Universidad Industrial de Santander

Legado académico y cultural de los santandereanos

# Diseño y Mejoras µCurrent

Brayan Andres Celis Godoy - 2191799
Jeiffer Bernal Tellez - 2194679

#LaUISqueQueremos

# OBJETIVOS

## Problema:

- La falta de herramientas accesibles y precisas para medir corriente en dispositivos de bajo consumo, como microcontroladores ESP32, afecta directamente la optimización de dispositivos IoT.
- Los modos operativos en microcontroladores varían desde microamperios (modo sueño) hasta miliamperios (modo activo), lo que requiere un sistema de medición versátil y de alta precisión.

## Objetivo general:

Diseñar un sistema de medición que optimice la eficiencia energética y permita transiciones automáticas entre rangos.

## Importancia:

Prolongar la vida útil de las baterías y mejorar el rendimiento de dispositivos portátiles o embebidos en aplicaciones IoT.


# OBJETIVOS ESPECÍFICOS

1.Diseño y simulación del circuito de medición:

Crear un modelo que permita verificar y optimizar las mediciones de corriente.

Circuito de medición, sistema automático, alimentación.

LTspice®

2.Desarrollo de visualización:

Proporcionar herramientas para monitoreo remoto a través de aplicaciones web y dedicadas.

ESP32 para adquisición y envío de datos.

3.Diseño del PCB:

Integrar todos los componentes seleccionados en un diseño compacto, económico y eficiente.

Diseño del PCB y análisis de costos.

EasyEDA

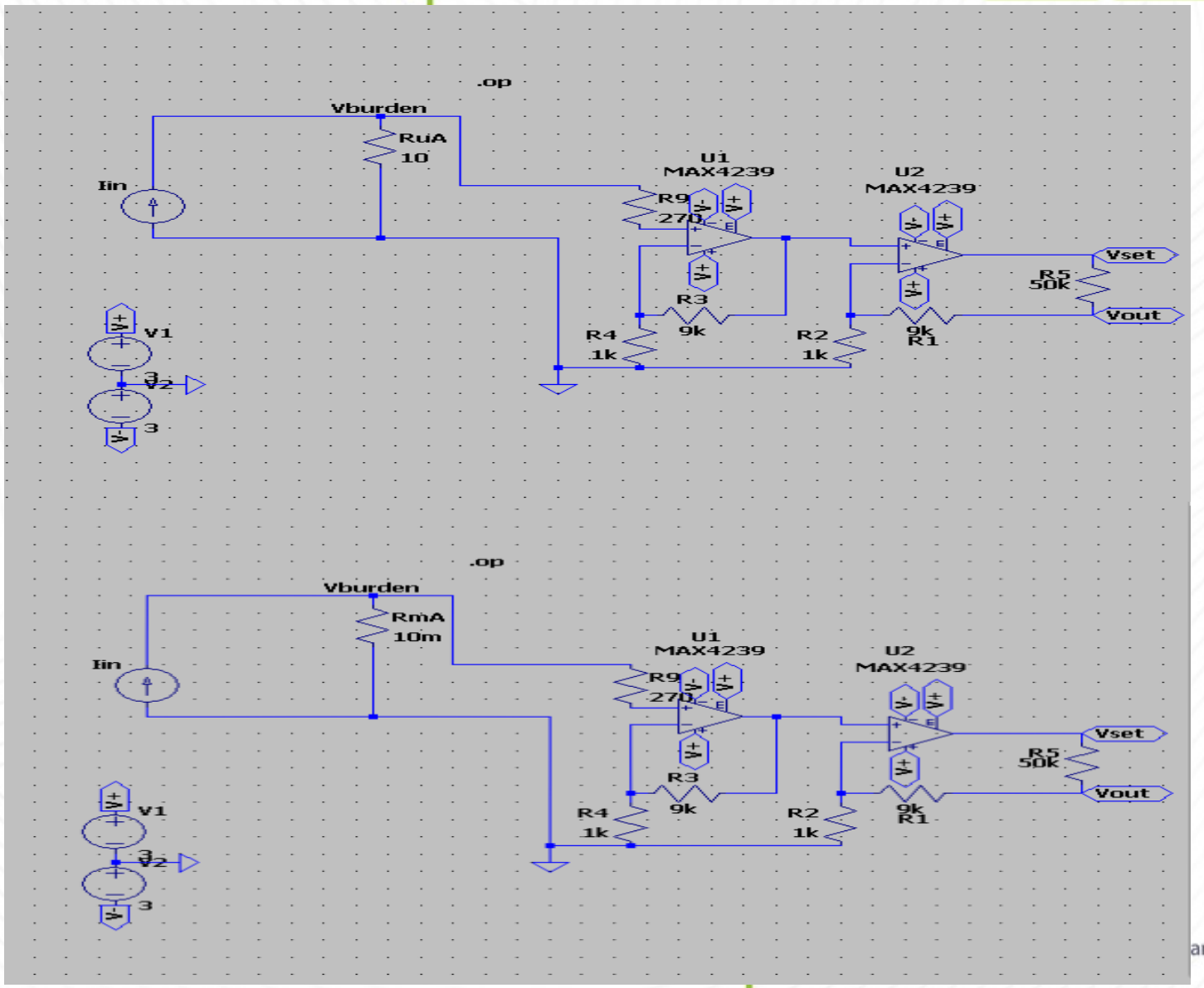
# Solución Técnica

## Diseño Base del Sistema de Medición:

- El sistema utiliza una combinación de resistencias shunt de baja impedancia (10 mΩ y 10 Ω) para medir corriente de manera precisa en dos rangos: miliamperios y microamperios.

## Amplificadores Operacionales:

- Los MAX4239 se emplean para amplificar las señales obtenidas de las resistencias shunt, asegurando linealidad y precisión en la medición.
- Estos amplificadores tienen un ruido bajo, un offset mínimo y una alta precisión, características fundamentales para evitar distorsiones en las mediciones.



