



UNIT DevLab desde cero

Versión 0.0.1

Cesar Bautista

04 de marzo de 2025

Contenido

1. MicroPython	3
1.1. Compatibilidad de Placas	3
1.2. Características Avanzadas	4
1.3. Comenzando con MicroPython	4
1.4. Bibliotecas y Módulos	5
1.5. IDEs y Herramientas	5
2. Guía de Instalación de Bibliotecas con MIP	7
2.1. Biblioteca DualMCU	8
3. Arduino IDE	11
3.1. Historia	11
3.2. Características Principales	11
3.3. Diferencias Clave entre Arduino y un Entorno Nativo	12
3.4. Casos de Uso	13
4. Guía de Instalación de Paquetes de Unit Electronics	15
4.1. Prerequisitos	15
4.2. Instalación Rápida	15
4.3. 1. Instalación del Paquete de Placa DualMCU-ONE	16
4.4. 2. Instalación del Paquete de Placa Cocket Nova CH552	16
5. Conector JST SH 1.0mm 4 pines	19
5.1. Ejemplo de integración del conector JST SH 1.0mm 4 pines	21
6. Salidas digitales	23
6.1. Parpadeo (blink)	23
7. Modulación por ancho de pulso (PWM)	27
7.1. Implementación	27
7.2. Aplicaciones	29
8. Entradas digitales	33
8.1. Pull-up y Pull-down	33
8.2. Lectura de Entradas	34
8.3. Arduino IDE y SDCC	35
9. Conversión de Analógico a Digital	37

9.1.	Definición de ADC	37
9.2.	Código de Ejemplo	39
9.3.	Arduino IDE y SDCC	40
10.	I2C (Inter-Integrated Circuit)	41
10.1.	Descripción General del I2C	41
10.2.	Ejemplo de aplicación Pantalla SSD1306	42
10.3.	Cocket Nova implementación de I2C	44
11.	Ejemplo de Integración	47
12.	SPI (Interfaz Periférica Serial)	51
12.1.	Visión General del SPI	51
12.2.	SDCard SPI	55
12.3.	SDIO (Interfaz de Entrada/Salida Segura Digital)	58
13.	Interfaz USB	65
13.1.	HID (Dispositivo de Interfaz Humano)	65
13.2.	CDC (Clase de Dispositivo de Comunicación)	66
13.3.	MIDI (Interfaz Digital de Instrumentos Musicales)	66
13.4.	Micrófono USB	66
14.	Control WS2812	69
14.1.	MicroPython y Arduino IDE	70
14.2.	Control WS2812 con SDCC	70
15.	Cómo Generar un Informe de Error	73
15.1.	Pasos para Crear un Informe de Error:	73
16.	Índices y tablas	75

UNIT DevLab desde cero. Presenta el ecosistema de desarrollo de UNIT Electronics el cual está diseñado para ofrecer el mayor soporte posible en aplicaciones específicas dentro del desarrollo tecnológico. Actualmente, la línea de productos incluye la familia DualMCU y la Cocket Nova, con la expectativa de expandirse hacia nuevas tecnologías en el futuro.

CAPÍTULO 1

MicroPython

MicroPython es una implementación ligera y eficiente de Python 3. Está optimizado para microcontroladores y entornos restringidos, lo que lo hace ideal para sistemas embebidos e IoT. Proporciona un conjunto amplio de bibliotecas y módulos para interactuar con hardware y periféricos.

1.1 Compatibilidad de Placas

La siguiente tabla enumera las placas compatibles con MicroPython:

Nota: Para descargar MicroPython y obtener más información sobre su instalación en placas específicas, visite el [sitio web oficial de MicroPython](#).

Tabla 1.1: Placas compatibles con MicroPython

Placa	Descripción y Recursos
ESP32	Microcontrolador de bajo costo con Wi-Fi y Bluetooth, ideal para aplicaciones IoT. Amplio soporte y documentación.
RP2040	Chip utilizado en Raspberry Pi Pico, perfecto para proyectos educativos y sistemas embebidos.
STM32	Microcontroladores de alto rendimiento para aplicaciones en tiempo real (p. ej., STM32F4 y STM32F7).
nRF52	Dispositivos de bajo consumo, adecuados para aplicaciones BLE, con soporte respaldado por ejemplos prácticos.
Pyboard	Placa oficial de MicroPython, diseñada específicamente para aprovechar las capacidades del lenguaje.

1.2 Características Avanzadas

MicroPython combina el rendimiento en dispositivos con pocos recursos con la versatilidad de Python 3. Entre sus principales características se destacan:

- REPL interactivo para pruebas y depuración en tiempo real.
- Bibliotecas optimizadas para el manejo de pines GPIO, I2C, SPI, UART, etc.
- Eficiencia en el uso de recursos, ideal para microcontroladores.
- Compatibilidad con gran parte del ecosistema Python.
- Facilidad de integración y extensión para soluciones IoT y embebidas.

1.2.1 Sintaxis y Tipado en MicroPython

La sintaxis de MicroPython es similar a Python 3, con tipado dinámico que facilita el desarrollo. A continuación, se presenta una comparativa de sus características:

Tabla 1.2: Comparativa de características

Característica	Descripción
Sintaxis Simplificada	Legibilidad y estructura claras, heredadas de Python 3.
Tipado Dinámico	No requiere declaración explícita de tipos.
Optimización para Hardware Limitado	Funciones adaptadas para minimizar el consumo de memoria.
Amplia Compatibilidad	Integración con gran parte del ecosistema y bibliotecas de Python.

1.2.2 Más Información y Recursos

Para profundizar en el uso de MicroPython se recomienda:

- Revisar la documentación oficial en el [sitio de MicroPython](#).
- Participar en comunidades y foros especializados.
- Utilizar recursos interactivos y cursos en línea que faciliten la adopción del entorno.

1.3 Comenzando con MicroPython

Pasos para iniciar tu proyecto:

1. Descarga el firmware adecuado para tu placa desde el [sitio oficial](#).

Truco: A continuación, se presentan enlaces directos a los firmwares de MicroPython para ESP32 y RP2040:

Tabla 1.3: Firmwares de MicroPython

Placa	Firmware
ESP32	Micropython DualMCU ESP32 bin
RP2040	Micropython DualMCU RP2040 uf2

Nota: Requerido para flashear el firmware en el ESP32 o [esptools](#)

2. Conecta tu dispositivo mediante Thonny o cualquier otro IDE compatible con MicroPython.
3. Consulta la [documentación y ejemplos para crear aplicaciones embebidas](#), desde prototipos hasta soluciones IoT.

1.4 Bibliotecas y Módulos

MicroPython ofrece un rico conjunto de módulos:

- **machine**: Interacción con hardware (GPIO, I2C, SPI, UART, etc.).
- **network**: Gestión de interfaces de red (Wi-Fi, Ethernet, etc.).

1.5 IDEs y Herramientas

Entre las herramientas recomendadas para trabajar con MicroPython se encuentran:

- **Thonny**: IDE sencillo y amigable, con soporte para MicroPython.
- **uPyCraft**: IDE ligero que facilita la edición, transferencia de archivos y comunicación serial.
- **rshell**: Herramienta de línea de comandos para gestionar archivos en la placa mediante conexión serial.

CAPÍTULO 2

Guía de Instalación de Bibliotecas con MIP

Nota: El soporte directo para mip en RP2040 no está disponible. Se utiliza la biblioteca *mip* para instalar la biblioteca *max1704x.py*.

- Dispositivo ESP32.
- IDE Thonny.
- Credenciales de Wi-Fi (SSID y contraseña).

Para instalar la biblioteca *max1704x.py*, siga estos pasos:

1. Conecte a Wi-Fi ejecutando el siguiente código en Thonny:

```
import mip
import network
import time

def connect_wifi(ssid, password):
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid, password)

    for _ in range(10):
        if wlan.isconnected():
            print('Conectado a la red Wi-Fi')
            return wlan.ifconfig()[0]
        time.sleep(1)

    print('No se pudo conectar a la red Wi-Fi')
    return None

ssid = "tu_ssid"
```

(continúe en la próxima página)

(proviene de la página anterior)

```
password = "tu_contraseña"

ip_address = connect_wifi(ssid, password)
print(ip_address)
```

2. Instale las bibliotecas utilizando mip:

```
mip.install('https://raw.githubusercontent.com/UNIT-Electronics/MAX1704X_lib/refs/heads/
↪main/Software/MicroPython/example/max1704x.py')
mip.install('https://raw.githubusercontent.com/Cesarbautista10/Libraries_compatibles_
↪with_micropython/refs/heads/main/Libs/oled.py')
mip.install('https://raw.githubusercontent.com/Cesarbautista10/Libraries_compatibles_
↪with_micropython/refs/heads/main/Libs/sdcard.py')
```

2.1 Biblioteca DualMCU

Para trabajar con la placa DualMCU ONE se debe instalar la biblioteca DualMCU.

1. Instale Thonny desde el [sitio web de Thonny](#).
2. En Thonny, vaya a **Herramientas -> Administrar paquetes**.
3. Busque `dualmcu` y haga clic en **Instalar**.

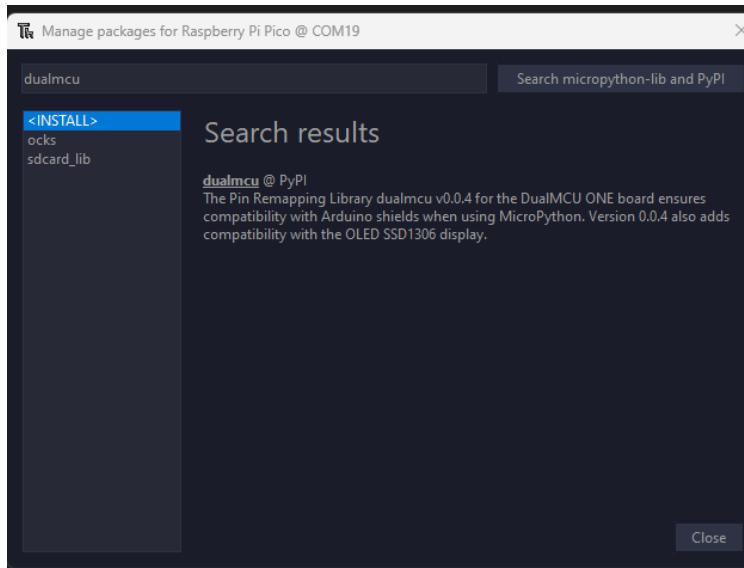


Figura 2.1: Biblioteca DualMCU

4. Verifique la instalación visual en:

Alternativamente, descargue la biblioteca desde [dualmcu.py](#).

La biblioteca DualMCU proporciona herramientas para trabajar con la placa DualMCU ONE. Ejemplo de uso básico:

```
import machine
from dualmcu import *
```

(continúe en la próxima página)

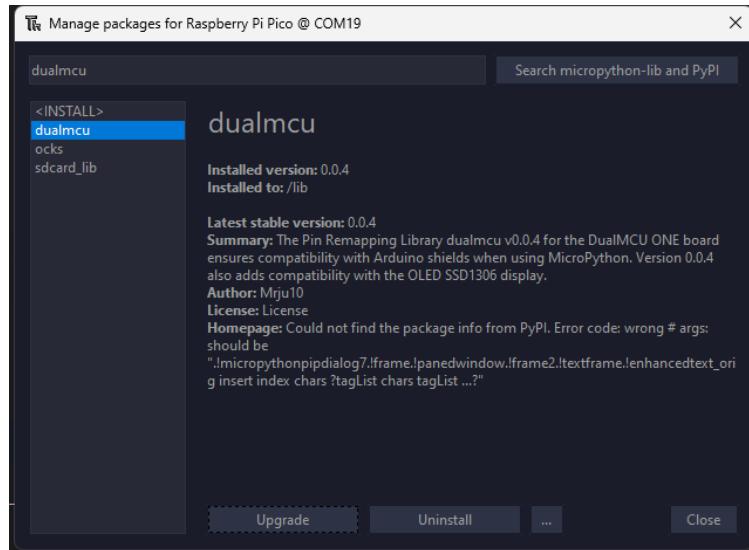


Figura 2.2: DualMCU instalada correctamente

(proviene de la página anterior)

```
i2c = machine.SoftI2C(scl=machine.Pin(22), sda=machine.Pin(21))

oled = SSD1306_I2C(128, 64, i2c)

oled.fill(1)
oled.show()

oled.fill(0)
oled.show()
oled.text('UNIT', 50, 10)
oled.text('ELECTRONICS', 25, 20)
oled.show()
```

- **Dualmcu:** Herramientas para trabajar con la placa DualMCU ONE.
- **Ocks:** Soporte para comunicación I2C.
- **SDcard-lib:** Gestión de tarjetas SD.

