PLC

PLC con microcontrolador ESP32

Carpeta Técnica

Della Torre, Joaquín

Erbino, Renzo

Romero Dominguez, Braulio

Villegas González, Alejandro

2024

Índice

1. Preámbulo

1.1. ¿Quiénes somos?

- Braulio Romero Domínguez
  Mail: braulioemilioromero7@gmail.com
  DNI: 48.838.123
- Villegas González Alejandro
  Mail: advillegasg@gmail.com
  DNI: 48.572.234
- Della Torre Joaquín
  Mail: jaoquin1234@gmail.com
  DNI: 48.240.913
- Erbino Renzo
  Mail: renzoerbino@gmail.com
  DNI: 48.944.402

1.2. Docentes a cargo

- Fabrizio Carlassara
- Agustin Palmieri Hise

1.3. Información adicional

1.3.1. Tiempo invertido

- Fecha de inicio: 6 de agosto de 2024.
- Duración: 16 semanas de trabajo.
- Esfuerzo del proyecto individual: 3 horas de trabajo semanales por integrante.
- Esfuerzo total del proyecto: 192 horas de trabajo.

1.3.2. Programas utilizados

- Usamos Visual Studio Code para programar el ESP32
- iot-ladder-editor, un programa que utilizamos para programar en Ladder. https://github.com/leofds/iot-ladder-editor/tree/main

1.3.3. Lenguajes de programación y frameworks utilizados

- Python
- Ladder
- C

2. Introducción

2.1. Resumen del proyecto

Se busca aplicar las funciones de un PLC tradicional, el cual a grandes rasgos se encarga de detectar diversos tipos de señales del proceso, y elaborar y enviar acciones de acuerdo con lo que se ha programado. Además, recibe configuraciones de los programadores y da reporte a los mismos, aceptando modificaciones de programación cuando son necesarias
Pero se utiliza buscando una simplificación en el ámbito de utilizar un microcontrolador ESP32 como núcleo de las operaciones, el cual estará conectado a una página web que mostrará un análisis de datos recaudados y el programa del PLC el cual podrá ser modificado siempre que se desee a través de esa página

2.2. ¿Por qué usar un PLC?

Los PLCs permiten controlar y automatizar procesos industriales, reduciendo la necesidad de intervención manual

y aumentando la precisión y consistencia en las operaciones. Son como cerebros programables que hacen que las cosas funcionen automáticamente y de manera eficiente. Además, con un microcontrolador abarataríamos costo, reduciríamos el tamaño, sería más fácil de programar e incluso se le podrían añadir funciones de Wifi o Bluetooth

2.3. PLCs en la industria

Los PLC se usan en la automatización industrial para controlar procesos y maquinaria en industrias como la automotriz, alimentaria y química, así como en sistemas de energía, edificios y transporte automatizado por ejemplo en automatización de procesos industriales, control de máquinas, sistemas de transporte y logística y gestión de energía.

2.4. Impacto medioambiental

Los PLCs optimizan procesos industriales, reduciendo el consumo energético al eliminar desperdicios y ajustando operaciones para usar solo la energía necesaria, además son esenciales en la operación de sistemas de energía renovable. Al mismo tiempo, contienen componentes electrónicos que requieren recursos como metales, plásticos y tierras raras para su fabricación, lo que puede generar impacto ambiental durante la extracción y procesamiento de estos materiales, teniendo en cuenta que su producción contribuye a emisiones de CO2 y que consumen energía durante su operación, y su impacto depende de la fuente de energía utilizada.

3. Desarrollo técnico

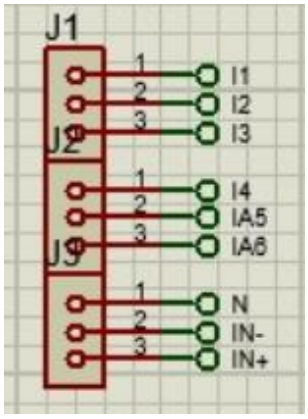
3.1. Descripción del funcionamiento

El PLC lee lo que hay en las entradas y según esto usa la lógica que dice que si tal entrada esta en tal estado activa tal salida, y dentro del ESP32 tiene varios archivos de configuración con un archivo de LCD que te muestra lo que

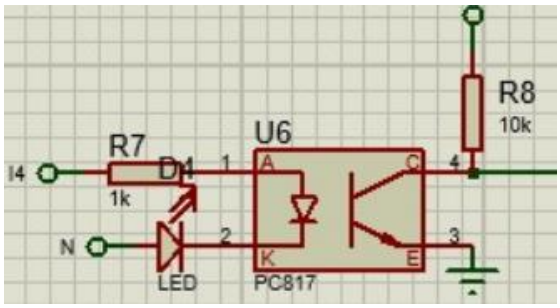
está sucediendo. Se conecta al Wifi e inicia un servidor web con su IP, donde lo accedes y ves el estado de entradas y salidas del PLC. Manda a correr el archivo del plc.c, donde está la lógica del PLC que al cambiar algún estado se refresca la pagina, después de iniciar el servidor web.

3.1.1 Descripción de las entradas

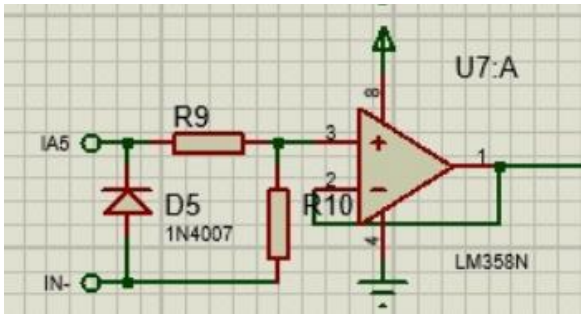
Las borneras tienen las entradas digitales y analógicas (I1 a I4 e IA5 a IA6 respectivamente) y también poseen los 24V y su GND. N solo cumple la función de neutro respecto a todas las entradas digitales, mientras que el neutro de las analógicas es IN-.