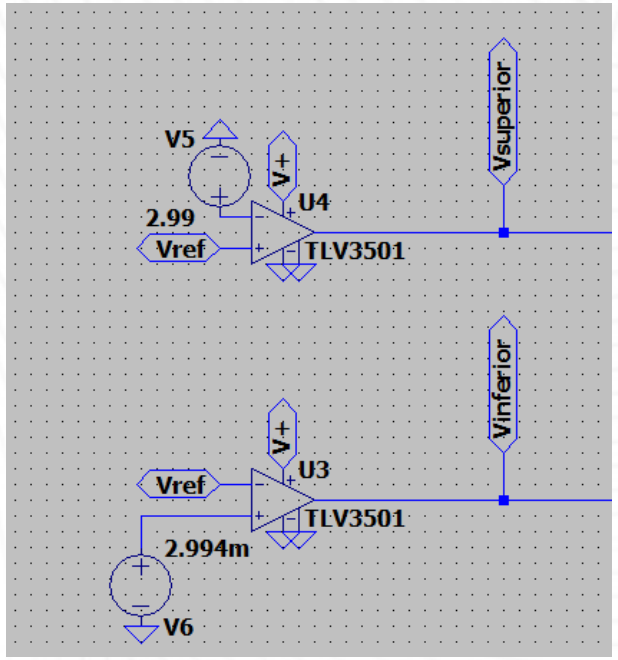
# Solución Técnica

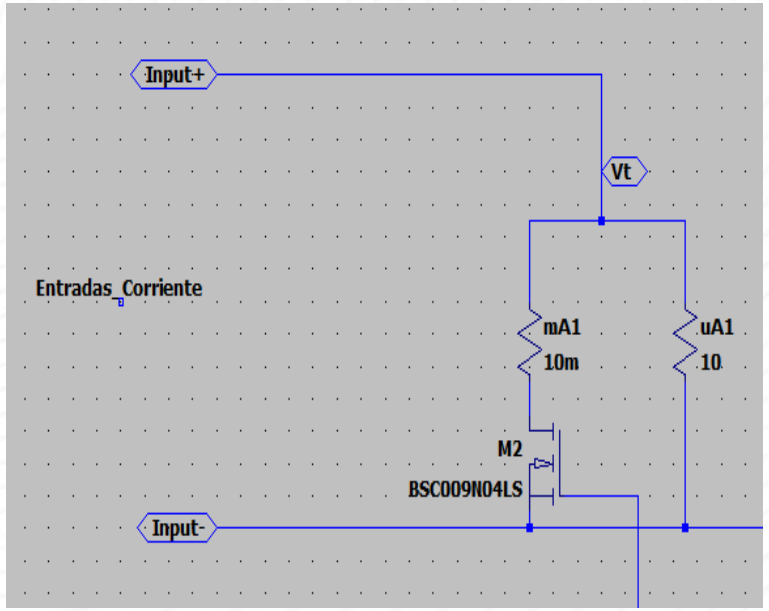
## MOSFETs de Baja Rds(on):

Utilizados para seleccionar entre las resistencias shunt dependiendo del rango de corriente, minimizando las pérdidas de potencia y asegurando eficiencia energética.

## Comparadores TLV3501:

Encargados de detectar automáticamente los cambios en los niveles de corriente, activando los MOSFETs según sea necesario. Su velocidad de respuesta rápida garantiza la transición sin errores.

