UNIT ELECTRONICS

DualMCU ONE Development Board

Release 0.0.1

Cesar Bautista

Jan 20, 2025

CONTENTS

Note: The information in this file is subject to change without notice to improve or correct the documentation as needed.

Tip: You are reading the most recent version available.

This guide is designed to help developers familiarize themselves with the DualMCU ONE development board and its capabilities. Whether you're a beginner or an experienced developer, this guide will provide you with the knowledge and resources you need to start developing with the DualMCU ONE.

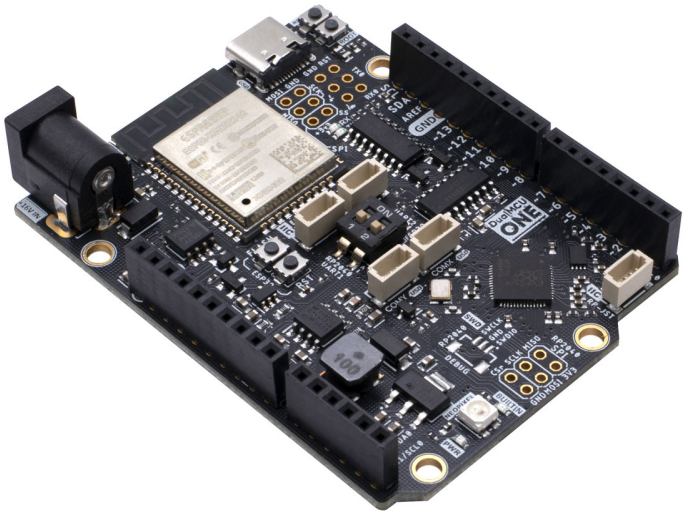


Fig. 1: DualMCU ONE board

CHAPTER

ONE

DUALMCU ONE DEVELOPMENT BOARD

1.1 Introduction

The DualMCU ONE development board is a versatile platform that combines the power of two microcontrollers: the RP2040 and the ESP32. This combination allows developers to take advantage of the unique features and capabilities of each microcontroller, making it ideal for a wide range of applications.

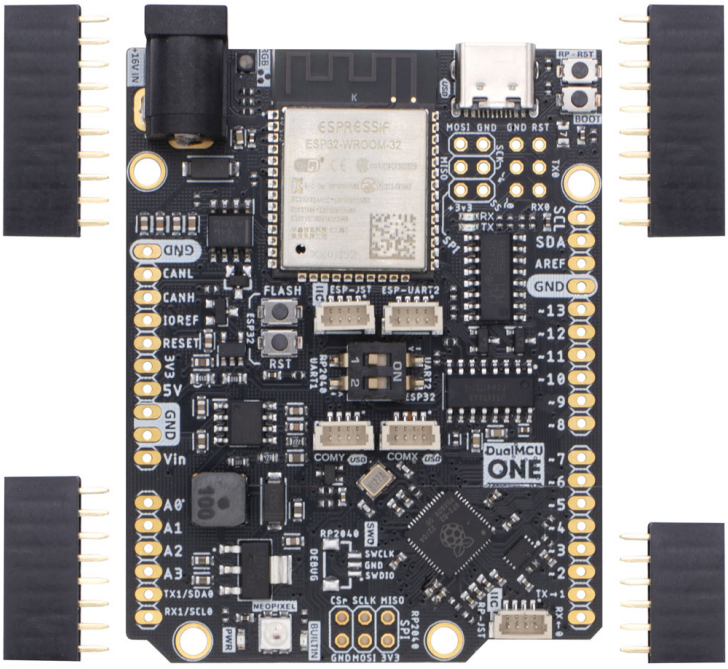


Fig. 1.1: DualMCU ONE board

The desing is inspired by the connections of Arduino UNO Rev3 , making it easy to use with a wide range of shields and accessories. The board also features a USB-C connector for power and data, a microSD card slot.

1.2 Squematic of the DualMCU ONE board

1.3 Pin Mapping

Digital and analog pins on the DualMCU ONE board are mapped to the corresponding GPIO pins on the RP2040 microcontroller. This mapping allows you to easily identify the pins you need to use for your project.

1.3.1 Distribution of analog pins

Analog pins A0 to A3, TX1, and RX1 are mapped to the corresponding GPIO pins on the RP2040 microcontroller.

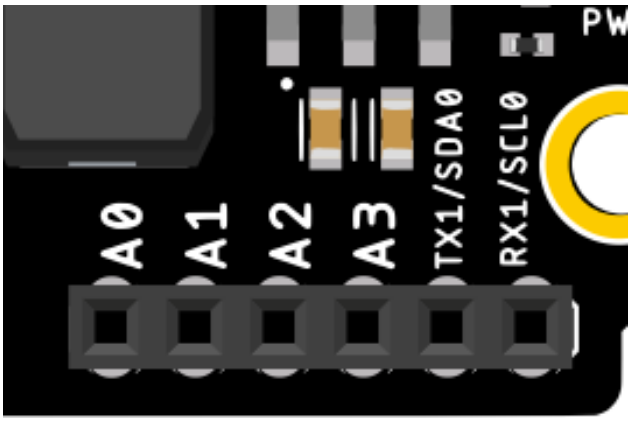


Fig. 1.2: DualMCU ONE pins A0 to A3, TX1, RX1

Table 1.1: Analog Pin Mapping

PIN BOARD	GPIO RP2040
A0	26
A1	27
A2	28
A3	29
TX1 / SDA0	22
RX1 / SCL0	23

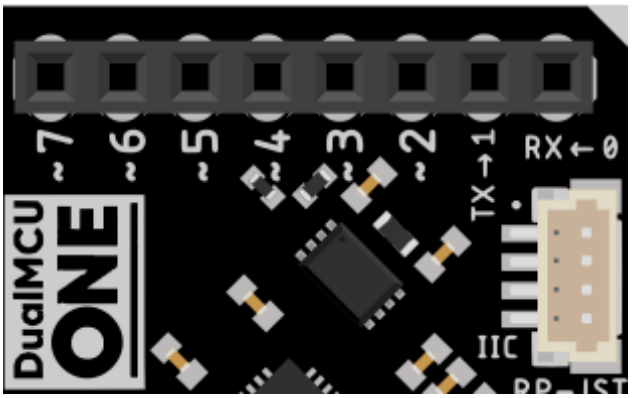


Fig. 1.3: DualMCU ONE pins D0 to D7

1.3.2 Distribution of digital pins D0 to D7

Table 1.2: Digital Pin Mapping

PIN BOARD	Special Function	GPIO RP2040
D0	RX0	GPIO1
D1	TX0	GPIO0
D2	RX1	GPIO5
D3	TX1	GPIO4
D4	•	GPIO9
D5	•	GPIO11
D6	•	GPIO8
D7	•	GPIO10

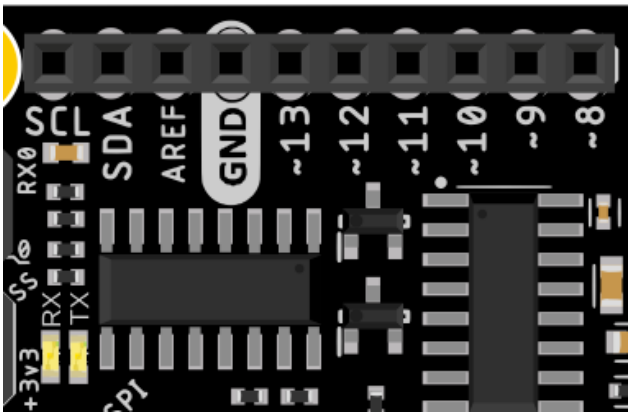


Fig. 1.4: DualMCU ONE pins D8 to D13

1.3.3 Distribution of digital pins D8 to D13

Table 1.3: Digital Pin Mapping 2

PIN BOARD	Special Function	GPIO RP2040
D8	•	GPIO2
D9	•	GPIO3
D10	•	GPIO17
D11	•	GPIO19
D12	•	GPIO16
D13	•	GPIO18
SDA0	•	GPIO20
SCL0	•	GPIO21

1.4 RP2040 Microcontroller

The RP2040 is a powerful microcontroller featuring a dual-core ARM Cortex-M0+ processor, 264KB of SRAM, and a wide range of peripherals.

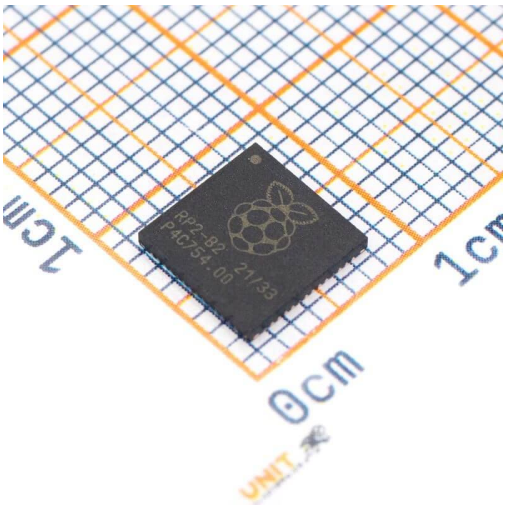


Fig. 1.5: RP2040 chip

The **RP2040** is Raspberry Pi's microcontroller, engineered to deliver high performance, cost-efficiency, and user-friendliness within the microcontroller domain.

1.4.1 Features

The RP2040 boasts several advanced features, including:

- **Symmetrical Dual-Core Processor**: Equipped with dual ARM Cortex-M0+ processors operating at 133MHz.
- **Large On-Chip Memory**: 264kB of SRAM divided into six independent banks.
- **Deterministic Bus Fabric**: Ensures reliable and predictable operation.
- **Rich Peripheral Set**: Enhanced with Raspberry Pi's unique Programmable I/O (PIO) subsystem.

Professional users will find the RP2040's power and flexibility unrivaled, thanks to its comprehensive documentation, polished MicroPython port, and UF2 bootloader in ROM, which lowers the entry barrier for beginners and hobbyists.

1.4.2 Memory and Storage

The RP2040 is a stateless device supporting cached execute-in-place from external QSPI memory. This design allows users to choose the appropriate density of non-volatile storage for their applications, benefiting from the low pricing of standard flash parts.

1.4.3 Manufacturing and Power Efficiency

Manufactured on a modern 40nm process node, the RP2040 delivers:

- **High Performance**
- **Low Dynamic Power Consumption**
- **Minimal Leakage**

It includes various low-power modes, enabling extended-duration operation on battery power.

1.4.4 Technical Specifications

Key Features:

- **Dual ARM Cortex-M0+ @ 133MHz**
- **264kB on-chip SRAM** in six independent banks
- **Support for up to 16MB of off-chip Flash memory** via dedicated QSPI bus
- **DMA controller**
- **Fully-connected AHB crossbar**
- **Interpolator and integer divider peripherals**
- **On-chip programmable LDO** to generate core voltage
- **2 on-chip PLLs** to generate USB and core clocks
- **30 GPIO pins**, 4 of which can be used as analog inputs

Peripherals:

- **2 UARTs**
- **2 SPI controllers**
- **2 I2C controllers**
- **16 PWM channels**
- **USB 1.1 controller and PHY**, with host and device support
- **8 PIO state machines**

Tip: For more information about the RP2040 microcontroller, refer to the official documentation available on the Raspberry Pi website.

1.5 ESP32 Microcontroller

The **ESP32** was initially released in September 2016 by Espressif Systems. Since then, it has gained immense popularity in the hardware development and IoT project communities due to its versatility, power, and affordability. It is used in a wide range of applications, from home projects to industrial solutions.

The ESP32 is a series of low-cost, low-power microcontrollers with integrated Wi-Fi and Bluetooth capabilities. Developed by Espressif Systems, it is widely used in various Internet of Things (IoT) applications, including home automation, wearable devices, and industrial automation. The ESP32 offers a rich set of peripherals, including GPIO, SPI,

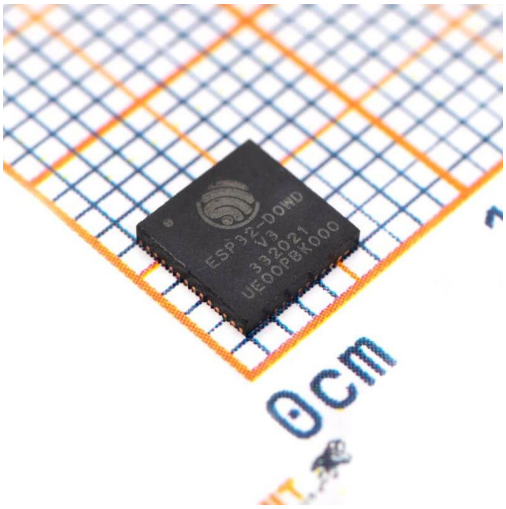


Fig. 1.6: ESP32 chip

I2C, UART, ADC, DAC, and more, making it versatile for a wide range of projects. Its popularity stems from its affordability, flexibility, and robustness. Developers often use the Arduino IDE or the ESP-IDF (Espressif IoT Development Framework) to program the ESP32.

1.5.1 Specifications

The specifications of the ESP32 can vary slightly between different variants and models, but here is an overview of common features:

Microcontroller: Dual-core Tensilica LX6/LX7 microprocessor

Clock Speed: Up to 240 MHz

RAM: 520 KB to 4 MB

Flash Memory: 4 MB to 16 MB

Wi-Fi: 802.11 b/g/n (2.4 GHz)

Bluetooth: Bluetooth 4.2 and Bluetooth 5.0 BLE (Bluetooth Low Energy)

1.5.2 Peripherals

- **GPIO (General Purpose Input/Output)**
- **SPI (Serial Peripheral Interface)**
- **I2C (Inter-Integrated Circuit)**
- **UART (Universal Asynchronous Receiver-Transmitter)**
- **ADC (Analog to Digital Converter)**
- **DAC (Digital to Analog Converter)**
- **PWM (Pulse Width Modulation)**
- **RTC (Real Time Clock)**
- **Touch sensor**

- **Interrupt controller**

1.5.3 Security

Support for cryptography, including SSL/TLS, WPA/WPA2, and AES.

1.5.4 Power Consumption

Low-power modes to conserve battery life in IoT applications.

1.5.5 Development Interfaces

Compatible with the Arduino IDE and Espressif's ESP-IDF for software development.

1.5.6 Physical Dimensions

Dimensions vary depending on the specific module, but they are generally compact and suitable for embedded applications.

Caution: These are the general specifications; however, depending on the manufacturer and the exact model of the ESP32, there may be differences in specific features and additional capabilities.

CHAPTER

TWO

PINOUT DUALMCU ONE BOARD

The section below provides an overview of the pinout for the DualMCU ONE development board. The pinout details the GPIO connections for both the ESP32 and RP2040 microcontrollers.

