UNIT NANO C6

Release 0.0.1

Cesar Bautista

Feb 11, 2025

CONTENTS

Note: You are reading the most recent version available.

Documentation for the UNIT NANO ESP32C6 Development Board. This board is a versatile ultra-low-power microcontroller with advanced connectivity and peripheral features, making it suitable for a wide range of IoT and embedded applications.

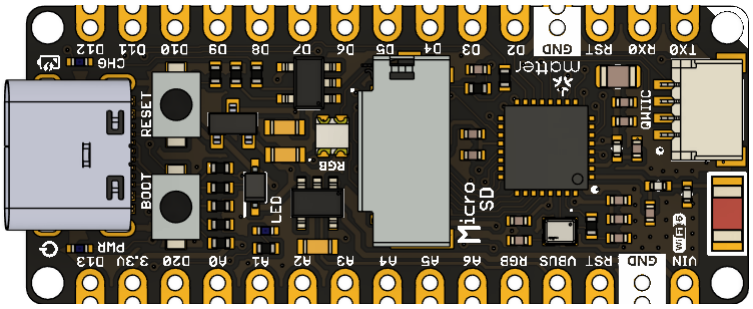


Fig. 1: esp32c6_nano

CHAPTER

ONE

UNIT NANO C6

1.1 Introduction

This guide will help you get started with the UNIT NANO C6 development board. The UNIT NANO C6 is a development board based on the ESP32C6 microcontroller. It is designed for prototyping and developing IoT applications. The board features a variety of interfaces, including GPIO, I2C, SPI, UART, and more. It also has built-in support for Wi-Fi and Bluetooth connectivity.

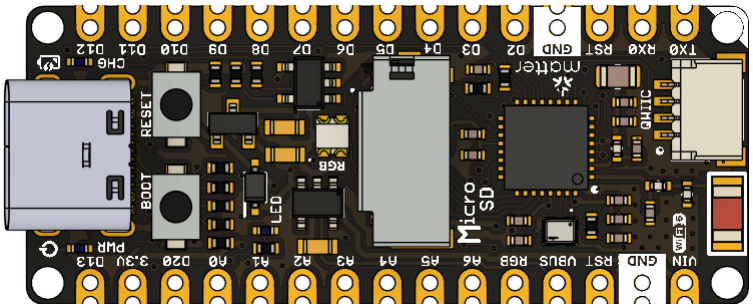


Fig. 1.1: UNIT NANO C6 Development Board

1.2 ESP32C6 Microcontroller

The ESP32-C6 is a versatile ultra-low-power microcontroller with advanced connectivity and peripheral features, making it suitable for a wide range of IoT and embedded applications.

1.2.1 Architecture

- **CPU Architecture**:
  - Based on **RISC-V 32-bit ISA**.
  - Includes:
    - A **High-Performance (HP) CPU** running up to **160 MHz** with a 4-stage pipeline.
    - A **Low-Power (LP) CPU** running up to **20 MHz** for energy-efficient tasks.
- **Memory**:
  - **320 KB ROM** for booting and essential functions.

  - **512 KB High-Performance SRAM** (HP SRAM) for data and instructions.
  - **16 KB Low-Power SRAM** (LP SRAM), retaining data during sleep modes.
  - Optional external flash and SRAM support through SPI, Dual SPI, Quad SPI, and QPI.
- **Security Features**:
  - Secure boot and memory encryption.
  - Cryptographic hardware accelerators for AES, RSA, SHA, ECC, and HMAC.
  - Support for Trusted Execution Environment (TEE).
- **Wireless Capabilities**:
  - **Wi-Fi 6 (2.4 GHz)**, Bluetooth 5.3, Zigbee, and Thread (802.15.4) for versatile connectivity options.
  - Integrated coexistence for simultaneous operation of Wi-Fi, Bluetooth, and 802.15.4.