Posee cuatro de las seis entradas digitales, estas funcionan con un optoacoplador el cual es generalmente utilizado para reducir la señal eléctrica que ingresa de 12V o 24V a un rango de 0 a 3.3V, lo que el microcontrolador es capaz de soportar. Este método es generalmente usado para las entradas de los microprocesadores.

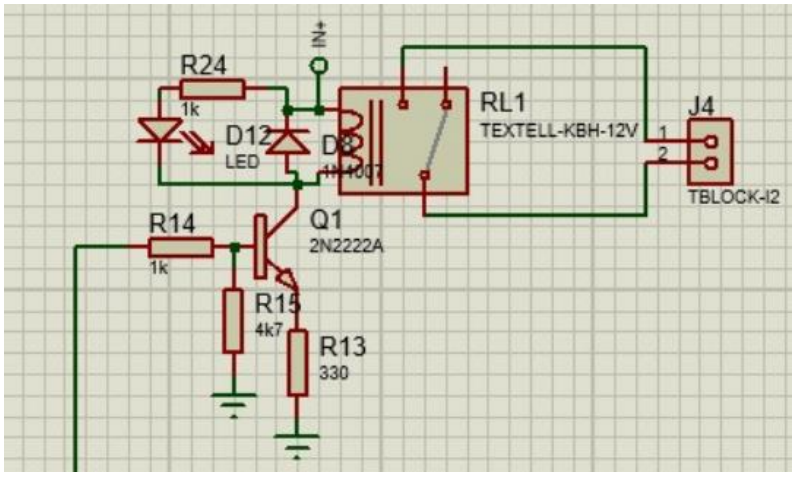


En las entradas analógicas tenemos una reducción de tensión, que pasará de 0V-10V a una menor, que pueda aceptar el microcontrolador, mediante una relación a través de la primera parte. Luego tenemos un seguidor de tensión que separa la entrada de la salida que luego va conectada al microcontrolador.

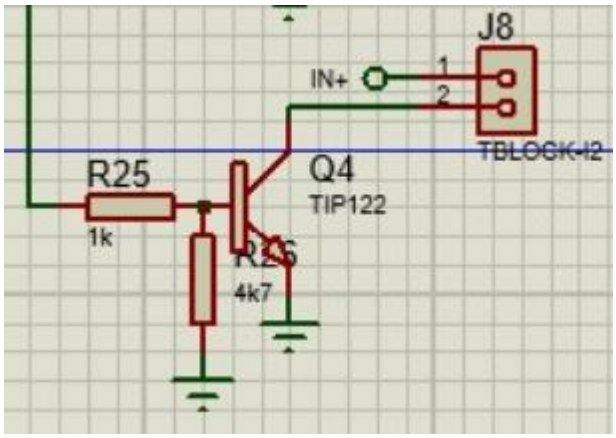


3.1.2 Descripción de las salidas

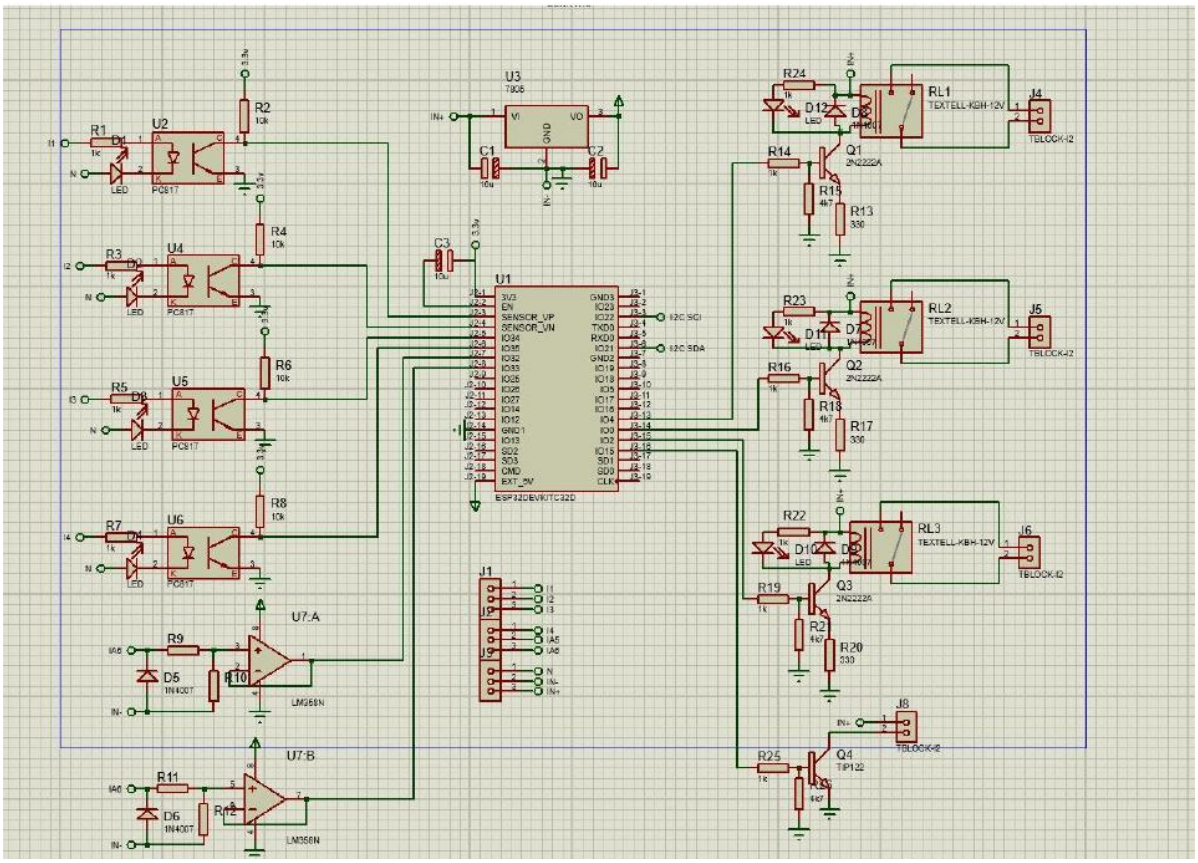
Tiene una salida con etapa de potencia de relé, la cual funciona a través del mecanismo básico de etapa de potencia de relé y es uno de los dos tipos de salida que posee el PLC. Esto se realiza de esta manera para tener el control de las salidas de grandes potencias de una manera más segura y eficiente.