2.1 Top View

2.2 Bottom View

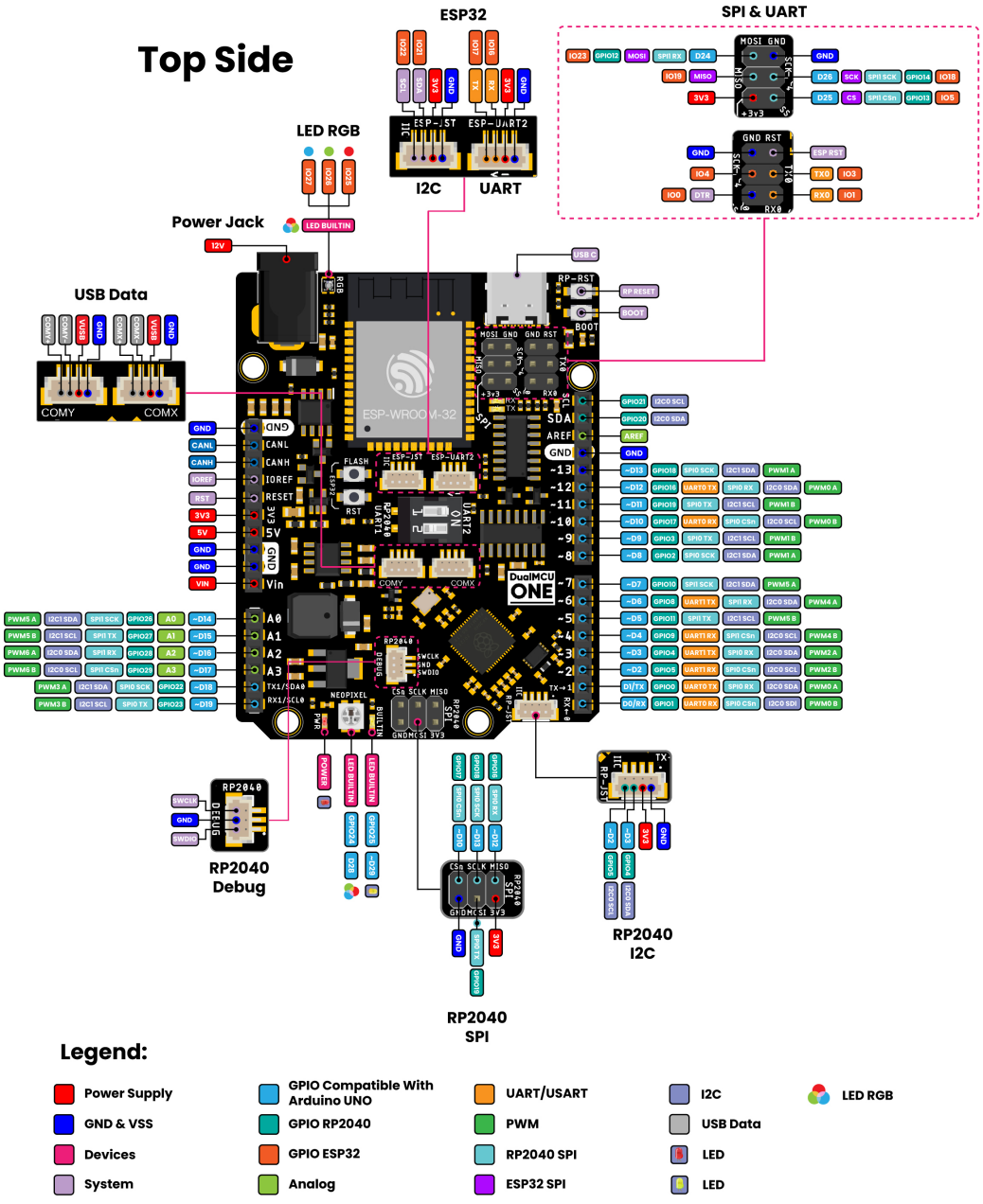


Fig. 2.1: DualMCU ONE board

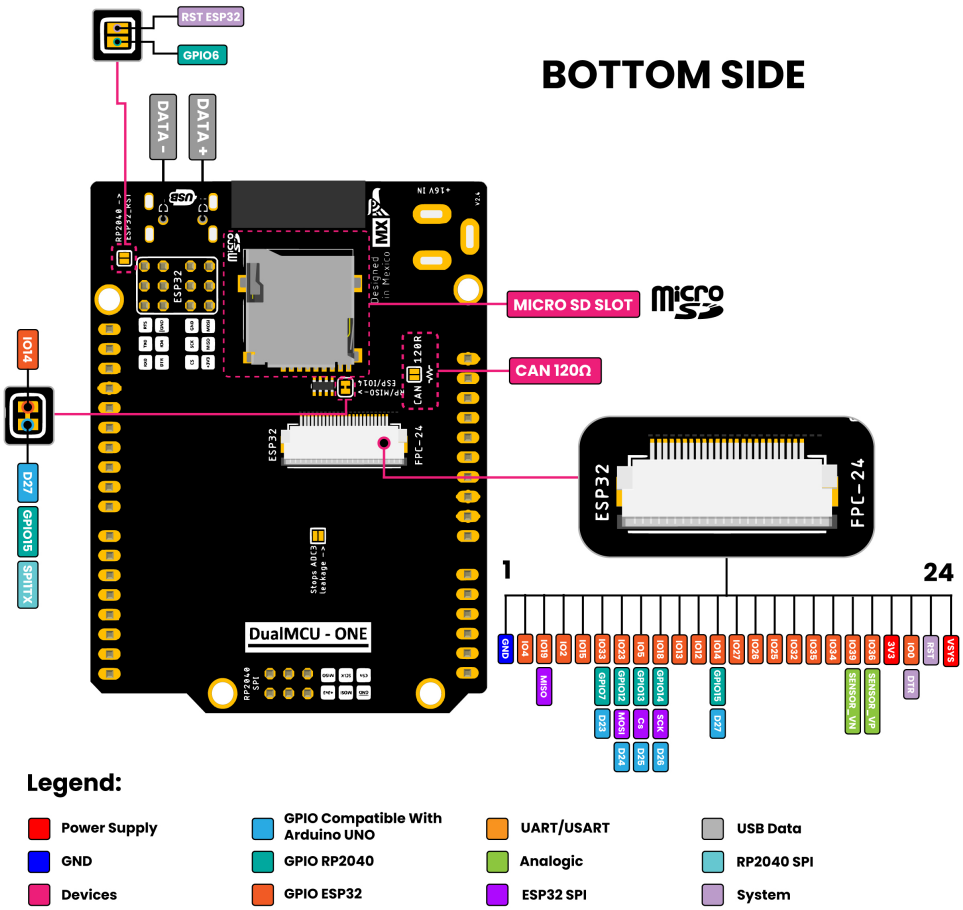


Fig. 2.2: DualMCU ONE board

CHAPTER

THREE

DESKTOP ENVIRONMENT

The environment setup is the first step to start working with the DualMCU ONE board. The following steps will guide you through the setup process.

1. Install the required software
2. Set up the development environment
3. Install the required libraries
4. Set up the board

3.1 Install the required software

The following software is required to start working with the DualMCU ONE board:

1. **Python 3.7 or later**: Python is required to run the scripts and tools provided by the DualMCU ONE board.
2. **Git**: Git is required to clone the DualMCU ONE board repository.
3. **MinGW**: MinGW is a native Windows port of the GNU Compiler Collection (GCC), with freely distributable import libraries and header files for building native Windows applications.
4. **Visual Studio Code**: Visual Studio Code is a code editor that is required to write and compile the code.

This section will guide you through the installation process of the required software.

3.2 Python 3.7 or later

Python is a programming language that is required to run the scripts and tools,

To install Python, follow the instructions below:

1. Download the Python installer from the:

2. Run the installer and follow the instructions.

> **Attention:** Make sure to check the box that says “Add Python to PATH” during the installation process.

Open a terminal and run the following command to verify the installation:

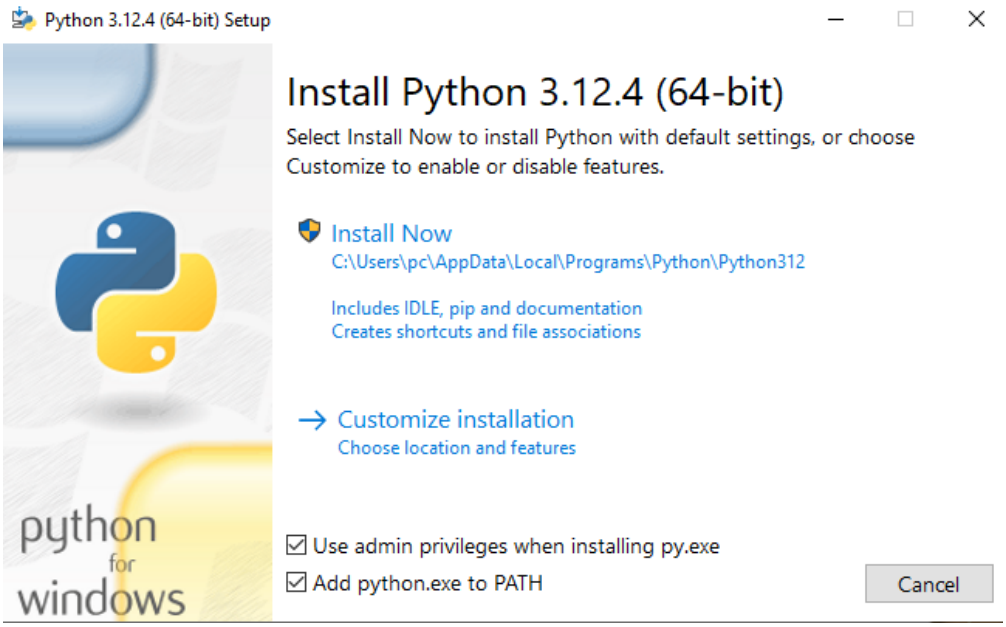


Fig. 3.1: Add python to PATH

```
python --version
```

If the installation was successful, you should see the Python version number.

3.3 Git

Git is a version control system that is required to clone the repositories in general. To install Git, follow the instructions below:

1. Download the Git installer from the

2. Run the installer and follow the instructions.
3. Open a terminal and run the following command to verify the installation:

```
git --version
```

If the installation was successful, you should see the Git version number.

3.4 MinGW

MinGW is a native Windows port of the GNU Compiler Collection (GCC), with freely distributable import libraries and header files for building native Windows applications. MinGW provides a complete Open Source programming toolset that is suitable for the development of native Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs. MinGW, being Minimalist, does not, and never will, attempt to provide a POSIX runtime environment for POSIX application deployment on MS-Windows. If you want POSIX application deployment on this platform, please consider Cygwin instead.

To install MinGW, follow the instructions below:

1. Download the MinGW installer from the

2. Run the installer and follow the instructions.

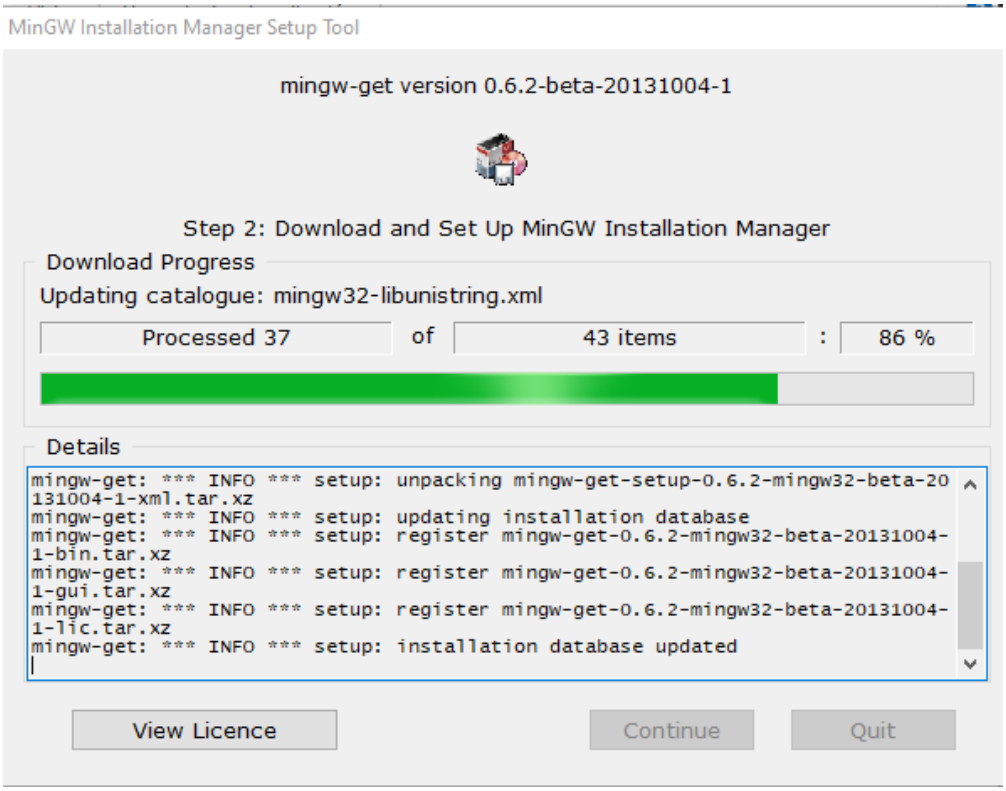


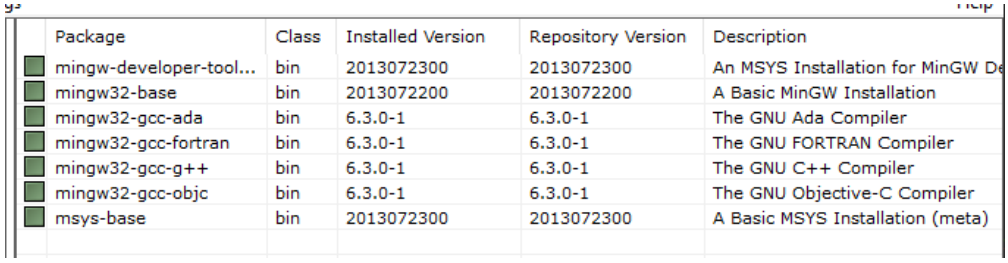
Fig. 3.2: MinGW installer

Note: During the installation process, make sure to select the following packages:

- mingw32-base
- mingw32-gcc-g++
- msys-base

---

3. Open a terminal and run the following command to verify the installation:



Package	Class	Installed Version	Repository Version	Description
mingw-developer-tool...	bin	2013072300	2013072300	An MSYS Installation for MinGW De
mingw32-base	bin	2013072200	2013072200	A Basic MinGW Installation
mingw32-gcc-ada	bin	6.3.0-1	6.3.0-1	The GNU Ada Compiler
mingw32-gcc-fortran	bin	6.3.0-1	6.3.0-1	The GNU FORTRAN Compiler
mingw32-gcc-g++	bin	6.3.0-1	6.3.0-1	The GNU C++ Compiler
mingw32-gcc-objc	bin	6.3.0-1	6.3.0-1	The GNU Objective-C Compiler
msys-base	bin	2013072300	2013072300	A Basic MSYS Installation (meta)

Fig. 3.3: MinGW installation

```
mingw --version
```

If the installation was successful, you should see the MinGW version number.