Para más información y actualizaciones, visite el [`repositorio de GitHub dualmcu`](#).

CAPÍTULO 3

Arduino IDE

3.1 Historia

El **Arduino Integrated Development Environment (IDE)** fue desarrollado para proporcionar una plataforma accesible y fácil de usar para programar placas Arduino y otros microcontroladores compatibles. Su primera versión se lanzó en 2005 junto con la primera placa Arduino, con el objetivo de democratizar el acceso a la programación de hardware para estudiantes, artistas y desarrolladores de todas las áreas.

A lo largo de los años, el IDE ha evolucionado, incorporando soporte para una amplia gama de microcontroladores y arquitecturas, desde los clásicos **ATmega328P** hasta plataformas avanzadas como **ESP32** y **RP2040**. Actualmente, existen dos versiones principales:

- **Arduino IDE 1.x:** Una versión clásica basada en Java, con una interfaz sencilla y un sistema de compilación estable.
- **Arduino IDE 2.x:** Una versión moderna basada en Electron, con funciones avanzadas como autocompletado, depuración en vivo y una mejor experiencia de usuario.

El ecosistema de **Arduino IDE** ha sido clave para el desarrollo de proyectos de electrónica y robótica debido a su facilidad de uso y compatibilidad con múltiples plataformas.

3.2 Características Principales

Arduino IDE se distingue por las siguientes características:

- **Interfaz Intuitiva:** Diseñada para facilitar la programación con una curva de aprendizaje baja.
- **Compatibilidad Multiplataforma:** Funciona en Windows, macOS y Linux.
- **Bibliotecas Integradas:** Permite el uso de cientos de bibliotecas para sensores, motores, comunicación inalámbrica, etc.
- **Gestor de Placas:** Soporte para agregar tarjetas adicionales como ESP32, STM32 y RP2040 mediante el **Board Manager**.

- **Compilador Simplificado:** Usa un preprocesador que oculta detalles avanzados de C++, como la declaración de prototipos de funciones.
- **Monitor Serie:** Herramienta integrada para depuración de datos en serie.

3.3 Diferencias Clave entre Arduino y un Entorno Nativo

3.3.1 Capa de Abstracción

Arduino proporciona una capa de abstracción que facilita la programación de microcontroladores mediante funciones de alto nivel:

```
pinMode(13, OUTPUT);
digitalWrite(13, HIGH);
delay(1000);
```

Los entornos nativos requieren configurar registros directamente, permitiendo un control más preciso sobre el hardware pero con mayor complejidad:

```
DDRB |= (1 << PB5); // Configura PB5 como salida
PORTB |= (1 << PB5); // Activa la salida
```

3.3.2 Compilador y Toolchain

- **Arduino** utiliza un **preprocesador** especial que maneja automáticamente detalles como la declaración de funciones.
- **Entornos nativos** como **ESP-IDF** y **Pico-SDK** utilizan compiladores como **GCC** o **Clang**, que permiten una mayor optimización del código.

3.3.3 Portabilidad y Optimización

Tabla 3.1: Comparación de Portabilidad y Optimización

Característica	Arduino IDE	Entornos Nativos
Portabilidad	Alto: el mismo código funciona en varias plataformas	Baja: optimizado para hardware específico
Rendimiento	Moderado	Alto: acceso directo al hardware
Consumo de memoria	Mayor debido a capas de abstracción	Reducido: control total del código

3.3.4 Soporte para Multitarea y RTOS

Arduino no ofrece soporte nativo para **sistemas operativos en tiempo real (RTOS)**. Sin embargo, algunos fabricantes han extendido la funcionalidad de Arduino con soporte multitarea:

- **ESP-IDF (ESP32):** Utiliza **FreeRTOS** para manejar múltiples tareas simultáneamente.
- **Pico-SDK (RP2040):** Permite gestionar tareas en **núcleos separados**, logrando procesamiento paralelo.

3.4 Casos de Uso

Arduino IDE es ampliamente utilizado en diversos ámbitos:

- **Educación:** Ideal para enseñanza de programación y electrónica.
- **Prototipado Rápido:** Desarrollo rápido de pruebas con sensores y actuadores.
- **IoT y Domótica:** Control de dispositivos conectados mediante Wi-Fi y Bluetooth.
- **Robótica:** Programación de robots autónomos y sistemas embebidos.

Gracias a su comunidad y ecosistema en crecimiento, **Arduino IDE** sigue siendo una herramienta fundamental para desarrolladores de hardware en todo el mundo.

CAPÍTULO 4

Guía de Instalación de Paquetes de Unit Electronics

Esta guía proporciona instrucciones paso a paso para instalar los paquetes de soporte de placas necesarios para programar las placas de desarrollo **DualMCU-ONE/DualMCU** (RP2040 + ESP32) y **Cocket Nova CH552** utilizando el entorno Arduino IDE. Estos paquetes permiten el desarrollo en el entorno de Arduino, asegurando una integración fluida con el hardware.

4.1 Prerequisitos

Antes de continuar, asegúrate de tener instaladas las siguientes herramientas:

- [Arduino IDE](#) – Requerido para programar las placas.
- [Controladores USB](#) – Necesarios para la comunicación con las placas.
- **Paquetes de soporte de placas:**
 - [DualMCU-ONE \(ESP32 + RP2040\)](#)
 - [Cocket Nova CH552](#)

4.2 Instalación Rápida

Copia y pega las siguientes URLs en el campo **URLs Adicionales del Gestor de Tarjetas** en las preferencias del Arduino IDE:

```
https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-ESP32-Arduino-Package/  
  ↵main/package_Uelectronics_esp32_index.json  
https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-RP2040-Arduino-Package/  
  ↵main/package_Uelectronics_rp2040_index.json  
https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-CH552-Arduino-Package/  
  ↵refs/heads/develop/package_duino_mcs51_index.json
```

Luego, busca las placas **Unit Electronics** en el **Gestor de Tarjetas** e instala todos los paquetes necesarios.

Si prefieres una instalación manual, sigue los pasos detallados a continuación.

4.3 1. Instalación del Paquete de Placa DualMCU-ONE

Paso 1: Instalar el Paquete ESP32

1. Abre **Arduino IDE**.
2. Ve a **Archivo > Preferencias**.
3. En el campo **URLs Adicionales del Gestor de Tarjetas**, ingresa la siguiente URL:

`https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-ESP32-Arduino-Package/main/package_Uelectronics_esp32_index.json`
4. Haz clic en **OK** para guardar las preferencias.
5. Ve a **Herramientas > Placa > Gestor de Tarjetas**.
6. Busca **DualMCU**.
7. Haz clic en **Instalar**.
8. Una vez instalado, selecciona **DualMCU** en el menú **Placas**.

Paso 2: Instalar el Paquete RP2040

1. Abre **Arduino IDE**.
2. Ve a **Archivo > Preferencias**.
3. En el campo **URLs Adicionales del Gestor de Tarjetas**, ingresa la siguiente URL:

`https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-RP2040-Arduino-Package/main/package_Uelectronics_rp2040_index.json`
4. Haz clic en **OK** para guardar las preferencias.
5. Ve a **Herramientas > Placa > Gestor de Tarjetas**.
6. Busca **RP2040**.
7. Haz clic en **Instalar**.
8. Una vez instalado, selecciona **RP2040** en el menú **Placas**.

4.4 2. Instalación del Paquete de Placa Cocket Nova CH552

Para programar la placa **Cocket Nova CH552** utilizando Arduino IDE, sigue estos pasos:

1. Abre **Arduino IDE**.
2. Ve a **Archivo > Preferencias**.
3. En el campo **URLs Adicionales del Gestor de Tarjetas**, ingresa la siguiente URL:

`https://raw.githubusercontent.com/UNIT-Electronics/Uelectronics-CH552-Arduino-Package/refs/heads/develop/package_duino_mcs51_index.json`

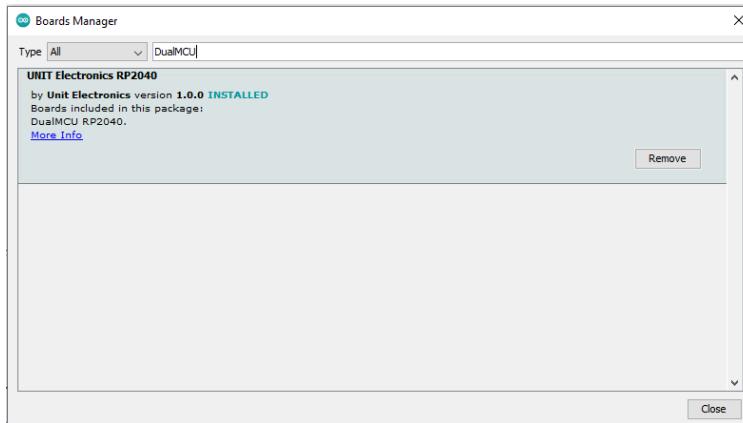


Figura 4.1: Ejemplo de instalación en el Gestor de Tarjetas.

4. Haz clic en **OK** para guardar las preferencias.
5. Ve a **Herramientas > Placa > Gestor de Tarjetas**.
6. Busca **Cocket Nova**.
7. Haz clic en **Instalar**.
8. Una vez instalado, selecciona **Cocket Nova** en el menú **Placas**.

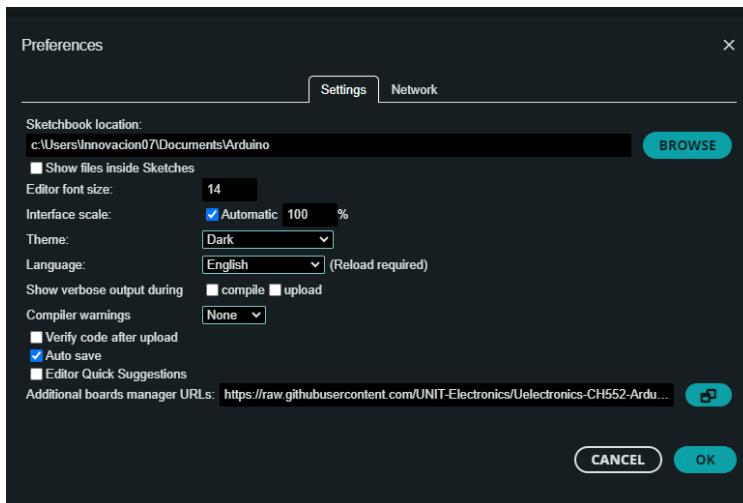


Figura 4.2: Ejemplo de instalación en el Gestor de Tarjetas.

Truco: Has instalado correctamente los paquetes necesarios para programar las placas de desarrollo **DualMCU-ONE** (ESP32 + RP2040) y **Cocket Nova CH552** en el Arduino IDE. ¡Ahora estás listo para comenzar a desarrollar tus proyectos!