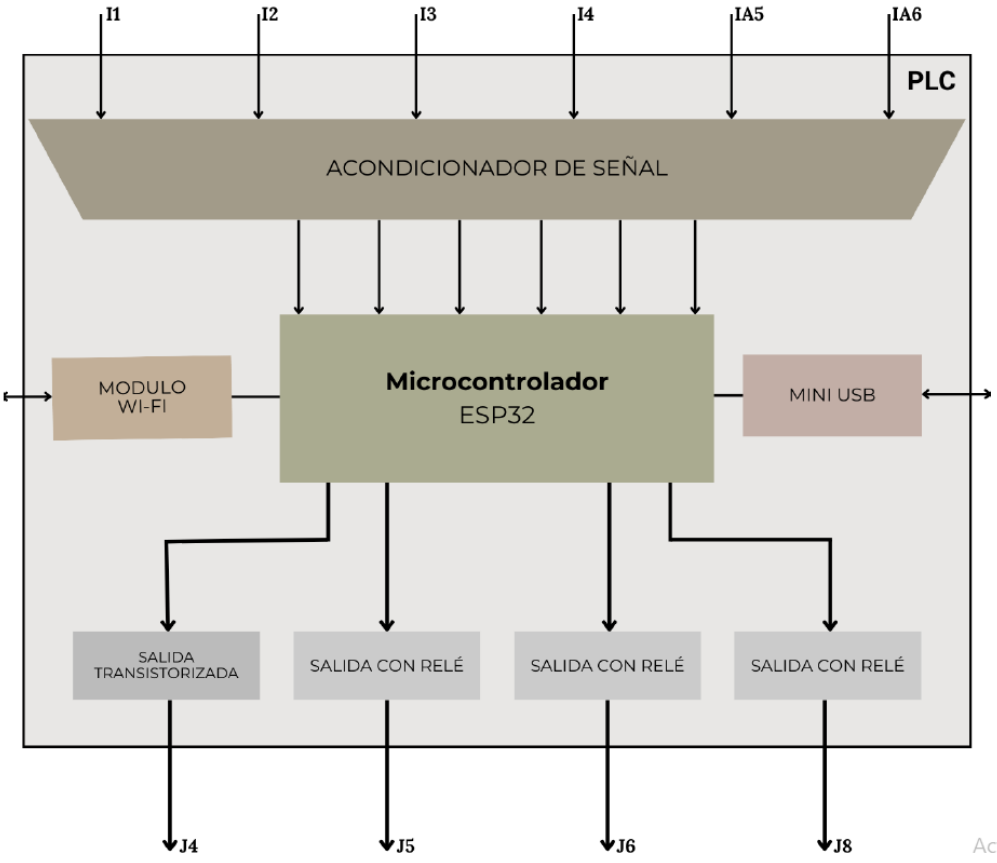


Las salidas transistorizadas en un PLC se utilizan cuando se requiere una respuesta rápida, un alto número de conmutaciones y el manejo de señales digitales, pero con la limitación de manejar cargas de menor potencia que las salidas de relé.



3.1.3. Diagramas y esquemáticos





4. Placa

4.1. Componentes

- U1 – ESP32
- U2, U4, U5, U6 – PC817
- U3 – LM7805
- U7 – LM358N
- Q1, Q2, Q3 – 2N2222A
- Q4 – TIP122
- 26x Resistencia
- 8x Bornera
- 12x Diodo
- 3x Relé
- 3x Capacitor

4.2. Costos

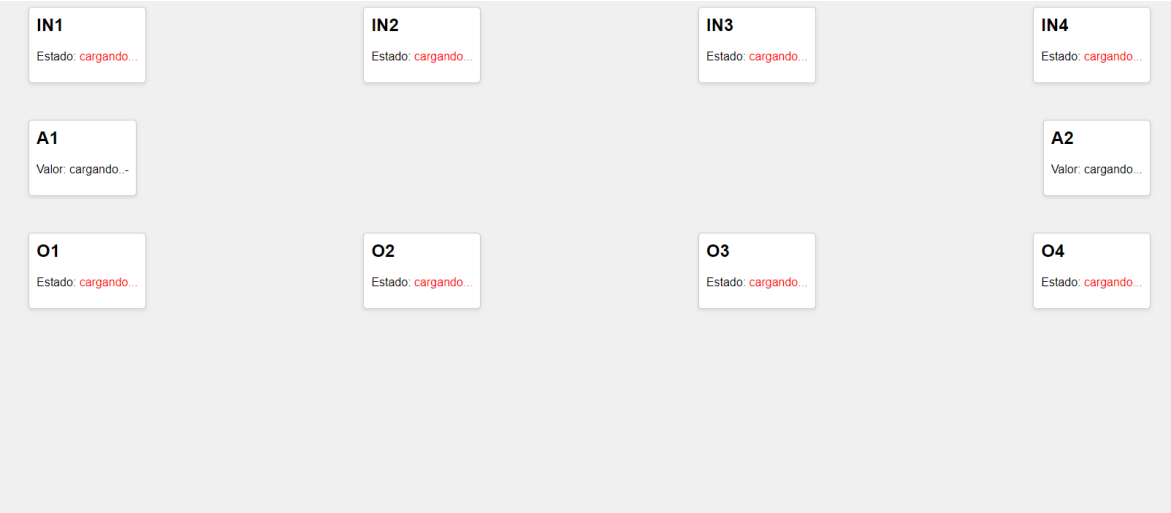
- ESP32 ≈ 10000$
- 4x PC817 ≈ 3000$
- LM7805 ≈ 2500$
- LM358N ≈ 3400$
- 3x 2N2222A ≈ 2300$
- TIP122 ≈ 2500$
- 26x Resistencia ≈ 2000$
- 8x Bornera ≈ 6000$
- 12x Diodo ≈ 5000$
- 3x Relé ≈ 4000$
- 3x Capacitor ≈ 1000$

