Documentación del proyecto ESP32 con GSR

En el presente documento se explica a detalle de que consta el código con imágenes sobre el funcionamiento del proyecto realizado con ESP32 el cual debe de dar descargas eléctricas cuando el nivel de estrés en el cuerpo del paciente sea bajo y disminuir las descargas eléctricas cuando el nivel de estrés en el usuario sea alto por medio de un control PID aplicado a un motor con potenciómetro, el cual moverá el potenciómetro que aumente o disminuya las descargas además de utilizar un servidor web donde se puede visualizar en tiempo real el sensor y la tabla donde se estiman los valores de acuerdo a un ADC de 8 bits.

```
// Configuración de red
const char* ssid = "Luisifer_2.4Gnormal";
const char* password = "cradleoffilth21";

// Configuración de hardware
const int GSR = 32;           // Sensor GSR (entrada de referencia)
const int encoder_pot = 35;    // Sensor de feedback (posición)
const int PWMPin = 33;        // Pin PWM para el motor
const int DirPin1 = 27;       // Dirección motor pin 1
const int DirPin2 = 14;       // Dirección motor pin 2
const int rs = 15, en = 2, d4 = 4, d5 = 16, d6 = 17, d7 = 5; // Pines LCD

// Objetos globales
WebServer server(80);
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

Lo que se encuentra encerrado de rojo se modifica con los datos de tu red móvil o red WiFi, el ssid es el nombre exacto de tu modem o red celular y el password es la contraseña.

```
// Declaraciones de funciones
void wifiTask(void *pvParameters);
void sensorTask(void *pvParameters);
void pidTask(void *pvParameters);
void motorTask(void *pvParameters);
void lcdTask(void *pvParameters);
void serverTask(void *pvParameters);
String sensorReading();
void initializeLittleFS();
void initializeLCD();
void connectToWiFi();
void initializeWebServer();
void WriteDriverVoltage(float V, float Vmax);
int scaleTo1023(int value);
```

Lo que esta encerrado en color rojo son las funciones empleadas en la configuración FreeRTOS para multitareas, las funciones abajo son para configuración y se mandan a llamar en el setup.

```ino
// Función para escalar ADC de 0-4095 a 0-1023
int scaleTo1023(int value) {
  return map(value, 0, 4095, 0, 1023);
}

// Función para controlar el motor
void WriteDriverVoltage(float V, float Vmax) {
  int PWMval = int(255 * abs(V) / Vmax);
  if (PWMval > 255) PWMval = 255;

  if (V > 0) {
    digitalWrite(DirPin1, HIGH);
    digitalWrite(DirPin2, LOW);
  }
  else if (V < 0) {
    digitalWrite(DirPin1, LOW);
    digitalWrite(DirPin2, HIGH);
  }
  else {
    digitalWrite(DirPin1, LOW);
    digitalWrite(DirPin2, LOW);
  }
  analogWrite(PWMPin, PWMval);
}
```

La función scaleTo1023 mapea el ADC de 10bits a un ADC de 8 bits ya que los valores se manejan de 0 a 1023, la función WriteDriverVoltage sirve para aumentar o disminuir el PWM para la velocidad del motor, así como la inversión de giro del motor.

```ino
String sensorReading() {
  long sum = 0;
  for (int i = 0; i < 10; i++) {
    int sensorValue = analogRead(GSR);
    if (sensorValue == 0 || sensorValue == 4095) {
      Serial.println("Error lectura sensor");
      return "ERROR";
    }
    sum += sensorValue;
    delay(5);
  }
  int gsr_average = sum / 10;
  return String(scaleTo1023(gsr_average)); // Convertir a 0-1023
}

void initializeLCD() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("Iniciando sistema");
  delay(1000);
}
```

La función sensorReading sirve para medir el sensor GSR además de parametrizar la medición del sensor, la función initializeLCD es para setear los valores de la pantalla LCD e inicializarla.

```
void initializeLittleFS() {
  if (!LittleFS.begin()) {
    Serial.println("Error con LittleFS");
    lcd.clear();
    lcd.print("Error LittleFS");
    while (1) delay(1000);
  }
}
```

Esta función esta encargada de arrancar el archivo “index.html” cargado previamente en la ESP32, ese archivo contiene el frontend del servidor web.

```
void connectToWiFi() {
  // Configuración de red
  esp_netif_init();
  esp_wifi_set_ps(WIFI_PS_NONE);
  WiFi.setSleep(false);

  // Conexión WiFi
  WiFi.begin(ssid, password);
  lcd.clear();
  lcd.print("Conectando WiFi...");

  unsigned long startTime = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - startTime < 30000) {
    delay(500);
    Serial.print(".");
    static int dots = 0;
    lcd.setCursor(0, 1);
    lcd.print("                ");
    lcd.setCursor(dots % 6, 1);
    lcd.print(".");
    dots++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    wifiConnected = true;
    lcd.clear();
    lcd.print("WiFi Conectado");
    lcd.setCursor(0, 1);
    lcd.print("IP: ");
    lcd.print(WiFi.localIP());
    delay(2000);
  } else {
    lcd.clear();
    lcd.print("Error WiFi");
    lcd.setCursor(0, 1);
    lcd.print("Reiniciando...");
    delay(2000);
    ESP.restart();
  }
}
```

La función connectToWiFi esta dedicada a la conexión de la ESP32 a lo que seria el modem o la red móvil definida en el código, lleva manejo de errores para asegurar la correcta conexión al WiFi.