1.2.2 General Features

- **GPIOs and I/O Functionality**:
  - Up to **30 GPIOs** (QFN40) or **22 GPIOs** (QFN32).
  - Multiple I/O functions through pin multiplexing.
  - Support for digital and analog configurations:
    - **12-bit SAR ADC** with up to 7 channels.
    - Integrated **Temperature Sensor**.
- **Peripheral Interfaces**:
  - Digital interfaces:
    - Two **UARTs**.
    - **I2C** and **I2S** for communication and audio processing.
    - **SPI** with multiple modes for fast data transfer.
  - PWM controllers:
    - **LED PWM** with up to 6 channels.
    - **Motor Control PWM (MCPWM)** for precision control.
  - **Pulse Counter** for frequency and signal measurement.
  - **USB Serial/JTAG Controller** for debugging and serial communication.
- **Timers**:
  - **52-bit System Timer** for accurate timekeeping.
  - Two **54-bit General-Purpose Timers**.
  - Multiple **Digital Watchdog Timers** for reliability.

1.2.3 Power Management

- Supports four power modes for optimal energy usage:
  - **Active**, **Modem-sleep**, **Light-sleep**, and **Deep-sleep**.
- Ultra-low power consumption in **Deep-sleep mode** (7 μA).
- Retains memory and critical functions in low-power modes.

1.2.4 Security and Hardware Acceleration

- **General DMA Controller** for efficient data transfers.
- Built-in hardware accelerators for cryptography:
  - **AES**, **RSA**, **SHA**, and **ECC**.
- Secure boot and flash encryption for system integrity.

1.2.5 Applications

The ESP32-C6 is ideal for various applications, including:

- Smart Home devices.
- Industrial Automation.
- IoT sensor hubs and data loggers.
- Consumer Electronics and more.

1.2.6 Development Support

- Fully compatible with Espressif's **ESP-IDF** (IoT Development Framework) for professional-grade development.
- **Arduino IDE** support for hobbyists and simpler programming tasks.
- Compatibility with third-party SDKs for integration into various workflows.

1.2.7 Physical Dimensions

- **Compact form factor** suitable for embedded applications.
- Available in QFN40 (5×5 mm) and QFN32 (5×5 mm) packages, ensuring versatility for different designs.

Caution: These are the general specifications; depending on the manufacturer and the specific ESP32-C6 module, there may be differences in features or additional capabilities.

CHAPTER

TWO

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

2.1 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.

2.2 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:

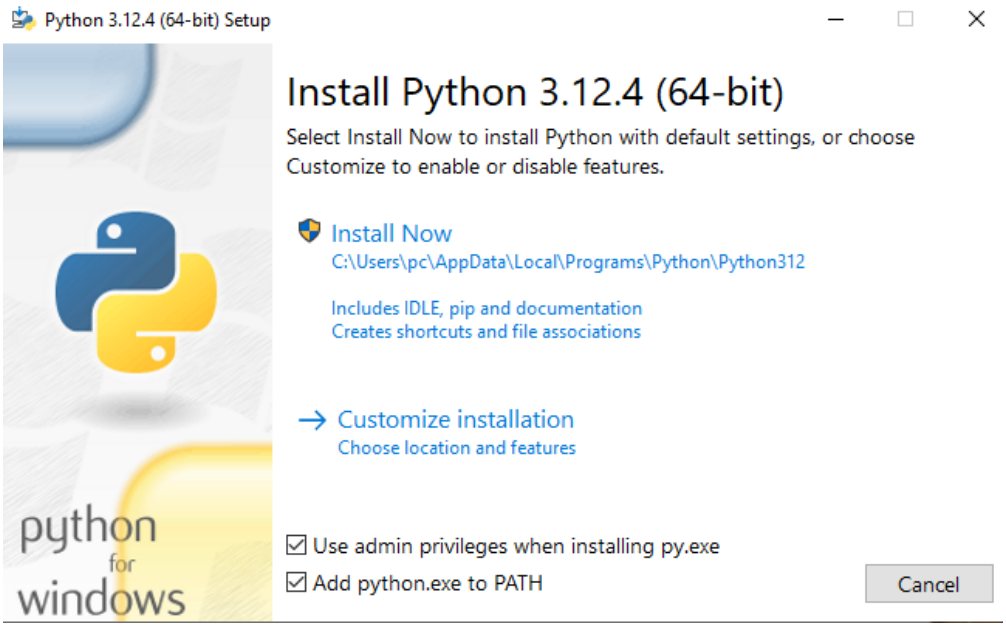


Fig. 2.1: Add python to PATH

```
python --version
```

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

2.3 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.

2.4 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.