Precio total ≈ 41700$

5. Página

5.1. Funcionamiento

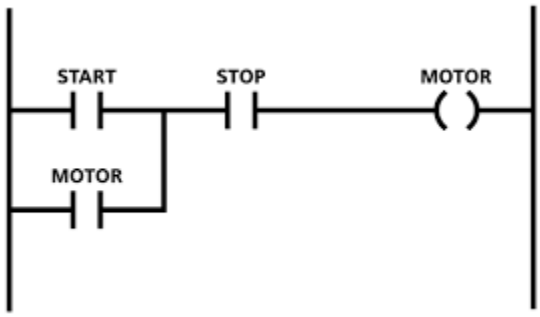
Al entrar a la página tienes un web_server.c, el código del servidor web que describe los eventos, es decir, que hacer cuando alguien entra a la página, que mostrarle y le devuelve el html con el javascript dentro y te muestra las entradas y salidas del PLC, de forma paralela es decir, que si se altera el estado de una de las entradas o salidas se refresca la página.

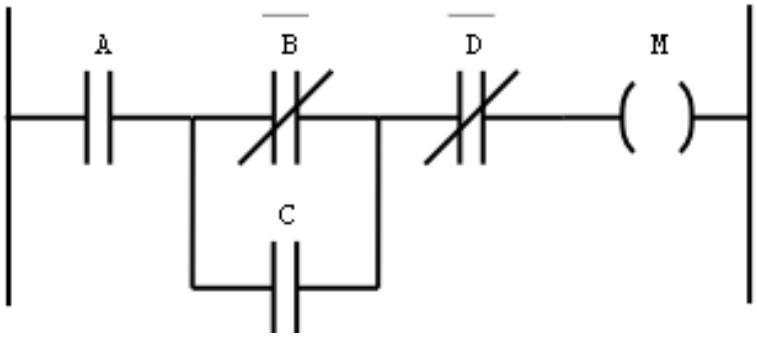


6. Código de la placa

6.1. Lenguaje Ladder

Este es un lenguaje de programación gráfica utilizado principalmente para programar PLCs. Es muy popular en la automatización industrial debido a su similitud con los diagramas eléctricos utilizados tradicionalmente en circuitos de control eléctrico y su facilidad de interpretación por parte de técnicos e ingenieros, especialmente para aquellos que migran de sistemas de relés a sistemas programables. Por otra parte, se optimizó el código para que no ocupe espacio en memoria





.

