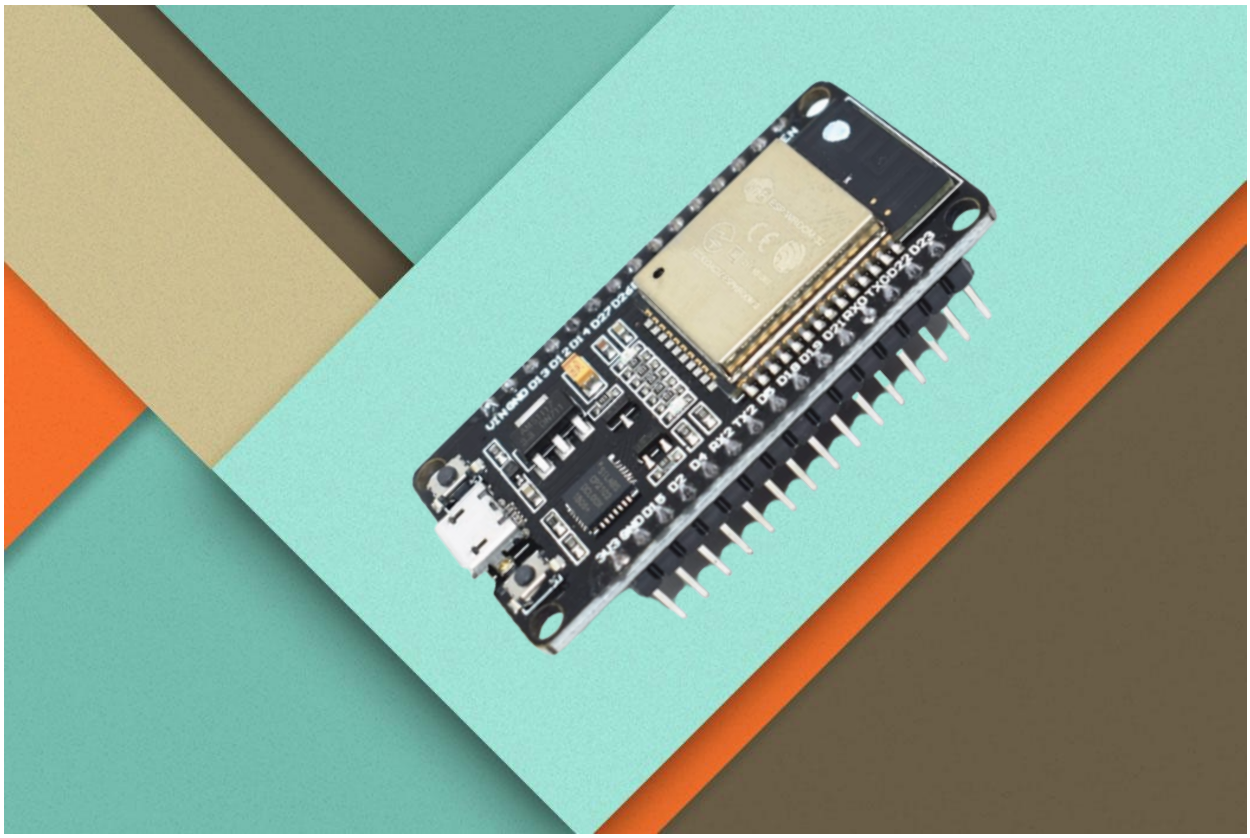


MANUAL DRIVER UART (ESP32)

DICIEMBRE 2022

Autores:
Axel Gay Díaz
Anahí González Holguin
Carlos Alberto González Vázquez
Carlos Eduardo López Lara

ÍNDICE

1. Generalidades

1.1 Recomendaciones del programador

Si nunca se ha hecho uso de la unidad ESP32, en el siguiente link se da información a detalle:

ESP32 Wi-Fi & Bluetooth MCU | Espressif Systems

En esta página se encuentran cosas como:

- Información técnica
- Información de Módulos y Kits de desarrollo
- Videos con ejemplos de aplicación

Se proporcionarán enlaces para la instalación de la IDE recomendada:

Guía de instalación Espressif-IDE (ES):

1) Espressif ESP32 - ESP-IDF - Instalación y Primer Ejemplo - YouTube

Guía de instalación Espressif-IDE (EN):