Para documentación adicional e ideas de proyectos, visita [**UNIT Electronics**](<https://uelectronics.com/>).

CAPÍTULO 5

Conecotor JST SH 1.0mm 4 pines

El conector JST SH 1.0mm 4 pines es un conector que se utiliza para conectar las placas **DualMCU**, **DualMCU ONE** y **Cocket Nova** a dispositivos como sensores, actuadores y otros periféricos. El conector tiene un paso de 1.0mm y se usa comúnmente en dispositivos electrónicos pequeños.

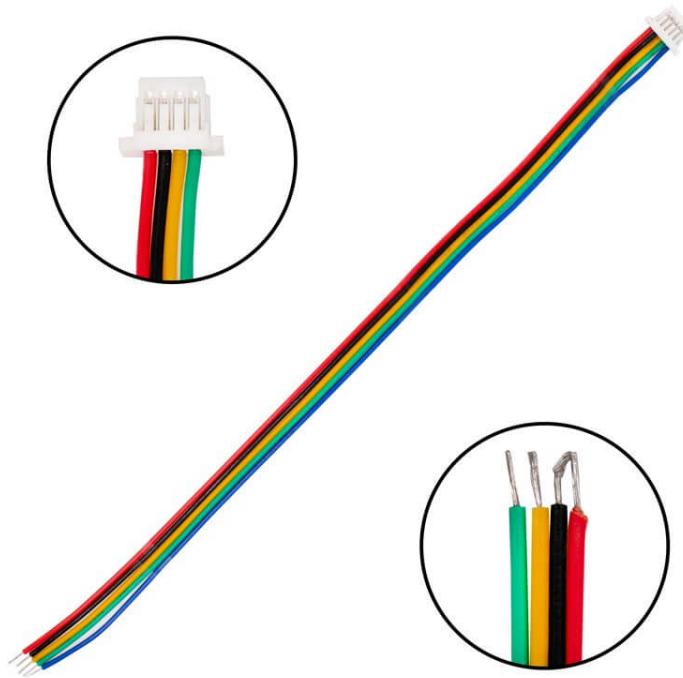


Figura 5.1: JST SH 1.0mm 4 pines

Para conectar el conector JST SH 1.0mm 4 pines a la placa DualMCU ONE, siga los pasos a continuación:

- Identificar el conector:** El conector JST SH 1.0mm 4 pines tiene un paso de 1.0mm y 4 pines.

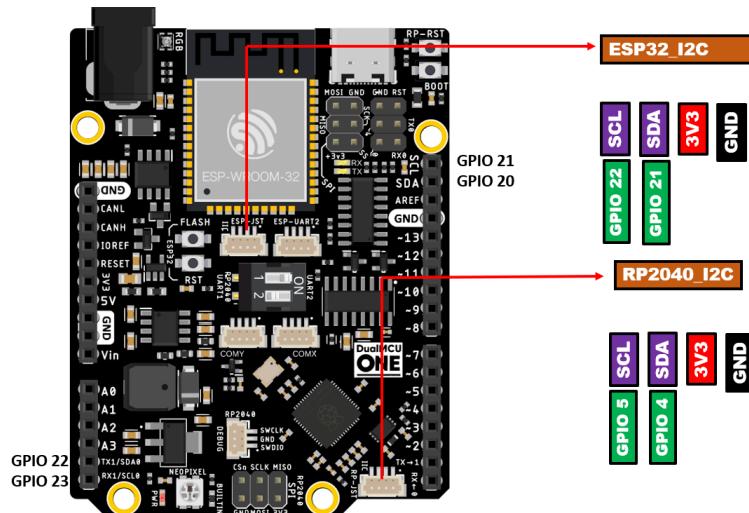


Figura 5.2: JST SH 1.0mm 4 pines (ejemplo para comunicación I2C)

- Conección:** Alinee el conector con los pinos en la placa DualMCU ONE y empuje suavemente el conector sobre los pinos.

Advertencia: A veces los colores del conector no corresponden a los pinos. Tenga cuidado al conectar el conector a la placa.

- Verificación:** Despues de conectar el conector JST SH 1.0mm 4 pines a la placa DualMCU ONE, verifique la conexión comprobando los pinos en la placa.



Figura 5.3: JST SH 1.0mm 4 pines conectado

Nota: El conector JST SH 1.0mm 4 pines es compatible con la placa desarrollada Cocket Nova.

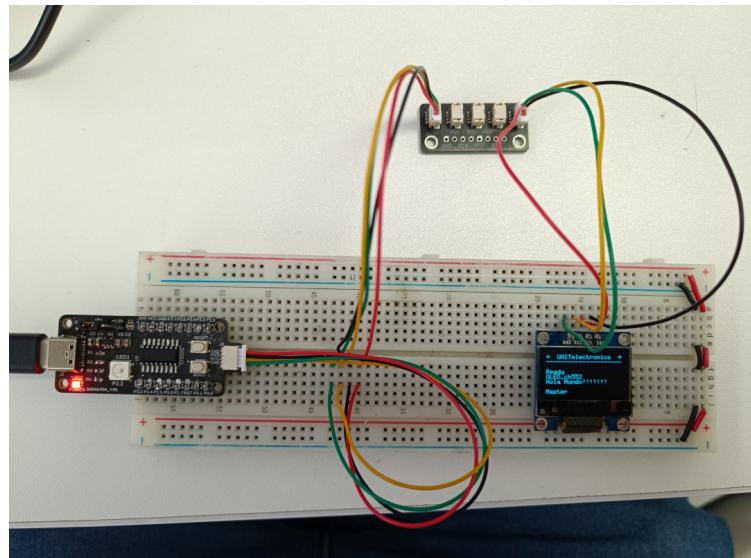


Figura 5.4: OLED a Cocket Nova

5.1 Ejemplo de integración del conector JST SH 1.0mm 4 pines

Este ejemplo demuestra cómo integrar el Cargador Boost LiPo UNIT y el Monitor I2C con un ESP32 y mostrar el estado de la batería en una pantalla OLED.

Para más detalles del ejemplo, visite el siguiente enlace: [Ejemplo de integración](#)

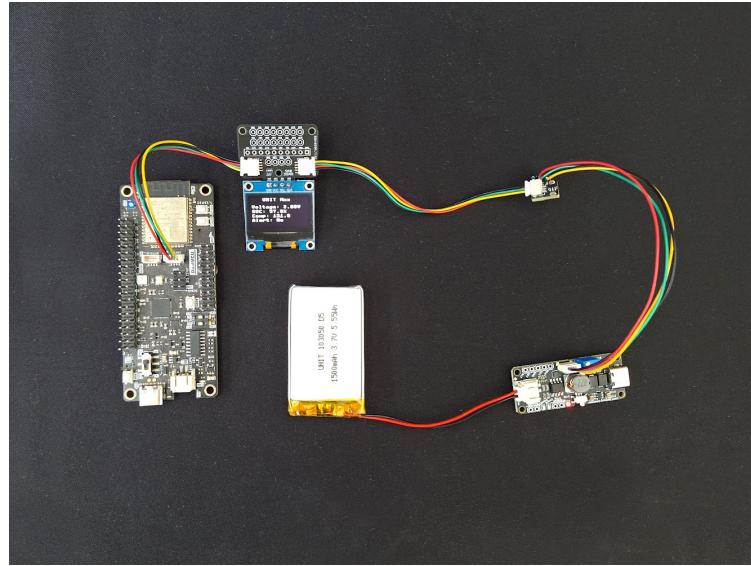


Figura 5.5: Ejemplo de integración deL conector JST a la placa DualMCU

CAPÍTULO 6

Salidas digitales

Las salidas digitales son una forma de interactuar con el mundo exterior. En la mayoría de los microcontroladores, las salidas digitales se utilizan para encender o apagar LEDs, activar relés, controlar motores y más.

Truco: Open-drain y Open-collector

Algunos microcontroladores tienen salidas de drenaje abierto (open-drain) o colector abierto (open-collector). Estas salidas son útiles para la conexión de dispositivos de alta corriente o para la comunicación bidireccional.

- **Open-drain:** La salida puede conectarse a tierra (GND) pero no a VCC.
 - **Open-collector:** La salida puede conectarse a VCC pero no a tierra (GND).
-

Advertencia: MicroPython no se encuentra disponible para la placa de desarrollo Cocket Nova su ejemplo es solo para SDCC.

6.1 Parpadeo (blink)

Un parpadeo de LED es un proyecto común para comenzar con microcontroladores. Lo que no te dicen es la equivalencia de un parpadeo en diferentes plataformas. Para Arduino IDE, MicroPython o SDCC puede diferir en la cantidad de líneas de código, pero el resultado es el mismo: un LED que parpadea.

Truco: Con frecuencia, las tarjetas de desarrollo tienen un LED integrado en un pin específico, como el pin 13 en Arduino Uno o el pin 25 en Raspberry Pi Pico.

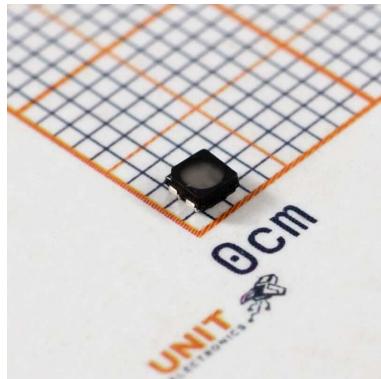


Figura 6.1: RGB_LED

6.1.1 MicroPython y Arduino IDE

Nota: En el siguiente ejemplo, se utiliza el pin 25 para el LED en la placa de desarrollo DualMCU RP2040. Modifica el pin según la placa de desarrollo que estés utilizando.

MicroPython

```
from machine import Pin
import time
led = Pin(25, Pin.OUT)
while True:
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
```

C++

```
#define LED_BUILTIN 25

void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(500);
    digitalWrite(LED_BUILTIN, LOW);
    delay(500);
}
```

6.1.2 Arduino IDE y SDCC

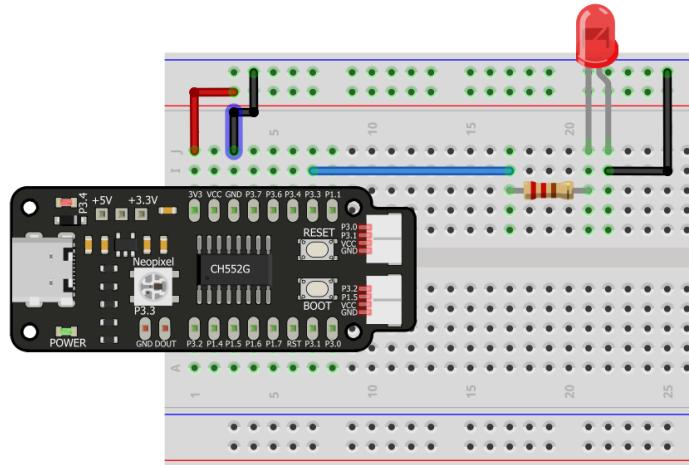


Figura 6.2: LEDs

C++

```
#define LED_BUILTIN 34

void setup() {
pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
digitalWrite(LED_BUILTIN, HIGH);
delay(500);
digitalWrite(LED_BUILTIN, LOW);
delay(500);
}
```

SDCC

```
#include "src/system.h"
#include "src/gpio.h"
#include "src/delay.h"

#define PIN_LED P34

void main(void)
{
    CLK_config();
    DLY_ms(5);

    PIN_output(PIN_LED);
    while (1)
    {
        PIN_toggle(PIN_LED);
        DLY_ms(500);
```

(continúa en la próxima página)

(proviene de la página anterior)

```
    }  
}
```

CAPÍTULO 7

Modulación por ancho de pulso (PWM)

La modulación por ancho de pulso (PWM) es una técnica utilizada para controlar la cantidad de energía entregada a un dispositivo. En los microcontroladores, el PWM se utiliza para controlar la velocidad de los motores, el brillo de los LEDs y más.