6.2. Códigos

“PLC.C”

```c

#include <stdio.h>
#include "driver/gpio.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_timer.h"
#include "esp_adc/adc_oneshot.h"

adc_oneshot_unit_handle_t adc_handle;// handle del adc
adc_oneshot_chan_cfg_t config_channel = {
  .bitwidth = ADC_BITWIDTH_DEFAULT,
  .atten = ADC_ATTEN_DB_12,
}; // creo el struct para la config de los canales


//pin inputs
#define PIN_I01 36
#define PIN_I02 39
#define PIN_I03 34
#define PIN_I04 35
#define PIN_I05 32
#define PIN_I06 33

int list_pins[6]= {PIN_I01,PIN_I02,PIN_I03,PIN_I04,PIN_I05,PIN_I06};
adc_channel_t list_channels[6];

//pin outpus
#define PIN_Q01 19
#define PIN_Q02 18
#define PIN_Q03 5
#define PIN_Q04 17
//estate vars for every pin
uint8_t LD_I1 = 0;
uint8_t LD_I2 = 0;
uint8_t LD_I3 = 0;
uint8_t LD_I4 = 0;
uint8_t LD_I5 = 0;
uint8_t LD_I6 = 0;

uint8_t LD_Q1 = 0;
uint8_t LD_Q2 = 0;
uint8_t LD_Q3 = 0;
uint8_t LD_Q4 = 0;

typedef struct {
  int32_t PRE;
  int32_t AC;
  int32_t B;
  int32_t DN;
  int32_t EN;
  uint64_t TT;
} LD_TIMER;

union {
  uint32_t p[2];
  uint64_t v;
} LD_TIME;

uint64_t getTime(){
  return LD_TIME.v;
}


LD_TIMER LD_T1;

void refreshTime64bit(){
unsigned long now = esp_timer_get_time()/1000;  if(now < LD_TIME.p[0]){
    LD_TIME.p[1]++;
  }
  LD_TIME.p[0] = now;
}

void readInputs(){
  //pin 33 --> input/6
  adc_oneshot_read(adc_handle, list_channels[5], &LD_I6);vTaskDelay(1);
  //pin 32 --> input/5
  adc_oneshot_read(adc_handle, list_channels[4], &LD_I5);vTaskDelay(1);
  //pin 35 --> input/4
  adc_oneshot_read(adc_handle, list_channels[3], &LD_I4);vTaskDelay(1);
  //pin 34 --> input/3
  adc_oneshot_read(adc_handle, list_channels[2], &LD_I3);vTaskDelay(1);
  //pin 39 --> input/2
  adc_oneshot_read(adc_handle, list_channels[1], &LD_I2);vTaskDelay(1);
```

```
    //pin 36 --> input/1
    adc_oneshot_read(adc_handle, list_channels[0], &LD_I1);vTaskDelay(1);

  }

  void writeOutputs(){

    gpio_set_level(PIN_Q01,LD_Q1);
    gpio_set_level(PIN_Q02,LD_Q2);
    gpio_set_level(PIN_Q03,LD_Q3);
    gpio_set_level(PIN_Q04,LD_Q4);

  }

  void rung001(void){
    uint8_t _LD_S0;
    _LD_S0 = 1;
    if(LD_I1){
      _LD_S0 = 0;
    }
    LD_Q1 = _LD_S0;
  }

  void rung002(void){
    uint8_t _LD_S0;
    uint64_t _LD_T1;
    uint64_t _LD_T2;
    _LD_S0 = 1;
    if(LD_I2){
      _LD_S0 = 0;
    }
    LD_T1.EN = _LD_S0;
    if(!_LD_S0){
      LD_T1.DN = 0;
      LD_T1.AC = 0;
      LD_T1.TT =  getTime();
    }else{
      if(!LD_T1.DN){
        _LD_T1 =  getTime();
        _LD_T2 = _LD_T1 - LD_T1.TT;
        if(_LD_T2 >= LD_T1.B){
          LD_T1.TT = _LD_T1;
          LD_T1.AC = LD_T1.AC + 1;
          if(LD_T1.AC >= LD_T1.PRE){
            LD_T1.DN = 1;
          }
        }
      }
    }
    _LD_S0 = LD_T1.DN;
    LD_Q2 = _LD_S0;
    if(_LD_S0){
      LD_T1.DN = 0;
      LD_T1.AC = 0;
      LD_T1.EN = 0;
      LD_T1.TT =  getTime();
    }
  }

  void rung003(void){
    uint8_t _LD_S0;
    _LD_S0 = 1;
  }

  void initContext(void){
    LD_T1.EN = 0;
    LD_T1.AC = 0;
    LD_T1.PRE = 1;
    LD_T1.B = 100;
    LD_T1.DN = 0;
    LD_T1.TT =  getTime();
  }

  void init(){
          LD_TIME.v = 0;
      refreshTime64bit();
     //INPUTS(with adc)-- inicializacion
    //adc_oneshot_unit_handle_t adc_handle;// handle del adc
    adc_oneshot_unit_init_cfg_t config_adc; //struct de config de canales
    config_adc.unit_id=ADC_UNIT_1;
    config_adc.ulp_mode= ADC_ULP_MODE_DISABLE;
    config_adc.clk_src= 0;

    adc_oneshot_new_unit(&config_adc, &adc_handle);//creo la unidad con el handle y config
    /*
    adc_oneshot_chan_cfg_t config_channel; // creo el struct para la config de los canales
    config_channel.bitwidth = ADC_BITWIDTH_DEFAULT;
    config_channel.atten = ADC_ATTEN_DB_12;
    */


    //adc_oneshot_config_channel(adc_handle, EXAMPLE_ADC1_CHAN0, config_adc);
    adc_channel_t c;adc_unit_t u;
    for(int in_pin =0;in_pin<6;in_pin++){

      adc_oneshot_io_to_channel(list_pins[in_pin],&u,&c);
      adc_oneshot_config_channel(adc_handle, c, &config_channel);
      //agrego cada channel al arrray
      list_channels[in_pin] = c;
      //muestro el canal asginado para cada pin.
      ESP_LOGI("ADC","pin: %d >> channel: %d",list_pins[in_pin],c);

    }
      /*
      pinMode(PIN_I01, INPUT);
      pinMode(PIN_I02, INPUT);
      pinMode(PIN_Q01, OUTPUT);
      */
      //OUTPUTS
      gpio_reset_pin(PIN_Q01);gpio_reset_pin(PIN_Q02);gpio_reset_pin(PIN_Q03);gpio_reset_pin(PIN_Q04);
      gpio_set_direction(PIN_Q01,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q02,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q03,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q04,GPIO_MODE_OUTPUT);
  }


  void app(){
     init();
      initContext();
    for(;;){
      vTaskDelay(1);
      readInputs();
      refreshTime64bit();
      rung001();
      rung002();
      rung003();
      writeOutputs();
    }
  }
```