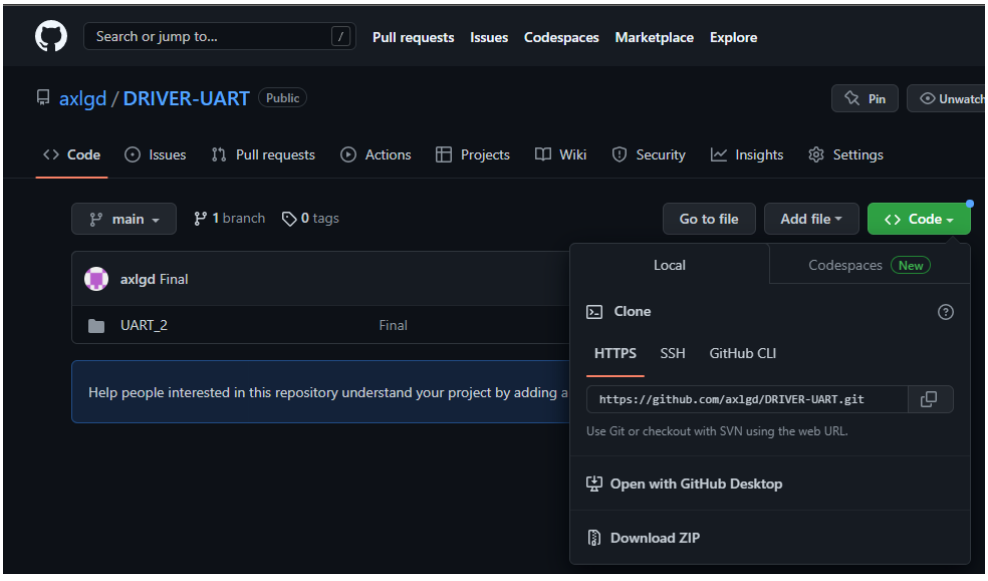
Getting started with ESP-IDF programming using espressif IDE - YouTube

1.2 Descarga e Implementación del Driver

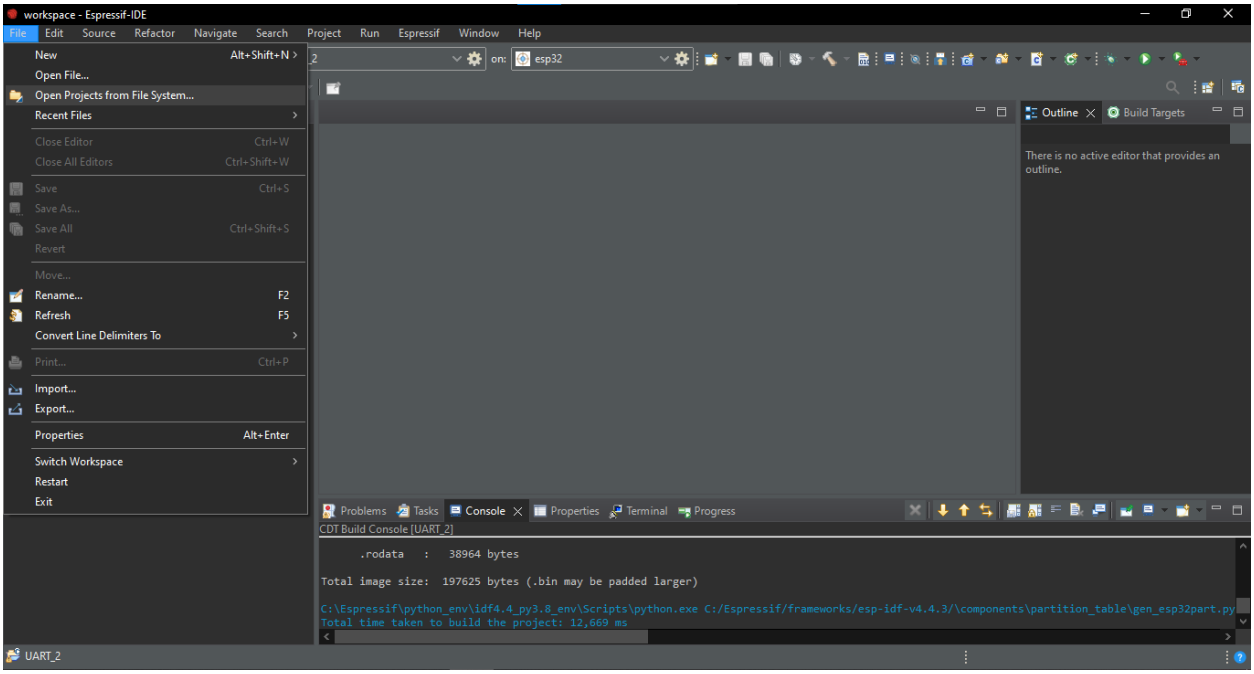
1. Accede al siguiente enlace de GitHub:

https://github.com/axlgd/DRIVER-UART.git

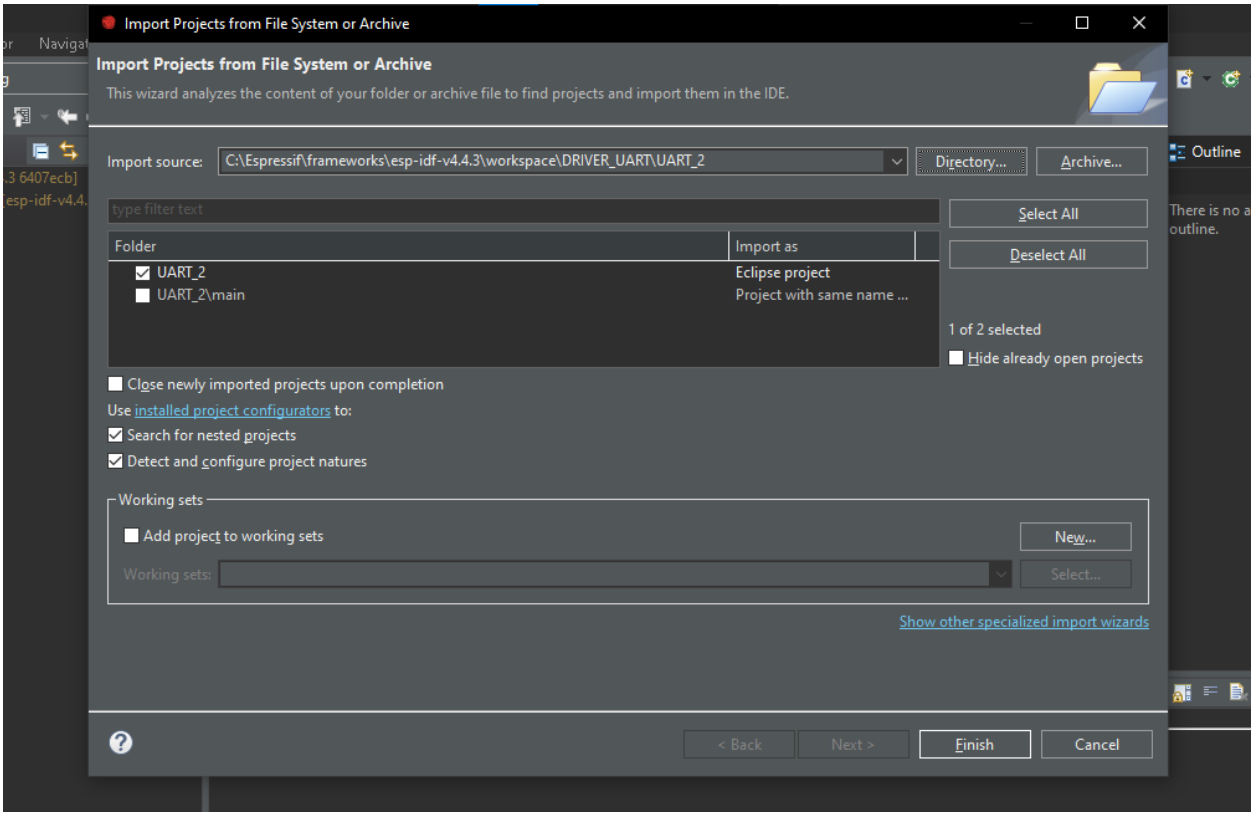
2. Da click en Code y en Download ZIP dentro del repositorio (UART_2):



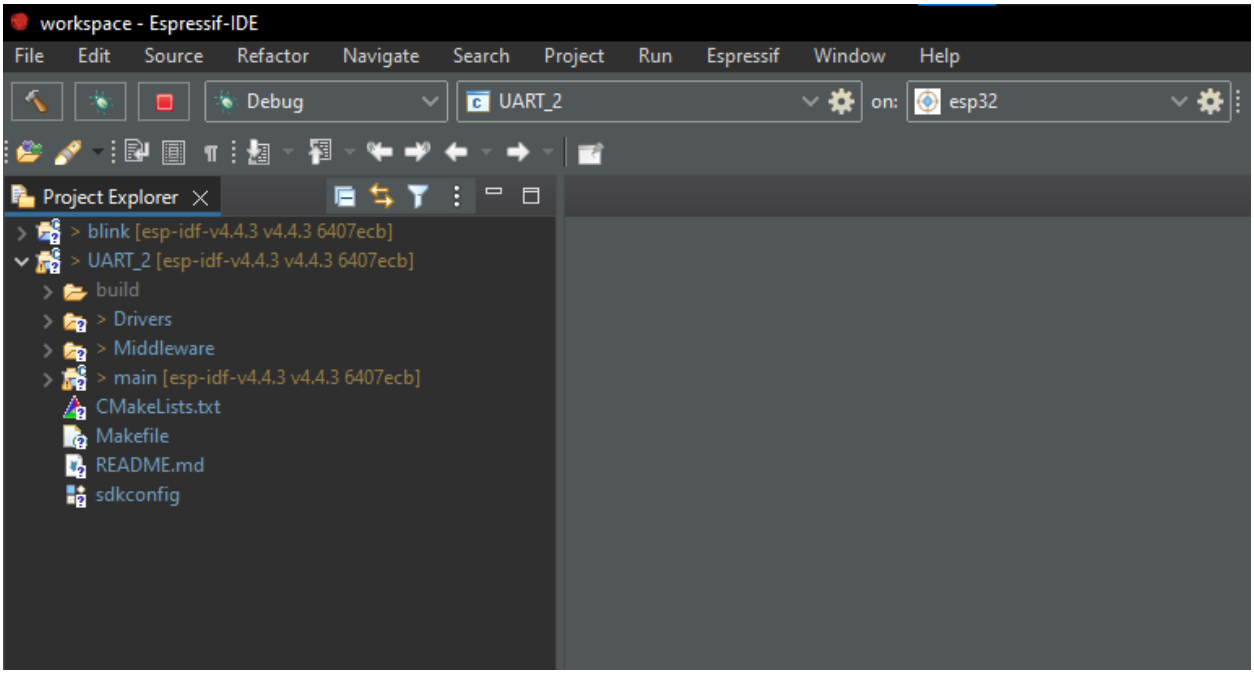
3. Seleccionamos la opción de abrir proyecto del sistema:



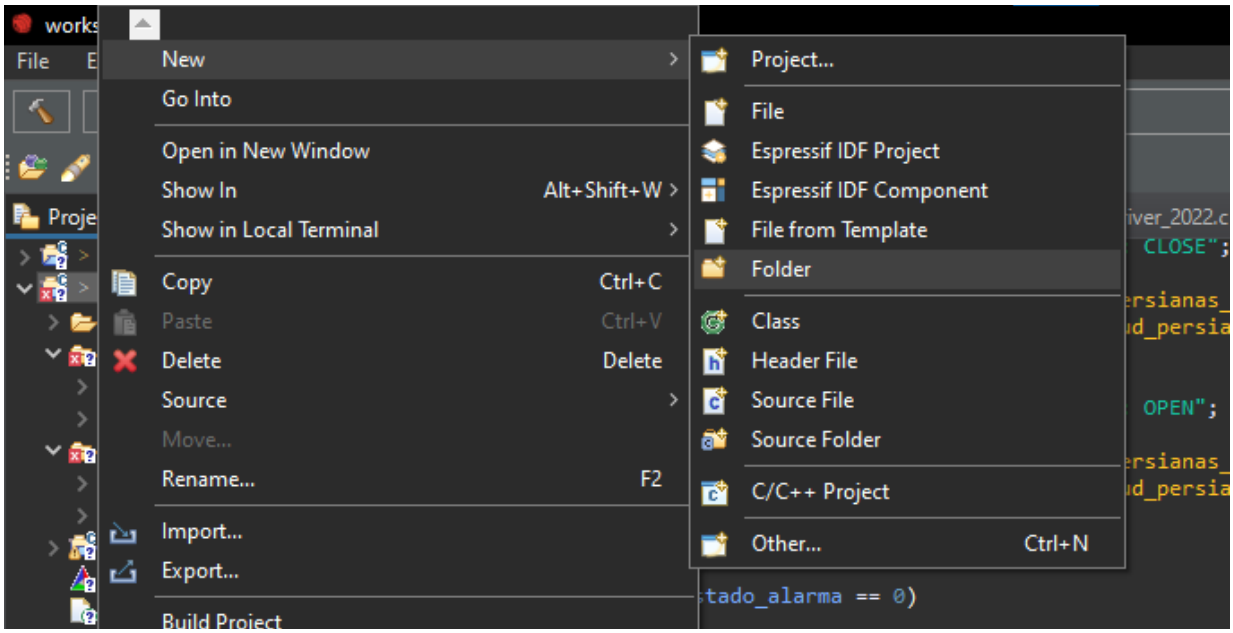
4. Seleccionamos la ruta del proyecto y damos click en Finish:



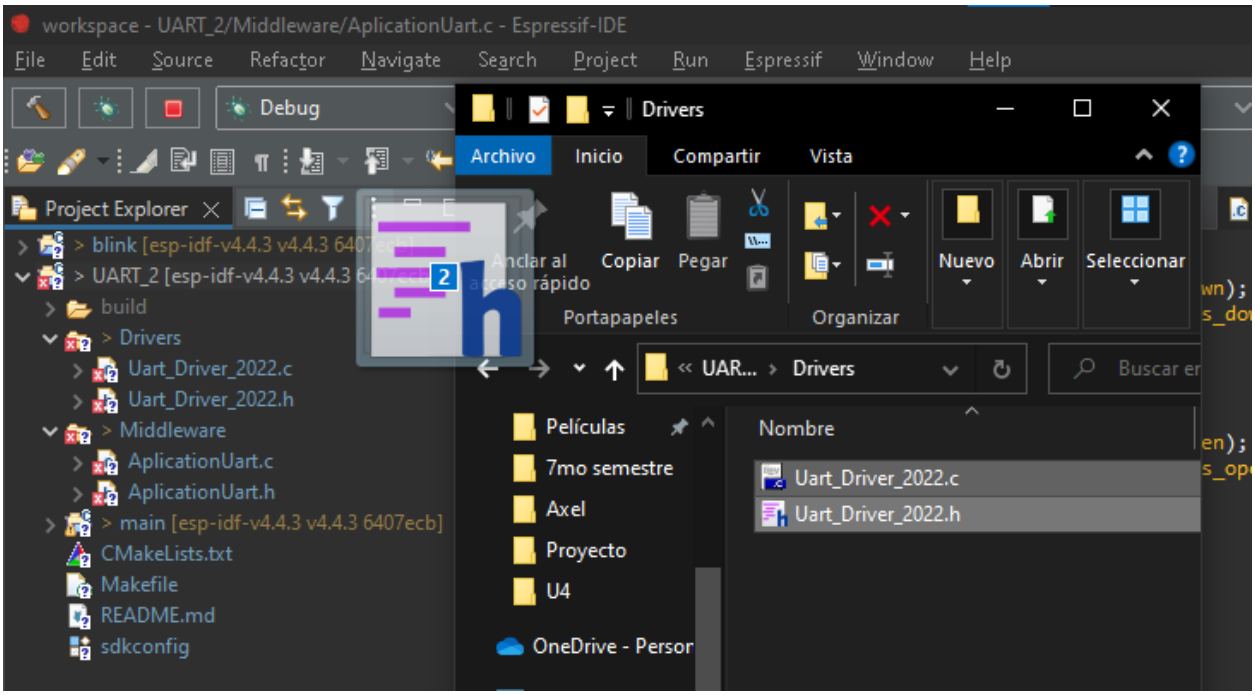
5. Después nos deberá aparecer el proyecto a la izquierda:



6. En caso de que requiera agregar los drivers a otro proyecto que no sea el de ejemplo, solo necesita seguir los siguientes pasos:
    a) Crear una nueva carpeta dentro de su proyecto y nombrarla.



b) Arrastrar los archivos Uart_Driver_2022.c y Uart_Driver_2022.h a la carpeta que recién creaste.



2. Funcionamiento General

2.1 Archivos de cabecera (.h)

En estos archivos se encuentran todas las definiciones y redefiniciones de funciones y constantes necesarias para la configuración del UART, pines y un botón externo.

2.1.1 Uart_Driver_2022.h

Uart_Driver_2022.h

```h
#pragma once

//Librerías necesarias
#include <stdio.h>
#include <stdlib.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"


//CONFIGURACIÓN PARA SELECCIONAR PINES DE LA ESP Y CARACTERÍSTICAS DE LA UART
#define ECHO_TEST_TXD (GPIO_NUM_17)         //PIN TXD
#define ECHO_TEST_RXD (GPIO_NUM_16)         //PIN RXD
#define ECHO_TEST_RTS (UART_PIN_NO_CHANGE)
#define ECHO_TEST_CTS (UART_PIN_NO_CHANGE)

#define ECHO_UART_PORT_NUM      (CONFIG_EXAMPLE_UART_PORT_NUM) //NUMERO DE PUERTO DE UART
#define ECHO_UART_BAUD_RATE     (CONFIG_EXAMPLE_UART_BAUD_RATE) //BAUD RATE
#define ECHO_TASK_STACK_SIZE    (CONFIG_EXAMPLE_TASK_STACK_SIZE)

//Redefiniciones del driver
#define TX_PIN (ECHO_TEST_TXD)
#define RX_PIN (ECHO_TEST_RXD)


#define UART_PORT   (ECHO_UART_PORT_NUM)
#define BAUD_RATE   (ECHO_UART_BAUD_RATE)
#define DATA_BITS   (UART_DATA_8_BITS) // BITS DE DATOS
#define PARITY      (UART_PARITY_DISABLE) // PARIDAD
#define STOP_BITS   (UART_STOP_BITS_1) //BITS DE PARO


static const char *TAG = "UART TEST";

#define BUF_SIZE (1024)

/*****************************************************************************
* Function: uart_config
* Preconditions: Ninguna.
* Overview: Accede a los registros por medio de una estructura para configurar la UART.
* Input: Ninguno.
* Output: Ninguno.
*
*****************************************************************************/
extern void uart_config(void);

/*****************************************************************************
* Function: uart_read
* Preconditions: data, len.
* Overview: Permite recibir información vía UART desde otra terminal.
* Input: *data. Dirección de memoria de la variable "data", el arreglo dentro
* de memoria dinámica que guarda lo que se reciba de la UART.
* Output: int len. Entero que guarda el índice del arreglo data donde se guardó la
* información que recibió la UART.
*****************************************************************************/
```

```
73  extern int uart_read(uint8_t *data);
74
75  /**************************************************************************
76   * Function: uart_write
77   * Preconditions: data, len.
78   * Overview: Permite escribir en la terminal.
79   * Input:
80   * *data. Dirección de memoria de la variable "data", el arreglo dentro
81   * de memoria dinámica que guarda lo que se recibe de la UART.
82   * int len. Entero que guarda el índice del arreglo data donde se guardó la
83   * información que recibió la UART.
84   * Output: Ninguna.
85   **************************************************************************/
86  extern void uart_write(uint8_t *data, int len);
87
```

2.2 Funciones (.c)

En estos archivos se encuentra la implementación de todas las funciones utilizadas en el Driver.