Advertencia: El soporte de ubicación para salidas PWM depende de la placa de desarrollo. Revisar la documentación de la placa para conocer los pines PWM disponibles.

7.1 Implementación

7.1.1 MicroPython y Arduino IDE

MicroPython

```
from machine import Pin, PWM
import time
pwm = PWM(Pin(25))
pwm.freq(1000)
while True:
    for duty_cycle in range(1024):
        pwm.duty(duty_cycle)
        time.sleep(0.01)
```

C++

```
void setup() {
    pinMode(9, OUTPUT);
    analogWrite(9, 128);
}
```

(continúe en la próxima página)

(proviene de la página anterior)

```
void loop() {
    for (int i = 0; i <= 255; i++) {
        analogWrite(9, i);
        delay(10);
    }
}
```

7.1.2 Arduino IDE y SDCC

SDCC

```
#include <stdio.h>
#include "src/config.h"
#include "src/system.h"
#include "src/gpio.h"
#include "src/delay.h"
#include "src/pwm.h"

#define MIN_COUNTER 10
#define MAX_COUNTER 254
#define STEP_SIZE    10

void change_pwm(int hex_value)
{
    PWM_write(PIN_PWM, hex_value);
}

void main(void)
{
    CLK_config();
    DLY_ms(5);
    PWM_set_freq(1);
    PIN_output(PIN_PWM);
    PWM_start(PIN_PWM);
    PWM_write(PIN_PWM, 0);
    while (1)
    {
        for (int i = MIN_COUNTER; i < MAX_COUNTER; i+=STEP_SIZE)
        {
            change_pwm(i);
            DLY_ms(20);
        }
        for (int i = MAX_COUNTER; i > MIN_COUNTER; i-=STEP_SIZE)
        {
            change_pwm(i);
            DLY_ms(20);
        }
    }
}
```

C++

```
#define led 34

int brightness = 0;
int fadeAmount = 5;

void setup() {
    pinMode(led, OUTPUT);
}

void loop() {
    analogWrite(led, brightness);
    brightness = brightness + fadeAmount;
    if (brightness <= 0 || brightness >= 255) {
        fadeAmount = -fadeAmount;
    }
    delay(30);
}
```

7.2 Aplicaciones

7.2.1 Control de servomotores - MicroPython RP2040

Los servomotores son dispositivos que se utilizan para controlar la posición de un objeto. Se utilizan en aplicaciones como robots, drones, juguetes y más.

Requieren de una señal PWM para controlar la posición del eje del motor. La mayoría de los servomotores aceptan una señal PWM con una frecuencia de 50 Hz y un ciclo de trabajo de 0.5 ms a 2.5 ms.

MicroPython

```
import machine
import utime

servo_pin = machine.Pin(0)
pwm_servo = machine.PWM(servo_pin)
pwm_servo.freq(50)

def set_servo_angle(angle):
    duty_cycle = int(1024 + (angle / 180) * 3072)
    pwm_servo.duty_u16(duty_cycle)

try:
    while True:
        for angle in range(0, 181, 10):
            set_servo_angle(angle)
            utime.sleep(0.1)
        for angle in range(180, -1, -10):
            set_servo_angle(angle)
            utime.sleep(0.1)
except KeyboardInterrupt:
    pwm_servo.deinit()
    print("\nPWM detenido. Recursos liberados.")
```

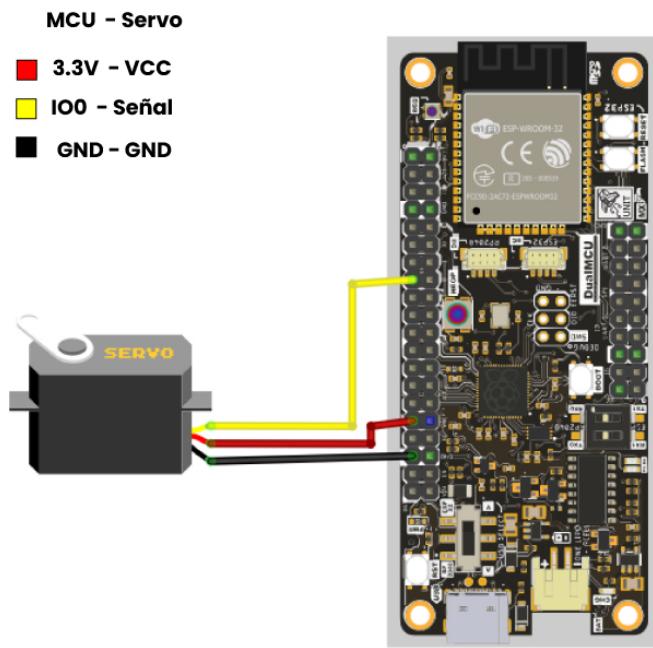


Figura 7.1: Diagrama de conexión del servomotor

C++

```
#define SERVO_PIN 0

void setup() {
    pinMode(SERVO_PIN, OUTPUT);
}

void loop() {
    for (int i = 40; i <= 115; i++) {
        analogWrite(SERVO_PIN, i);
        delay(500);
    }
}
```


CAPÍTULO 8

Entradas digitales

Las entradas digitales son una forma de interactuar con el mundo exterior. En la mayoría de los microcontroladores, las entradas digitales se utilizan para leer el estado de un botón, un sensor digital, un interruptor y más.

8.1 Pull-up y Pull-down

La mayoría de los microcontroladores tienen resistencias pull-up y pull-down internas. Estas resistencias se utilizan para mantener un valor lógico alto o bajo en una entrada digital cuando no se aplica una señal externa.

- **Pull-up:** La resistencia pull-up conecta la entrada a VCC (nivel alto).
- **Pull-down:** La resistencia pull-down conecta la entrada a tierra (nivel bajo).

Estas resistencias pueden activarse o desactivarse mediante software. En algunos microcontroladores, como el ESP32, las resistencias pull-up y pull-down se pueden configurar con un valor de resistencia específico.

También es posible utilizar resistencias externas para pull-up y pull-down. Esto es útil cuando se necesita un valor de resistencia específico o cuando las resistencias internas no son suficientes.

Advertencia: MicroPython no se encuentra disponible para la placa de desarrollo Cocket Nova su ejemplo es solo para SDCC.

8.2 Lectura de Entradas

La lectura de una entrada digital en un microcontrolador es un proceso sencillo. La entrada puede estar en uno de dos estados: alto (HIGH) o bajo (LOW). En la mayoría de los microcontroladores, el estado alto corresponde a un valor lógico de 1 y el estado bajo a un valor lógico de 0.

Nota: En el siguiente ejemplo, se utiliza el pin 26 para la entrada digital en la placa de desarrollo DualMCU RP2040. Modifica el pin según la placa de desarrollo que estés utilizando.

MicroPython

```
from machine import Pin
import time

# Configuración de la entrada digital
button = Pin(26, Pin.IN, Pin.PULL_UP) # Inicializar pin 26 para entrada con resistencia pull-up

while True:
    if button.value() == 0: # Si el botón está presionado
        print("Botón presionado")
    else:
        print("Botón no presionado")
    time.sleep(0.1)
```

C++

```
#define BUTTON_PIN 26

void setup() {
    pinMode(BUTTON_PIN, INPUT_PULLUP); // Inicializar pin 26 para entrada con resistencia pull-up
}

void loop() {
    if (digitalRead(BUTTON_PIN) == LOW) { // Si el botón está presionado
        Serial.println("Botón presionado");
    } else {
        Serial.println("Botón no presionado");
    }
    delay(100);
}
```

8.3 Arduino IDE y SDCC

C++

```
#include <Serial.h>

void setup() {
// No need to init USBSerial
pinMode(33, INPUT);
pinMode(34, OUTPUT);
}

void loop() {
// Leer el valor del botón en una variable
int sensorVal = digitalRead(11);
// Imprimir el valor del botón en el monitor serial
USBSerial.println(sensorVal);
if (sensorVal == HIGH) {
    digitalWrite(33, LOW);
} else {
    digitalWrite(33, HIGH);
}

delay(10);
}
```

SDCC

```
#include "src/system.h"
#include "src/gpio.h"
#include "src/delay.h"

#define PIN_LED P34
#define PIN_BUTTON P33

void main(void)
{
CLK_config();
DLY_ms(5);
PIN_input(PIN_BUTTON);
PIN_output(PIN_LED);
while (1)
{
    if (PIN_read(PIN_BUTTON)){
        PIN_high(PIN_LED);
    }
    else{
        PIN_low(PIN_LED);
    }
}
}
```


CAPÍTULO 9

Conversión de Analógico a Digital

9.1 Definición de ADC

La conversión de analógico a digital (ADC) es un proceso que convierte señales analógicas en valores digitales. Los microcontroladores utilizan ADC para leer señales analógicas de sensores y otros dispositivos.

Advertencia: El voltaje de referencia del ADC varía según el microcontrolador. Consulta la hoja de datos del microcontrolador para obtener información específica.

9.1.1 Cuantificación y Codificación de Señales Analógicas

Las señales analógicas son continuas y pueden tomar cualquier valor dentro de un rango dado. En cambio, las señales digitales son discretas y solo pueden adoptar valores específicos. La conversión de una señal analógica a digital implica dos pasos: cuantificación y codificación.

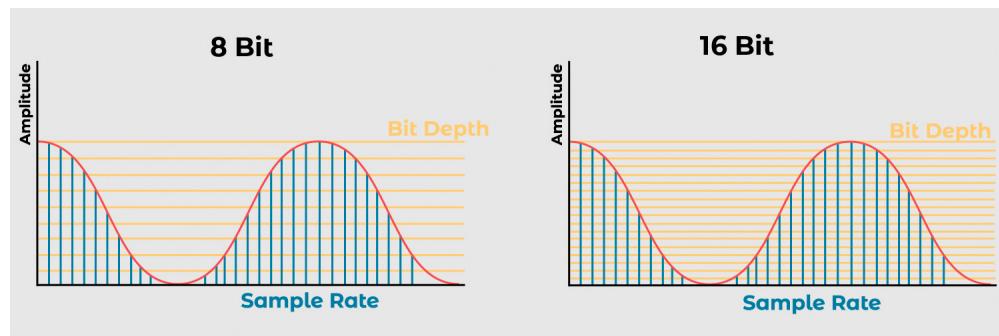


Figura 9.1: Conversión de Analógico a Digital

Tabla 9.1: Ejemplos de Codificación y Cuantificación

Resolución	Niveles de Cuantificación	Código Digital
8 bits	256	0x00 a 0xFF
12 bits	4,096	0x000 a 0xFFFF
16 bits	65,536	0x0000 a 0xFFFF

Cuantificación

Divide la señal analógica en niveles discretos. El número de niveles determina la resolución del ADC.

Nota: La resolución de un ADC se mide en bits y se calcula como 2^n , donde n es el número de bits.

Tabla 9.2: Resoluciones de ADC

Resolución	Niveles de Cuantificación	Descripción
8 bits	256	Un ADC de 8 bits tiene 256 niveles de cuantificación, lo que significa que puede representar la señal analógica con 256 valores diferentes.
12 bits	4,096	Un ADC de 12 bits tiene 4,096 niveles de cuantificación, lo que permite representar la señal analógica con 4,096 valores distintos.
16 bits	65,536	Un ADC de 16 bits tiene 65,536 niveles de cuantificación, permitiendo representar la señal analógica con 65,536 valores distintos.

Codificación

Asigna un código digital a cada nivel de cuantificación. Este código digital representa el valor de la señal analógica en dicho nivel.