“PLC1.C”

```c
#include <stdio.h>
#include "driver/gpio.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_timer.h"
#include "esp_adc/adc_oneshot.h"

adc_oneshot_unit_handle_t adc_handle;// handle del adc
adc_oneshot_chan_cfg_t config_channel = {
  .bitwidth = ADC_BITWIDTH_DEFAULT,
  .atten = ADC_ATTEN_DB_12,
}; // creo el struct para la config de los canales


//pin inputs
#define PIN_I01 36
#define PIN_I02 39
#define PIN_I03 34
#define PIN_I04 35
#define PIN_I05 32
#define PIN_I06 33

int list_pins[6]= {PIN_I01,PIN_I02,PIN_I03,PIN_I04,PIN_I05,PIN_I06};
adc_channel_t list_channels[6];

//pin outpus
#define PIN_Q01 19
#define PIN_Q02 18
#define PIN_Q03 5
#define PIN_Q04 17
//estate vars for every pin
uint8_t LD_I1 = 0;
uint8_t LD_I2 = 0;
uint8_t LD_I3 = 0;
uint8_t LD_I4 = 0;
uint8_t LD_I5 = 0;
uint8_t LD_I6 = 0;

uint8_t LD_Q1 = 0;
uint8_t LD_Q2 = 0;
uint8_t LD_Q3 = 0;
uint8_t LD_Q4 = 0;

union {
  uint32_t p[2];
  uint64_t v;
} LD_TIME;
```

```c
uint64_t getTime(){
  return LD_TIME.v;
}

void refreshTime64bit(){
    unsigned long now = esp_timer_get_time()/1000;
    if(now < LD_TIME.p[0]){
        LD_TIME.p[1]++;
    }
    LD_TIME.p[0] = now;
}



void readInputs(){
  //pin 33 --> input/6
  adc_oneshot_read(adc_handle, list_channels[5], &LD_I6);vTaskDelay(1);
  //pin 32 --> input/5
  adc_oneshot_read(adc_handle, list_channels[4], &LD_I5);vTaskDelay(1);
  //pin 35 --> input/4
  adc_oneshot_read(adc_handle, list_channels[3], &LD_I4);vTaskDelay(1);
  //pin 34 --> input/3
  adc_oneshot_read(adc_handle, list_channels[2], &LD_I3);vTaskDelay(1);
  //pin 39 --> input/2
  adc_oneshot_read(adc_handle, list_channels[1], &LD_I2);vTaskDelay(1);
  //pin 36 --> input/1
  adc_oneshot_read(adc_handle, list_channels[0], &LD_I1);vTaskDelay(1);
}
void writeOutputs(){
  gpio_set_level(PIN_Q01,LD_Q1);
  gpio_set_level(PIN_Q02,LD_Q2);
  gpio_set_level(PIN_Q03,LD_Q3);
  gpio_set_level(PIN_Q04,LD_Q4);
}

void init(){
    LD_TIME.v = 0;
    refreshTime64bit();
   //INPUTS(with adc)-- inicializacion
  //adc_oneshot_unit_handle_t adc_handle;// handle del adc
  adc_oneshot_unit_init_cfg_t config_adc; //struct de config de canales
  config_adc.unit_id=ADC_UNIT_1;
  config_adc.ulp_mode= ADC_ULP_MODE_DISABLE;
  config_adc.clk_src= 0;

  adc_oneshot_new_unit(&config_adc, &adc_handle);//creo la unidad con el handle y config
```

```c
  /*
  adc_oneshot_chan_cfg_t config_channel; // creo el struct para la config de los canales
  config_channel.bitwidth = ADC_BITWIDTH_DEFAULT;
  config_channel.atten = ADC_ATTEN_DB_12;
  */


  //adc_oneshot_config_channel(adc_handle, EXAMPLE_ADC1_CHAN0, config_adc);
  adc_channel_t c;adc_unit_t u;
  for(int in_pin =0;in_pin<6;in_pin++){

    adc_oneshot_io_to_channel(list_pins[in_pin],&u,&c);
    adc_oneshot_config_channel(adc_handle, c, &config_channel);
    //agrego cada channel al arrray
    list_channels[in_pin] = c;
    //muestro el canal asginado para cada pin.
    ESP_LOGI("ADC","pin: %d >> channel: %d",list_pins[in_pin],c);

  }
    /*
    pinMode(PIN_I01, INPUT);
    pinMode(PIN_I02, INPUT);
    pinMode(PIN_Q01, OUTPUT);
    */
    //OUTPUTS
    gpio_reset_pin(PIN_Q01);gpio_reset_pin(PIN_Q02);gpio_reset_pin(PIN_Q03);gpio_reset_pin(PIN_Q04);
    gpio_set_direction(PIN_Q01,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q02,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q03,GPIO_MODE_OUTPUT);gpio_set_direction(PIN_Q04,GPIO_MODE_OUTPUT);
}

/*
void initcontext(){

}
*/



/*
void refreshTime64bit(){
  unsigned long now = millis();
  if(now < LD_TIME.p[0]){
    LD_TIME.p[1]++;
  }
  LD_TIME.p[0] = now;
}
*/
```

```c

void rung001(){
  uint8_t _LD_S0;
  _LD_S0 = 1;
  if(_LD_S0){
    if(LD_I1 > 200){
      _LD_S0 = 0;
    }
  }
  LD_Q1 = _LD_S0;
}

void rung002(){
  uint8_t _LD_S0;
  _LD_S0 = 1;
  if(!LD_I2){
    _LD_S0 = 0;
  }
  LD_Q1 = _LD_S0;
}


void app(void)
{
    init();

    while(1){
        vTaskDelay(1);
        readInputs();
        refreshTime64bit();
        rung001();
        //rung002();
        writeOutputs();
    }
}
```

