VALUE; // write enable
TIMERG1.wdt_feed=1;                       // feed dog
  TIMERG1.wdt_wprotect=0;                   // write protect*/
}
```

II. Bare-Metal

FirstESP32_BareMetal.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"
/*****************************************************************************************/
/*                          INTERNAL DEFINES                                             */
/*****************************************************************************************/

/*****************************************************************************************/
unsigned long start_time, curT;
/*****************************************************************************************/
/*                          OBJECTS                                                      */
/*****************************************************************************************/

// create SerialBT object to control Bluetooth connection
//create serial Bluetooth terminal to send and receive commands
Bluetooth serial 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_NeoPixelstrip(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);

/*****************************************************************************************/
/*                          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 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 displayString   (String str, int displaytime);
void feedTheDog   (void);
```

```
void displayclear   (void);
/*********************************************************************************************/



void setup()
{
Serial.begin(115200);
  /*********************************************************************************************/
  /*                          CONFIGURATIONS                                                   */
  /*********************************************************************************************/
  //call the initialization function
StatusIndicator_init();

Init_Task();
Serial.println("setup done");

}

void loop()
{
  //int Start = millis();
  //Bluetooth_Task(NULL);
  //Task1(NULL);
  //Task2(NULL);
  //Task3(NULL);
  Task4(NULL);

 }
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();
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",0.5);
delay(1000);
displayclear();
  //delay(1000);
}

/*****************************************************************************************/
/*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 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 connects to a Bluetooth and gets 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);
delay(100);
  }
}
/*****************************************************************************************/
/*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)
  {
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
delay(500);
   //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
delay(500);
  }

}
/*****************************************************************************************/
/*NOTES: task actions are given in requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void IRAM_ATTR Task2  (void *parameter)
{
   int schedTime;
  //while(1)
  {
   //unsigned long start_time, curT;
start_time = micros();
   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);
delay(50);
     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);
   }
delay(100);
schedTime = start_time-curT;
curT = micros();
Serial.printf("Task 2 start_time %lu  Stop %luExecTime %d Schd_Time %d\r\n", start_time ,curT ,
curT - start_time, schedTime );
curT = micros();
  }
}

/*****************************************************************************************/
/*NOTES: task actions are given in requirements                                          */
/*INPUTS   : No INPUTS                                                                   */
/*RETURNES : NO OUTPUTS                                                                  */
/*****************************************************************************************/
void  IRAM_ATTR Task3  (void *parameter)
{
  int schedTime;

start_time = micros();
  //while(1)
  {
//disableCore0WDT();
   // disableCore1WDT();
    //disableLoopWDT();
feedTheDog();
RunIndicator();
   bool ButtonPressed = false;
RunIndicator();
   //check if the button1 is pressed or not
   if(digitalRead(Push_Button1) == 1)
   {
 // to handel debouncing
delay(10);
     if(digitalRead(Push_Button1) == 1)
     {
       //update flage state to indicate its pressed
ButtonPressed = true;
     }
   }

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

feedTheDog();
     //delay for 5 s
delay(5000);
     //Before Rotating Stepper change colour to blue
RGB_Change_Color(0, 0, 255);
feedTheDog();
     // Rotate Stepper Back
```

```
myStepper.step(-Steps_Per_Revolution);
feedTheDog();
    }
else{
RGB_Change_Color(0, 0, 0);
    }
feedTheDog();

    //enableCore0WDT();
    //enableCore1WDT();
    //enableLoopWDT();

 }
schedTime = start_time-curT;
curT = micros();
Serial.printf("Task 3 start_time %lu  Stop %luExecTime %d Schd_Time %d\r\n", start_time ,curT , curT - start_time, schedTime );
curT = micros();
}

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

void IRAM_ATTR Task4  (void *parameter)
{
 int schedTime;
 //while(1)
 //unsigned long start_time, curT;
start_time = micros();
 {
RunIndicator();
   //Check if received data is available and if yes, try to decode it.
   //Decoded result is in the receiver. decoded data 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
delay(1000);
motor.brake(motor1);
delay(500);
   }
delay(50);
schedTime = start_time-curT;
curT = micros();
Serial.printf("Task 4 start_time %lu  Stop %luExecTime %d Schd_Time %d\r\n", start_time ,curT , curT - start_time, schedTime );
curT = micros();

 }
```

```
}

/*****************************************************************************************/
/*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();
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);
delay(500);
digitalWrite(BT_TSK_LED,LOW);
delay(500);
     }
    }
   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);
delay(100);
digitalWrite(BT_TSK_LED,LOW);
delay(100);
     }

   }
}
/*****************************************************************************************/
/*NOTES    : Function to print string on display                                        */
/*INPUTS   : Display Strting                                                            */
/*RETURNES : NO OUTPUTS                                                                 */
/*****************************************************************************************/

void displayString(String str,intdisplaytime){
displayclear();
  if(str.length() > 0) {
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.drawString(0, 10, str);
   // Display it on the screen
display.display();
   delay(displaytime*1000);
  }
};

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