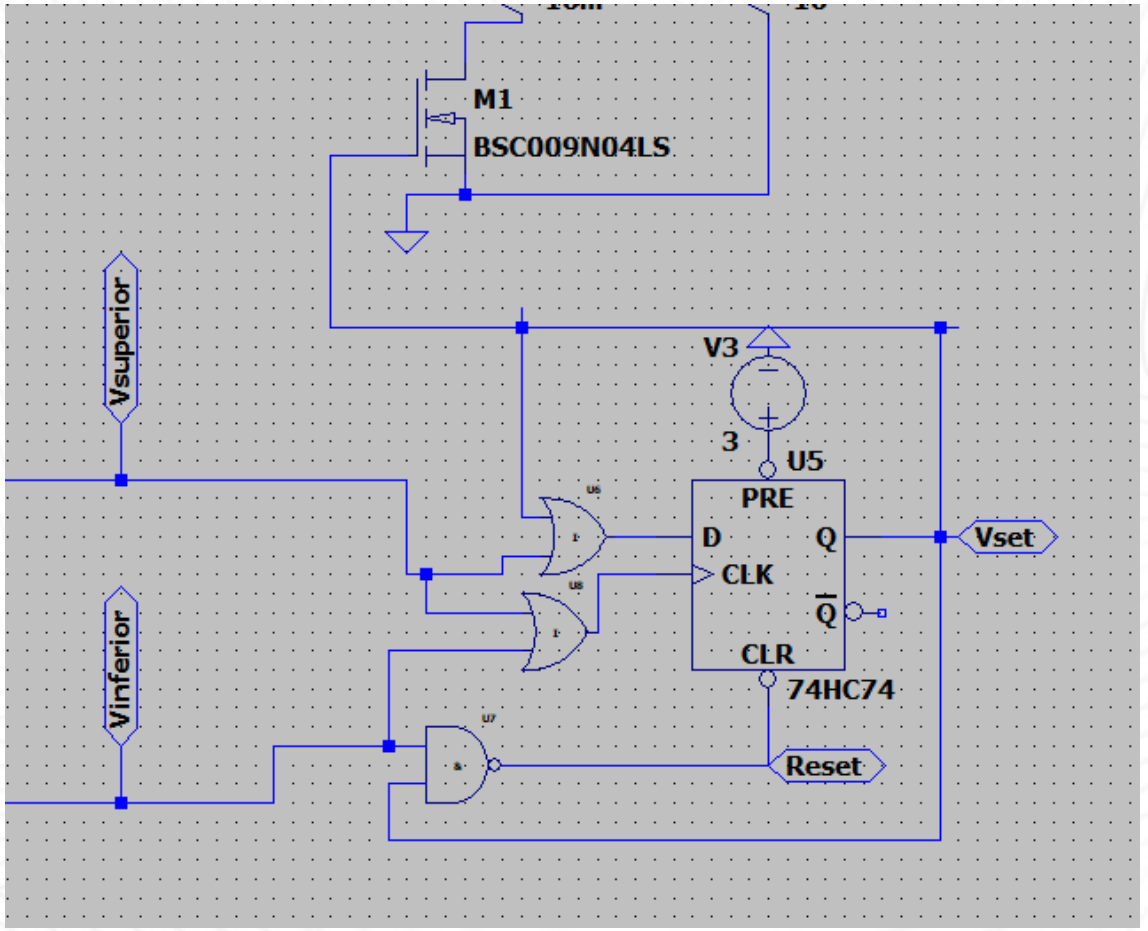


# Solución Técnica

## Rango Automático de Corriente:

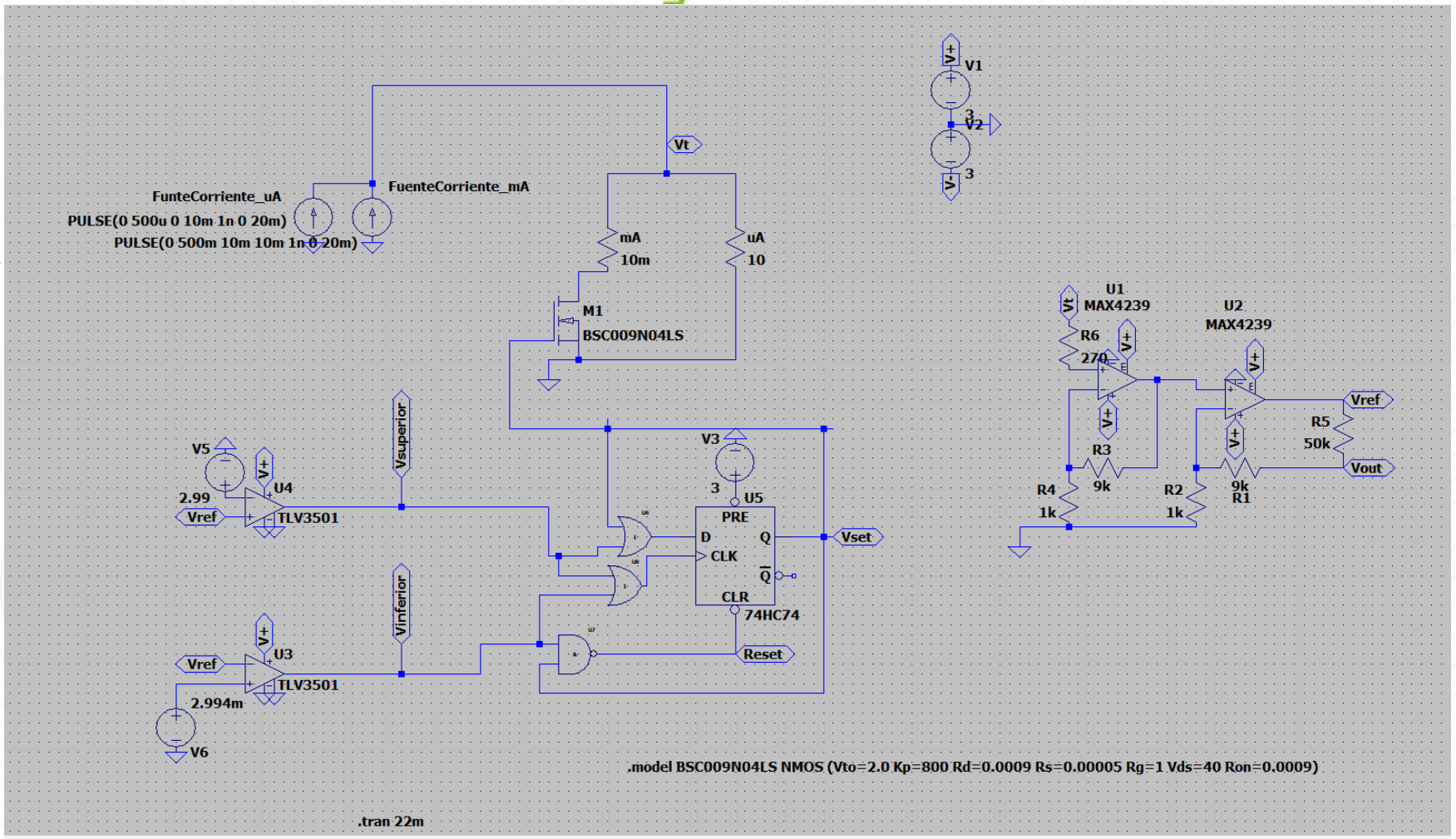
El sistema alterna automáticamente entre los rangos de microamperios y miliamperios, eliminando la necesidad de intervención manual. Esto simplifica su uso en aplicaciones prácticas y reduce el riesgo de errores operativos.



# Solución Técnica

## Resultados en Simulaciones:

Simulaciones realizadas en LTspice validan la funcionalidad del circuito en ambos rangos de corriente, mostrando una salida estable de 500 mV y un Burden Voltage mínimo, lo que garantiza precisión.

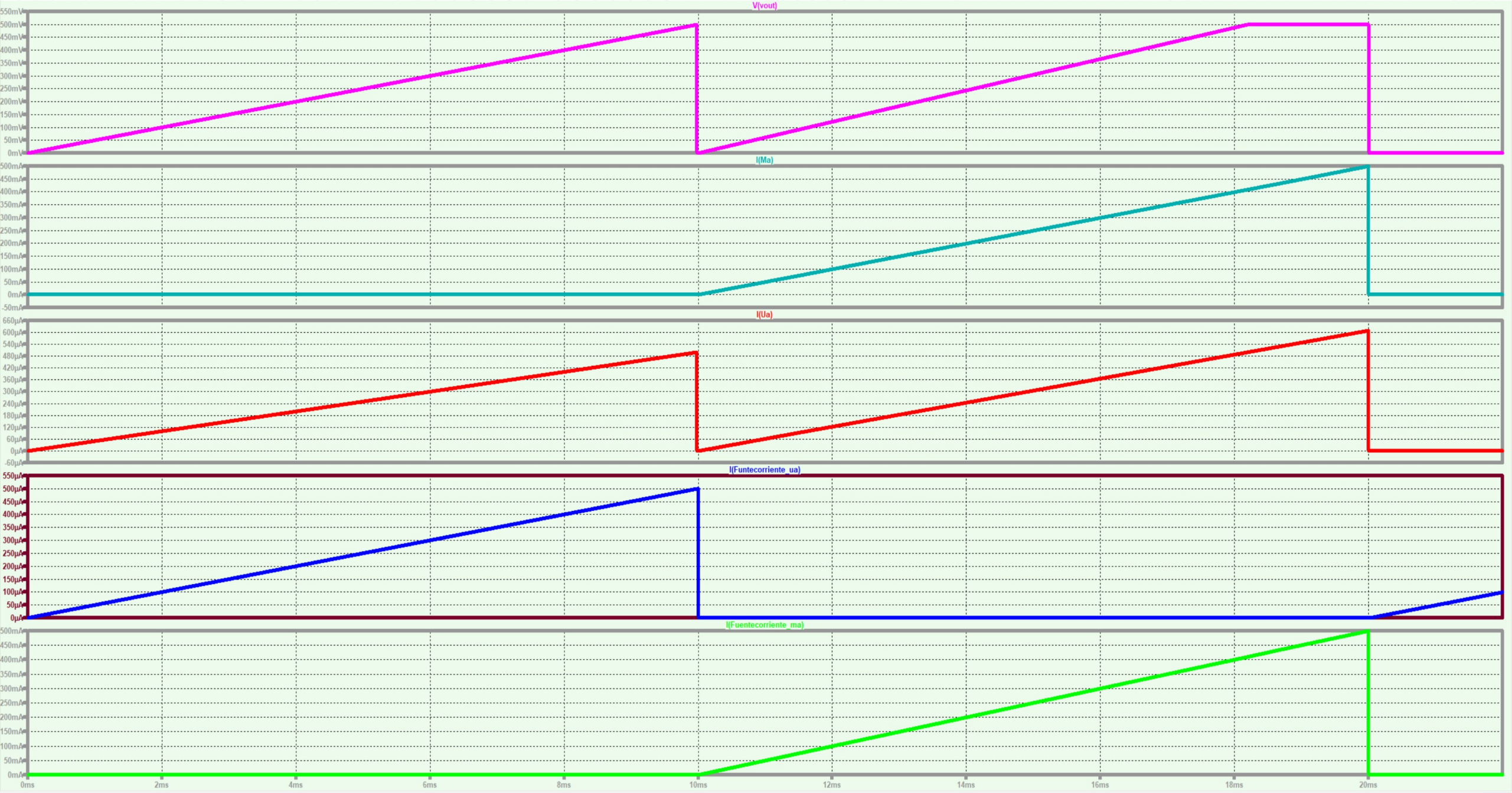


# Comparativa de elementos

| Característica | 74HC74 | SN74LV74 | SN74AUC7 | MC74VH1H74 |
|---|---|---|---|---|
| Vcc | 2 - 6V | 1.8 - 3.6V | 0.8 - 2.7V | 1.65 - 5.5V |
| Corriente | 20 µA | 5 µA | <5 µA | 1 µA |
| Delay | 75 ns at 3V | 10 ns | 1.8 ns | 3 ns |
| Precio | 0.2 | 0.28 | 0.71 | 0.15 |