“WEB_SERVER.C”

```c
#include "esp_http_server.h"
#include <string.h>
#include <esp_system.h>
#include <nvs_flash.h>
#include <sys/param.h>
#include <stdio.h>
#include "esp_log.h"
//#include "plc.c"

extern uint8_t LD_I1;
extern uint8_t LD_I2;
extern uint8_t LD_I3;
extern uint8_t LD_I4;
extern uint8_t LD_I5;
extern uint8_t LD_I6;

extern uint8_t LD_Q1;
extern uint8_t LD_Q2;
extern uint8_t LD_Q3;
extern uint8_t LD_Q4;

//#define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))

/* Our URI handler function to be called during GET /uri request */
esp_err_t get_handler(httpd_req_t *req)
{
    /* Send a simple response */
    extern unsigned char view_start[] asm("_binary_webpage_html_start");
    extern unsigned char view_end[] asm("_binary_webpage_html_end");
    size_t view_len = view_end - view_start;
    //memcpy(viewHtml, view_start, view_len);
    /*
    char viewHtml[view_len];
    memcpy(viewHtml, view_start, view_len);
    ESP_LOGI("index", "URI: %s", req->uri);
    */
    //ESP_LOGI("index", "URI: %s", viewHtml);

    //const char resp[] = "hola";
    httpd_resp_send_chunk(req, (const char *)view_start, view_len);
    return ESP_OK;
}

esp_err_t Iodata(httpd_req_t *req)
{
    //update_values(LD_I1,LD_I2,LD_I3,LD_I4,LD_I5,LD_I6,LD_Q1,LD_Q2,LD_Q3,LD_Q4);
    //extern unsigned char view_start[] asm("_binary_Iodata_json_start");
    //extern unsigned char view_end[] asm("_binary_Iodata_json_end");
    //size_t view_len = view_end - view_start;
    char viewJson[] = "{ \"I1\":%d,\"I2\":%d,\"I3\":%d,\"I4\":%d,\"I5\":%d,\"I6\":%d,\"O1\":%d,\"O2\":%d,\"O3\":%d,\"O4\":%d}";
    //memcpy(viewJson, view_start, view_len);


    char *viewJsonUpdated;
    asprintf(&viewJsonUpdated, viewJson, LD_I1,LD_I2,LD_I3,LD_I4,LD_I5,LD_I6,LD_Q1,LD_Q2,LD_Q3,LD_Q4);

    ESP_LOGI("json", "%s", viewJsonUpdated);
    httpd_resp_set_type(req,"application/json");
    httpd_resp_sendstr(req, viewJsonUpdated);
    free(viewJsonUpdated);
    return ESP_OK;
}



/* Our URI handler function to be called during POST /uri request */
esp_err_t post_handler(httpd_req_t *req)
{
    /* Destination buffer for content of HTTP POST request.
     * httpd_req_recv() accepts char* only, but content could
     * as well be any binary data (needs type casting).
     * In case of string data, null termination will be absent, and
     * content length would give length of string */
    char content[100];

    /* Truncate if content length larger than the buffer */
    size_t recv_size = MIN(req->content_len, sizeof(content));

    int ret = httpd_req_recv(req, content, recv_size);
    if (ret <= 0) {  /* 0 return value indicates connection closed */
        /* Check if timeout occurred */
        if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
            /* In case of timeout one can choose to retry calling
             * httpd_req_recv(), but to keep it simple, here we
             * respond with an HTTP 408 (Request Timeout) error */
            httpd_resp_send_408(req);
        }
        /* In case of error, returning ESP_FAIL will
         * ensure that the underlying socket is closed */
        return ESP_FAIL;
    }

    /* Send a simple response */
    const char resp[] = "URI POST Response";
    httpd_resp_send(req, resp, HTTPD_RESP_USE_STRLEN);
    return ESP_OK;
}

/* URI handler structure for GET /uri */
httpd_uri_t uri_get = {
    .uri      = "/",
    .method   = HTTP_GET,
    .handler  = get_handler,
    .user_ctx = NULL
};

httpd_uri_t uri_infovalues = {
    .uri      = "/info_values",
    .method   = HTTP_GET,
    .handler  = Iodata,
    .user_ctx = NULL
};



/* URI handler structure for POST /uri */
httpd_uri_t uri_post = {
    .uri      = "/uri",
    .method   = HTTP_POST,
    .handler  = post_handler,
    .user_ctx = NULL
};

/* Function for starting the webserver */
httpd_handle_t start_webserver(void)
{
    /* Generate default configuration */
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    /* Empty handle to esp_http_server */
    httpd_handle_t server = NULL;

    /* Start the httpd server */
    if (httpd_start(&server, &config) == ESP_OK) {
        /* Register URI handlers */
        httpd_register_uri_handler(server, &uri_get);
        httpd_register_uri_handler(server, &uri_infovalues);

        httpd_register_uri_handler(server, &uri_post);
```

```c

        }
        /* If server failed to start, handle will be NULL */
        return server;
}

/* Function for stopping the webserver */
void stop_webserver(httpd_handle_t server)
{
    if (server) {
        /* Stop the httpd server */
        httpd_stop(server);
    }
}
```

“WEBPAGE.HTML”

```html
<html lang='es'>
<head>
    <meta charset='UTF-8'>
    <meta name='viewport' content='width=device-width, initial-scale=1.0'>
    <title>Estado de Variables</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f0f0;
        }

        .container {
            width: 80%;
            margin: auto;
            display: flex;
            justify-content: space-between;
            align-items: center;
            padding: 20px;
        }

        .variable {
            padding: 10px;
            margin-bottom: 10px;
            background-color: #fff;
            border: 1px solid #ccc;
            border-radius: 5px;
            box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
        }

        .variable h2 {
            margin-top: 0;
        }

        .on {
            color: green;
        }

        .off {
            color: red;
        }
    </style>
</head>
<body>
    <script>
        function requestDataFromESP32() {
            fetch('/info_values')
                .then(response => response.json())
                .then(data => {
                    console.log('I1', data.I1);
                    const estado_I1 = document.getElementById('I1_value');
                    estado_I1.textContent = (data.I1 === 255) ? 'ON' : 'OFF';
                    estado_I1.className = (data.I1 === 255) ? 'on' : 'off';

                    const estado_I2 = document.getElementById('I2_value');
                    estado_I2.textContent = (data.I2 === 255) ? 'ON' : 'OFF';
                    estado_I2.className = (data.I2 === 255) ? 'on' : 'off';

                    const estado_I3 = document.getElementById('I3_value');
                    estado_I3.textContent = (data.I3 === 255) ? 'ON' : 'OFF';
                    estado_I3.className = (data.I3 === 255) ? 'on' : 'off';

                    const estado_I4 = document.getElementById('I4_value');
                    estado_I4.textContent = (data.I4 === 255) ? 'ON' : 'OFF';
                    estado_I4.className = (data.I4 === 255) ? 'on' : 'off';

                    const estado_I5 = document.getElementById('I5_value');
                    estado_I5.textContent = data.I5;

                    const estado_I6 = document.getElementById('I6_value');
                    estado_I6.textContent = data.I6;

                    const estado_O1 = document.getElementById('O1_value');
                    estado_O1.textContent = (data.O1 === 1) ? 'ON' : 'OFF';
                    estado_O1.className = (data.O1 === 1) ? 'on' : 'off';

                    const estado_O2 = document.getElementById('O2_value');
                    estado_O2.textContent = (data.O2 === 1) ? 'ON' : 'OFF';
                    estado_O2.className = (data.O2 === 1) ? 'on' : 'off';

                    const estado_O3 = document.getElementById('O3_value');
                    estado_O3.textContent = (data.O3 === 1) ? 'ON' : 'OFF';
                    estado_O3.className = (data.O3 === 1) ? 'on' : 'off';

                    const estado_O4 = document.getElementById('O4_value');
                    estado_O4.textContent = (data.O4 === 1) ? 'ON' : 'OFF';
                    estado_O4.className = (data.O4 === 1) ? 'on' : 'off';
                })
                .catch(error => console.error('Error al obtener datos del sensor:', error));
        }

        setInterval(requestDataFromESP32, 500); // 500 ms = 0.5 segundos
    </script>
```

```html

    <div class='container'>
        <div class='variable'>
            <h2>IN1</h2>
            <p>Estado: <span id='I1_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>IN2</h2>
            <p>Estado: <span id='I2_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>IN3</h2>
            <p>Estado: <span id='I3_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>IN4</h2>
            <p>Estado: <span id='I4_value' class='off'>cargando...</span></p>
        </div>
    </div>

    <div class='container'>
        <div class='variable'>
            <h2>A1</h2>
            <p>Valor: <span id='I5_value'>cargando..</span></p>
        </div>
        <div class='variable'>
            <h2>A2</h2>
            <p>Valor: <span id='I6_value'>cargando...</span></p>
        </div>
    </div>

    <div class='container'>
        <div class='variable'>
            <h2>O1</h2>
            <p>Estado: <span id='O1_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>O2</h2>
            <p>Estado: <span id='O2_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>O3</h2>
            <p>Estado: <span id='O3_value' class='off'>cargando...</span></p>
        </div>
        <div class='variable'>
            <h2>O4</h2>
            <p>Estado: <span id='O4_value' class='off'>cargando...</span></p>
        </div>
    </div>

</body>
</html>
```