3.4.1 Environment Variable Configuration

Remember that for Windows operating systems, an extra step is necessary, which is to open the environment variable -> Edit environment variable:

```
C:\MinGW\bin
```

3.4.2 Locate the file

After installing MinGW, you will need to locate the *mingw32-make.exe* file. This file is typically found in the *C:/MinGW/bin* directory. Once located, rename the file to *make.exe*.

3.4.3 Rename it

After locating *mingw32-make.exe*, rename it to *make.exe*. This change is necessary for compatibility with many build scripts that expect the command to be named *make*.

Warning: If you encounter any issues, create a copy of the file and then rename the copy to *make.exe*.

3.4.4 Add the path to the environment variable

Next, you need to add the path to the MinGW bin directory to your system's environment variables. This allows the *make* command to be recognized from any command prompt.

1. Open the Start Search, type in "env", and select "Edit the system environment variables".
2. In the System Properties window, click on the "Environment Variables" button.
3. In the Environment Variables window, under "System variables", select the "Path" variable and click "Edit".
4. In the Edit Environment Variable window, click "New" and add the path:

   ```
   C:\MinGW\bin
   ```

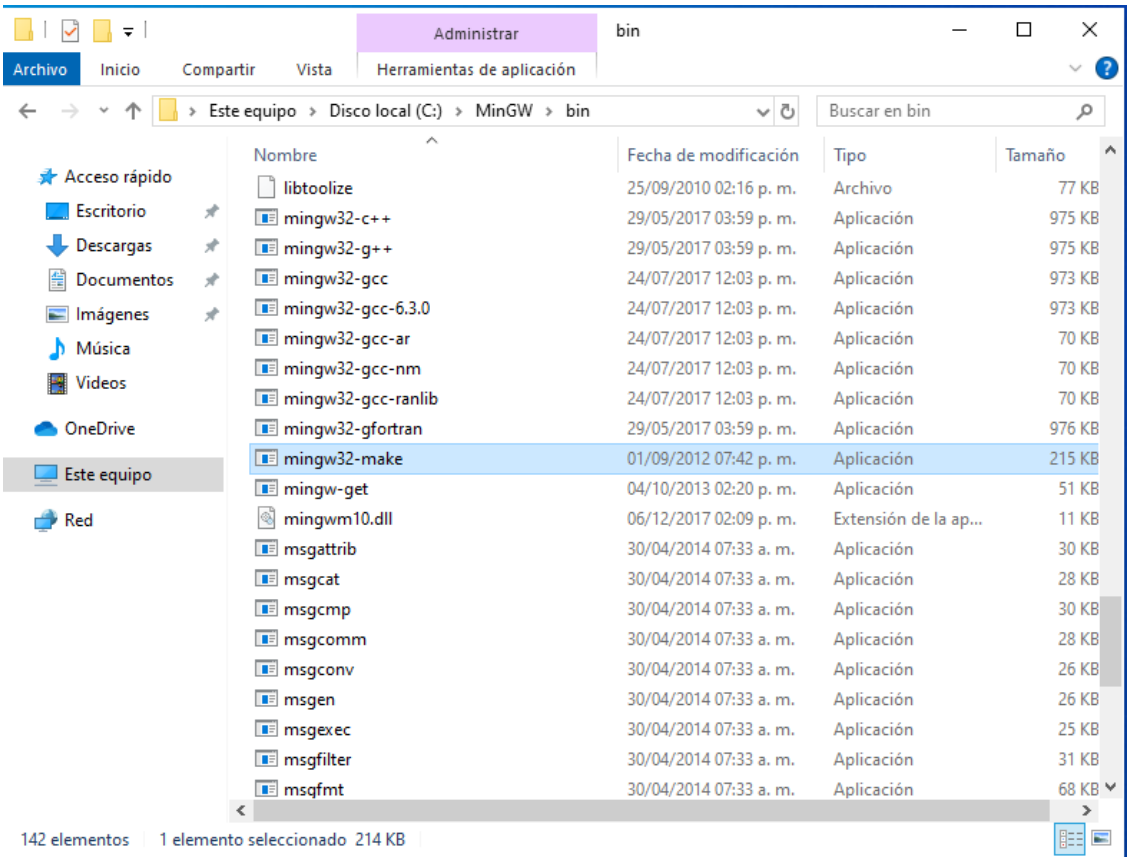


Fig. 3.4: Locating the *mingw32-make.exe* file

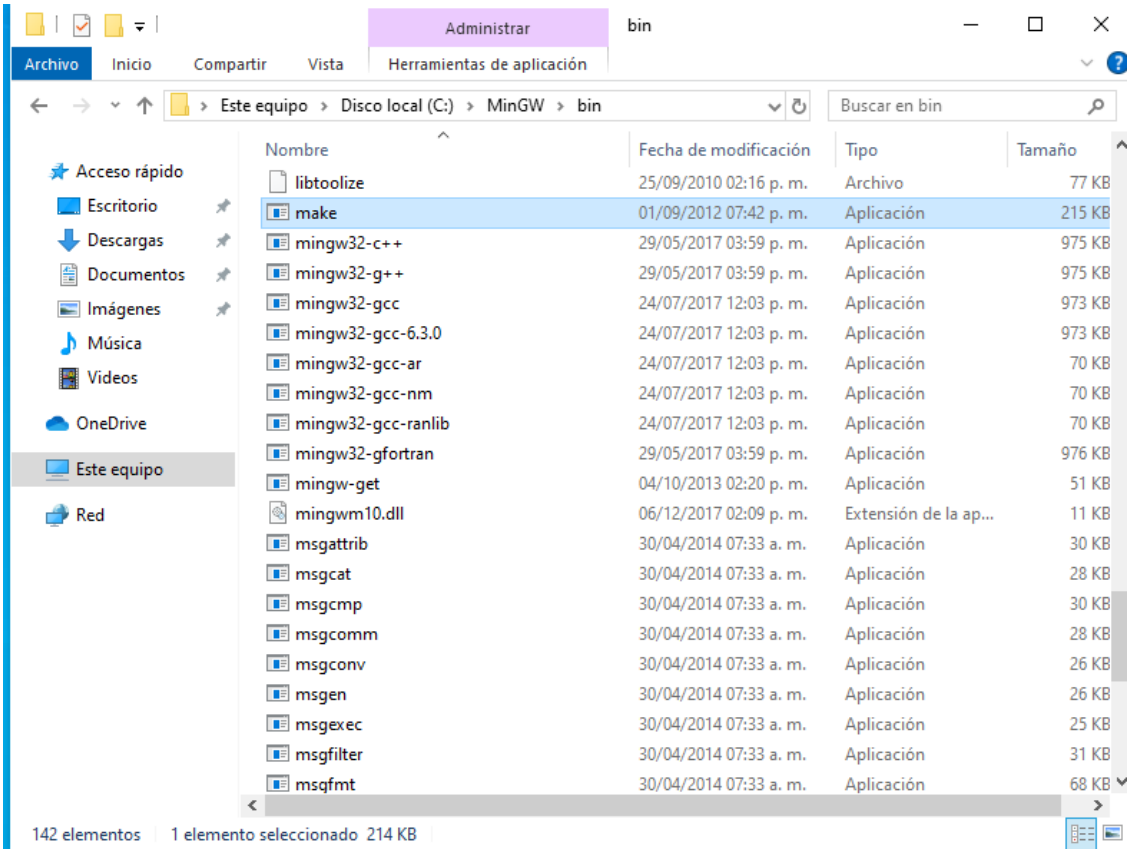


Fig. 3.5: Renaming *mingw32-make.exe* to *make.exe*

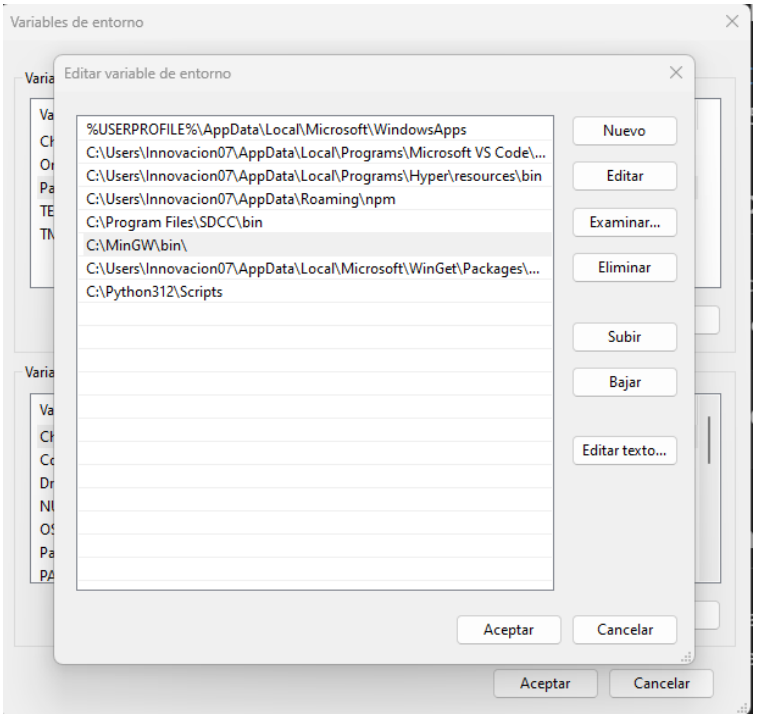


Fig. 3.6: Adding MinGW bin directory to environment variables

3.5 Visual Studio Code

Visual Studio Code is a code editor that is required to write and compile the code.

To install Visual Studio Code, follow the instructions below:

1. Download the Visual Studio Code installer from the

2. Run the installer and follow the instructions.

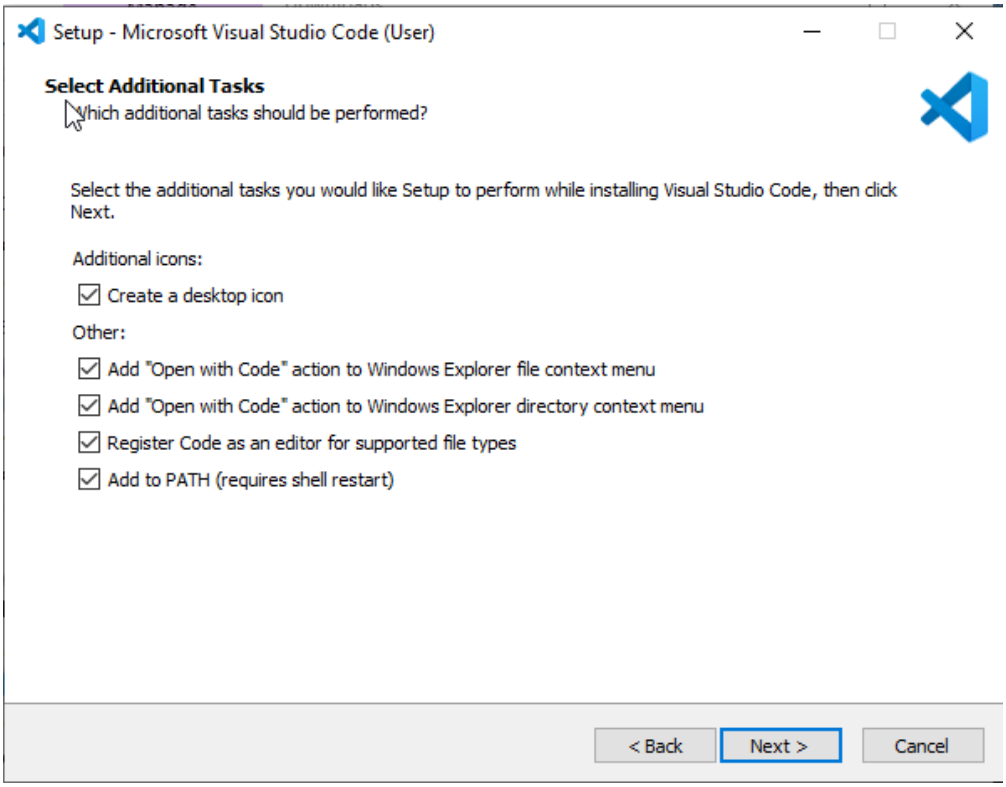


Fig. 3.7: Visual Studio Code installer

Note: During the installation process, make sure to check the box that says "Open with Code".

3. Open a terminal and run the following command to verify the installation:

```
code --version
```

4. Install extensions for Visual Studio Code:

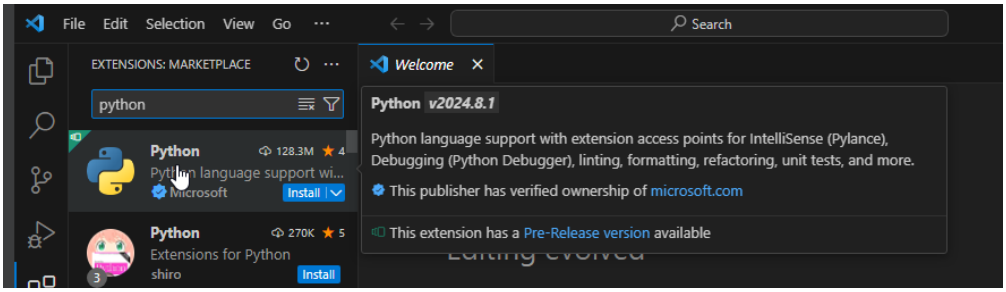


Fig. 3.8: Visual Studio Code extensions

CHAPTER

FOUR

ENVIRONMENT SETUP

4.1 MicroPython Installation on DualMCU

This repository contains a basic example of how to install MicroPython on the DualMCU using the ESP32 microcontroller. The main goal is for you to find this repository useful and be able to incorporate parts of this implementation into your projects.

4.1.1 Environment Setup

Before you start, it is recommended to perform the following setup:

Install MicroPython

This will allow you to download the firmware to the DualMCU using the Thonny IDE.