| Características | TLV3601 | LTC6752 | AD790 | TLV3501 |
|---|---|---|---|---|
| Propagacion tipica | 2.5 ns | 2.9 ns | 4 ns | 4.5 ns |
| Vcc | 2.7 - 5.5 V | 2.7 - 5.5 V | 2.7 - 5.5 V | 2.7 - 5.5 V |
| Corriente de reposo | 4.5 mA | 2.9 mA | 1 mA | 1.8 mA |
| Frecuencia maxima | 140 MHz | 100 MHz | 50 MHz | 80 MHz |
| Precio | 4.02 | 6.54 | 14.51 | 3.6 |

| caracteristicas | TLV333 | OPA391 | AD8538 | MCP6V01 | LTC6078 | MAX4239 |
|---|---|---|---|---|---|---|
| Offset tipico | 3 µV | 5 µV | 1 mV | 2 µV | 5 µV | 2 µV |
| Deriva de offset | 0.02 µV/°C | 0.05 µV/°C | 3 µV/°C | 0.6 µV/°C | 0.2 µV/°C | 0.02 µV/°C |
| Ruido de entrada(0.1-10hz) | 1.1 µV_pp | 1 µV_pp | 0.5 µV_pp | 1.6 µV_pp | 0.9 µV_pp | 1.1 µV_pp |
| Corriente Reposo | 17 µA | 12 µA | 45 µA | 115 µA | 55 µA | 1 mA |
| Vcc | 1.8 - 5.5 V | 1.7 - 5.5 V | 2.7 - 5.5 V | 1.8 - 5.5 V | 2.7 - 5.5 V | 2.85 - 5.5 V |
| Ancho de banda | 350 kHz | 10 MHz | 5 MHz | 1 MHz | 3 MHz | 4 MHz |
| Precio | $5.435 | $10.918 | $11.061 | $10.775 | $41.669 | $20.167 |


V(vout)
550mV
500mV
450mV
400mV
350mV
300mV
250mV
200mV
150mV
100mV
50mV
0mV
I(Ma)
500mA
450mA
400mA
350mA
300mA
250mA
200mA
150mA
100mA
50mA
0mA
-50mA
I(Ua)
660µA
600µA
540µA
480µA
420µA
360µA
300µA
240µA
180µA
120µA
60µA
0µA
-60µA
I(Funtecorriente_ua)
550µA
500µA
450µA
400µA
350µA
300µA
250µA
200µA
150µA
100µA
50µA
0µA
I(Fuentecorriente_ma)
500mA
450mA
400mA
350mA
300mA
250mA
200mA
150mA
100mA
50mA
0mA
0ms
2ms
4ms
6ms
8ms
10ms
12ms
14ms
16ms
18ms
20ms

# Visualización

## Servidor Web:

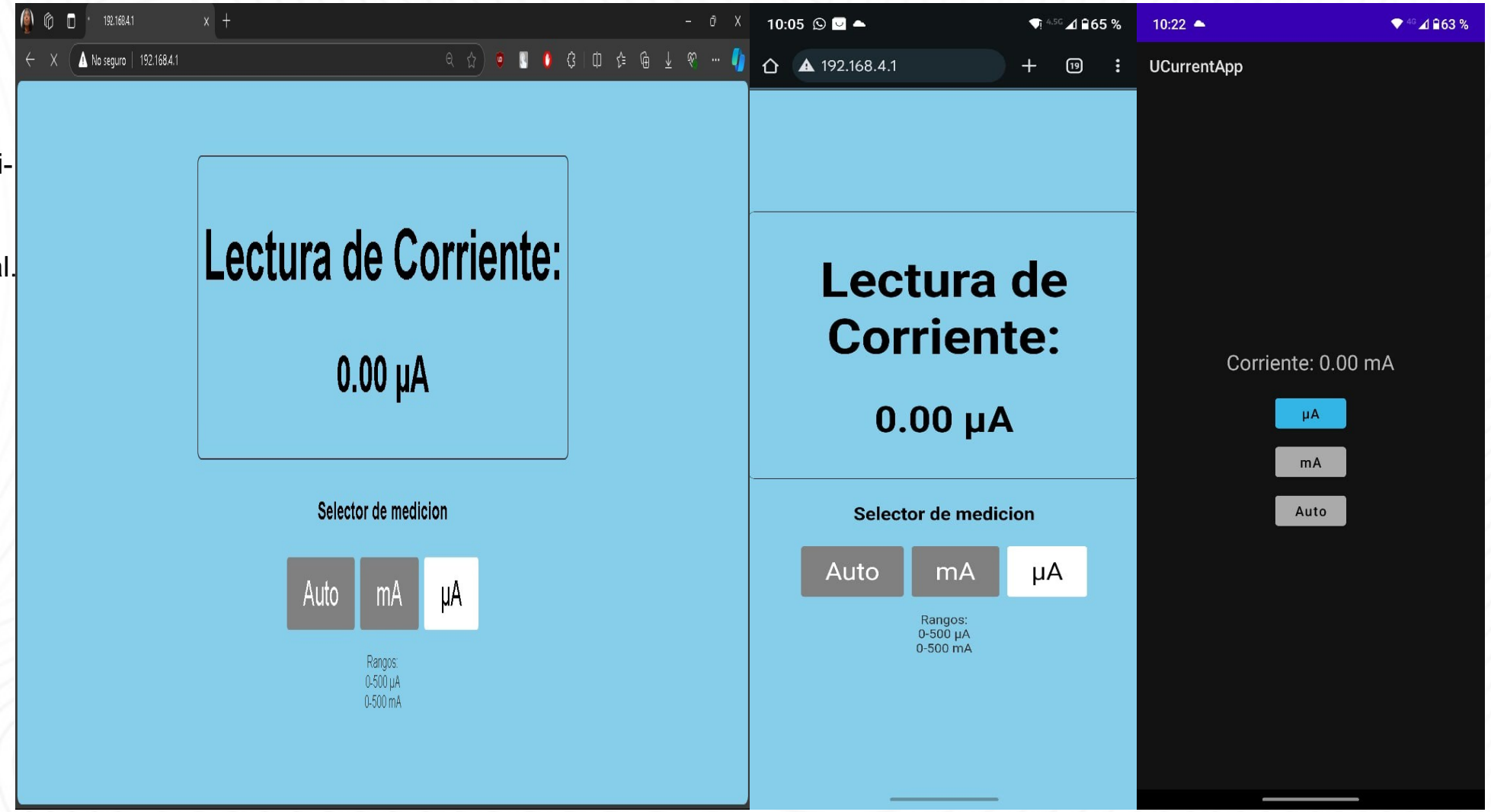
- Proporciona una interfaz accesible desde cualquier navegador.
- La ESP32 genera una red Wi-Fi que permite conectar dispositivos para monitorear las mediciones en tiempo real.
- Tecnología utilizada: HTML y comunicación HTTP para transmisión de datos.