2.2.1 Uart_Driver_2022.c

uart_config()

Esta función es la que se usa para configurar el UART accediendo a memoria mediante una estructura.

```c
void uart_config(void)
{
    uart_config_t uart_config = {
        .baud_rate = BAUD_RATE, //Baud Rate 115 200
        .data_bits = DATA_BITS, //8 bits de datos
        .parity    = PARITY,    //Sin paridad
        .stop_bits = STOP_BITS, //1 bit de paro
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
        .source_clk = UART_SCLK_APB,
    };
    int intr_alloc_flags = 0;

#if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
#endif

    //Instalación del driver para la UART
    ESP_ERROR_CHECK(uart_driver_install(UART_PORT, BUF_SIZE * 2, 0, 0, NULL, intr_alloc_flags));
    //Configuración de la configuración de la UART ANTES DE ENTONCES ANTES definida
    ESP_ERROR_CHECK(uart_param_config(UART_PORT, &uart_config));
    //SETTER de los pines RXD, TXD y el número de puerto de la UART.
    ESP_ERROR_CHECK(uart_set_pin(UART_PORT, ECHO_TEST_TXD, ECHO_TEST_RXD, ECHO_TEST_RTS, ECHO_TEST_CTS));
}
```

uart_read()

Esta función es la que permite recibir información desde otra terminal a través de UART.

```c
int uart_read(uint8_t *data)
{
    int len = uart_read_bytes(ECHO_UART_PORT_NUM, data, (BUF_SIZE - 1), 20 / portTICK_RATE_MS);
    return len;
}
```

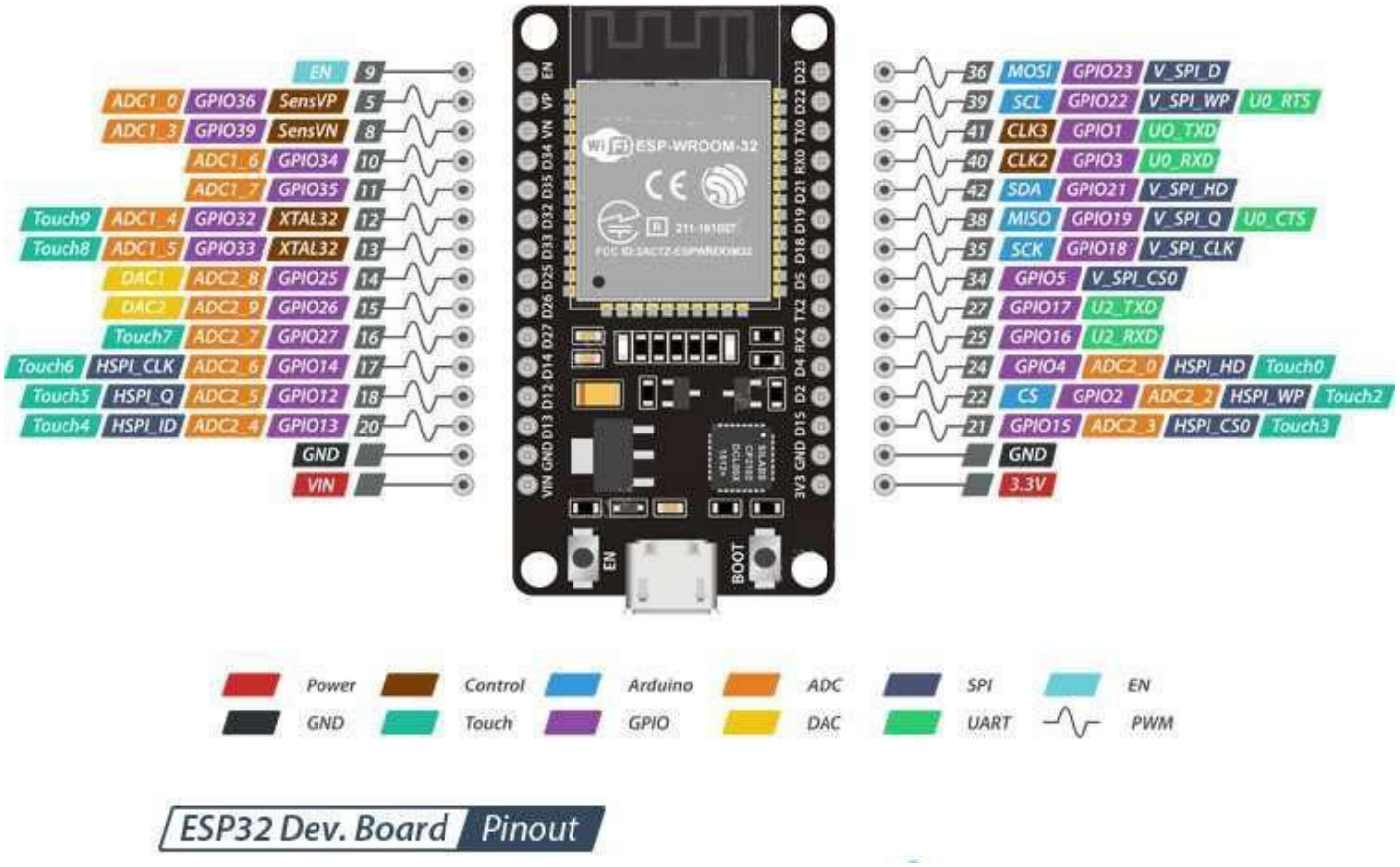
uart_write()

Esta función nos permite escribir en la terminal.