1. Go to **"Run"** -> **"Configure interpreter"** to complete the setup.

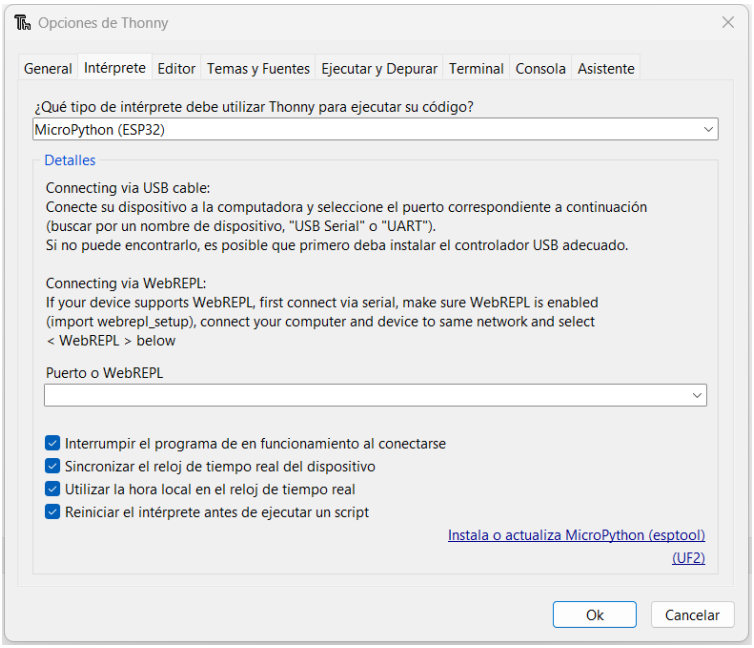


Fig. 4.1: Interpreter Configuration

4.1.2 Firmware Update

ESP32

To use MicroPython, it is recommended to consider an update. Follow these steps to update your DualMCU ESP32:

1. Start your DualMCU ESP32 by pressing the FLASH button.
2. Click on **"Install or Update MicroPython"**.
3. A new window will open. Use the following configuration:
    - **Variant**: Espessif ESP32/WROOM
    - **Version**: 1.20.0

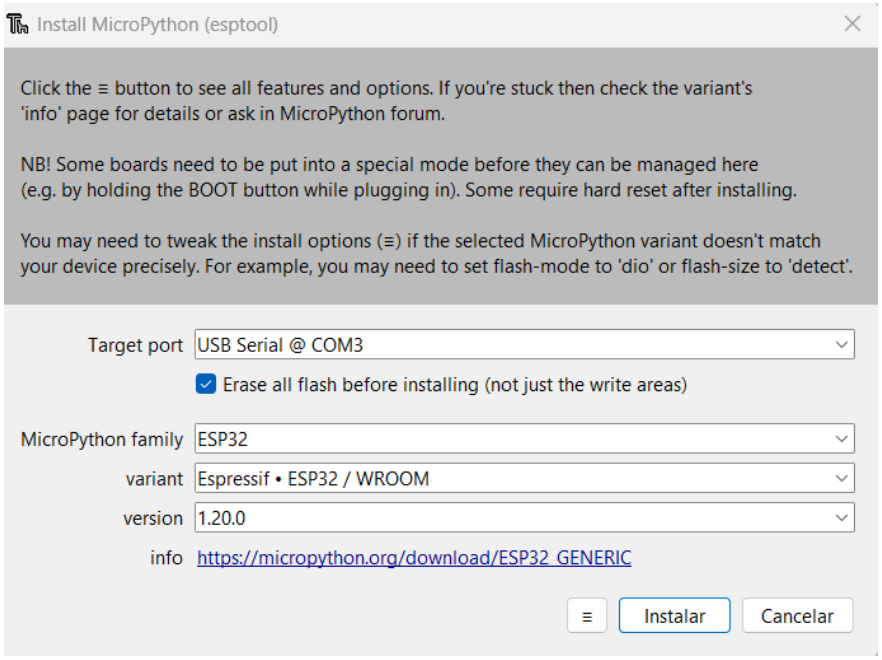


Fig. 4.2: ESP32 Installer Configuration

4. Press **Install** and wait for the installation to finish.

These steps will allow you to update and properly configure MicroPython on your DualMCU ESP32.

RP2040

To use MicroPython, it is recommended to consider an update. Follow these steps to update your DualMCU RP2040:

1. Start your DualMCU RP2040 by pressing the FLASH button.
2. Click on **"Install or Update MicroPython"**.
3. A new window will open. Use the following configuration:
    - **Variant**: Raspberry Pi Pico / Pico H
    - **Version**: 1.23.0
4. Press **Install** and wait for the installation to finish.

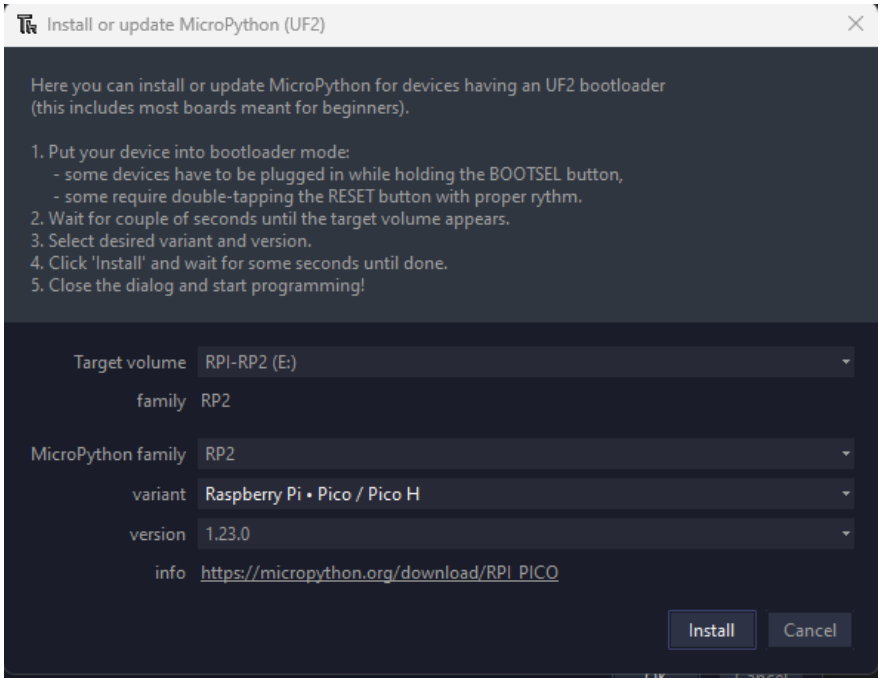


Fig. 4.3: RP2040 Installer Configuration

These steps will allow you to update and properly configure MicroPython on your DualMCU ESP32.

4.1.3 Running the Example

Once you have set up the environment, you can open and run the example, open Thonny, and follow these steps:

1. Go to the bottom right corner and select the **"MicroPython (ESP32)"** option.

Fig. 4.4: ESP32 Interpreter

Blink Example

Inside the **Examples** folder, you will find a basic example called "blink" that you can use to verify that the setup was applied correctly.

```
'''
Unit Electronics 2023
       (o_
(o_    //\
(/)_   V_/_

version: 0.0.1
revision: 0.0.1
context: This code is a basic configuration of three RGB LEDs
'''
import machine
import time

led_pin = machine.Pin(4, machine.Pin.OUT)
led_pin2 = machine.Pin(26, machine.Pin.OUT)
led_pin3 = machine.Pin(25, machine.Pin.OUT)

def loop():
     while True:
        led_pin.on()
        led_pin2.on()
        led_pin3.on()
        time.sleep(1)
        led_pin.off()
        led_pin2.off()
        led_pin3.off()
        time.sleep(1)

loop()
```

4.2 Arduino IDE Installation environment

Uelectronics Arduino core is a ported version of the Raspberry Pi Pico Arduino Core based on the great work of earlephilhower Earle F. Philhower, III. This port of the RP2040 uses the Raspberry Pi Pico SDK and a custom GCC 10.3/Newlib 4.0 toolchain, the same as earlephilhower version 2.6.4.

4.3 Documentation

See https://github.com/UNIT-Electronics/DualMCU along with the examples for more detailed usage information.

4.4 Supported Boards

- DualMCU RP2040
- Raspberry Pi Pico
- Raspberry Pi Pico W
- Generic (configurable flash, I/O pins)

4.5 Installing via Arduino Boards Manager

Open up the Arduino IDE and go to File->Preferences.

In the dialog that pops up, enter the following URL in the "Additional Boards Manager URLs" field:

https://github.com/UNIT-Electronics/Uelectronics-RP2040-Arduino-Package/releases/download/v1.0.0/package_Uelectronics_rp2040_index.json

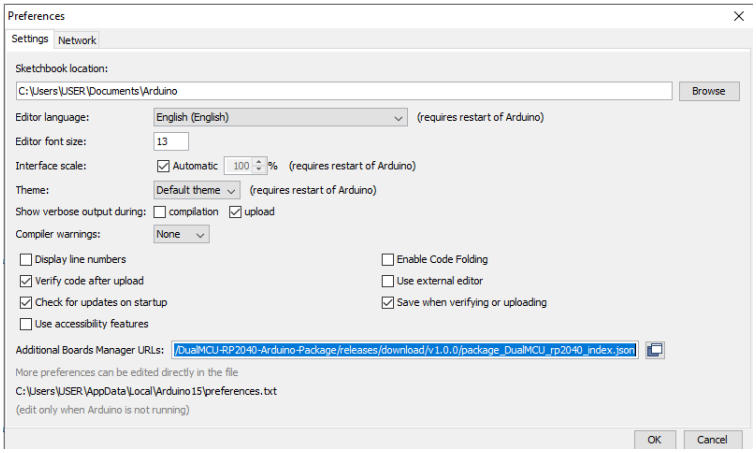


Fig. 4.5: Preferences-AditionalBoardsManagerURL

Hit OK to close the dialog.

Go to Tools->Boards->Board Manager in the IDE

Type "DualMCU" in the search box and select "Add":

The Uelectronics-ESP32-Arduino package is a collection of software tools that enable users to program and control devices using the ESP32 MCU on the DualMCU and the Arduino platform. The package includes a set of libraries and tools for programming the ESP32 using the Arduino (IDE).

The package includes a range of sample code and examples to help users get started with programming the ESP32 and creating connected devices.

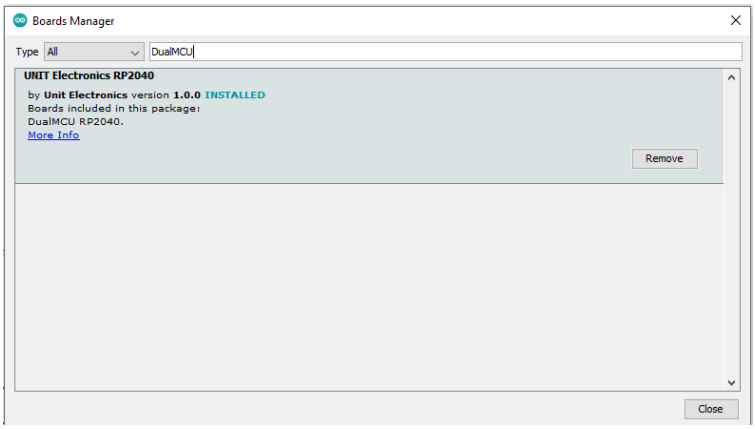


Fig. 4.6: BoardsManager

4.6 Supported Boards

- UNIT DualMCU ESP32
- ESP32 Dev Module
- ESP32S3 Dev Module
- ESP32C3 Dev Module
- ESP32S2 Dev Module

4.7 Installing via Arduino Boards Manager

Open up the Arduino IDE and go to File->Preferences.

- Stable release link: *https://github.com/UNIT-Electronics/Uelectronics-ESP32-Arduino-Package/releases/download/v1.0.0/package_Uelectronics_ESP32_index.json*

Arduino allows installation of third-party platform packages using Boards Manager.

- Install the current upstream Arduino IDE at the 1.8 level or later. The current version is at the Arduino website.
- Start Arduino and open Preferences window.
- Enter one of the release links above into *Additional Board Manager URLs* field. You can add multiple URLs, separating them with commas.
- Open Boards Manager from Tools > Board menu and install *esp32* platform (and don't forget to select your ESP32 board from Tools > Board menu after installation).

4.7.1 Support

The DualMCU development board is compatible with both the MicroPython Integrated Development Environment (IDE), such as Thonny, and the Arduino development environment. This compatibility allows you to program the DualMCU using MicroPython, CircuitPython, or the Arduino programming language.

MicroPython IDE support includes an interactive console (REPL) for executing commands immediately in MicroPython and CircuitPython, enabling quick and easy code testing and debugging.

Furthermore, support for the Arduino development environment allows you to leverage the extensive tools and community resources available through Arduino. This can be particularly beneficial if you are already familiar with Arduino and wish to utilize its user-friendly features and abundant resources for developing projects with the DualMCU development board.

Arduino Package RP2040	**https://github.com/UNIT-Electronics/Uelectronics-RP2040-Arduino-Package**
Arduino Package ESP32	**https://github.com/UNIT-Electronics/Uelectronics-ESP32-Arduino-Package**

CHAPTER

FIVE

INSTALLING PACKAGES

This section will guide you through the installation process of the required libraries using the pip package manager.

5.1 Unit Electronics Library Development

Use method below to install different libraries developed by Unit Electronics.

Open a terminal and run the following command to install the library using pip:

```
pip install <name-library>
```

For example, to install the library chatos, run the following command:

```
pip install chatos
```

If the installation was successful, open a terminal and run the following command to verify the installation:

```
python -m chatos
```

5.1.1 Libraries available

- Chatos : The library provides a set of tools to help developers work with the Chatos board. Stablish a communication between the computer and the microcontroller CH552 using the serial port to 9600 baud rate.
- Loadupch : The library is a tool by load the firmware to the CH552 microcontroller.

5.1.2 DualMCU Library

Firstly, you need install Thonny IDE. You can download it from the Thonny website.

1. Open Thonny.
2. Navigate to **Tools** -> **Manage Packages**.
3. Search for `dualmcu` and click **Install**.
4. Successfully installed the library.

Alternatively, download the library from dualmcu.py.