## Aplicación Dedicada:

Mayor personalización y control del sistema.

Desarrollada para dispositivos móviles y PC, con procesamiento eficiente y visualización intuitiva.

Tecnología utilizada: comunicación basada en endpoints con el ESP32.



# SERVIDOR WEB

```cpp
#include <WiFi.h>
#include <WebServer.h>

const char* ssid = "ESP32_AP";
const char* password = "123456789";

// Crear una instancia del servidor web
WebServer server(80);

// Definir los pines de entrada
const int pinAutoInput = 19;      // Pin para la entrada Auto (GPIO19)
const int pinRangeInput = 5;      // Pin para la entrada de rango (uA/mA) (GPIO5)
const int inputPin = 34;          // Pin 34 como entrada de señal analógica

// Función para leer la corriente sin conversión
float readCurrent() {
  int sensorValue = analogRead(inputPin);
  float voltage = (sensorValue / 4095.0) * 3300.0;  // Convertir a milivoltios (0-3300mV)
  return voltage;
}

// Función para la página principal
void handleRoot() {
  // Leer los estados lógicos de las entradas
  bool isAuto = digitalRead(pinAutoInput);
  bool isMilliAmp = digitalRead(pinRangeInput);  // HIGH = mA, LOW = uA

  // Definir la unidad de medida dependiendo del estado del pin de rango
  String displayUnit = "";
  if (isAuto) {
    displayUnit = "Auto";
  } else if (isMilliAmp) {
    displayUnit = "mA";
  } else {
    displayUnit = "µA";
  }

  float currentValue = readCurrent();  // Leer la corriente

  // Crear el contenido HTML
  String html = "<html><head>";
  html += "<meta charset='UTF-8'>";  // Añadir la codificación UTF-8
  html += "<style>";
  html += "html, body { height: 100%; margin: 0; font-family: Arial; font-size: 1.6em; background-color: #87CEEB; }"; // Fondo celeste
  html += "body { display: flex; justify-content: center; align-items: center; }";
  html += ".container { text-align: center; }";
  html += ".box { border: 1px solid black; display: inline-block; padding: 20px; margin-bottom: 20px; border-radius: 15px; font-size: 1.6em; }";
  html += ".button { padding: 30px 60px; font-size: 1.6em; margin: 10px; background-color: grey; color: white; border: none; cursor: pointer; border-radiu
  html += ".button.selected { background-color: white; color: black; }";  // Botón seleccionado en blanco
  html += ".small-text { font-size: 0.8em; color: #333; }";
  html += "</style>";

  // Añadir JavaScript para auto-actualización
  html += "<script>";
  html += "setInterval(function() { window.location.reload(); }, 1000);";  // Refrescar cada 1 segundo
  html += "</script>";

  html += "</head><body>";

  html += "<div class='container'>";

  html += "<div class='box'><h1>Lectura de Corriente:</h1>";
  html += "<h2>" + String(currentValue, 2) + " " + displayUnit + "</h2></div>";

  html += "<h3>Selector de medicion</h3>";
```

```cpp
  // Crear el contenido HTML
  String html = "<html><head>";
  html += "<meta charset='UTF-8'>";  // Añadir la codificación UTF-8
  html += "<style>";
  html += "html, body { height: 100%; margin: 0; font-family: Arial; font-size: 1.6em; background-color: #87CEEB; }"; // Fondo celeste
  html += "body { display: flex; justify-content: center; align-items: center; }";
  html += ".container { text-align: center; }";
  html += ".box { border: 1px solid black; display: inline-block; padding: 20px; margin-bottom: 20px; border-radius: 15px; font-size: 1.6em; }";
  html += ".button { padding: 30px 60px; font-size: 1.6em; margin: 10px; background-color: grey; color: white; border: none; cursor: pointer; border-radius:
  html += ".button.selected { background-color: white; color: black; }";  // Botón seleccionado en blanco
  html += ".small-text { font-size: 0.8em; color: #333; }";
  html += "</style>";

  // Añadir JavaScript para auto-actualización
  html += "<script>";
  html += "setInterval(function() { window.location.reload(); }, 1000);";  // Refrescar cada 1 segundo
  html += "</script>";

  html += "</head><body>";

  html += "<div class='container'>";

  html += "<div class='box'><h1>Lectura de Corriente:</h1>";
  html += "<h2>" + String(currentValue, 2) + " " + displayUnit + "</h2></div>";

  html += "<h3>Selector de medicion</h3>";

  // Botones con estado visual según las entradas
  html += "<input class='button " + String((isAuto) ? "selected" : "") + "' type='button' value='Auto'>";
  html += "<input class='button " + String((!isAuto && isMilliAmp) ? "selected" : "") + "' type='button' value='mA'>";
  html += "<input class='button " + String((!isAuto && !isMilliAmp) ? "selected" : "") + "' type='button' value='µA'>";

  html += "<p class='small-text'>Rangos:<br>0-500 µA<br>0-500 mA</p>";
  html += "</div>";

  html += "</body></html>";

  server.send(200, "text/html", html);
}

void setup() {
  Serial.begin(115200);
  pinMode(inputPin, INPUT);
  pinMode(pinAutoInput, INPUT);
  pinMode(pinRangeInput, INPUT);

  WiFi.softAP(ssid, password);
  Serial.println("Punto de acceso iniciado");
  Serial.print("IP del ESP32: ");
  Serial.println(WiFi.softAPIP());

  server.on("/", handleRoot);
  server.begin();
}

void loop() {
  server.handleClient();
  if (WiFi.status() == WL_CONNECTED) {
    esp_sleep_enable_timer_wakeup(5000000);  // Configurar para despertar cada 5 segundos
    esp_light_sleep_start();                 // Entrar en light sleep
  }
}
```