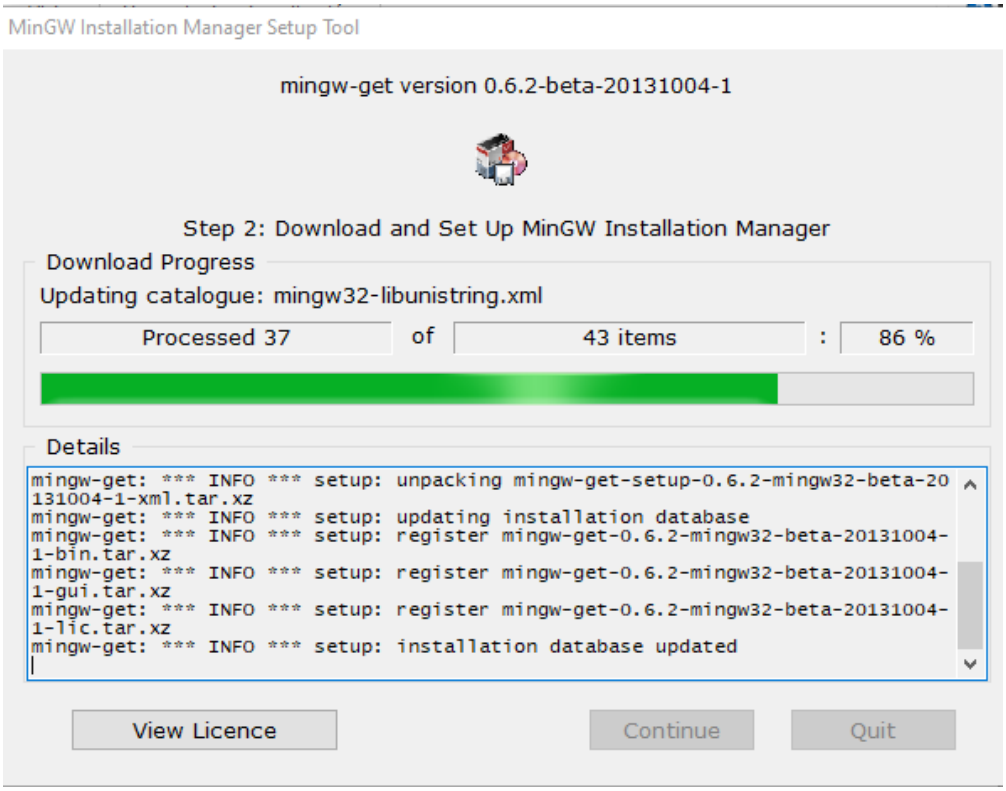
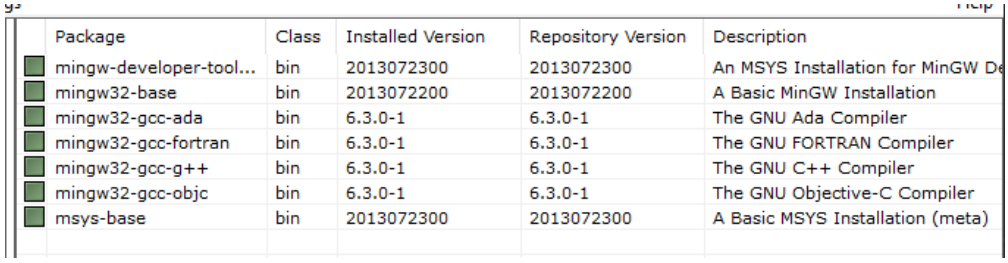


Fig. 2.2: MinGW installer

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. 2.3: MinGW installation

```
mingw --version
```

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

2.4.1 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
```

2.4.2 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*.

2.4.3 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*.

2.4.4 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
   ```