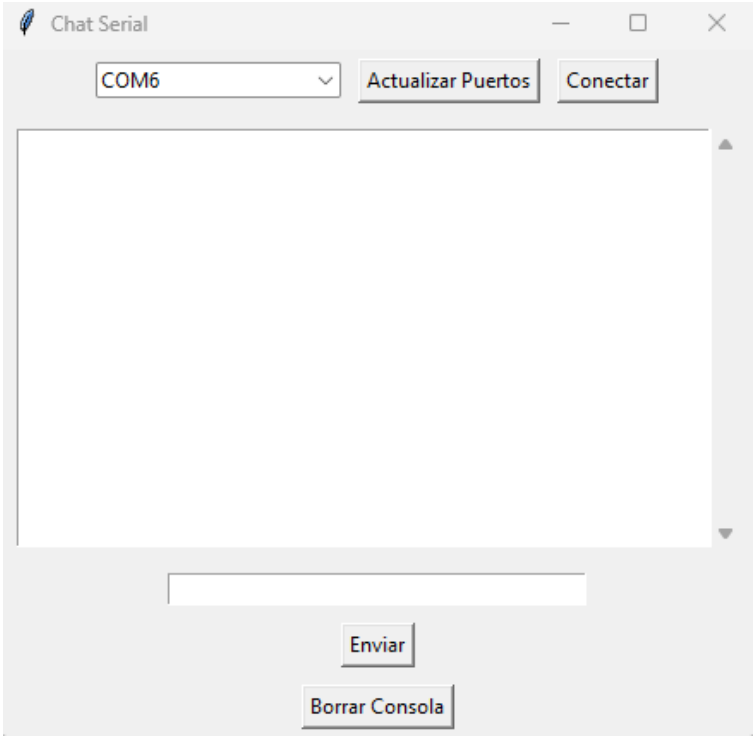


Fig. 5.1: Chatos Library Successfully Installed

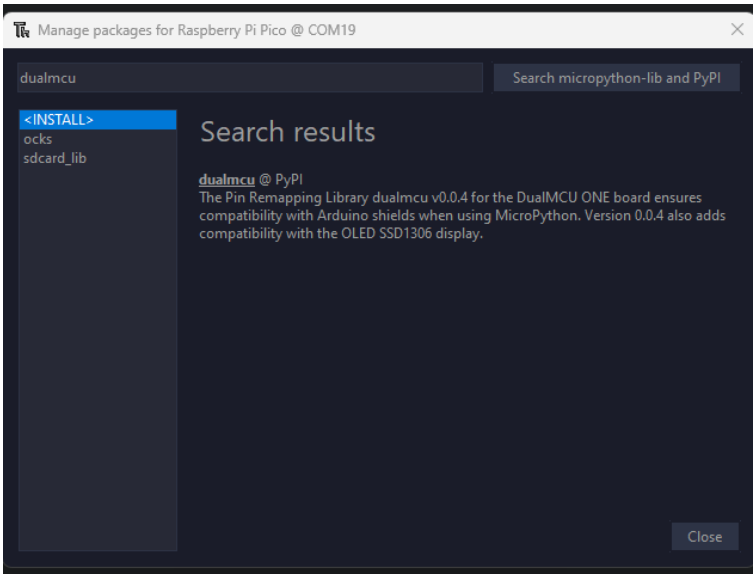


Fig. 5.2: DualMCU Library

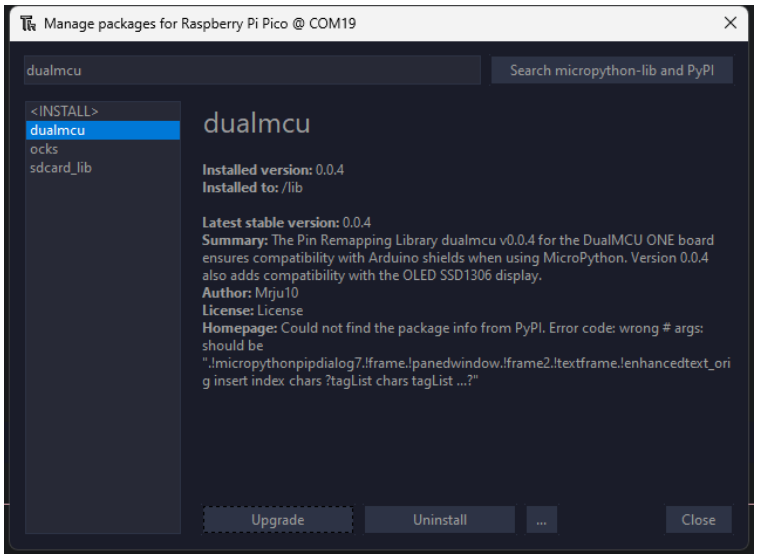


Fig. 5.3: DualMCU Library Successfully Installed

Usage

The library provides a set of tools to help developers work with the DualMCU ONE board. The following are the main features of the library:

- **I2C Support**: The library provides support for I2C communication protocol, making it easy to interface with a wide range of sensors and devices.
- **Arduino Shields Compatibility**: The library is compatible with Arduino Shields, making it easy to use a wide range of shields and accessories with the DualMCU ONE board.
- **SDcard Support**: The library provides support for SD cards, allowing developers to easily read and write data to SD cards.

Examples of the library usage:

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

oled.show()
```

Libraries available

- Dualmcu : The library provides a set of tools to help developers work with the DualMCU ONE board. The library is actively maintained and updated to provide the best experience for developers working with the DualMCU ONE board. For more information and updates, visit the *dualmcu GitHub repository* `
- Ocks : The library provides support for I2C communication protocol.
- SDcard-lib : The library provides support for SD cards, allowing developers to easily read and write data to SD cards; all rights remain with the original author.

The library is actively maintained and updated to provide the best experience for developers working with the DualMCU ONE board. For more information and updates, visit the *dualmcu GitHub repository* `

CHAPTER

SIX

GENERAL PURPOSE INPUT/OUTPUT (GPIO) RP2040 & ESP32

The General Purpose Input/Output (GPIO) pins on microcontrollers like the RP2040 and ESP32 are versatile pins that can be configured as either input or output pins. They can be used to read digital signals from sensors, control LEDs, drive motors, and interface with other digital devices.

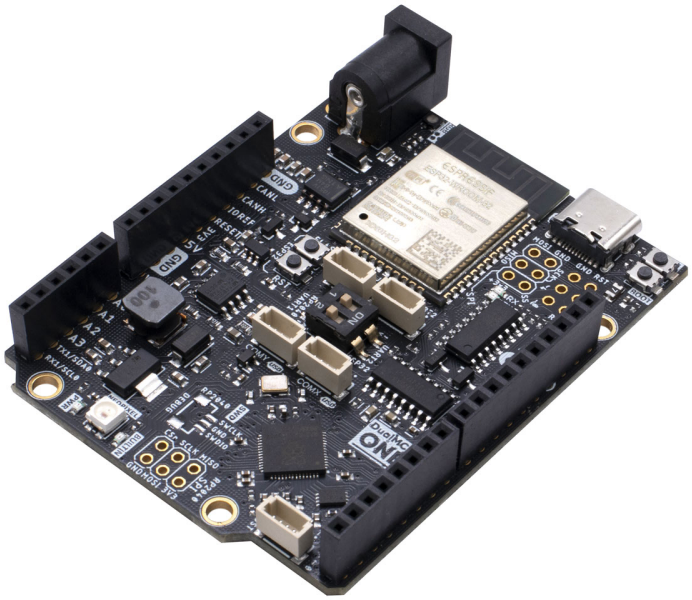


Fig. 6.1: DualMCU ONE board

The DualMCU ONE development board is equipped with 22 GPIO pins, 4 of which can be used as analog inputs. The ESP32 microcontroller also has a large number of GPIO pins but these pins require a expansion board to be used.

Let's start with a simple example: blinking an LED. This example will demonstrate how to control GPIO pins on the DualMCU ONE board using both MicroPython.

6.1 LEDs ESP32 & RP2040

In this section, we will learn how to work an RGB LED using a microcontroller. We will learn how to use the LED to different GPIO pins and control its on and off states with a simple program.

6.1.1 Hardware Description

An RGB LED has three LEDs in one: red, green, and blue. You can control the color of the LED by varying the intensity of each of the LEDs. Here is an image of the RGB LED:

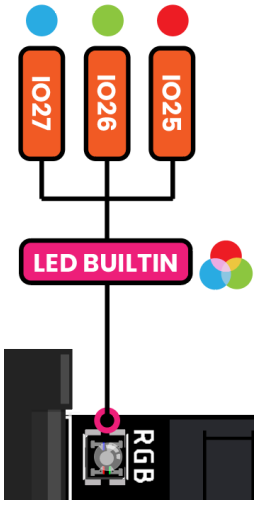


Fig. 6.2: RGB LED

6.1.2 Pin Connections

Table 6.1: RGB LED Connections

PIN	GPIO ESP32	GPIO RP2040
BLUE	27	
RED	25	25
GREEN	26	

6.1.3 RGB LED Code ESP32

Tip: This code block is designed to work exclusively with the RGB LED on the DualMCU development board when using the ESP32 microcontroller.

MicroPython

```
import machine
import time
```

(continues on next page)

(continued from previous page)

```
led_pin = machine.Pin(27, machine.Pin.OUT)
led_pin2 = machine.Pin(26, machine.Pin.OUT)
led_pin3 = machine.Pin(25, machine.Pin.OUT)

def loop():
    while True:
        led_pin.on()
        led_pin2.on()
        led_pin3.on()
        time.sleep(1)
        led_pin.off()
        led_pin2.off()
        led_pin3.off()
        time.sleep(1)

loop()
```

C++

```
#define LED 25

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(1000);                       // wait for a second
    digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
    delay(1000);                       // wait for a second
}
```

6.1.4 LED Blink Code RP2040

Tip: This code block is designed to work exclusively with RP2040 microcontroller on the DualMCU development board.

Caution: A single LED is connected to GPIO 25 on the RP2040 microcontroller.

MicroPython

```
import machine
import time
```

(continues on next page)

(continued from previous page)

```
led = machine.Pin(25, machine.Pin.OUT)

def loop():
    while True:
        led.on()
        time.sleep(1)
        led.off()
        time.sleep(1)

loop()
```

C++

```
#define LED 25

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(1000);                       // wait for a second
    digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
    delay(1000);                       // wait for a second
}
```

CHAPTER

SEVEN

ANALOG TO DIGITAL CONVERSION

Learn how to read analog sensor values using the ADC module on the DualMCU ONE board. This section will cover the basics of analog input and conversion techniques.

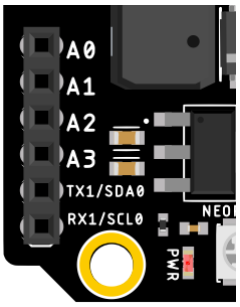


Fig. 7.1: ADC Pins

7.1 ADC Definition

Analog-to-digital conversion (ADC) is a process that converts analog signals into digital values. The DualMCU ONE development board, equipped with the RP2040 microcontroller, includes four analog pins. These pins are capable of reading analog voltages and converting them into digital values. Below you will find the details on how to utilize these pins for ADC operations.

7.1.1 Quantification and Codification of Analog Signals

Analog signals are continuous signals that can take on any value within a given range. Digital signals, on the other hand, are discrete signals that can only take on specific values. The process of converting an analog signal into a digital signal involves two steps: quantification and codification.

- **Quantification**: This step involves dividing the analog signal into discrete levels. The number of levels determines the resolution of the ADC. For example, an 8-bit ADC can divide the analog signal into 256 levels.
- **Codification**: This step involves assigning a digital code to each quantization level. The digital code represents the value of the analog signal at that level.

7.2 Distribution of Pins

Below is a table showing the distribution of analog pins on the DualMCU board along with their corresponding GPIO pins on the RP2040.

Table 7.1: Pin Mapping

PIN	GPIO RP2040
A0	26
A1	27
A2	28
A3	29

ADC includes four pins: A0, A1, A2, and A3. These pins can be used to read analog values from sensors and other devices. RP2040 has a 12-bit ADC, which means it can read analog values from 0 to 4095.

Caution: The fourth analog channel (A3) on the RP2040 microcontroller is dedicated to the internal temperature sensor and is not available for external connections.

7.3 Class ADC

The `machine.ADC` class is used to create ADC objects that can interact with the analog pins.

class machine.**ADC**(*pin*)

The constructor for the ADC class takes a single argument: the pin number.

7.4 Example Definition

To define and use an ADC object, follow this example:

MicroPython

```
import machine
adc = machine.ADC(0)  # Initialize ADC on pin A0
```

C++

```
#define ADC0 26
```

7.5 Reading Values

To read the analog value converted to a digital format:

MicroPython

```
adc_value = adc_value = adc.read() # Read the ADC value
print(adc_value)  # Print the ADC value
```

C++

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

7.6 Example Code

Below is an example that continuously reads from an ADC pin and prints the results:

Note: The following code is designed to work with the RP2040 microcontroller on the DualMCU development board.

MicroPython

```
import machine
import time

# Setup
A0 = machine.Pin(26, machine.Pin.IN)  # Initialize pin A0 for input
adc = machine.ADC(A0)                 # Create ADC object

# Continuous reading
while True:
    adc_value = adc.read_u16()        # Read the ADC value
    print(f"ADC Reading: {adc_value:.2f}")  # Print the ADC value
    time.sleep(1)                     # Delay for 1 second
```

C++

```
// Potentiometer is connected to GPIO 26 (Analog ADC0)
const int potPin = 26;

// variable for storing the potentiometer value
int potValue = 0;

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

void loop() {
  // Reading potentiometer value
  potValue = analogRead(potPin);
```

(continues on next page)

(continued from previous page)