# APLICATIVO DEDICADO



```ino
#include <WiFi.h>
#include <WebServer.h>

const char* ssid = "ESP32_AP";
const char* password = "123456789";

// Crear una instancia del servidor web
WebServer server(80);

// Definir el pin de salida para el MOSFET
const int mosfetPin = 18;          // Pin para controlar el MOSFET (GPIO18)

// Variables para guardar el estado del sistema
String mode = "uA";                // Modo actual (uA, mA, Auto)
float currentValue = 0.0;          // Valor de corriente leído
String range = "µA";               // Escala de medición actual (uA o mA)

// Función para leer la corriente sin conversión
float readCurrent() {
  int sensorValue = analogRead(34);  // Pin 34 como entrada de señal analógica
  float voltage = (sensorValue / 4095.0) * 500.0;  // Convertir a milivoltios (0-3300mV)
  return voltage;
}

// Función para procesar el modo automático
void handleAutoMode(float currentValue) {
  if (currentValue >= 495) {
    // Activar el MOSFET cuando la corriente supere los 495mV
    digitalWrite(mosfetPin, HIGH);
    range = "mA";  // Cambiar la escala a miliamperios cuando el MOSFET esté encendido
  } else if (currentValue < 1) {
    // Apagar el MOSFET cuando la corriente sea inferior a 1mV
    digitalWrite(mosfetPin, LOW);
    range = "µA";  // Cambiar la escala a microamperios cuando el MOSFET esté apagado
  }
}

// Función para manejar el cambio de modo desde la aplicación
void handleSetMode() {
  if (server.hasArg("mode")) {
    mode = server.arg("mode");

    // Lógica para los modos
    if (mode == "mA") {
      digitalWrite(mosfetPin, HIGH);  // Encender MOSFET en mA
      range = "mA";                   // Fijar el rango en miliamperios
    } else if (mode == "uA") {
      digitalWrite(mosfetPin, LOW);   // Apagar MOSFET en uA
      range = "µA";                   // Fijar el rango en microamperios
    }
    // No encendemos/apagamos directamente en modo Auto, lo hace handleAutoMode()
  }

  // Responder a la aplicación confirmando el cambio de modo
  server.send(200, "text/plain", "Mode changed to " + mode);
}

// Función para devolver los datos actuales en formato JSON
void handleData() {
  currentValue = readCurrent();  // Leer la corriente

  // Si el modo es "Auto", procesar automáticamente y cambiar la escala
  if (mode == "Auto") {
    handleAutoMode(currentValue);
  }
```

```ino
    digitalWrite(mosfetPin, LOW);    // Apagar MOSFET en uA
    range = "µA";                    // Fijar el rango en microamperios
  }
  // No encendemos/apagamos directamente en modo Auto, lo hace handleAutoMode()
}

  // Responder a la aplicación confirmando el cambio de modo
  server.send(200, "text/plain", "Mode changed to " + mode);
}

// Función para devolver los datos actuales en formato JSON
void handleData() {
  currentValue = readCurrent();  // Leer la corriente

  // Si el modo es "Auto", procesar automáticamente y cambiar la escala
  if (mode == "Auto") {
    handleAutoMode(currentValue);
  }

  // Crear el JSON con las variables que queremos enviar
  String jsonData = "{";
  jsonData += "\"current\": " + String(currentValue, 2) + ",";  // Agregar lectura de corriente
  jsonData += "\"mode\": \"" + mode + "\",";  // Enviar el modo actual
  jsonData += "\"range\": \"" + range + "\"";  // Enviar la escala actual (mA o µA)
  jsonData += "}";

  // Enviar los datos en formato JSON
  server.send(200, "application/json", jsonData);
  void enterLightSleep() {
  Serial.println("Entrando en Light Sleep...");

  // Configurar el tiempo de sueño (por ejemplo, 10 segundos)
  esp_sleep_enable_timer_wakeup(10 * 1000000);  // 10 segundos en microsegundos

  Serial.flush();  // Limpiar el buffer serial
  esp_light_sleep_start();  // Entrar en light sleep

  wakeup_count++;  // Contador de veces que ha despertado
  Serial.println("Despertado del Light Sleep");
  }
 enterLightSleep();
}

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

  // Configurar los pines
  pinMode(34, INPUT);  // Pin 34 para la señal de corriente
  pinMode(mosfetPin, OUTPUT);  // Pin para el MOSFET

  // Configurar el WiFi
  WiFi.softAP(ssid, password);
  Serial.println("Punto de acceso iniciado");
  Serial.print("IP del ESP32: ");
  Serial.println(WiFi.softAPIP());

  // Configurar las rutas del servidor
  server.on("/setMode", handleSetMode);   // Endpoint para cambiar el modo
  server.on("/data", handleData);         // Endpoint para obtener los datos actuales
  server.begin();                         // Iniciar el servidor
}

void loop() {
  server.handleClient();  // Manejar las solicitudes del cliente
}
```

# Desarrollo del PCB

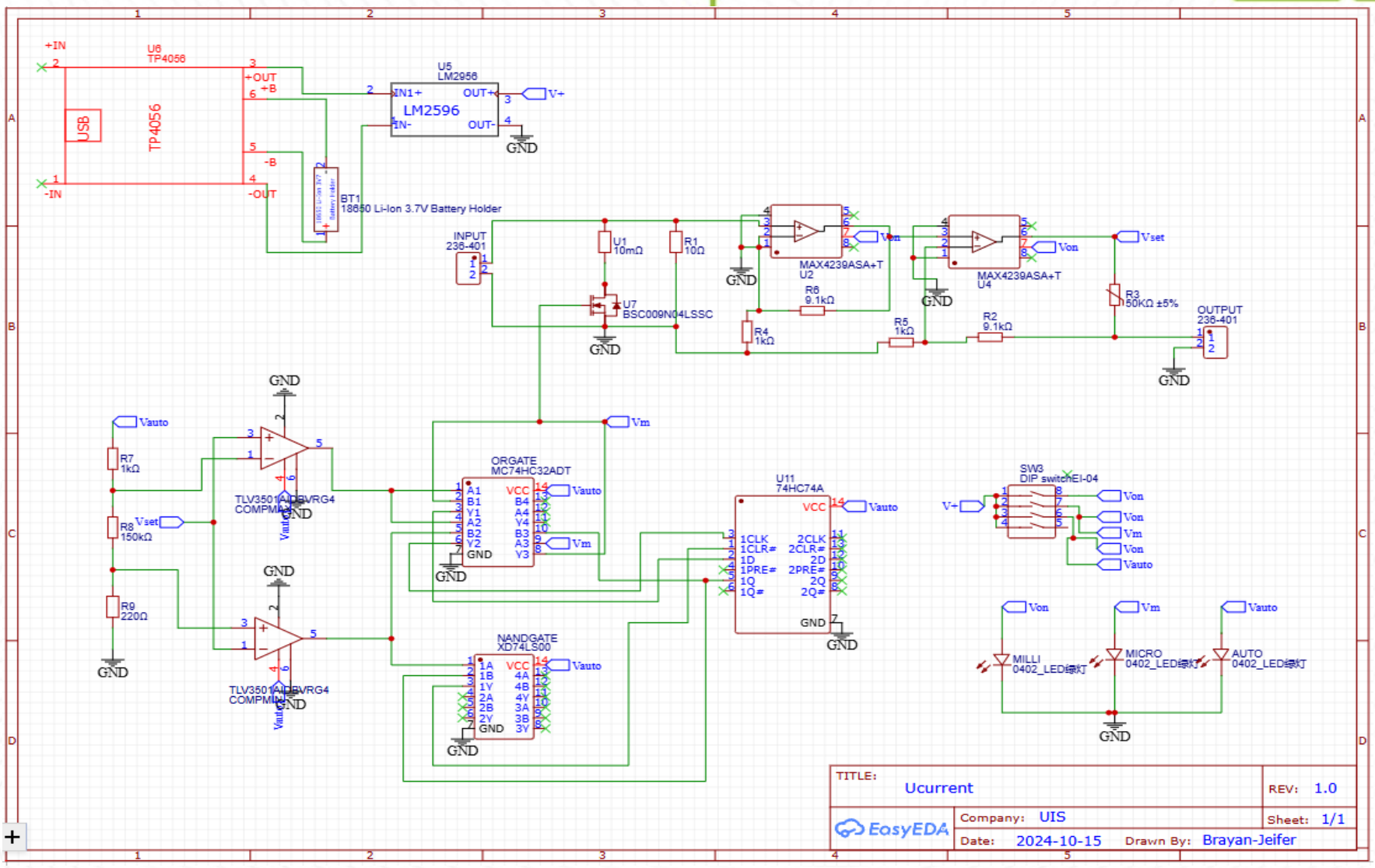
Herramienta Utilizada: EasyEDA.

