



Introducción al Ecosistema de Desarrollo UNIT

Versión 0.0.1

Cesar Bautista

06 de febrero de 2025

1. MicroPython	3
1.1. Comenzando con MicroPython	3
1.2. Características de MicroPython	4
1.3. Comenzando con MicroPython	4
1.4. Bibliotecas y Módulos de MicroPython	4
1.5. IDEs y Herramientas de MicroPython	4
2. Arduino IDE	5
2.1. Historia	5
2.2. Características Principales	5
2.3. Diferencias Clave entre Arduino y un Entorno Nativo	6
2.4. Casos de Uso	7
3. Guía de Instalación de Paquetes de Unit Electronics	9
3.1. Prerequisitos	9
3.2. Instalación Rápida	9
3.3. 1. Instalación del Paquete de Placa DualMCU-ONE	10
3.4. 2. Instalación del Paquete de Placa Cocket Nova CH552	10
4. Salidas digitales	13
4.1. Parpadeo (blink)	13
5. Modulación por ancho de pulso (PWM)	17
5.1. Implementación	17
5.2. Aplicaciones	19
6. Entradas digitales	23
6.1. Pull-up y Pull-down	23
6.2. Lectura de Entradas	24
6.3. Arduino IDE y SDCC	25
7. Conversión de Analógico a Digital	27
7.1. Definición de ADC	27
7.2. Código de Ejemplo	28
7.3. Clase ADC	30
7.4. Ejemplo de Definición	30
7.5. Lectura de Valores	30

8. Cómo Generar un Informe de Error	31
8.1. Pasos para Crear un Informe de Error:	31
9. Índices y tablas	33
Índice	35

El ecosistema de desarrollo de UNIT Electronics 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.

MicroPython es una implementación ligera y eficiente del lenguaje de programación Python 3 que está optimizada para ejecutarse en microcontroladores. Está diseñado para ser utilizado en entornos restringidos, lo que lo hace adecuado para el desarrollo de sistemas embebidos. MicroPython proporciona un conjunto rico de bibliotecas y módulos que permiten a los desarrolladores interactuar con componentes de hardware y periféricos, lo que lo convierte en una opción ideal para proyectos de IoT y prototipos.

1.1 Comenzando con MicroPython

Comenzar con MicroPython es fácil. La siguiente tabla enumera las placas que son compatibles con MicroPython.

Placa	Descripción
ESP32	ESP32 es una serie de microcontroladores de sistema en un chip de bajo costo y bajo consumo con Wi-Fi integrado y Bluetooth de doble modo. Es ampliamente utilizado en aplicaciones de IoT y es compatible con MicroPython.
RP2040	RP2040 es un chip microcontrolador desarrollado por Raspberry Pi. Se utiliza en la placa de desarrollo Raspberry Pi Pico y es compatible con MicroPython.
STM32	STM32 es una serie de chips microcontroladores desarrollados por STMicroelectronics. Es ampliamente utilizado en sistemas embebidos y es compatible con MicroPython.
nRF52	nRF52 es una serie de chips microcontroladores desarrollados por Nordic Semiconductor. Se utiliza en aplicaciones de Bluetooth Low Energy (BLE) y es compatible con MicroPython.

1.2 Características de MicroPython

MicroPython ofrece varias características que lo hacen adecuado para el desarrollo de sistemas embebidos:

- **REPL Interactivo:** MicroPython proporciona un bucle de lectura-evaluación-impresión (REPL) interactivo que permite a los desarrolladores ejecutar comandos de Python directamente en el microcontrolador, lo que facilita la creación rápida de prototipos y pruebas.
- **Interfaz de Hardware:** MicroPython incluye bibliotecas y módulos que permiten a los desarrolladores interactuar con componentes de hardware como pines GPIO, I2C, SPI, UART y más, lo que facilita la interfaz con sensores, actuadores y otros periféricos.