```
  Serial.println(potValue);
  delay(500);
}
```

CHAPTER

EIGHT

I2C (INTER-INTEGRATED CIRCUIT)

Discover the I2C communication protocol and learn how to communicate with I2C devices using the DualMCU ONE board. This section will cover I2C bus setup and communication with I2C peripherals.

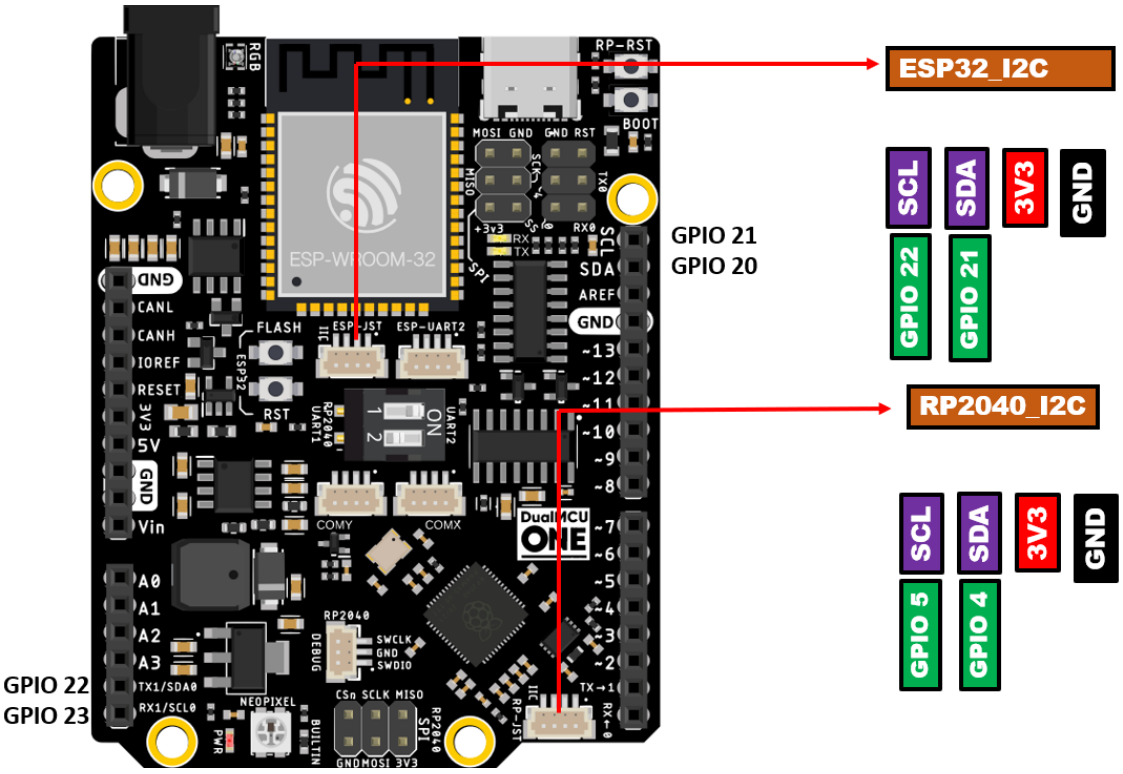


Fig. 8.1: I2C Pins

8.1 I2C Overview

I2C (Inter-Integrated Circuit) is a synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication bus. It is commonly used to connect low-speed peripherals to processors and microcontrollers. The DualMCU ONE development board features I2C communication capabilities, allowing you to interface with a wide range of I2C devices.

8.2 Pinout Details

Below is the pinout table for the I2C connections on the DualMCU ONE, detailing the corresponding GPIO connections for both the ESP32 and RP2040 microcontrollers.

Table 8.1: I2C Pinout

PIN	GPIO ESP32	GPIO RP2040
SCLK	22	5 / 21 / 23
SDA	21	4 / 20 / 22

8.3 Scanning for I2C Devices

To scan for I2C devices connected to the bus, you can use the following code snippet:

MicroPython

```
import machine

i2c = machine.SoftI2C(0, scl=machine.Pin(5), sda=machine.Pin(4))
devices = i2c.scan()

for device in devices:
    print("Device found at address: {}".format(hex(device)))
```

C++

```
#include <Wire.h>

void setup() {
  // in setup
  Wire.setSDA(4);
  Wire.setSCL(5);
  Wire.begin();
  Serial.begin(9600); // Start serial communication at 9600 baud rate
  while (!Serial); // Wait for serial port to connect
  Serial.println("\nI2C Scanner");
}

void loop() {
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ ) {
    // The i2c_scanner uses the return value of the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
```

(continues on next page)

(continued from previous page)

```
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address, HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4) {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

8.4 SSD1306 Display

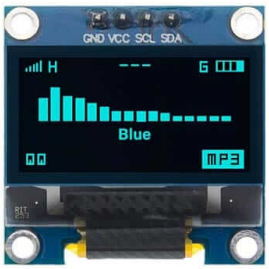


Fig. 8.2: SSD1306 Display

The display 128x64 pixel monochrome OLED display equipped with an SSD1306 controller is connected using a JST

1.25mm 4-pin connector. The following table provides the pinout details for the display connection.

Table 8.2: SSD1306 Display Pinout

Pin	Connection
1	GND
2	VCC
3	SDA
4	SCL

8.4.1 Library Support

MicroPython

The *dualmcu.py* library for MicroPython on ESP32 & RP2040 is compatible with the SSD1306 display controller.

Installation

1. Open Thonny.
2. Navigate to **Tools** -> **Manage Packages**.
3. Search for `dualmcu` and click **Install**.

for more information, Check the section

DualMCU Library (MicroPython)

DualMCU library is a project that aims to provide a set of tools to help developers work with the DualMCU ONE board. The library is compatible with MicroPython on ESP32 & RP2040 and provides support for the SSD1306 display controller, Arduino Shields compatibility, and more.,

Installation

1. Open Thonny.
2. Navigate to **Tools** -> **Manage Packages**.
3. Search for `dualmcu` and click **Install**.
4. Successfully installed the library.

Alternatively, download the library from dualmcu.py.

Usage

The library provides a set of tools to help developers work with the DualMCU ONE board. The following are the main features of the library:

- **SSD1306 Display Controller**: The library provides support for the SSD1306 display controller, allowing developers to easily interface with OLED displays.
- **Arduino Shields Compatibility**: The library is compatible with Arduino Shields, making it easy to use a wide range of shields and accessories with the DualMCU ONE board.

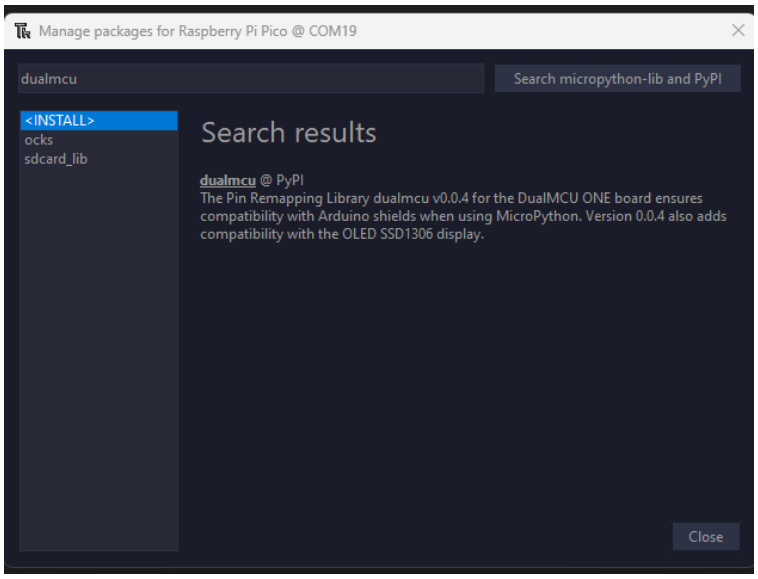


Fig. 8.3: DualMCU Library 1

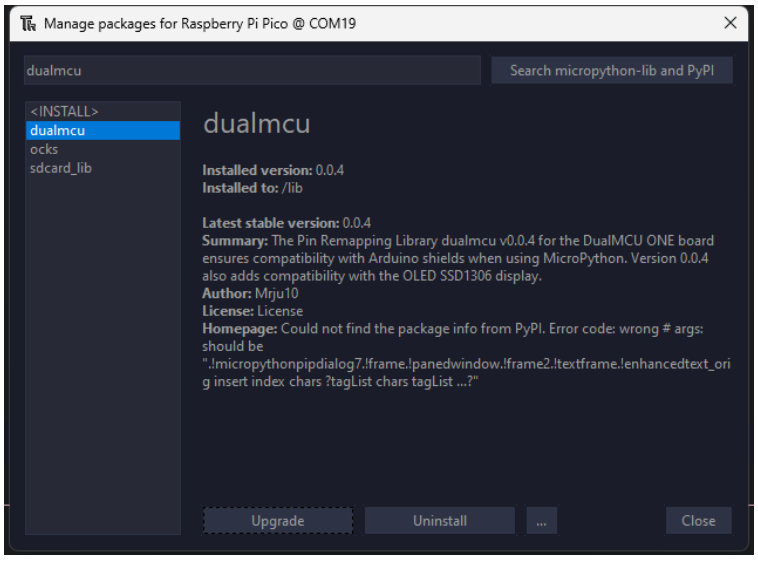


Fig. 8.4: DualMCU Library Successfully Installed

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

oled.show()
```

The library is actively maintained and updated to provide the best experience for developers working with the DualMCU ONE board. For more information and updates, visit the *dualmcu GitHub repositorY*`

Alternatively, download the library from dualmcu.py.

Microcontroller Configuration

```
SoftI2C(scl, sda, *, freq=400000, timeout=50000)
```

Change the following line depending on your microcontroller:

For ESP32:

```
>>> i2c = machine.SoftI2C(freq=400000, timeout=50000, sda=machine.Pin(21), scl=machine.
↪Pin(22))
```

For RP2040:

```
>>> i2c = machine.SoftI2C(freq=400000, timeout=50000, sda=machine.Pin(4), scl=machine.
↪Pin(5))
```

Example Code

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

(continues on next page)

(continued from previous page)

```
oled.show()
```

This code initializes the display, fills the screen with different colors, and displays text.

`sda=machine.Pin(*)` and `scl=machine.Pin(*)` should be replaced with the appropriate GPIO pins for your setup.

This version clarifies the structure, pin configurations, library usage, installation instructions, and example code for your project. Adjust the placeholders marked with * in your actual code based on your specific GPIO pin assignments.

C++

The *Adafruit_SSD1306* library for Arduino is compatible with the SSD1306 display controller.

Installation

1. Open the Arduino IDE.
2. Navigate to **Tools** -> **Manage Libraries**.
3. Search for `Adafruit_SSD1306` and click **Install**.

Description code

The provided Arduino sketch is designed to interface an RP2040 microcontroller with an SSD1306 OLED display using the I2C communication protocol. It begins by including the necessary libraries for controlling the display and initializing serial communication for debugging purposes. The sketch defines constants for the display dimensions and I2C pins (SDA on GPIO 4 and SCL on GPIO 5) and initializes an Adafruit SSD1306 object.

Example Code

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

// OLED display TWI (I2C) interface
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_WIDTH   128 // OLED display width, in pixels
#define SCREEN_HEIGHT  64  // OLED display height, in pixels
#define SDA_PIN        4   // SDA pin
#define SCL_PIN        5   // SCL pin

// Declare an instance of the class (specify width and height)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(9600);

  // Initialize I2C
  Wire.setSDA(4);
  Wire.setSCL(5);
  Wire.begin();
  // Start the OLED display
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
```

(continues on next page)

(continued from previous page)

```
  // Clear the buffer
  display.clearDisplay();

  // Set text size and color
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0,0);
  display.println(F("UNIT ELECTRONICS!"));
  display.display();  // Show initial text
  delay(4000);        // Pause for 2 seconds
}

void loop() {
  // Increase a counter
  static int counter = 0;

  // Clear the display buffer
  display.clearDisplay();
  display.setCursor(0, 10); // Position cursor for new text
  display.setTextSize(2);   // Larger text size

  // Display the counter
  display.print(F("Count: "));
  display.println(counter);

  // Refresh the display to show the new count
  display.display();

  // Increment the counter
  counter++;

  // Wait for half a second
  delay(500);
}
```

CHAPTER

NINE

CONECTOR JST SH 1.0MM 4 PINS

The JST SH 1.0mm 4 pins is a connector that is used to connect the DualMCU ONE board to devices such as sensors, actuators, and other peripherals. The connector has a pitch of 1.0mm and is commonly used in small electronic devices.

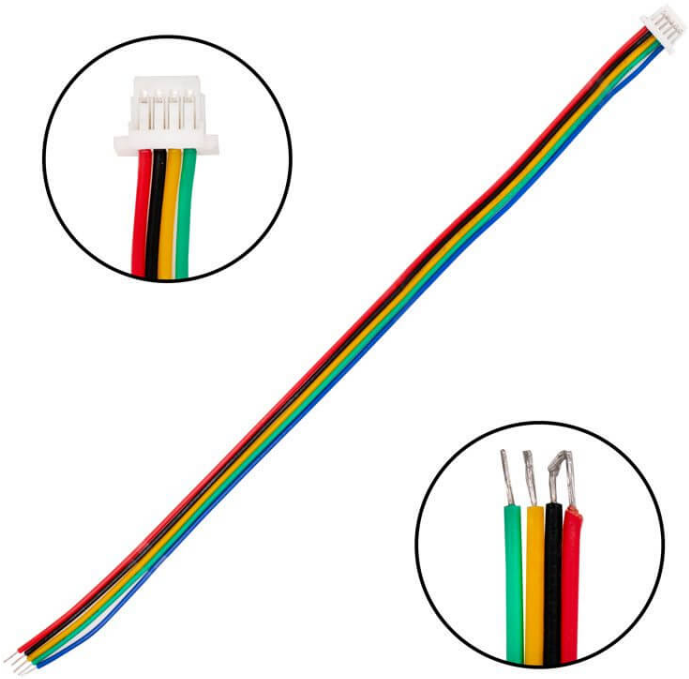


Fig. 9.1: JST SH 1.0mm 4 pins

For connecting the JST SH 1.0mm 4 pins connector to the DualMCU ONE board, follow the steps below:

1. **Identify the connector**: The JST SH 1.0mm 4 pins connector has a pitch of 1.0mm and 4 pins.
2. **Connect the connector**: Align the connector with the pins on the DualMCU ONE board and gently push the connector onto the pins.

Caution: Sometimes the connector colors do not correspond to the pins. Be careful when connecting the connector to the board.

3. **Verify the connection**: After connecting the JST SH 1.0mm 4 pins connector to the DualMCU ONE board, verify the connection by checking the pins on the board.