“WIFI_CONNECTION.H”

```c
/* WiFi station Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this
   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
   CONDITIONS OF ANY KIND, either express or implied.
*/
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"

#include "lwip/err.h"
#include "lwip/sys.h"

/* The examples use WiFi configuration that you can set via project configuration menu

   If you'd rather not, just change the below entries to strings with
   the config you want - ie #define EXAMPLE_WIFI_SSID "mywifissid"
*/
#define EXAMPLE_ESP_WIFI_SSID      CONFIG_ESP_WIFI_SSID
#define EXAMPLE_ESP_WIFI_PASS      CONFIG_ESP_WIFI_PASSWORD
#define EXAMPLE_ESP_MAXIMUM_RETRY  CONFIG_ESP_MAXIMUM_RETRY

#if CONFIG_ESP_WPA3_SAE_PWE_HUNT_AND_PECK
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HUNT_AND_PECK
#define EXAMPLE_H2E_IDENTIFIER ""
#elif CONFIG_ESP_WPA3_SAE_PWE_HASH_TO_ELEMENT
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_HASH_TO_ELEMENT
#define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
#elif CONFIG_ESP_WPA3_SAE_PWE_BOTH
#define ESP_WIFI_SAE_MODE WPA3_SAE_PWE_BOTH
#define EXAMPLE_H2E_IDENTIFIER CONFIG_ESP_WIFI_PW_ID
#endif
#if CONFIG_ESP_WIFI_AUTH_OPEN
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_OPEN
#elif CONFIG_ESP_WIFI_AUTH_WEP
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WEP
#elif CONFIG_ESP_WIFI_AUTH_WPA_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA_WPA2_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA_WPA2_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WPA2_WPA3_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WPA2_WPA3_PSK
#elif CONFIG_ESP_WIFI_AUTH_WAPI_PSK
#define ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD WIFI_AUTH_WAPI_PSK
#endif

/* FreeRTOS event group to signal when we are connected*/
static EventGroupHandle_t s_wifi_event_group;

/* The event group allows multiple bits for each event, but we only care about two events:
 * - we are connected to the AP with an IP
 * - we failed to connect after the maximum amount of retries */
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT      BIT1

static const char *TAG = "wifi station";

static int s_retry_num = 0;


static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data)
{
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < EXAMPLE_ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

void wifi_init_sta(void)
{
    s_wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                        ESP_EVENT_ANY_ID,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_any_id));
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                        IP_EVENT_STA_GOT_IP,
                                                        &event_handler,
                                                        NULL,
                                                        &instance_got_ip));

    wifi_config_t wifi_config = {
        .sta = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .password = EXAMPLE_ESP_WIFI_PASS,
            /* Authmode threshold resets to WPA2 as default if password matches WPA2 standards (password len => 8).
             * If you want to connect the device to deprecated WEP/WPA networks, Please set the threshold value
             * to WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK and set the password with length and format matching to
             * WIFI_AUTH_WEP/WIFI_AUTH_WPA_PSK standards.
             */
            .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
            .sae_pwe_h2e = ESP_WIFI_SAE_MODE,
            .sae_h2e_identifier = EXAMPLE_H2E_IDENTIFIER,
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK(esp_wifi_start() );

    ESP_LOGI(TAG, "wifi_init_sta finished.");

    /* Waiting until either the connection is established (WIFI_CONNECTED_BIT) or connection failed for the maximu
     * number of re-tries (WIFI_FAIL_BIT). The bits are set by event_handler() (see above) */
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
            WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
            pdFALSE,
            pdFALSE,
            portMAX_DELAY);

    /* xEventGroupWaitBits() returns the bits before the call returned, hence we can test which event actually
     * happened. */
    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
                 EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
}

}

void connect_wifi()
{
    //Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
    wifi_init_sta();
}
```