1.3 Comenzando con MicroPython

Para comenzar con MicroPython, necesitarás una placa de microcontrolador que sea compatible con MicroPython, como el ESP32 o el RP2040. Aquí están los pasos generales para comenzar con MicroPython:

1. **Instalar el Firmware de MicroPython:** Descarga el firmware de MicroPython para tu placa de microcontrolador desde el sitio web oficial de MicroPython y flashealo en la placa usando una herramienta como esptool o rshell.
2. **Conectar a la Placa:** Usa un programa de terminal serial para conectarte a la placa a través de una conexión serial (UART) para acceder al REPL de MicroPython.
3. **Escribir y Ejecutar Código Python:** Escribe código Python usando el REPL interactivo o un editor de texto, y ejecútalo en la placa para interactuar con componentes de hardware y periféricos.

1.4 Bibliotecas y Módulos de MicroPython

MicroPython proporciona un conjunto rico de bibliotecas y módulos que permiten a los desarrolladores trabajar con componentes de hardware y periféricos. Algunas de las bibliotecas y módulos clave disponibles en MicroPython incluyen:

- **machine:** Proporciona clases y funciones para interactuar con componentes de hardware como pines GPIO, I2C, SPI, UART y más.
- **network:** Incluye clases y funciones para trabajar con interfaces de red, como Wi-Fi y Ethernet, lo que permite aplicaciones de IoT.

1.5 IDEs y Herramientas de MicroPython

Hay varios Entornos de Desarrollo Integrados (IDEs) y herramientas disponibles para trabajar con MicroPython, tales como:

- **Thonny:** Un IDE amigable para principiantes para Python que incluye soporte para MicroPython, proporcionando características como edición de código, REPL y transferencia de archivos.
- **uPyCraft:** Un IDE ligero para el desarrollo de MicroPython que incluye características como edición de código, transferencia de archivos y comunicación serial con la placa.
- **rshell:** Una herramienta de línea de comandos para acceder y gestionar archivos en una placa de MicroPython a través de una conexión serial, lo que permite la transferencia de archivos y el acceso al REPL.

2.1 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.

2.2 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.

2.3 Diferencias Clave entre Arduino y un Entorno Nativo

2.3.1 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
```

2.3.2 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.

2.3.3 Portabilidad y Optimización

Tabla 2.1: 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

2.3.4 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.

2.4 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.

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.

3.1 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**

3.2 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.

3.3 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**.

3.4 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
```

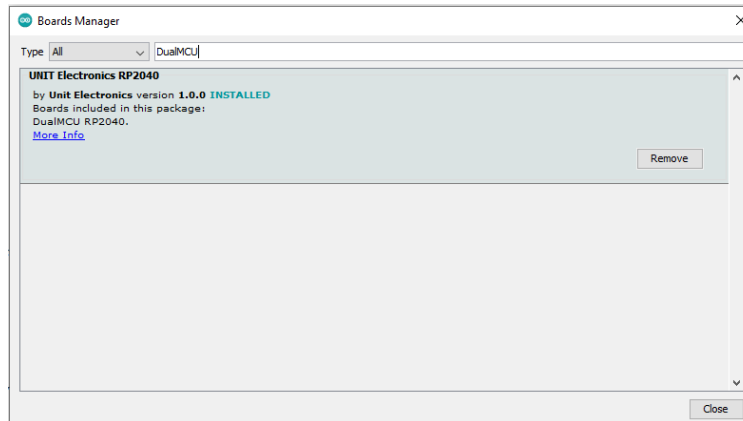


Figura 3.1: Ejemplo de instalación en el Gestor de Tarjetas.

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**.