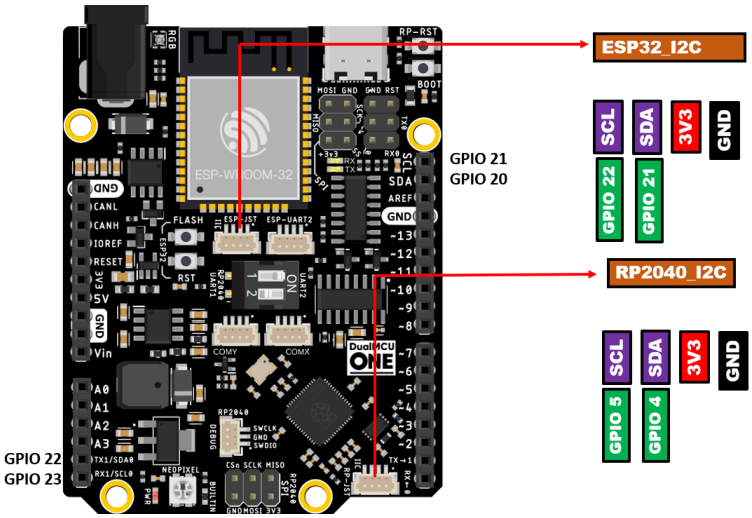


Fig. 9.2: JST SH 1.0mm 4 pins (example for communication I2C)

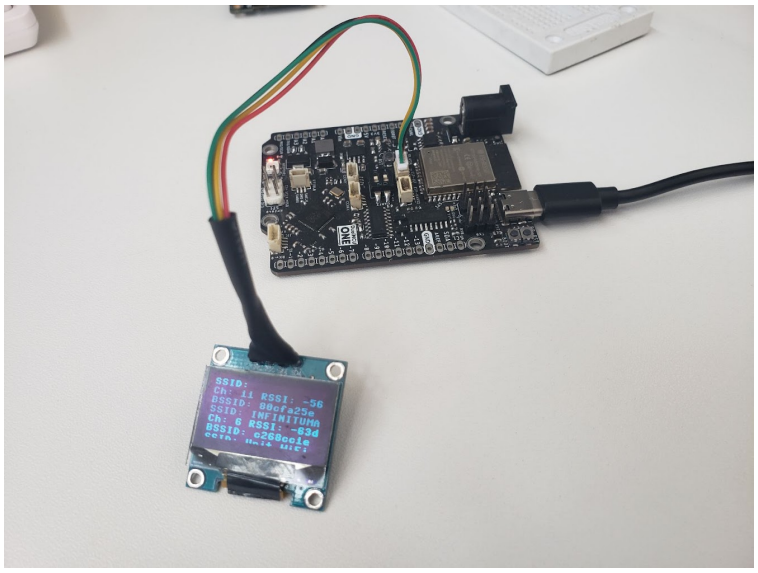


Fig. 9.3: JST SH 1.0mm 4 pins connected

Note: The JST SH 1.0mm 4 pins connector is compatible with Cocket Nova developed board.

CHAPTER

TEN

SPI (SERIAL PERIPHERAL INTERFACE)

Explore SPI communication protocol and learn how to interface with SPI devices using the DualMCU ONE board. Well walk you through setting up SPI communication and communicating with SPI devices.

10.1 SPI Overview

SPI (Serial Peripheral Interface) is a synchronous, full-duplex, master-slave communication bus. It is commonly used to connect microcontrollers to peripherals such as sensors, displays, and memory devices. The DualMCU ONE development board features SPI communication capabilities, allowing you to interface with a wide range of SPI devices.

10.2 Pinout Details

In the next figure, you can see the SPI pins on the DualMCU ONE board and the corresponding GPIO pins on the ESP32 and RP2040 microcontrollers.

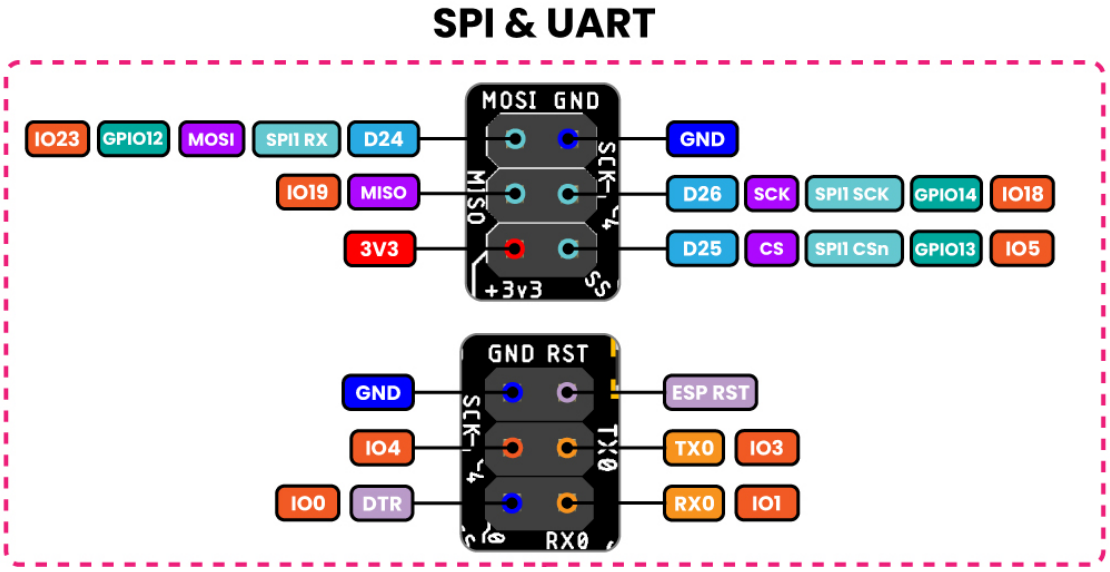


Fig. 10.1: SPI Pins

Below is the pinout table for the SPI connections on the DualMCU ONE, detailing the corresponding GPIO connections for both the ESP32 and RP2040 microcontrollers.

Table 10.1: SPI Pinout

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

Caution: ESP32 and RP2040 have conections sharing the same pins, so you recommend to use only one microcontroller at a time.

Table 10.2: sharing pins

GPIO ESP32	GPIO RP2040
5	13
18	14
23	12

10.3 SPI between ESP32 & RP2040

DualMCU ONE board was designed to be compatible with both ESP32 and RP2040 microcontrollers. So it is possible to share information between the two microcontrollers using the SPI protocol. For this, we will use the following pins:

Table 10.3: SPI Connections

Pin	SPI RP2040	Pin	SPI ESP32
SCK	14	SCK	18
MISO	12	MOSI	23
MOSI	15	MISO	14
SS	13	SS	5
READY	7	READY	33
RESET	16	RESET	RST

Caution: The connection RST does not exist physically; therefore, it is necessary to establish an external connection.

10.4 SDCard SPI

Warning: Ensure that the Micro SD contain data. We recommend saving multiple files beforehand to facilitate the use.

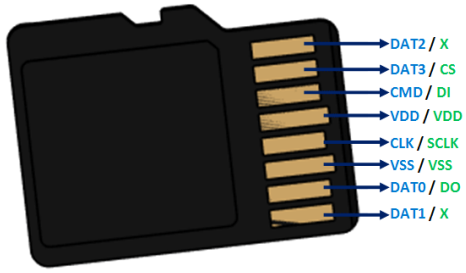


Fig. 10.2: Micro SD Card Pinout

10.4.1 Library (MicroPython)

The *dualmcu.py* library for MicroPython on ESP32 & RP2040 is compatible with the reader and writer of the Micro SD card. The library provides a simple interface for reading and writing files on the SD card. The library is available on PyPi and can be installed using the Thonny IDE.

Installation

1. Open Thonny.
2. Navigate to **Tools** -> **Manage Packages**.
3. Search for `dualmcu` and click **Install**.

for more information, Check the section

- DualMCU ONE Library

Alternatively, download the library from dualmcu.py.

VSPI & HSPI **VSPI Interfacing.** .. _figura-micro-sd-card-reader:

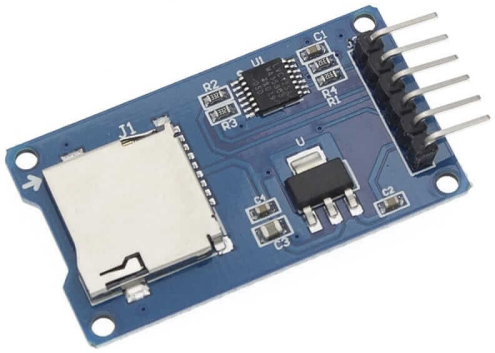


Fig. 10.3: Micro SD Card external reader

The conections are as follows:

This table illustrates the connections between the SD card and the GPIO pins on the ESP32 and RP2040 microcontrollers.

Table 10.4: VSPI Connections

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

Descriptions

- SCK (Serial Clock)
- SS (Slave Select)

```
import machine, os
from dualmcu import *

SCK_PIN = 18
MOSI_PIN = 23
MISO_PIN = 19
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("files ...")
print(os.listdir("/sd"))
```

HSPI Interfacing.

This table details the connections between the SD card and the ESP32 microcontroller.

Table 10.5: HSPI Connections

SD Card	ESP32	PIN
D2		12
D3	SS (Slave Select)	13
CMD	MOSI	15
VSS	GND	
VDD	3.3V	
CLK	SCK (Serial Clock)	14
VSS	GND	
D0	MISO	2
D1		4

For the test , we will utilize an ESP32 WROM-32E and a SanDisk Micros Ultra card with a capacity of 32 GB.

```
import machine
import os
from dualmcu import *

# Initialize SPI interface for the SD card
spi = machine.SPI(2, baudrate=1000000, polarity=0, phase=0, sck=machine.Pin(14),␣
↪mosi=machine.Pin(15), miso=machine.Pin(2))

# Initialize the SD card
sd = SDCard(spi, machine.Pin(13))

# Mount the filesystem
vfs = os.VfsFat(sd)
os.mount(vfs, "/sd")

# List files in the root of the SD card
print("Files in the root of the SD card:")
print(os.listdir("/sd"))


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

10.4.2 SPI (Arduino IDE)

Arduino IDE is compatible for reader and writer of the Micro SD card. The library provides a simple interface for reading and writing files on the SD card.

VSPI

```

```

CHAPTER

ELEVEN

WS2812 CONTROL

Harness the power of WS1280 LED strips with the DualMCU ONE board. Learn how to control RGB LED strips and create dazzling lighting effects using MicroPython.

This section describes how to control WS2812 LED strips using the DualMCU ONE board. The DualMCU ONE board has a GPIO pin embebbed connected to the single WS2812 LED.

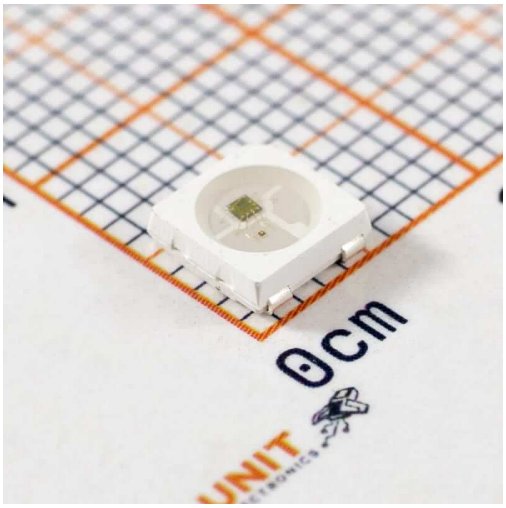


Fig. 11.1: WS2812 LED Strip

11.1 Code Example

Below is an example that demonstrates how to control WS1280 LED strips using the DualMCU ONE board:

```
from machine import Pin
from neopixel import NeoPixel
np = NeoPixel(Pin(24), 1)
np[0] = (255, 128, 0) # set to red, full brightness

np.write()
```

Tip: for more information on the NeoPixel library, refer to the NeoPixel Library Documentation.

CHAPTER

TWELVE

COMMUNICATION

Unlock the full communication potential of the DualMCU ONE board with built-in Wi-Fi, Bluetooth, and Serial capabilities.

12.1 Wi-Fi

Learn how to set up and use Wi-Fi communication on the DualMCU ONE board.

```
import machine
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('your-ssid', 'your-password')

while not wlan.isconnected():
    pass

print('Connected to Wi-Fi')

# Check the IP address
print(wlan.ifconfig())
```

12.1.1 Scan Wi-Fi Networks

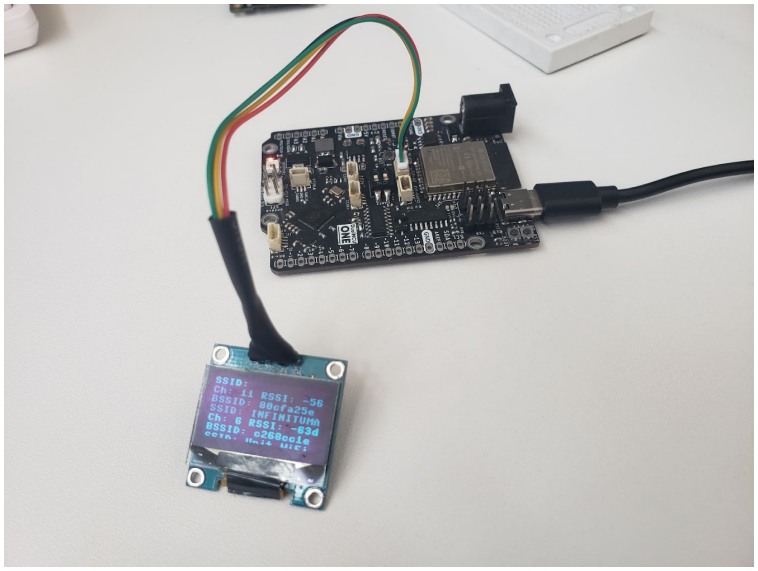
Scan for available Wi-Fi networks and display the results on an OLED display.

```
import network
import time
import machine
from machine import I2C, Pin
from ocks import SSD1306_I2C

# Initialize I2C for the OLED display
# i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
i2c = machine.SoftI2C(freq=400000, timeout=50000, sda=machine.Pin(21), scl=machine.
↪Pin(22))
oled_width = 128
oled_height = 64
```

(continues on next page)

(continued from previous page)

```
oled = SSD1306_I2C(oled_width, oled_height, i2c)

# Initialize WiFi in station mode
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)


def display_networks(networks, start_index, num_display=3):
    oled.fill(0)  # Clear the display
    y = 0
    for i in range(start_index, start_index + num_display):
        if i >= len(networks):
            break
        ssid, bssid, channel, RSSI, authmode, hidden = networks[i]
        ssid_str = ssid.decode('utf-8')
        bssid_str = bssid.hex()
        if len(ssid_str) > 10:
            ssid_str = ssid_str[:10]  # Truncate SSID to fit the display
        display_text1 = f"SSID: {ssid_str}"
        display_text2 = f"Ch: {channel} RSSI: {RSSI}dBm"
        display_text3 = f"BSSID: {bssid_str[:8]}"
        oled.text(display_text1, 0, y)
        oled.text(display_text2, 0, y + 10)
        oled.text(display_text3, 0, y + 20)
        print(display_text1,display_text2,bssid_str)
        y += 30
        if y >= oled_height:
            break
    oled.show()

print("Scanning for networks...")

while True:
    networks = sta_if.scan()
    num_networks = len(networks)
    start_index = 0

    while True:
        display_networks(networks, start_index)
        time.sleep(2)  # Display each set for 2 seconds
        start_index += 3  # Move to the next set of networks
        if start_index >= num_networks:
            start_index = 0  # Wrap around to the beginning
        if len(networks) < 3:
            break  # If there are fewer than 3 networks, don't loop indefinitely
```



12.2 Bluetooth

Explore Bluetooth communication capabilities and learn how to connect to Bluetooth devices.