```
void initializeWebServer() {
  server.on("/", []() {
    if (LittleFS.exists("/index.html")) {
      File file = LittleFS.open("/index.html", "r");
      server.streamFile(file, "text/html");
      file.close();
    } else {
      server.send(200, "text/plain", "Archivo no encontrado");
    }
  });

  server.on("/humanResistance", []() {
    String reading;
    if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
      reading = String(gsrValue);
      xSemaphoreGive(xMutex);
    }
    server.send(200, "text/plain", reading);
  });
  server.begin();
}
```

La función initializeWebServer es la encargada de generar las rutas y generar las peticiones HTTP necesarias para lo que seria el servidor web, asimismo esta configurada con los parámetros de multitarea para poder configurar el servidor de manera correcta.

```
// Tarea WiFi
void wifiTask(void *pvParameters) {
  connectToWiFi();
  initializeWebServer();

  for (;;) {
    if (WiFi.status() != WL_CONNECTED) {
      wifiConnected = false;
      connectToWiFi();
    }
    vTaskDelay(10000 / portTICK_PERIOD_MS);
  }
}

// Tarea Sensor
void sensorTask(void *pvParameters) {
  for (;;) {
    int gsr = sensorReading().toInt();
    int encoder = scaleTo1023(analogRead(encoder_pot));

    if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
      gsrValue = gsr;
      encoderValue = 1023 - encoder; // Relación inversa
      xSemaphoreGive(xMutex);
    }
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

Estas funciones son parte de las multitareas con FreeRTOS, la primera es para consultar si la ESP32 no se ha desconectado del WiFi o red móvil, la segunda esta encargada de leer el sensor, convertirlo en un numero entero y convertirla a la escala de ADC de 8 bits.

```
// Tarea PID
void pidTask(void *pvParameters) {
  for (;;) {
    if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
      unsigned long t = millis();
      int dt = t - t_prev;

      if (dt > 0) {
        float Theta = encoderValue;
        float Theta_d = gsrValue;
        float e = Theta_d - Theta;
        float inte = inte_prev + (dt * (e + e_prev) / 2);

        // Cálculo PID
        float V = kp * e + ki * inte + (kd * (e - e_prev) / dt);

        // Anti-windup
        if (V > Vmax) {
          V = Vmax;
          inte = inte_prev;
        }
        else if (V < Vmin) {
          V = Vmin;
          inte = inte_prev;
        }

        motorVoltage = V;
        t_prev = t;
        e_prev = e;
        inte_prev = inte;
      }
      xSemaphoreGive(xMutex);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

La tarea del PID se encarga de realizar un control en lazo cerrado de la posición de un motor encargado de aumentar o disminuir la intensidad de corriente de un dispositivo de descargas eléctricas con base en la medición sensada a través del sensor GSR, tiene un anti windup para evitar la sobrecarga de la acción integradora.

```
// Tarea Motor
void motorTask(void *pvParameters) {
  pinMode(DirPin1, OUTPUT);
  pinMode(DirPin2, OUTPUT);
  pinMode(PWMPin, OUTPUT);

  for (;;) {
    if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
      WriteDriverVoltage(motorVoltage, Vmax);
      xSemaphoreGive(xMutex);
    }
    vTaskDelay(20 / portTICK_PERIOD_MS);
  }
}
```

La siguiente tarea esta diseñada para llamar la función encargada de la velocidad e inversión de giro del motor además de configurar los pines necesarios para el correcto funcionamiento del motor.

```
// Tarea LCD
void lcdTask(void *pvParameters) {
  for (;;) {
    int valor, encoder;
    float voltage;

    if (xSemaphoreTake(xMutex, portMAX_DELAY) == pdTRUE) {
      valor = gsrValue;
      encoder = encoderValue;
      voltage = motorVoltage;
      xSemaphoreGive(xMutex);
    }

    String estado;
    if (valor == 0) estado = "Error Sensor";
    else if (valor < 60) estado = "Fuera Rango";
    else if (valor < 100) estado = "Relajacion";
    else if (valor < 300) estado = "Neutro";
    else if (valor < 500) estado = "Estres Leve";
    else if (valor < 700) estado = "Estres Alto";
    else estado = "Dolor";

    lcd.clear();
    lcd.print("GSR:");
    lcd.print(valor);
    lcd.print(" E:");
    lcd.print(encoder);

    lcd.setCursor(0, 1);
    lcd.print(estado);
    lcd.print(" ");
    lcd.print(voltage, 1);
    lcd.print("V");

    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```

Esta función esta diseñada para leer los valores de posición del motor, de voltaje del motor y los valores medidos por el sensor GSR además de imprimir en la LCD información acorde a lo sensado.

```
// Tarea Servidor
void serverTask(void *pvParameters) {
  for (;;) {
    server.handleClient();
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

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

  // Inicializar LCD
  initializeLCD();
  Serial.println("\nIniciando...");

  // Inicializar LittleFS
  initializeLittleFS();

  // Crear mutex
  xMutex = xSemaphoreCreateMutex();

  // Crear tareas
  xTaskCreatePinnedToCore(wifiTask, "WiFiTask", 4096, NULL, 1, NULL, 0);
  xTaskCreatePinnedToCore(sensorTask, "SensorTask", 2048, NULL, 3, NULL, 1);
  xTaskCreatePinnedToCore(pidTask, "PIDTask", 2048, NULL, 2, NULL, 1);
  xTaskCreatePinnedToCore(motorTask, "MotorTask", 2048, NULL, 2, NULL, 1);
  xTaskCreatePinnedToCore(lcdTask, "LCDTask", 2048, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(serverTask, "ServerTask", 4096, NULL, 1, NULL, 0);

  vTaskDelete(NULL);
}

void loop() {}
```

Las ultimas tareas están diseñadas para manejar el servidor web, para configurar todas las tareas empleadas en FreeRTOS asi como en que procesador se estaranejecutando dichas tareas, debido al estilo de programación multitarea no es necesario ejecutar líneas de código en el void loop.