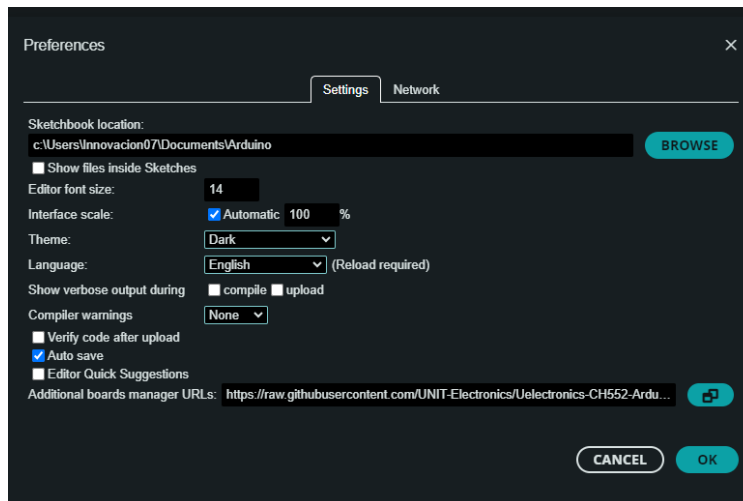


Figura 3.2: Ejemplo de instalación en el Gestor de Tarjetas.

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/).

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.

4.1 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.

4.1.1 MicroPython y Arduino IDE

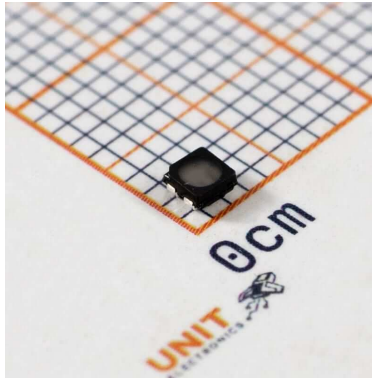


Figura 4.1: RGB_LED

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);
}
```

4.1.2 Arduino IDE y SDCC

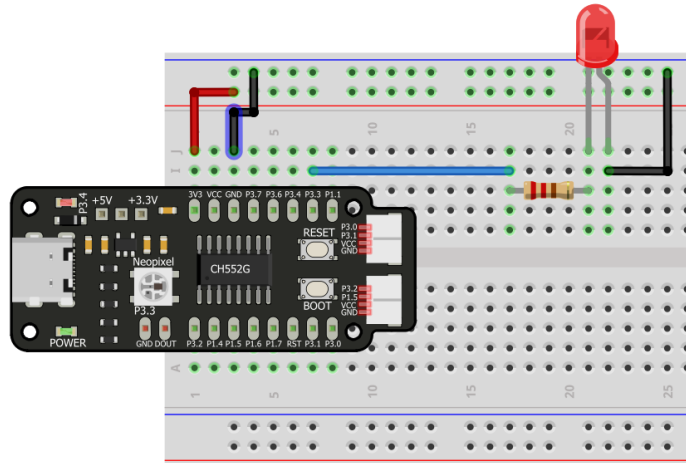


Figura 4.2: LEDs

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);
    }
}
```

(continúe en la próxima página)

(proviene de la página anterior)

```
}  
}
```

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.

5.1 Implementación

5.1.1 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);  
    }  
}
```

5.1.2 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);
}

```

5.2 Aplicaciones

5.2.1 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

# Configuración del pin de control del servomotor (puedes cambiarlo según tus conexiones)
servo_pin = machine.Pin(0) # Cambia a tu pin deseado

# Crea un objeto PWM para controlar el servomotor
pwm_servo = machine.PWM(servo_pin)

# Frecuencia del PWM para el servomotor (generalmente alrededor de 50 Hz)
pwm_servo.freq(50)

def set_servo_angle(angle):
    # Convierte el ángulo deseado (en grados) a un valor de ciclo de trabajo
    # Ten en cuenta que los valores específicos pueden variar según el servo
    duty_cycle = int(1024 + (angle / 180) * 3072)
    pwm_servo.duty_u16(duty_cycle)