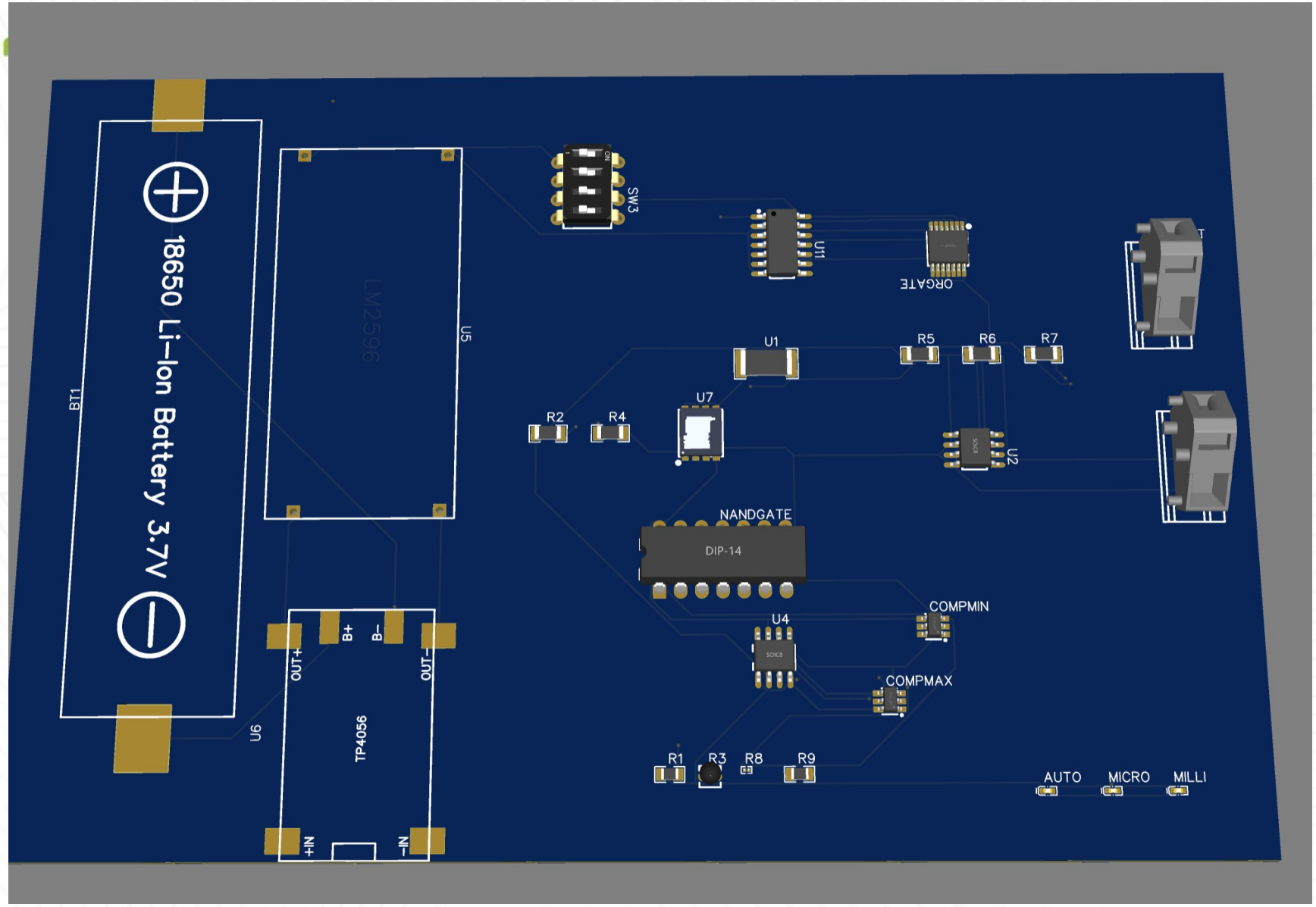
- Software en línea que facilita el diseño de PCBs y ofrece bibliotecas actualizadas de componentes.
- Permite generar listas de materiales (BOM) con costos y disponibilidad.

Diseño Modular:

Se integraron los siguientes bloques:

- Circuito de comparación: Amplificación y selección de señales.
- Circuito de control automático: Lógica para el cambio de rangos.
- Alimentación: Conversor buck-boost para mantener estabilidad.
- Interfaz con ESP32: Conexiones para transmisión de datos.




18650 Li-Ion Battery 3.7V
BT1
LM2596
U5
SW3
U11
ORGATE
U1
R5
R6
R7
U7
R2
R4
U2
NANDGATE
DIP-14
COMPMIN
U4
COMPMAX
TP4056
U6
OUT+
B+
B-
OUT-
+IN
-IN
R1
R3
R8
R9
AUTO
MICRO
MILLI

# Costos

## Balance Total del Proyecto:

- Costo total estimado: 30,809 USD.

Componentes principales y su impacto:

- TLV333: Alta precisión y estabilidad, representa el mayor costo en componentes activos.
- LM2956: Seleccionado por su velocidad y compatibilidad con el sistema.
- Buck-Boost (LM2623): Estabilidad garantizada en condiciones variables.

## Costos secundarios:

- Resistencias y capacitores de bajo costo que complementan el circuito.