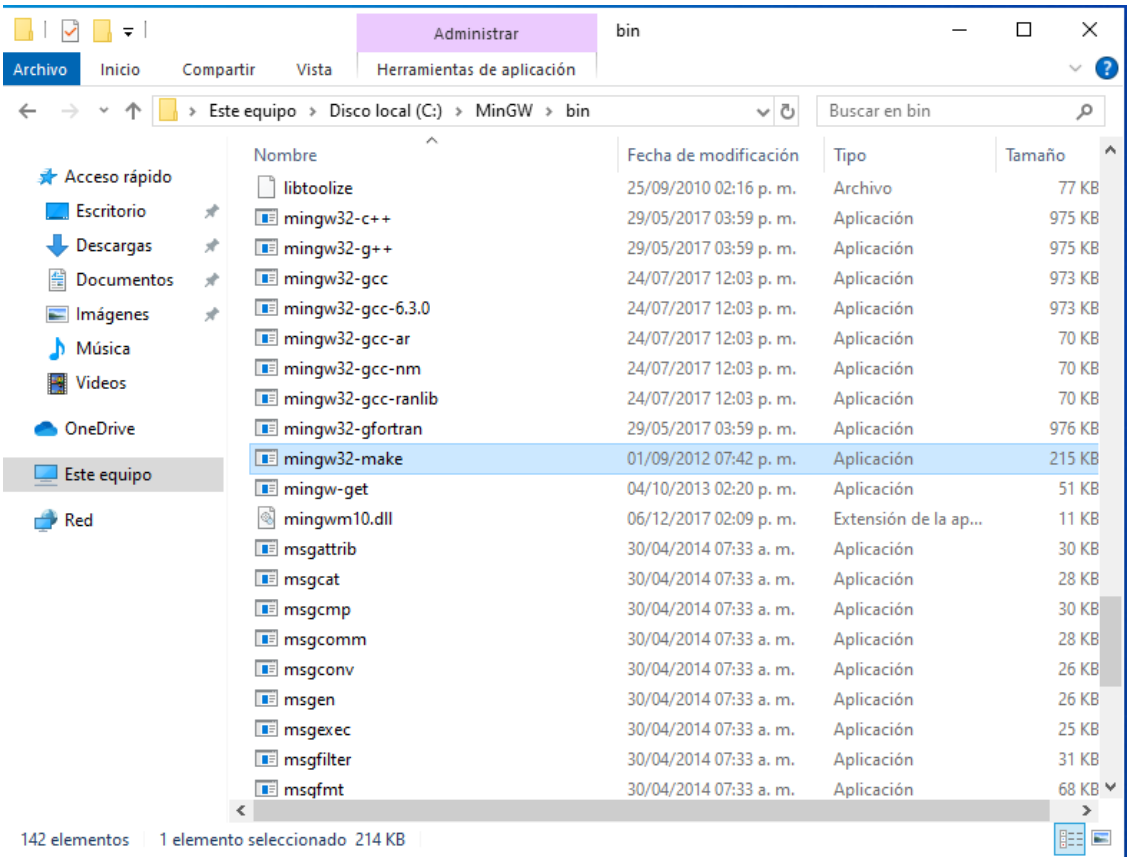


Fig. 2.4: Locating the *mingw32-make.exe* file

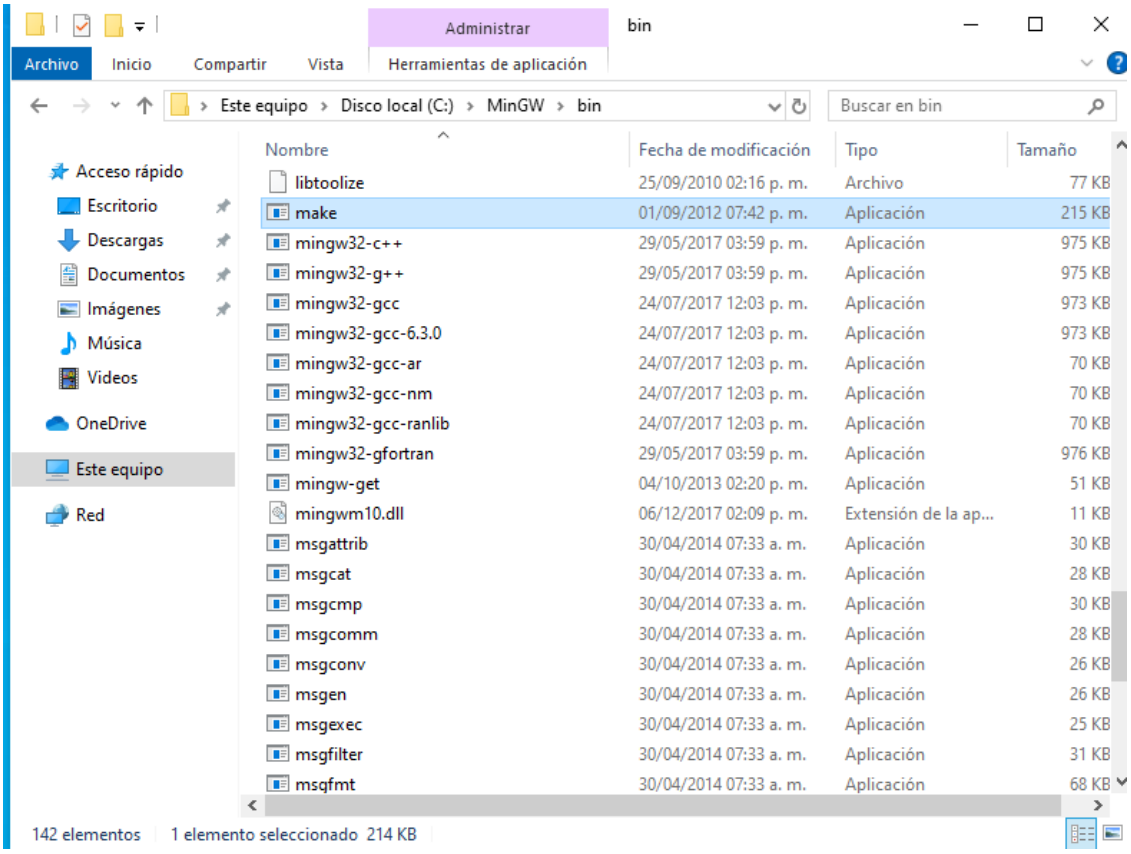


Fig. 2.5: Renaming *mingw32-make.exe* to *make.exe*