```
 /* // feed dog 0
TIMERG0.wdt_wprotect=TIMG_WDT_WKEY_VALUE; // write enable
  TIMERG0.wdt_feed=1;                       // feed dog
  TIMERG0.wdt_wprotect=0;                   // write protect
  // feed dog 1
  TIMERG1.wdt_wprotect=TIMG_WDT_WKEY_VALUE; // write enable
  TIMERG1.wdt_feed=1;                       // feed dog
  TIMERG1.wdt_wprotect=0;                   // write protect*/
}
```

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 frequancy for MI
#define BUZZER_Resoluation12  //pwmresoluation
#define BUZZER_Frequency        1000//pwmfrequancy


//** 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
```

V. Conclusion

There are many reasons why a developer should plan to use an RTOS. In many circumstances, a basic bare-metal scheduler will do average enough to perform basic tasks. In connected and IoT applications, where complex connectivity stacks need to be included, beginning with an RTOS. Many modern embedded systems that are becoming to be internet-connected have more complex scheduling needs than a typical stand-alone system. A few example components include file systems, USB, TCP/IP, and GUI components just to call a few of. Attempting to integrate these components into a bare-metal system wouldn't only be time-consuming but

would be a nightmare to the standard developer. There are times when a software application must stop what it's doing and concentrate on a special task or activity. For a bare-metal scheduler, the only way that this may happen is that if an interrupt is used then the code returns to where it had been previously executing. An RTOS is supposed to allow each task to tend a priority that helps confirm that the task is executed in a deterministic manner. A bare-metal scheduler is often designed to mimic preemption but during a real real-time system, the important deal is required which may only be found in an RTOS. Every RTOS comes with a way to make tasks, destroy tasks, synchronous tasks, communicate between tasks, and lock resources. Some RTOS may even allow data pools to be created for various application purposes. In a bare-metal implementation, if dedicated APIs are necessary then the developer is required to create and test them, this leads to a bug-prone code base and costly development cycle in terms of time as well as money. Re-inventing the wheel not only costs time but also comes with validation and maintenance issues.

From the above charts, we'll conclude that there is no much difference in scheduling and execution time when tasks remained ideal but when input is given we'll see the spikes in time of execution and when the task is performed any Input-Output and CPU is free it is often utilized to schedule another task. FreeRTOS allows modularization where are Bare Metal doesn't allow such changes. From the above chart, we'll see the responsiveness of the execution of the task where as soon as Input is given the execution starts, but in Bare Metal the responsiveness is often seen slightly less with low spikes in charts.

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

EE.Ahmad A A Alsaleh. "Analysis of Free RTOS Vs Bare Metal using ESP32." *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)*, 16(2), (2021): pp. 39-68.