9.1.2 Equivalencia de lectura de ADC en diferentes alternativas

MicroPython

```
adc_value = adc.read() # Leer el valor del ADC
```

C++

```
voltage_write = analogRead(ADC0);
```

SDCC

```
int data = ADC_read(); // Leer ADC (0 - 255, 8 bits)
```

9.2 Código de Ejemplo

Advertencia: MicroPython no se encuentra disponible para la placa de desarrollo Cocket Nova su ejemplo es solo para SDCC.

A continuación, se muestra un ejemplo de código para leer continuamente un valor ADC e imprimirllo:

9.2.1 MicroPython y Arduino IDE

Nota: El siguiente código está diseñado para funcionar con el microcontrolador RP2040 en la placa de desarrollo DualMCU.

MicroPython

```
import machine
import time

# Configuración del ADC
A0 = machine.Pin(26, machine.Pin.IN)      # Inicializar pin A0 para entrada
adc = machine.ADC(A0)                      # Crear objeto ADC

# Lectura continua
while True:
    adc_value = adc.read_u16()            # Leer el valor del ADC
    print(f'Lectura ADC: {adc_value:.2f}') # Imprimir el valor
    time.sleep(1)                         # Retraso de 1 segundo
```

C++

```
// El potenciómetro está conectado al GPIO 26 (ADC0 analógico)
const int potPin = 26;

// Variable para almacenar el valor del potenciómetro
int potValue = 0;

void setup() {
    Serial.begin(115200);
    analogReadResolution(12);
    delay(1000);
}

void loop() {
    // Leer el valor del potenciómetro
    potValue = analogRead(potPin);
    Serial.println(potValue);
    delay(500);
}
```

9.3 Arduino IDE y SDCC

C++

```
#define LED_BUILTIN 34

int sensorPin = 11;
int ledPin = LED_BUILTIN;
int sensorValue = 0;

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(sensorPin, INPUT);
}

void loop() {
    sensorValue = analogRead(sensorPin);
    digitalWrite(ledPin, HIGH);
    delay(sensorValue);
    digitalWrite(ledPin, LOW);
    delay(sensorValue);
}
```

SDCC

```
#include "src/system.h"
#include "src/gpio.h"
#include "src/delay.h"

#define PIN_ADC P11

void main(void)
{
    CLK_config();
    DLY_ms(5);

    ADC_input(PIN_ADC);
    ADC_enable();

    while (1)
    {
        int data = ADC_read(); // Leer valor ADC (0 - 255, 8 bits)
    }
}
```

CAPÍTULO 10

I2C (Inter-Integrated Circuit)

10.1 Descripción General del I2C

I2C (Inter-Integrated Circuit o Inter-IC) es un bus de comunicación serial síncrono, multi-maestro, multi-esclavo, de conmutación de paquetes y referencia única. Se utiliza comúnmente para conectar periféricos de baja velocidad a procesadores y microcontroladores. La gran mayoría de las placas de desarrollo de UNIT ELECTRONICS, como DualMCU, DualMCU ONE y Cocket Nova, ofrecen capacidades de comunicación I2C, lo que te permite interconectar una amplia gama de dispositivos I2C. A pesar de que la Cocket Nova no cuenta con I2C nativo, se implementa bajo una técnica llamada bit-banging.

Bit-banging es una técnica de programación en la que se manipulan directamente los pines de un microcontrolador para implementar un protocolo de comunicación. En el caso de I2C, se utilizan dos pines, SDA (Serial Data) y SCL (Serial Clock), para la comunicación entre dispositivos.

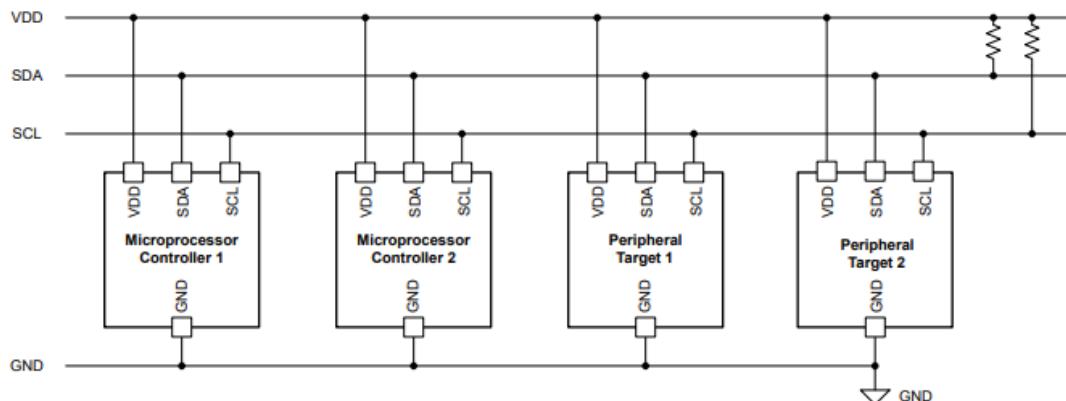


Figura 10.1: Pines I2C Imagen obtenida de [Application Note A Basic Guide to I2C](#)

Nota: Las tarjetas de desarrollo de UNIT ELECTRONICS son compatibles con Conector JST

10.2 Ejemplo de aplicación Pantalla SSD1306



Figura 10.2: Pantalla SSD1306

La pantalla OLED monocromática de 128x64 píxeles equipada con un controlador SSD1306 se conecta mediante un conector JST de 1.25mm de 4 pines. La siguiente tabla proporciona los detalles de conexión para la pantalla.

Tabla 10.1: Asignación de Pines de la Pantalla SSD1306

Pin	Conexión
1	GND
2	VCC
3	SDA
4	SCL

10.2.1 MicroPython y Arduino IDE

MicroPython

```
import machine
from dualmcu import *

i2c = machine.SoftI2C( scl=machine.Pin(22), sda=machine.Pin(21))

oled = SSD1306_I2C(128, 64, i2c)

oled.fill(1)
oled.show()

oled.fill(0)
```

(continúa en la próxima página)

(proviene de la página anterior)

```
oled.show()
oled.text('UNIT', 50, 10)
oled.text('ELECTRONICS', 25, 20)

oled.show()
```

Arduino

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET      -1
#define SCREEN_WIDTH    128
#define SCREEN_HEIGHT   64
#define SDA_PIN         4
#define SCL_PIN         5

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(9600);
  Wire.setSDA(4);
  Wire.setSCL(5);
  Wire.begin();
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("Error en la inicialización de la pantalla"));
    for(;;);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0,0);
  display.println(F("UNIT ELECTRONICS!"));
  display.display();
  delay(4000);
}

void loop() {
  display.clearDisplay();
  display.setCursor(0, 10);
  display.setTextSize(2);
  display.print(F("Contador: "));
  display.println(millis()/1000);
  display.display();
  delay(500);
}
```

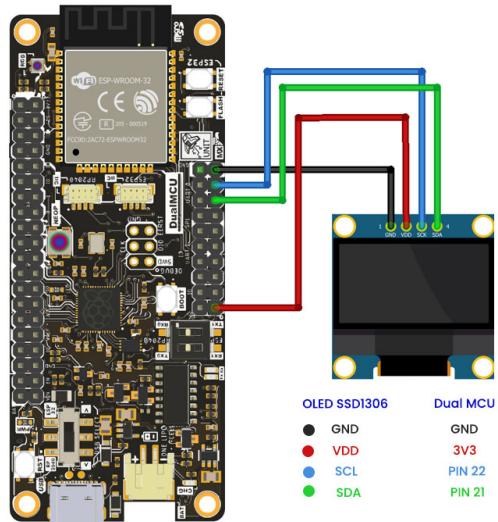


Figura 10.3: Diagrama de Conexión de la Pantalla SSD1306 DualMCU

10.3 Cocket Nova implementación de I2C

```
#include "src/config.h"
#include "src/system.h"
#include "src/gpio.h"
#include "src/delay.h"
#include "src/oled.h"

void main(void) {
    CLK_config();
    DLY_ms(5);

    OLED_init();

    OLED_print("* UNITElectronics *");
    OLED_print("-----\n");
    OLED_print("Ready\n");
    while(1) {

    }
}
```

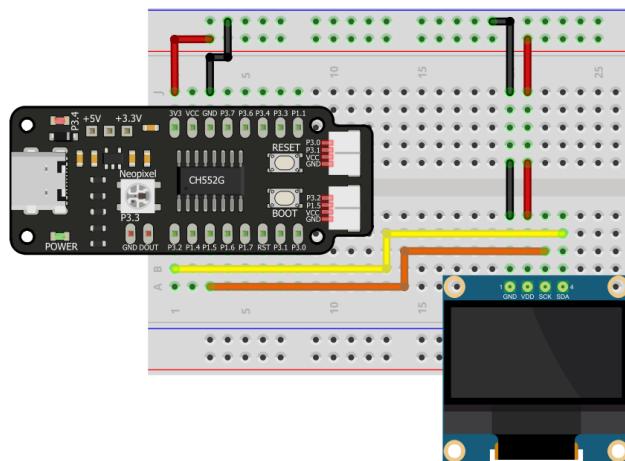


Figura 10.4: Pantalla OLED Cocket Nova

CAPÍTULO 11

Ejemplo de Integración

Este ejemplo demuestra cómo integrar el Cargador Boost LiPo & Monitor I2C de UNIT con un ESP32 y mostrar el estado de la batería en una pantalla OLED.

Prudencia: Dado que el estándar Qwiic opera a 3.3V, se recomienda usar un regulador de voltaje después de la salida JST del módulo para reducir VSYS a 3.3V y proteger los dispositivos Qwiic conectados.

Pasos para integrar el [Cargador Boost LiPo & Monitor I2C](#) de UNIT con el ESP32 y la pantalla OLED usando el conector JST y la comunicación I2C:

- 1. Conectar el Conector JST:** Conecta el conector JST a la placa DualMCU usando el [conector JST SH de 1.0mm y 4 pines](#).
- 2. Conectar el ESP32:** Conecta el ESP32 a la placa DualMCU mediante el protocolo de comunicación I2C.

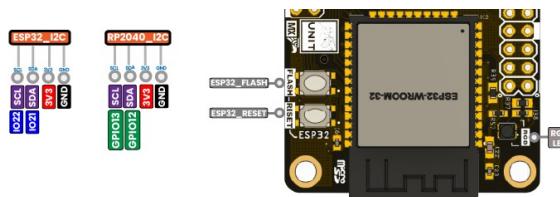


Figura 11.1: Conector JST SH de 1.0mm y 4 pines conectado al ESP32.

- 3. Soldar la Configuración de Pines de la Batería:** Suelda la configuración apropiada para el cargador de batería. Configura la corriente de carga (0.2 mA) según las especificaciones de la batería.

Advertencia: Asegúrate de verificar la polaridad de la batería antes de conectarla al cargador.

- 4. Conectar la Pantalla OLED:** Conecta la pantalla OLED al ESP32 usando el protocolo de comunicación I2C.
- 5. Subir el Código al ESP32:** Copia el siguiente código al ESP32 y ejecútalo usando Thonny:

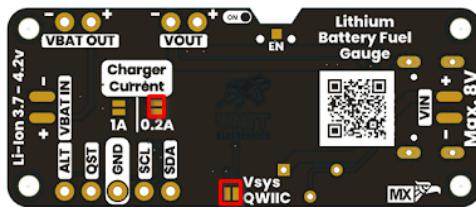


Figura 11.2: Configuración de Pines de la Batería