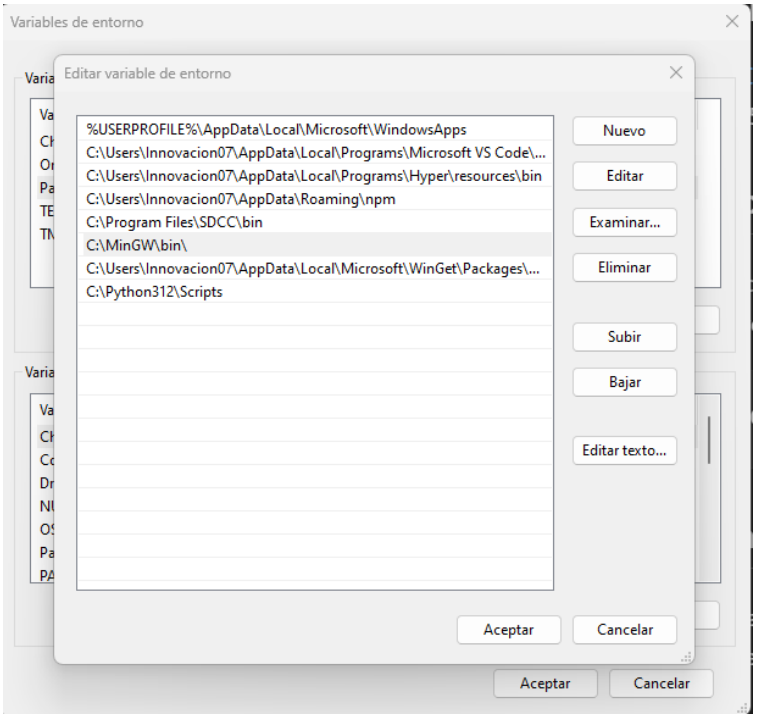


Fig. 2.6: Adding MinGW bin directory to environment variables

2.5 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.