## Consideraciones:

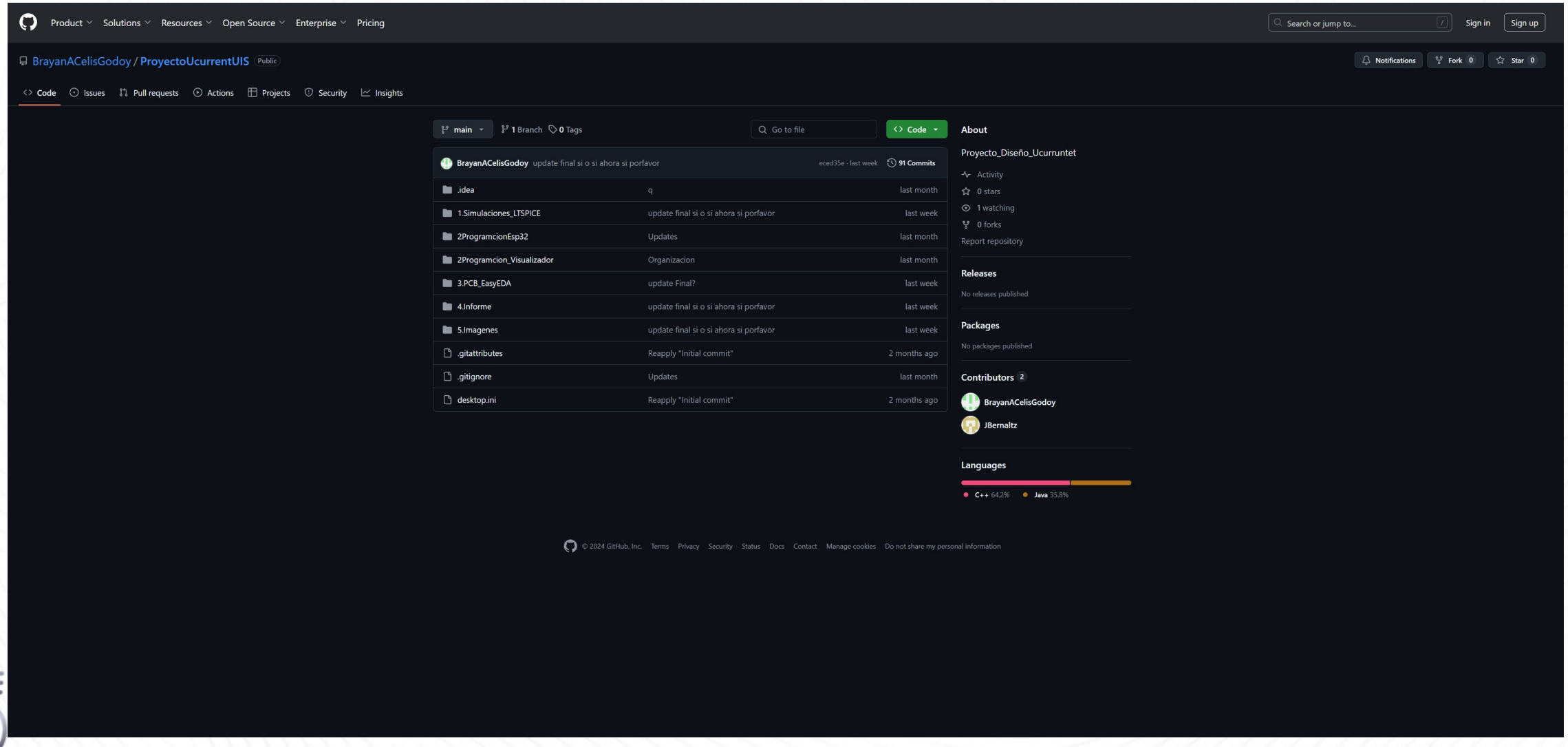
- Posibilidad de reducir costos mediante la integración en un módulo único.

| ID | Name | Quantity | Price | Total |
|---|---|---|---|---|
| 1 | Leds | 3 | 0,15 | 0,45 |
| 2 | TLV3501AIDBVR | 2 | 3,48 | 6,96 |
| 3 | 236-401 | 2 | 1,26 | 2,52 |
| 4 | XD74LS00 | 1 | 0,173 | 0,173 |
| 5 | MC74HC32ADT | 1 | 0,163 | 0,163 |
| 6 | 10Ω | 1 | 1,19 | 1,19 |
| 7 | 9.1kΩ | 2 | 0,034 | 0,068 |
| 8 | 50KΩ ±5% | 1 | 0,41 | 0,41 |
| 9 | 1kΩ | 3 | 0,1 | 0,3 |
| 10 | 150kΩ | 1 | 0,41 | 0,41 |
| 11 | 220Ω | 1 | 0,008 | 0,008 |
| 12 | DIP switchEI-04 | 2 | 0,45 | 0,9 |
| 13 | 10mΩ | 1 | 0,1 | 0,1 |
| 14 | TLV333 | 2 | 5,4335 | 10,867 |
| 15 | LM2956 | 1 | 1 | 1 |
| 16 | TP4056 | 1 | 2 | 2 |
| 17 | BSC009N04LSSC | 1 | 3,09 | 3,09 |
| 18 | 74HC74A | 1 | 0,2 | 0,2 |
| | | | Total | 30,809 |

# REPOSITORIO GitHub

#LaUISqueQueremos

Universidad Industrial de Santander




76 Aniversario UIS 1948 - 2024

Legado académico y cultural de los santandereanos

# Futuras Mejoras

## Conectividad Global:

Integrar la ESP32 a una red Wi-Fi existente, permitiendo monitoreo remoto a través de una base de datos en la nube.

Ejemplo de aplicación: Google Firebase para almacenar y visualizar datos globalmente.

## Pantalla Integrada:

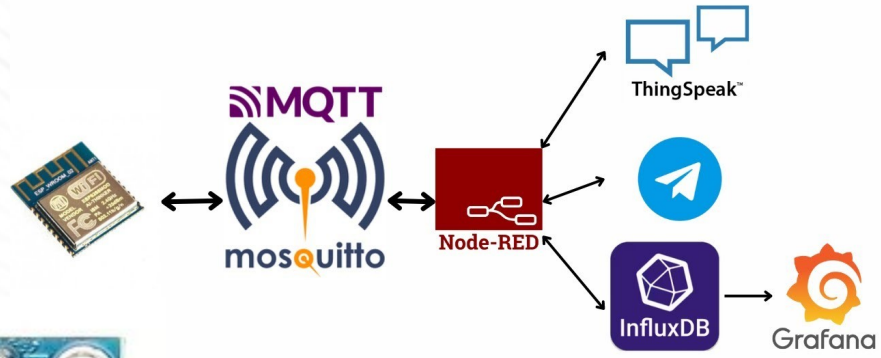
Añadir un módulo OLED para visualizar las mediciones directamente en el dispositivo.

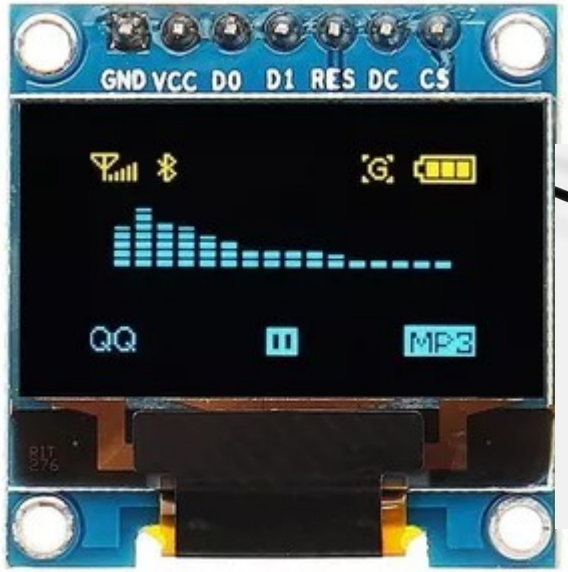
## Protección Física:

Diseñar una carcasa 3D para proteger los componentes y facilitar su manejo.

Optimización Lógica:

Reemplazar la lógica cableada por dispositivos programables (PAL o FPGA) para mayor flexibilidad.




Universidad Industrial de Santander

76 Aniversario UIS 1948 - 2024

Legado **académico** y **cultural** de los **santandereanos**

# ¡Gracias!

#LaUISqueQueremos