```

import machine
import time
from ocks import SSD1306_I2C
from max1704x import max1704x

i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))
oled = SSD1306_I2C(128, 64, i2c)

my_sensor = max1704x(sda_pin=21, scl_pin=22)

def update_display():
    oled.fill(0)
    oled.text('UNIT Max', 25, 0)

    vcell = my_sensor.getVCell()
    soc = my_sensor.getSoc()
    compensate_value = my_sensor.getCompensateValue()
    alert_threshold = my_sensor.getAlertThreshold()
    in_alert = my_sensor.inAlert()

    oled.text("Voltaje: {:.2f}V".format(vcell), 0, 16)
    oled.text("SOC: {:.1f}%".format(soc), 0, 26)
    oled.text("Comp: {:.1f}%".format(compensate_value), 0, 36)
    oled.text("Alerta: {}".format("Sí" if in_alert else "No"), 0, 46)

    oled.show()

while True:
    update_display()
    time.sleep(2)

```

6. **Monitorear el Estado de la Batería:** La pantalla OLED mostrará el estado de la batería, incluyendo el voltaje, estado de carga, valor de compensación y estado de alerta.
7. **Cargar la Batería:** Conecta la batería al cargador y monitorea el estado de carga usando la pantalla OLED.

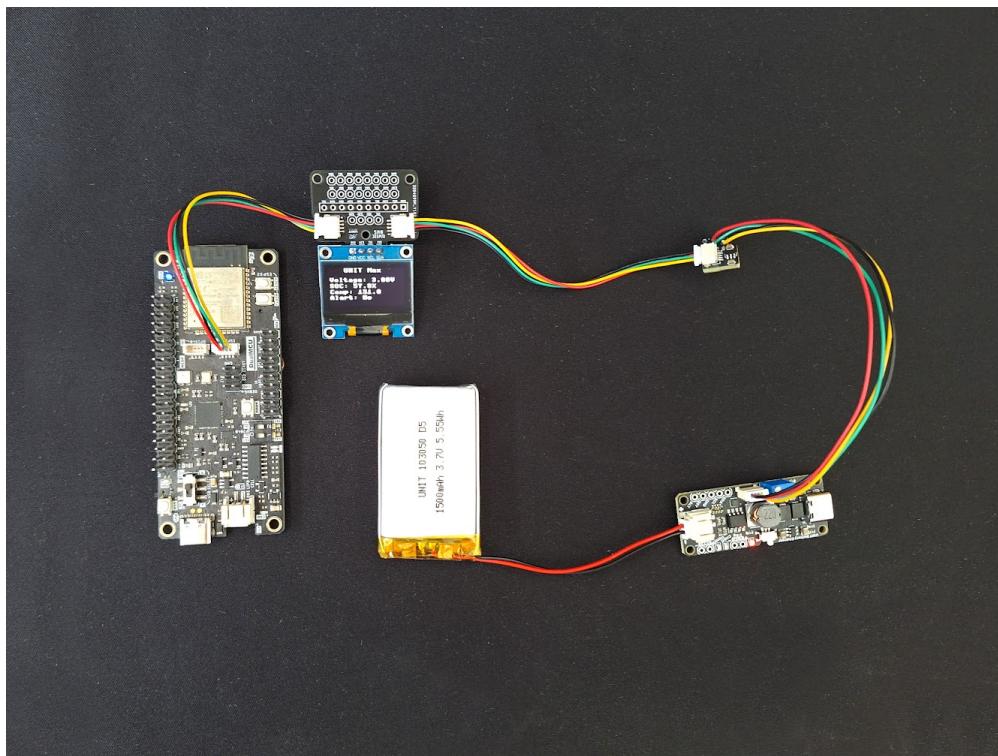


Figura 11.3: Ejemplo de Integración del Cargador Boost LiPo & Monitor I2C de UNIT con ESP32 y Pantalla OLED.

CAPÍTULO 12

SPI (Interfaz Periférica Serial)

Advertencia: El soporte para la tarjeta de desarrollo Cocket Nova (CH552) se encuentra en desarrollo. Por favor, mantente atento a futuras actualizaciones.

12.1 Visión General del SPI

SPI (Interfaz Periférica Serial) es un bus de comunicación síncrono, full-duplex y maestro-esclavo. Se utiliza comúnmente para conectar microcontroladores a periféricos como sensores, pantallas y dispositivos de memoria. La placa de desarrollo DualMCU ONE ofrece capacidades de comunicación SPI, lo que te permite interconectar una amplia gama de dispositivos SPI.

A continuación, se muestra la tabla de asignación de pines para las conexiones SPI en la DualMCU ONE, detallando las conexiones GPIO correspondientes para los microcontroladores ESP32 y RP2040.

Tabla 12.1: Asignación de Pines SPI

PIN	GPIO ESP32	GPIO RP2040
SCK	18	18
MISO	19	16
MOSI	23	19
SS	5	17

12.1.1 DualMCU ONE RP2040 y ESP32

La placa de desarrollo DualMCU ONE está equipada con dos microcontroladores, el ESP32 y el RP2040. Ambos microcontroladores ofrecen soporte para la comunicación SPI, lo que te permite conectar dispositivos SPI a la placa de desarrollo.

Tabla 12.2: Configuración SPI cruzado entre ESP32 y RP2040

ESP32	GPIO	RP2040	GPIO
SCK	18	SCK	14
MISO	23	MOSI	15
MOSI	19	MISO	12
SS	5	SS	13

MicroPython RP2040

```
from machine import Pin, SPI
import time

# RP2040 pin configuration
rp2040_cs = Pin(13, Pin.OUT)    # Chip Select (CS)
spi = SPI(1, baudrate=1000000, polarity=0, phase=0, sck=Pin(14), mosi=Pin(15), ↴
          miso=Pin(12))

# rp2040_cs = Pin(21, Pin.OUT)    # Chip Select (CS)
# spi = SPI(0, baudrate=1000000, polarity=0, phase=0, sck=Pin(18), mosi=Pin(19), ↴
#           miso=Pin(20))

def spi_send_command(command, response_length=3):
    # Seleccionar el dispositivo SPI (bajar CS)
    rp2040_cs.value(0)
    time.sleep_us(10)  # Breve pausa para estabilizar CS

    # Asegurarte de que ambos buffers (envío y respuesta) sean del mismo tamaño
    send_buffer = bytearray(response_length)
    response = bytearray(response_length)  # Buffer de respuesta de 2 bytes, igual que ↴
    send_buffer

    # Copiar el comando en el buffer de envío (solo el primer byte)
    send_buffer[0] = command[0]

    # Enviar el comando y recibir la respuesta
    spi.write_readinto(send_buffer, response)

    # Deseleccionar el dispositivo SPI (subir CS)
    time.sleep_us(1)
    rp2040_cs.value(1)

    return response

def main():
    # Lista de comandos para enviar al ESP32 (cada comando es de 1 byte)
```

(continúe en la próxima página)

(proviene de la página anterior)

```

commands = [
    b'\x01', # Comando 0x01: Solicitar estado general de memoria
    b'\x02', # Comando 0x02: Solicitar memoria libre
    b'\x03', # Comando 0x03: Solicitar estado de los registros
    b'\x21' # Comando 0x04: Solicitar temperatura
]

while True:
    for command_to_send in commands:
        # Enviar el comando y recibir la respuesta
        response = spi_send_command(command_to_send, response_length=3) # Ajustado a
        ↵para 2 bytes

        # Imprimir el comando enviado y la respuesta recibida
        print(f"Sent command: {command_to_send.hex()} | Received response:", ↵
        ↵response)

        time.sleep(1) # Tiempo de espera entre comandos

main()

```

ESP-IDF ESP32

```

#include <stdio.h>
#include <string.h>
#include "driver/spi_slave.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "esp_system.h" // Para funciones del sistema como esp_get_free_heap_size
#include "esp_heap_caps.h" // Para asignación de memoria compatible con DMA

// ESP32 WROOM 32E

#define PIN_NUM_MISO 23
#define PIN_NUM_MOSI 14
#define PIN_NUM_CLK 18
#define PIN_NUM_CS 5
#define SPIHOST HSPI_HOST
#define ESP32_state 0

#define CMD_START_RX 0XE0
#define CMD_END_RX 0xEE

static const char *TAG = "SPI_SLAVE";

// Buffer de 2 bytes para la transmisión SPI
uint8_t PK_BUFF[2];

// Función para procesar el comando recibido por SPI
void process_command(uint8_t *command) {

```

(continúe en la próxima página)

(proviene de la página anterior)

```

// delete start and end command choose the command
// command length is 1 byte

ESP_LOGI(TAG, "Received command: %02X", command[0]);

// Simular una respuesta según el comando recibido
switch (command[0]) {
    case 0x01: // Comando 0x01: Retornar estado de la memoria
        PK_BUFF[0] = 0x53; // Ejemplo de estado de memoria
        PK_BUFF[1] = 0x4f;
        break;
    case 0x02: // Comando 0x02: Retornar memoria disponible
        PK_BUFF[0] = 0x59; // Simular datos
        PK_BUFF[1] = 0x2D;
        break;
    case 0x03: // Comando 0x03: Retornar estado de registros
        PK_BUFF[0] = 0x28;
        PK_BUFF[1] = 0x29;
        break;
    case 0x21: // Comando 0x04: Name ascii 32, S3, C6
        PK_BUFF[0] = 0x4f;
        PK_BUFF[1] = 0x4b;

        break;
    default:
        PK_BUFF[0] = 0x00; // Comando desconocido
        PK_BUFF[1] = 0x00;
        break;
}
}

void app_main(void) {
    // Configuración del bus SPI
    spi_bus_config_t buscfg = {
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
    };

    // Configuración de la interfaz SPI como esclavo
    spi_slave_interface_config_t slvcfg = {
        .spics_io_num = PIN_NUM_CS,
        .flags = 0,
        .queue_size = 6, // Aumentar el tamaño de la cola para mejorar el rendimiento
        .mode = 0,
    };

    // Inicializar el bus SPI como esclavo
    esp_err_t ret = spi_slave_initialize(SPIHOST, &buscfg, &slvcfg, SPI_DMA_CH_AUTO);
}

```

(continúe en la próxima página)

(proviene de la página anterior)

```

ESP_ERROR_CHECK(ret);

// Asignar buffers de 4 bytes para enviar y recibir datos
uint8_t *sendbuf = (uint8_t *)heap_caps_malloc(4, MALLOC_CAP_DMA);
uint8_t *recvbuf = (uint8_t *)heap_caps_malloc(4, MALLOC_CAP_DMA);

if (sendbuf == NULL || recvbuf == NULL) {
    ESP_LOGE(TAG, "Failed to allocate DMA-capable memory");
    return;
}

memset(sendbuf, 0, 4); // Inicializar el buffer de envío

spi_slave_transaction_t t;
memset(&t, 0, sizeof(t)); // Inicializar la estructura de transacción SPI

while (1) {
    // Esperar un comando desde el maestro
    t.length = 8 * 2; // Recibir 1 byte de comando y enviar 2 bytes de respuesta
    t.tx_buffer = sendbuf;
    t.rx_buffer = recvbuf;

    // Realizar la transacción SPI
    ret = spi_slave_transmit(SPIHOST, &t, portMAX_DELAY);
    ESP_ERROR_CHECK(ret);

    // Procesar el comando recibido
    process_command(recvbuf);