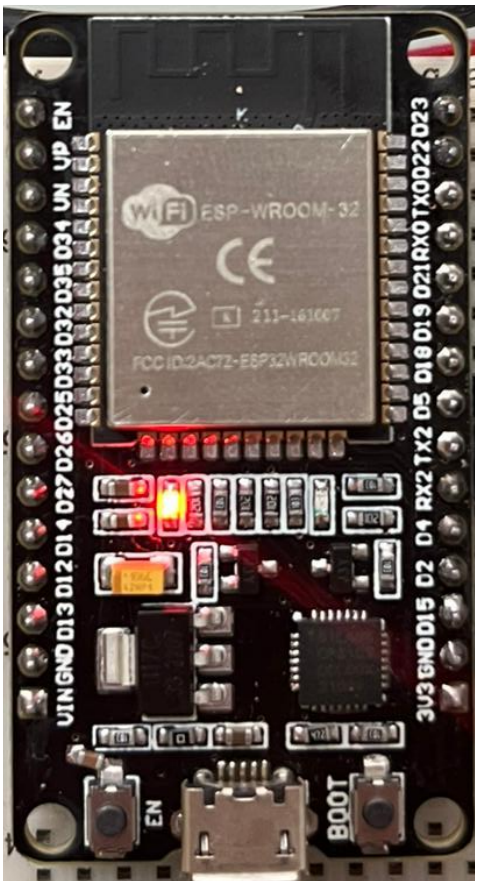
```
89  void uart_write(uint8_t *data, int len)
90  {
91      uart_write_bytes(ECHO_UART_PORT_NUM, (const char *) data, len);
92              if (len) {
93                  data[len] = '\0';
94                  ESP_LOGI(TAG, "Recv str: %s", (char *) data);
95               }
96  }
```

3. Configuración del DevKitC V1

3.1 Pinout


EN 9
ADC1_0 GPIO36 SensVP 5
ADC1_3 GPIO39 SensVN 8
ADC1_6 GPIO34 10
ADC1_7 GPIO35 11
Touch9 ADC1_4 GPIO32 XTAL32 12
Touch8 ADC1_5 GPIO33 XTAL32 13
DAC1 ADC2_8 GPIO25 14
DAC2 ADC2_9 GPIO26 15
Touch7 ADC2_7 GPIO27 16
Touch6 HSPI_CLK ADC2_6 GPIO14 17
Touch5 HSPI_Q ADC2_5 GPIO12 18
Touch4 HSPI_ID ADC2_4 GPIO13 20
GND
VIN
36 MOSI GPIO23 V_SPI_D
39 SCL GPIO22 V_SPI_WP U0_RTS
41 CLK3 GPIO1 U0_TXD
40 CLK2 GPIO3 U0_RXD
42 SDA GPIO21 V_SPI_HD
38 MISO GPIO19 V_SPI_Q U0_CTS
35 SCK GPIO18 V_SPI_CLK
34 GPIO5 V_SPI_CS0
27 GPIO17 U2_TXD
25 GPIO16 U2_RXD
24 GPIO4 ADC2_0 HSPI_HD Touch0
22 CS GPIO2 ADC2_2 HSPI_WP Touch2
21 GPIO15 ADC2_3 HSPI_CS0 Touch3
GND
3.3V
ESP-WROOM-32
EN
BOOT
Power
Control
Arduino
ADC
SPI
EN
GND
Touch
GPIO
DAC
UART
PWM
ESP32 Dev. Board Pinout

3.2 Características físicas



Algunas de sus principales características físicas son:

- SoC Tensilica Xtensa 32-bit LX6
- 2 botones, uno para activar el modo Bootloader, y otro para Reset
- 30 pines, entre los que encontramos para GPIO, UART, GND, alimentación de 5V o 3.3V, ADC, etc.
- Módulos WiFi y Bluetooth
- Puerto Micro USB
- LED indicador