try:
    while True:
        # Mueve el servomotor de 0 a 180 grados

```

(continúe en la próxima página)

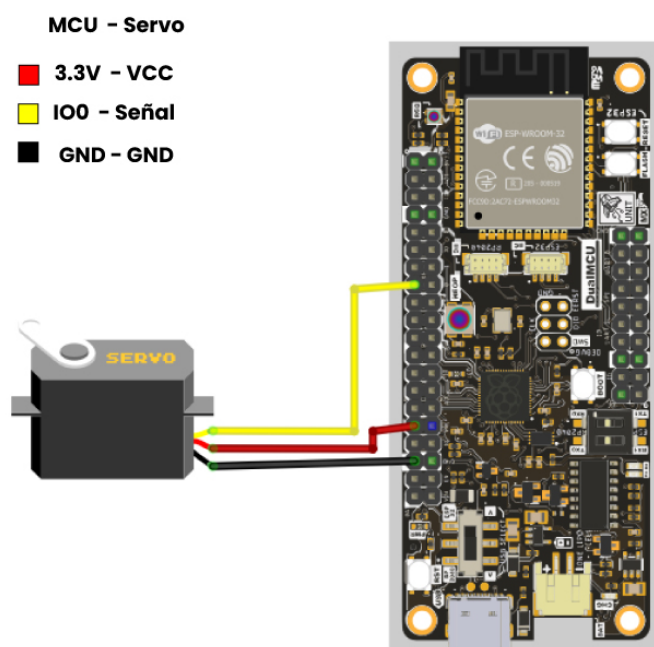


Figura 5.1: Diagrama de conexión del servomotor

(proviene de la página anterior)

```
for angle in range(0, 181, 10):
    set_servo_angle(angle)
    utime.sleep(0.1)

# Mueve el servomotor de 180 a 0 grados
for angle in range(180, -1, -10):
    set_servo_angle(angle)
    utime.sleep(0.1)

except KeyboardInterrupt:
    # Detén el PWM y limpia los recursos al interrumpir el programa con Ctrl+C
    pwm_servo.deinit()
    print("\nPWM detenido. Recursos liberados.")
```

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);
    }
}
```

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.

6.1 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.

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

6.2 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);
}
```

6.3 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);
    }
  }
}
```

Conversión de Analógico a Digital

7.1 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.

7.1.1 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.

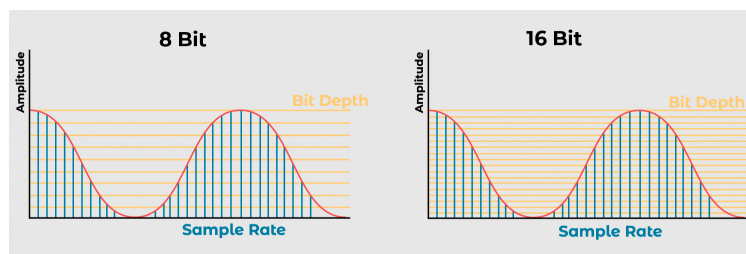


Figura 7.1: Conversión de Analógico a Digital

Tabla 7.1: Ejemplos de Codificación y Cuantificación

Resolución	Niveles de Cuantificación	Código Digital
8 bits	256	0x00 a 0xFF
12 bits	4,096	0x000 a 0xFFF
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 7.2: 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.

7.2 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 imprimirlo:

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);
}

```

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)
    }
}

```

(continúe en la próxima página)

(proviene de la página anterior)

```
}  
}
```

7.3 Clase ADC

La clase `machine.ADC` se utiliza para crear objetos ADC que pueden interactuar con los pines analógicos.

class `machine.ADC(pin)`

El constructor de la clase ADC toma un solo argumento: el número de pin.

7.4 Ejemplo de Definición

Para definir y usar un objeto ADC, sigue este ejemplo:

MicroPython

```
import machine  
adc = machine.ADC(0) # Inicializar ADC en el pin A0
```

C++

```
#define ADC0 26
```

7.5 Lectura de Valores

Para leer el valor analógico convertido a formato digital:

MicroPython

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

C++

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

SDCC

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

Cómo Generar un Informe de Error

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

8.1 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 9

Índices y tablas

- genindex
- modindex
- search

M

`machine.ADC` (*clase incorporada*), 30