```
import bluetooth
import time

# Initialize Bluetooth
ble = bluetooth.BLE()
ble.active(True)

# Helper function to convert memoryview to MAC address string
def format_mac(addr):
    return ':'.join('{:02x}'.format(b) for b in addr)

# Helper function to parse device name from advertising data
def decode_name(data):
    i = 0
    length = len(data)
    while i < length:
        ad_length = data[i]
        ad_type = data[i + 1]
        if ad_type == 0x09:  # Complete Local Name
            return str(data[i + 2:i + 1 + ad_length], 'utf-8')
        elif ad_type == 0x08:  # Shortened Local Name
            return str(data[i + 2:i + 1 + ad_length], 'utf-8')
        i += ad_length + 1
    return None

# Callback function to handle advertising reports
def bt_irq(event, data):
    if event == 5:  # event 5 is for advertising reports
        addr_type, addr, adv_type, rssi, adv_data = data
```

(continues on next page)

(continued from previous page)

```
        mac_addr = format_mac(addr)
        device_name = decode_name(adv_data)
        if device_name:
            print(f"Device found: {mac_addr} (RSSI: {rssi}) Name: {device_name}")
        else:
            print(f"Device found: {mac_addr} (RSSI: {rssi}) Name: Unknown")
#             pass

# Set the callback function
ble.irq(bt_irq)

# Start active scanning
ble.gap_scan(10000, 30000, 30000, True)  # Active scan for 10 seconds with interval and␣
↪window of 30ms

# Keep the program running to allow the callback to be processed
while True:
    time.sleep(1)
```

CHAPTER

THIRTEEN

WEB SERVER - MICROPYTHON

ESP32 Web Server is a simple web server that knows how to handle HTTP requests such as GET and POST and can only support one simultaneous client. It supports the following HTTP request methods:

- GET
- POST
- PUT
- DELETE
- PATCH

The server can be started by calling the *start* method. The server will start listening on the specified port and will call the *handler* method when a request is received.

13.1 Network basics

The basic connection using MicroPython usage the method *connect* from the *network* module. The *connect* method receives the SSID and the password as parameters. The method *ifconfig* returns the IP address, the subnet mask, the gateway and the DNS server.

Example of a simple connection to a Wi-Fi network:

```
import machine
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('your-ssid', 'your-password')

while not wlan.isconnected():
    pass

print('Connected to Wi-Fi')

# Check the IP address
print(wlan.ifconfig())
```

The code-block stablish a simple connection with a Wi-Fi network. The *while* loop waits until the connection is established. The *ifconfig* method returns a tuple with the IP address, the subnet mask, the gateway and the DNS server.

Tip: The *ifconfig* method returns a tuple with the IP address, the subnet mask, the gateway and the DNS server. For more information consult the network module documentation.

13.2 Web Server - Setup and Usage

Clone the repository:

```
git clone https://github.com/UNIT-Electronics/DualMCU_Advanced_Projects.git
```

Go to the *web_server* directory:

```
cd Projects/1. Web Server
```

This directory contains the following files:

- *Desktop* - Contains the files for the desktop web server
- *ESP32-MicroPython* - Contains the files for the ESP32 web server

13.2.1 Desktop

The directory *Desktop* contains the files for the desktop web server. Although the server is running on the desktop, it can be used to control an ESP32 running a web server.

Why use a desktop web server?, you may ask. The desktop web server can be used to test the ESP32 web server and interact with it without having to run the code on the ESP32. This can be useful for debugging and testing the web server without having to upload the code to the ESP32 every time.

The code describe methods to handle the HTTP requests. Each method receives the request and the response as parameters. for example, the *get* method handles the GET request and the *post* method handles the POST request.

Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

To install Flask, run the following command:

```
pip install Flask
```

Run the desktop web server by running the following command:

```
python app.py --ip <ESP32_IP>
```

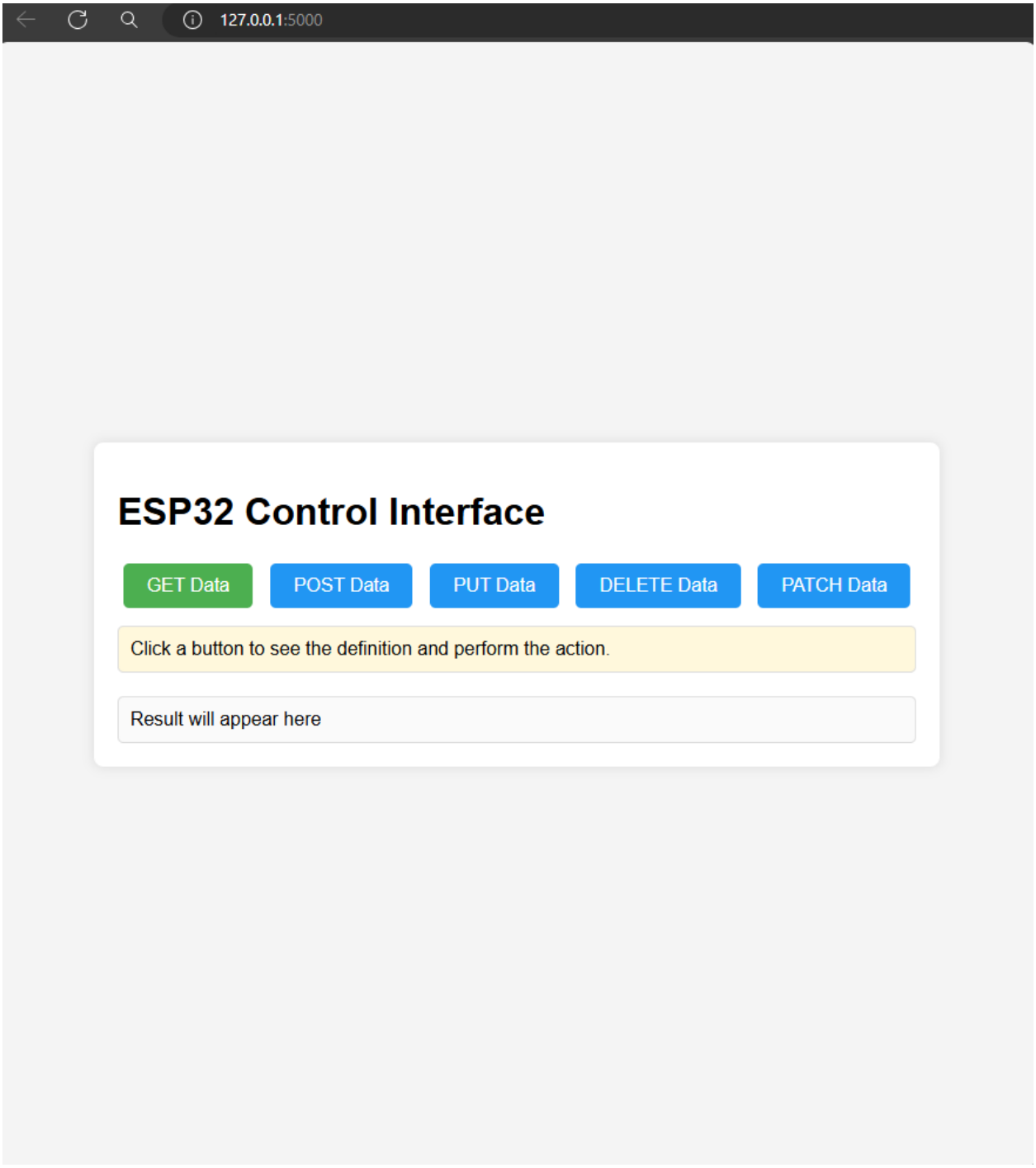


Fig. 13.1: Desktop Web Server

```
p> python .\app.py --ip 192.168.3.65
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 498-439-593
```

Fig. 13.2: Desktop Web Server

13.2.2 ESP32-MicroPython

The directory *ESP32-MicroPython* contains the files for the ESP32 web server. The *main.py* file contains the code for the ESP32 web server.

```
Connection established
IP address: 192.168.3.65
HTTP server running
Client connected from ('192.168.3.250', 51315)
Method: GET
Path: /api/data
Request: GET /api/data HTTP/1.1
Host: 192.168.3.65
User-Agent: python-requests/2.31.0
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: keep-alive
```

Fig. 13.3: ESP32 Web Server

13.3 Extra information

- How to install MicroPython on ESP32: MicroPython ESP32
- How to install MicroPython on DualMCU-ESP32: MicroPython ESP32
- DualMCU - First generation: DualMCU
- DualMCU - Getting started: DualMCU - Getting started
- DualMCU - Libraries: DualMCU - Libraries

CHAPTER

FOURTEEN

ARDUINO SHIELDS COMPATIBILITY

The DualMCU ONE board is compatible with a wide range of Arduino shields and accessories. The board features the same pinout as the Arduino UNO Rev3, making it easy to use with existing shields. The board also features a USB-C connector for power and data, a microSD card slot, and a dual-core architecture with an RP2040 and an ESP32 microcontrollers.

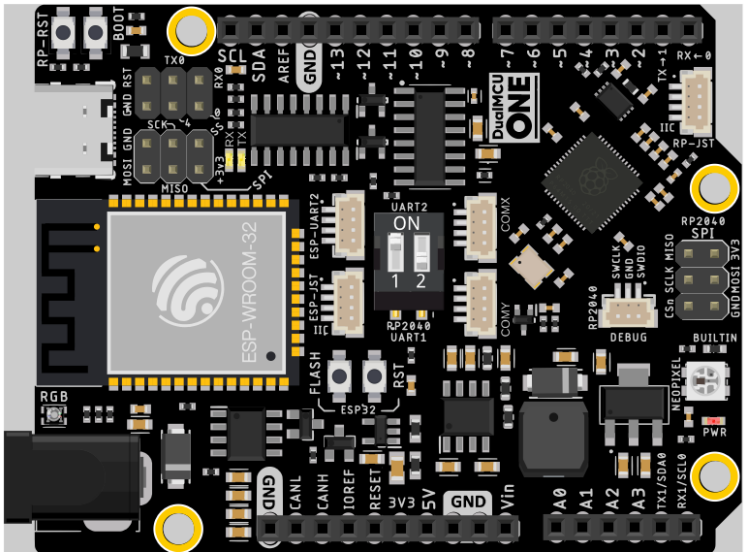


Fig. 14.1: DualMCU ONE board

14.1 Multi-purpose Shield for DualMCU ONE

The Multi-purpose Shield is a versatile accessory for the DualMCU ONE board. It features a variety of sensors and components, including:

- **2 LED indicators**: Show the program status.
- **2 switches**: For external interrupt experiments.
- **Reset button**: Allows resetting the board.
- **DHT11 sensor**: Measures temperature and humidity.
- **Potentiometer**: Provides analog input.
- **Passive buzzer**: Functions as an alarm.

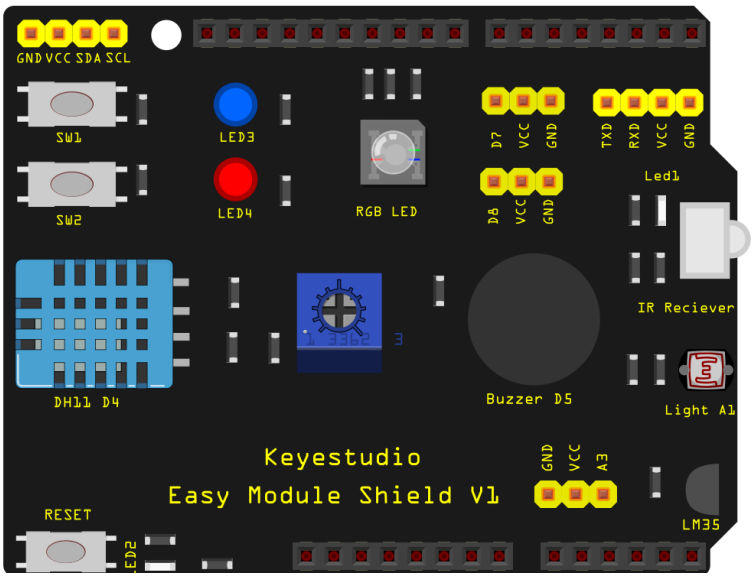


Fig. 14.2: Multi-purpose Shield

- **Full-color RGB LED**: Provides multiple color options.
- **Photocell**: Detects the brightness of light.
- **LM35D temperature sensor**: Measures temperature.
- **Infrared receiver**: Enables infrared communication.
- **Digital pins**: 2 digital pins (D7 and D8).
- **Analog pin**: 1 analog pin (A3).
- **IIC interface**: For I2C communication.
- **TTL serial pin**: For serial communication.

> **Warning:** The Multi-purpose Shield ADC sensors are connected to a 5V voltage supply, so caution is needed when connecting them to the DualMCU ONE board.

```
from machine import Pin
import time
from dht import DHT11
from dualmcu import *

# Create an instance of Shield
my_shield = Shield(
    pin_red=D9, pin_green=D10, pin_blue=D11,
    pin_buzzer=D5, pin_led1=D13, pin_led2=D12,
    pin_button1=D2, pin_button2=D3, pin_analog=A0
)

# Configure DHT11 sensor
sensor_dht = DHT11(Pin(D4))
```

(continues on next page)

(continued from previous page)

```
try:
    while True:
        # Read DHT11 sensor
        sensor_dht.measure()
        temp_dht = sensor_dht.temperature()
        hum = sensor_dht.humidity()

        # Read analog sensor
        analog_value = my_shield.read_analog()

        # Print the values read
        print(f'Temperature: {temp_dht}°C Humidity: {hum}% Analog: {analog_value}')

        # Read buttons and control LEDs
        my_shield.set_led1(my_shield.read_button1() == 0)
        my_shield.set_led2(my_shield.read_button2() == 0)

        # Play a sequence of tones and change the LED colors
        for color, freq in zip(
            ['red', 'green', 'blue', 'yellow', 'cyan', 'magenta', 'white', 'off'],
            [262, 294, 330, 349, 392, 440, 494, 523]
        ):
            my_shield.set_led(color)
            my_shield.play_tone(freq, 0.5)
            time.sleep(0.1)

except KeyboardInterrupt:
    my_shield.deinit()
```

CHAPTER

FIFTEEN

HOW GENERATE REPORT OF ERRORS

This is a guide to generate report of errors.

Unit Electronics is a company that produces electronic devices. The company has a quality control department that is responsible for checking the quality of the devices designed.

you can get the report of errors by following the steps below:

1. Got to the Unit Electronics github repository.
2. Click on the issues tab.
3. Click on the new issue button.
4. Fill in the title and description of the issue.
5. Click on the submit button.

The quality control department will review the issue and take the necessary action to resolve it.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

M