4. Ejemplos de uso

4.1 AplicationUart.h

En este archivo de cabecera encontramos las definiciones de funciones y constantes a utilizar en el ejemplo de aplicación.

```h
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include "C:\Espressif\frameworks\esp-idf-v4.4.3\workspace\UART_2\Drivers\Uart_Driver_2022.h"
#include "C:\Espressif\frameworks\esp-idf-v4.4.3\workspace\UART_2\Drivers\Uart_Driver_2022.c"
#include <unistd.h>
#include <TIME.H>
#include "esp_log.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "esp_log.h"

//Definiciones para el código de aplicación
#define On      1
#define Off     0
#define Open    1
#define Close   0

/*********************************************************************
* Function: sistema_init
* Preconditions:
* Overview: Imprime que ha iniciado el progcama.
* Input: Ninguna.
* Output: Ninguna.
*********************************************************************/
extern void sistema_init(void);

/*********************************************************************
* Function: imprimir_menu
* Preconditions:
* Overview: Imprime el menú de la aplicación
* Input: Ninguna.
* Output: Ninguna.
*********************************************************************/
extern void imprimir_menu(void);

/*********************************************************************
* Function: imprimir_estado
* Preconditions:
* Overview: Imprime el estado de cada componente del sistema.
* Input: uint8_t estado_luces, uint8_t estado_persianas, uint8_t estado_alarma
* Output: Ninguna.
*********************************************************************/
extern void imprimir_estado(uint8_t estado_luces, uint8_t estado_persianas, uint8_t estado_alarma);

```

4.2 AplicationUart.c

En este archivo se encuentra la implementación de cada una de las funciones a utilizar en el ejemplo de aplicación.

sistema_init()

Esta función nos indica que se ha iniciado el programa, imprimiendo en la terminal e indicándonos que presionemos una tecla para iniciar.

```
40  void sistema_init(void)
41  {
42      char cadena[] = "El programa ha iniciado\n\r Presione una tecla para iniciar";
43      uint8_t longitud;
44      longitud = sizeof(cadena);
45      uart_write(&cadena, longitud);
46  }
```

imprimir_menu()

Esta función imprime el menú de la aplicación, desplegando todas las opciones disponibles, las cuales son para el control de las luces, persianas, alarma o apagar el sistema.

```
void imprimir_menu(void)
{
    char luces[] = "1.Prender/Apagar luces";
        uint8_t longitud_luces;
        longitud_luces = sizeof(luces);
        uart_write(&luces, longitud_luces);
    char persianas[] = "2.Subir/Bajar persianas";
            uint8_t longitud_persianas;
            longitud_persianas = sizeof(persianas);
            uart_write(&persianas, longitud_persianas);
    char alarma[] = "3.Prender/Apagar alarma";
                uint8_t longitud_alarma;
                longitud_alarma = sizeof(alarma);
                uart_write(&alarma, longitud_alarma);
    char apagar[] = "4.Apagar el sistema";
                    uint8_t longitud_apagar;
                    longitud_apagar = sizeof(apagar);
                    uart_write(&apagar, longitud_apagar);
    char opcion[] = "Presione el boton de menu:";
                        uint8_t longitud_opcion;
                        longitud_opcion = sizeof(opcion);
                        uart_write(&opcion, longitud_opcion);
}
```

imprimir_estado()

Esta función imprime el estado de cada componente del sistema .

```
void imprimir_estado(uint8_t estado_luces, uint8_t estado_persianas, uint8_t estado_alarma)
{
    char Micasa[] = "Mi casa";
    uint8_t longitud_micasa;
    longitud_micasa = sizeof(Micasa);
    uart_write(&Micasa, longitud_micasa);

    if(estado_luces == 0)
    {
        char luces_off[] = "Luces: OFF";
        uint8_t longitud_luces_off;
        longitud_luces_off = sizeof(luces_off);
        uart_write(&luces_off, longitud_luces_off);
    }else
    {
        char luces_on[] = "Luces: ON";
        uint8_t longitud_luces_on;
        longitud_luces_on = sizeof(luces_on);
        uart_write(&luces_on, longitud_luces_on);
    }
    if(estado_persianas == 0)
    {
        char persianas_down[] = "Persianas: CLOSE";
        uint8_t longitud_persianas_down;
        longitud_persianas_down = sizeof(persianas_down);
        uart_write(&persianas_down, longitud_persianas_down);
    }else
    {
        char persianas_open[] = "Persianas: OPEN";
        uint8_t longitud_persianas_open;
        longitud_persianas_open = sizeof(persianas_open);
        uart_write(&persianas_open, longitud_persianas_open);
    }


    if(estado_alarma == 0)
    {
        char alarma_off[] = "Alarma: Off";
        uint8_t longitud_alarma_off;
        longitud_alarma_off = sizeof(alarma_off);
        uart_write(&alarma_off, longitud_alarma_off);
    }else
    {
        char alarma_on[] = "Alarma: On";
        uint8_t longitud_alarma_on;
        longitud_alarma_on = sizeof(alarma_on);
        uart_write(&alarma_on, longitud_alarma_on);
    }


}
```

4.3 Enlaces

Enlaces:

-Archivo comprimido del Driver UART:

https://github.com/axlgd/DRIVER-UART.git

-Videotutoriales (Lista de Reproducción):

https://www.youtube.com/playlist?list=PLUqGq5mzysq71o7x0OJAr6qn0i-qRyWpq