    // Copiar el resultado de PK_BUFF al buffer de envío
    sendbuf[0] = PK_BUFF[0];
    sendbuf[1] = PK_BUFF[1];
}
}
}

```

12.2 SDCard SPI

Advertencia: Asegúrate de que la Micro SD contenga datos. Se recomienda guardar múltiples archivos de antemano para facilitar su uso.

12.2.1 Biblioteca (MicroPython)

La biblioteca *dualmcu.py* para MicroPython en ESP32 y RP2040 es compatible con la lectura y escritura en la Micro SD. La biblioteca proporciona una interfaz sencilla para leer y escribir archivos en la tarjeta SD. La biblioteca está disponible en PyPi y se puede instalar mediante el IDE Thonny.

12.2.2 VSPI y HSPI

La diferencia entre VSPI y HSPI radica en la velocidad de transferencia de datos. VSPI es más lento que HSPI, pero es más fácil de configurar. HSPI, por otro lado, es más rápido pero requiere una configuración más detallada.

Interfaz VSPI

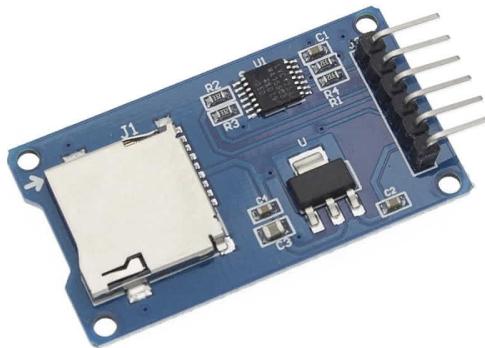


Figura 12.1: Lector externo de Micro SD

Las conexiones son las siguientes:

Esta tabla ilustra las conexiones entre la tarjeta SD y los pines GPIO en los microcontroladores ESP32 y RP2040.

Tabla 12.3: Conexiones VSPI

Tarjeta SD	Nombre del Pin	ESP32	RP2040
D3	SS	5	17
CMD	MOSI	23	19
VSS	GND		
VDD	3.3V		
CLK	SCK	18	18
D0	MISO	19	16

Descripciones

- SCK (Reloj Serial)
- SS (Selección del Esclavo)

```
import machine, os
from dualmcu import *

SCK_PIN = 18
MOSI_PIN = 23
MISO_PIN = 19
```

(continúe en la próxima página)

(proviene de la página anterior)

```
CS_PIN = 5

spi = machine.SPI(1, baudrate=100000, polarity=0, phase=0, sck=machine.Pin(SCK_PIN), mosi=machine.Pin(MOSI_PIN), miso=machine.Pin(MISO_PIN))
spi.init()
sd = SDCard(spi, machine.Pin(CS_PIN))
os.mount(sd, '/sd')
os.listdir('/')

print("archivos ...")
print(os.listdir("/sd"))
```

Interfaz HSPI

Esta tabla detalla las conexiones entre la tarjeta SD y el microcontrolador ESP32.

Tabla 12.4: Conexiones HSPI

Tarjeta SD	ESP32	PIN
D2		
D3	SS (Selección del Esclavo)	15
CMD	MOSI	13
VSS	GND	
VDD	3.3V	
CLK	SCK (Reloj Serial)	14
VSS	GND	
D0	MISO	12
D1		

Para la prueba, utilizaremos un ESP32 WROM-32E y una tarjeta SanDisk Micros Ultra con una capacidad de 32 GB.

```
import machine
import os
from dualmcu import *

# Inicializa la interfaz SPI para la tarjeta SD
spi = machine.SPI(2, baudrate=1000000, polarity=0, phase=0, sck=machine.Pin(14), mosi=machine.Pin(13), miso=machine.Pin(12))

# Inicializa la tarjeta SD
sd = SDCard(spi, machine.Pin(15))

# Monta el sistema de archivos
vfs = os.VfsFat(sd)
os.mount(vfs, "/sd")

# Lista los archivos en la raíz de la tarjeta SD
print("Archivos en la raíz de la tarjeta SD:")
print(os.listdir("/sd"))

os.umount("/sd")
```

12.3 SDIO (Interfaz de Entrada/Salida Segura Digital)

SDIO (Secure Digital Input/Output) es una interfaz de comunicación de alta velocidad que permite la transferencia de datos entre un microcontrolador y una tarjeta SD. La placa de desarrollo DualMCU ONE ofrece soporte para la interfaz SDIO, lo que te permite leer y escribir datos en una tarjeta SD de forma rápida y eficiente.

La interfaz SDIO es una interfaz de comunicación de alta velocidad que permite la transferencia de datos entre un microcontrolador y una tarjeta SD. La placa de desarrollo DualMCU ONE ofrece soporte para la interfaz SDIO, lo que te permite leer y escribir datos en una tarjeta SD de forma rápida y eficiente.

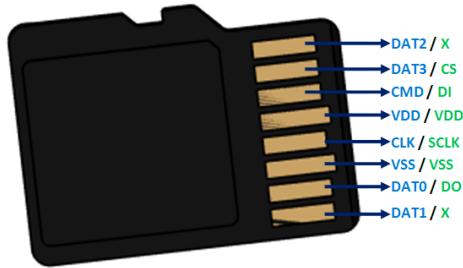


Figura 12.2: Pinout de la Micro SD

Esta tabla detalla las conexiones entre la tarjeta SD y el microcontrolador ESP32.

Tabla 12.5: Conexiones SDIO

Nombre del Pin	Pines Correspondientes en Modo SPI	Número GPIO (Slot 1)	Número GPIO (Slot 2)
CLK	SCLK	6	14
CMD	MOSI	11	15
DAT0	MISO	7	2
DAT1	Interrupción	8	4
DAT2	Sin Conexión (pullup)	9	12
DAT3	CS	10	13

12.3.1 Ejemplo de aplicación

```
/* Sketch for testing the ESP32 HSPI interface on the DualMCU ONE.
 * Connect the SD card to the following pins:
 *
 * SD Card / ESP32
 * D2      12
 * D3      13
 * CMD     15
 * VSS     GND
 * VDD     3.3V
 * CLK     14
 * VSS     GND
 * D0      2  (add 1K pull up after flashing)
 * D1      4
```

(continúe en la próxima página)

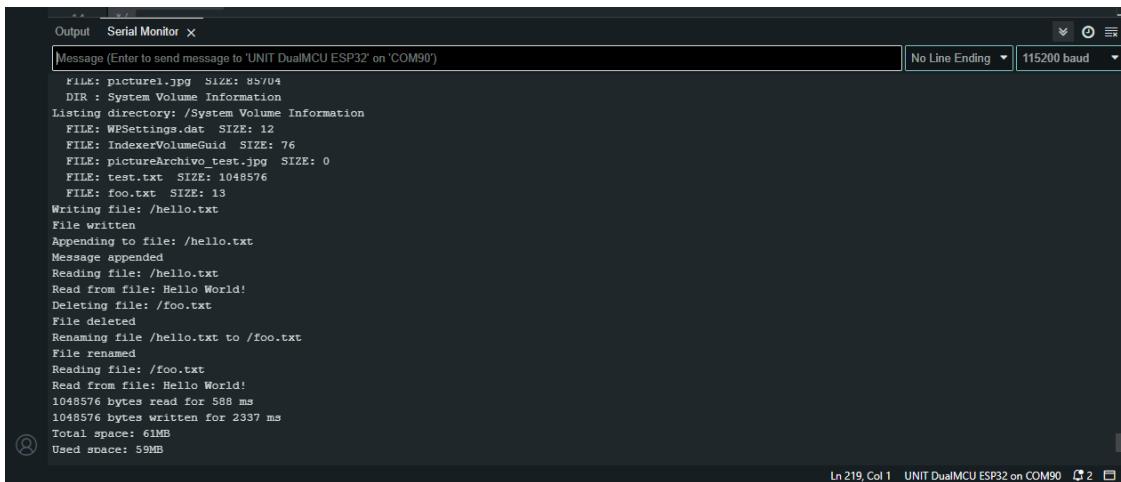


Figura 12.3: Lector de Micro SD

(proviene de la página anterior)

```

*/
#include "FS.h"
#include "SD_MMC.h"

void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
    Serial.printf("Listing directory: %s\n", dirname);

    File root = fs.open(dirname);
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print(" DIR : ");
            Serial.println(file.name());
            if(levels){
                listDir(fs, file.path(), levels -1);
            }
        } else {
            Serial.print(" FILE: ");
            Serial.print(file.name());
            Serial.print(" SIZE: ");
            Serial.println(file.size());
        }
        file = root.openNextFile();
    }
}

```

(continúe en la próxima página)

(proviene de la página anterior)

```

        }
    }

void createDir(fs::FS &fs, const char * path){
    Serial.printf("Creating Dir: %s\n", path);
    if(fs.mkdir(path)){
        Serial.println("Dir created");
    } else {
        Serial.println("mkdir failed");
    }
}

void removeDir(fs::FS &fs, const char * path){
    Serial.printf("Removing Dir: %s\n", path);
    if(fs.rmdir(path)){
        Serial.println("Dir removed");
    } else {
        Serial.println("rmdir failed");
    }
}

void readFile(fs::FS &fs, const char * path){
    Serial.printf("Reading file: %s\n", path);

    File file = fs.open(path);
    if(!file){
        Serial.println("Failed to open file for reading");
        return;
    }

    Serial.print("Read from file: ");
    while(file.available()){
        Serial.write(file.read());
    }
}

void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file){
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)){
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
}

void appendFile(fs::FS &fs, const char * path, const char * message){

```

(continúe en la próxima página)

(proviene de la página anterior)

```

Serial.printf("Appending to file: %s\n", path);

File file = fs.open(path, FILE_APPEND);
if(!file){
    Serial.println("Failed to open file for appending");
    return;
}
if(file.print(message)){
    Serial.println("Message appended");
} else {
    Serial.println("Append failed");
}
}

void renameFile(fs::FS &fs, const char * path1, const char * path2){
Serial.printf("Renaming file %s to %s\n", path1, path2);
if (fs.rename(path1, path2)) {
    Serial.println("File renamed");
} else {
    Serial.println("Rename failed");
}
}

void deleteFile(fs::FS &fs, const char * path){
Serial.printf("Deleting file: %s\n", path);
if(fs.remove(path)){
    Serial.println("File deleted");
} else {
    Serial.println("Delete failed");
}
}

void testFileIO(fs::FS &fs, const char * path){
File file = fs.open(path);
static uint8_t buf[512];
size_t len = 0;
uint32_t start = millis();
uint32_t end = start;
if(file){
    len = file.size();
    size_t flen = len;
    start = millis();
    while(len){
        size_t toRead = len;
        if(toRead > 512){
            toRead = 512;
        }
        file.read(buf, toRead);
        len -= toRead;
    }
    end = millis() - start;
    Serial.printf("%u bytes read for %u ms\n", flen, end);
}
}

```

(continúe en la próxima página)

(proviene de la página anterior)

```

        file.close();
    } else {
        Serial.println("Failed to open file for reading");
    }

file = fs.open(path, FILE_WRITE);
if(!file){
    Serial.println("Failed to open file for writing");
    return;
}

size_t i;
start = millis();
for(i=0; i<2048; i++){
    file.write(buf, 512);
}
end = millis() - start;
Serial.printf("%u bytes written for %u ms\n", 2048 * 512, end);
file.close();
}

void setup(){
Serial.begin(115200);
if(!SD_MMC.begin()){
    Serial.println("Card Mount Failed");
    return;
}
uint8_t cardType = SD_MMC.cardType();

if(cardType == CARD_NONE){
    Serial.println("No SD_MMC card attached");
    return;
}

Serial.print("SD_MMC Card Type: ");
if(cardType == CARD_MMC){
    Serial.println("MMC");
} else if(cardType == CARD_SD){
    Serial.println("SDSC");
} else if(cardType == CARD_SDHC){
    Serial.println("SDHC");
} else {
    Serial.println("UNKNOWN");
}

uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
Serial.printf("SD_MMC Card Size: %lluMB\n", cardSize);

listDir(SD_MMC, "/");
createDir(SD_MMC, "/mydir");
listDir(SD_MMC, "/", 0);

```

(continúe en la próxima página)

(proviene de la página anterior)

```
removeDir(SD_MMC, "/mydir");
listDir(SD_MMC, "/", 2);
writeFile(SD_MMC, "/hello.txt", "Hello ");
appendFile(SD_MMC, "/hello.txt", "World!\n");
readFile(SD_MMC, "/hello.txt");
deleteFile(SD_MMC, "/foo.txt");
renameFile(SD_MMC, "/hello.txt", "/foo.txt");
readFile(SD_MMC, "/foo.txt");
testFileIO(SD_MMC, "/test.txt");
Serial.printf("Total space: %lluMB\n", SD_MMC.totalBytes() / (1024 * 1024));
Serial.printf("Used space: %lluMB\n", SD_MMC.usedBytes() / (1024 * 1024));
}

void loop(){
```

}

CAPÍTULO 13

Interfaz USB

La interfaz USB es un protocolo de comunicación que permite la conexión de dispositivos electrónicos a una computadora. En el caso de los microcontroladores, la interfaz USB permite la comunicación con una computadora para la transferencia de datos, actualización de firmware, depuración, etc.

13.1 HID (Dispositivo de Interfaz Humano)

HID (Human Interface Device) es un protocolo de comunicación USB que permite a los dispositivos electrónicos interactuar con un usuario. Los dispositivos HID incluyen teclados, ratones, joysticks, gamepads, etc.

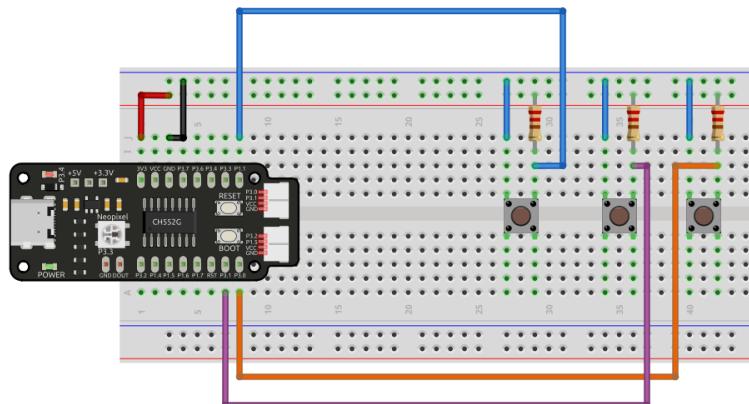


Figura 13.1: Ejemplo de Dispositivo HID

13.2 CDC (Clase de Dispositivo de Comunicación)

CDC (Communication Device Class) es una clase de dispositivos USB que permite la comunicación entre un dispositivo y una computadora. Los dispositivos CDC incluyen módems, adaptadores de red, etc.

13.3 MIDI (Interfaz Digital de Instrumentos Musicales)

MIDI (Musical Instrument Digital Interface) es un protocolo de comunicación que permite a los dispositivos electrónicos comunicarse entre sí para la creación, edición y reproducción de música. Los dispositivos MIDI incluyen teclados, sintetizadores, controladores, etc.

13.4 Micrófono USB

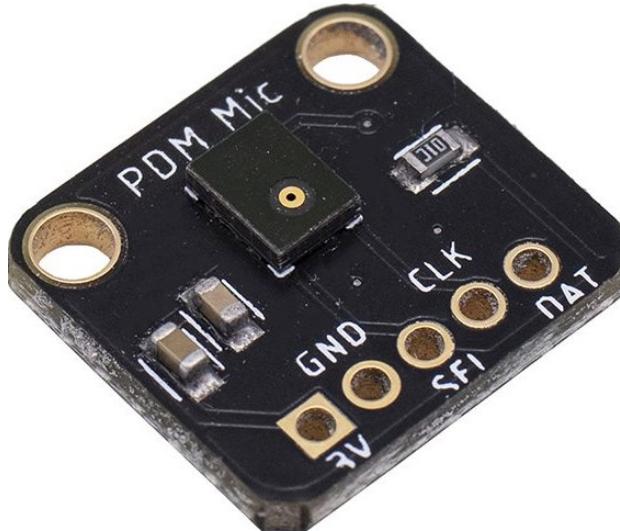


Figura 13.2: Micrófono USB DualMCU

Este archivo transforma el DualMCU en un micrófono USB de alto rendimiento utilizando el microcontrolador RP2040. Para utilizar esta funcionalidad, conecta un micrófono MEMS PDM (Pulse Density Modulation), como:

- **MP34DT06J**
- **UNIT PDM MEMS Microphone MP34DT05**

Esta funcionalidad es ideal para videoconferencias o aplicaciones de audio en general, proporcionando un rendimiento de sonido de alta calidad.

13.4.1 Conexión de un micrófono PDM

Para conectar un **micrófono PDM UNIT MP34DT05TR-A** o un **módulo de micrófono PDM de Adafruit** al RP2040 en la placa DualMCU, sigue estos pasos:

1. Asegúrate de tener el hardware necesario: un módulo de micrófono PDM y la placa DualMCU.
2. Ubica los pines GPIO12 y GPIO13 en la placa DualMCU [(Diagrama de pines)](<https://github.com/UNIT-Electronics/DualMCU/blob/main/Hardware/Resources/EU0002-DUALMCU%20V3.1.2.jpg>) y conecta el micrófono de la siguiente manera:

Tabla 13.1: Conexión entre DualMCU y PDM-MIC

DualMCU	PDM-MIC
3.3V	3.3V
GND	GND
GPIO12	Señal SCL
GPIO13	Señal DAT

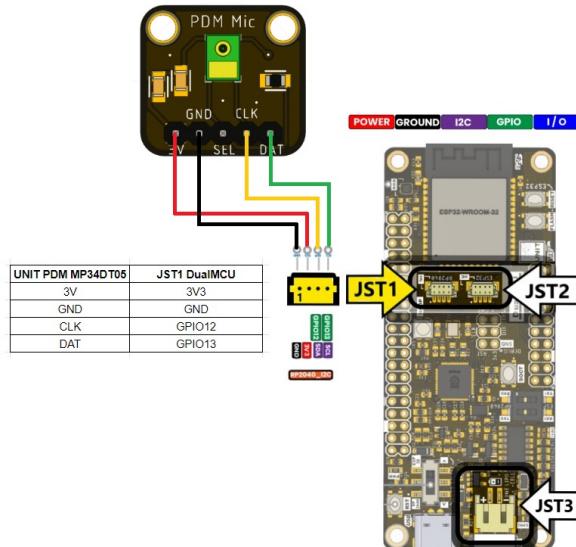


Figura 13.3: Micrófono USB DualMCU

3. Enciende la placa DualMCU y el módulo de micrófono.
4. Entra en modo BOOT en el RP2040 y arrastra y suelta el archivo *usb_microphone.uf2* en la placa DualMCU.

Ejemplo Completo en [UNIT-PDM-MEMS-Microphone-Breakout-Guide-UF2](#)

CAPÍTULO 14

Control WS2812

El led WS2812 es un led RGB que se puede controlar con un solo pin de datos. Es muy popular en proyectos de iluminación y decoración. En este tutorial, aprenderás cómo controlar un led WS2812 con MicroPython y Arduino.

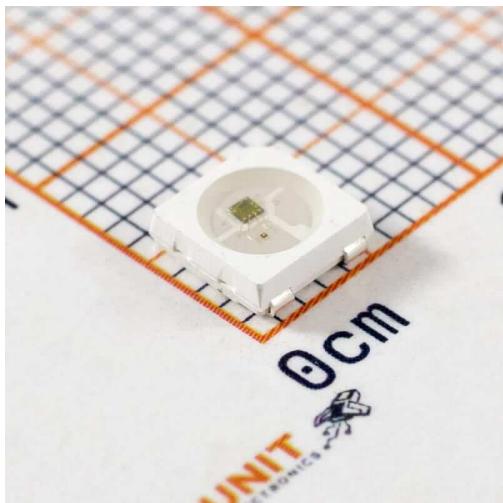


Figura 14.1: Tira LED WS2812

14.1 MicroPython y Arduino IDE

MicroPython

```
from machine import Pin
from neopixel import NeoPixel
np = NeoPixel(Pin(24), 1)
np[0] = (255, 128, 0)

np.write()
```

C++

```
#include <Adafruit_NeoPixel.h>
#define PIN 24
#define NUMPIXELS 1
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
pixels.setPixelColor(0, pixels.Color(255, 128, 0));
pixels.show();
```

Truco: para obtener más información sobre la biblioteca NeoPixel, consulta la [Documentación de la Biblioteca NeoPixel](#).

14.2 Control WS2812 con SDCC

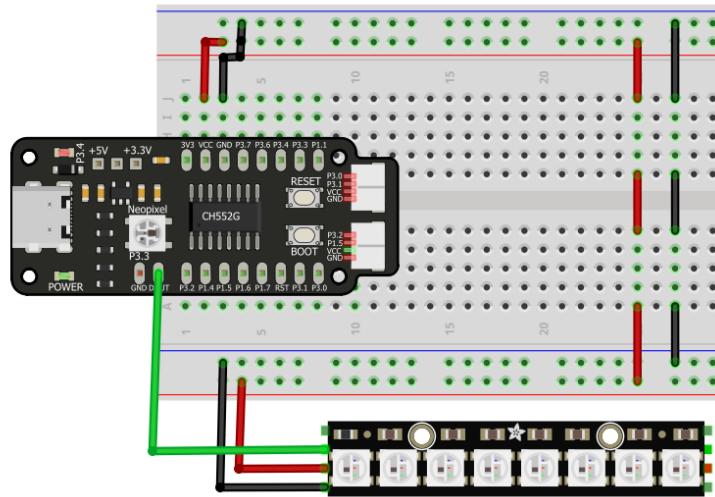


Figura 14.2: Tira LED WS2812

SDCC

```
#include "src/config.h"
#include "src/system.h"
#include "src/delay.h"
```

(continúa en la próxima página)

(proviene de la página anterior)

```

#include "src/neo.h"
#include <stdlib.h>

#define delay 100
#define NeoPixel 2
#define level 100

void randomColorSequence(void) {

    for(int j=0; j<NeoPixel; j++){
        uint8_t red = rand() % level;
        uint8_t green = rand() % level;
        uint8_t blue = rand() % level;
        uint8_t num = rand() % NeoPixel;

        for(int i=0; i<num; i++){
            NEO_writeColor(0, 0, 0);
        }
        NEO_writeColor(red, green, blue);
        DLY_ms(delay);
        NEO_writeColor(0, 0, 0);
    }

    for(int l=0; l<9; l++){
        NEO_writeColor(0, 0, 0);
    }
}

void main(void) {
    NEO_init();
    CLK_config();
    DLY_ms(delay);

    while (1) {
        randomColorSequence();
        DLY_ms(10);
    }
}

```

Arduino-IDE

CAPÍTULO 15

Cómo Generar un Informe de Error

Esta guía explica cómo generar un informe de error utilizando repositorios de GitHub.

15.1 Pasos para Crear un Informe de Error:

1. **Acceder al Repositorio de GitHub** Navega al repositorio de GitHub donde se aloja el proyecto.
2. **Abrir la Pestaña de Issues** Haz clic en la pestaña «Issues» ubicada en el menú del repositorio.
3. **Crear un Nuevo Issue**
 - Haz clic en el botón «New Issue».
 - Proporciona un título claro y conciso para el issue.
 - **Añade una descripción detallada, incluyendo información relevante como:**
 - Pasos para reproducir el error.
 - Resultados esperados y reales.
 - Cualquier registro, captura de pantalla o archivo relacionado.
4. **Enviar el Issue** Una vez que el formulario esté completo, haz clic en el botón «Submit».

El equipo de desarrollo o los mantenedores revisarán el issue y tomarán las medidas apropiadas para abordarlo.

CAPÍTULO 16

Índices y tablas

- genindex
- modindex
- search