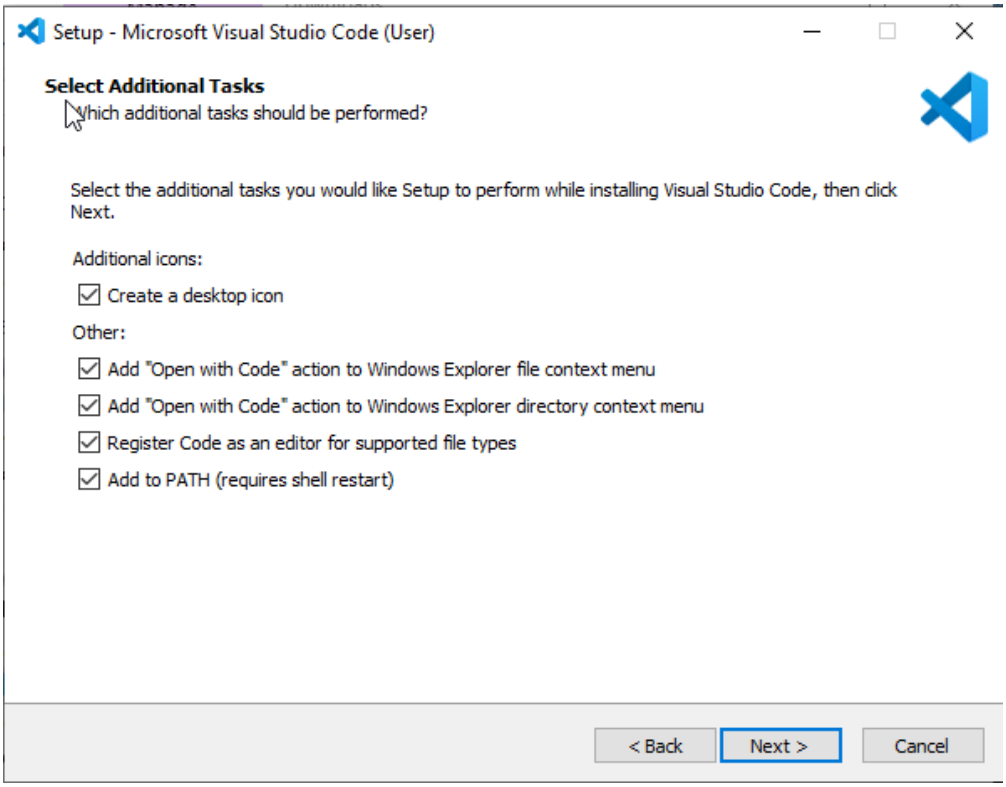


Fig. 2.7: Visual Studio Code installer

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:

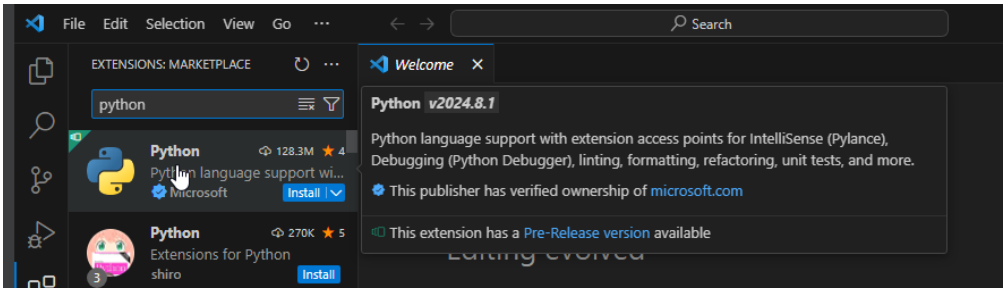


Fig. 2.8: Visual Studio Code extensions

CHAPTER

THREE

INSTALLING PACKAGES

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

3.1 Installation Guide Using MIP Library

Note: Direct support for mip on RP2040 is not available. The *mip* library is utilized to install other libraries on the NANOC6 board.

3.1.1 Requirements

- ESP32C6 device
- Thonny IDE
- Wi-Fi credentials (SSID and Password)

3.1.2 Installation Instructions

Follow the steps below to install the *max1704x.py* library:

3.1.3 Connect to Wi-Fi

Copy and run the code below in Thonny to connect your ESP32 to a Wi-Fi network:

```
import mip
import network
import time

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

   for _ in range(10):
      if wlan.isconnected():
            print('Connected to the Wi-Fi network')
```

(continues on next page)

(continued from previous page)

```
            return wlan.ifconfig()[0]
        time.sleep(1)

    print('Could not connect to the Wi-Fi network')
    return None


ssid = "your_ssid"
password = "your_password"

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

3.2 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
```

3.2.1 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.

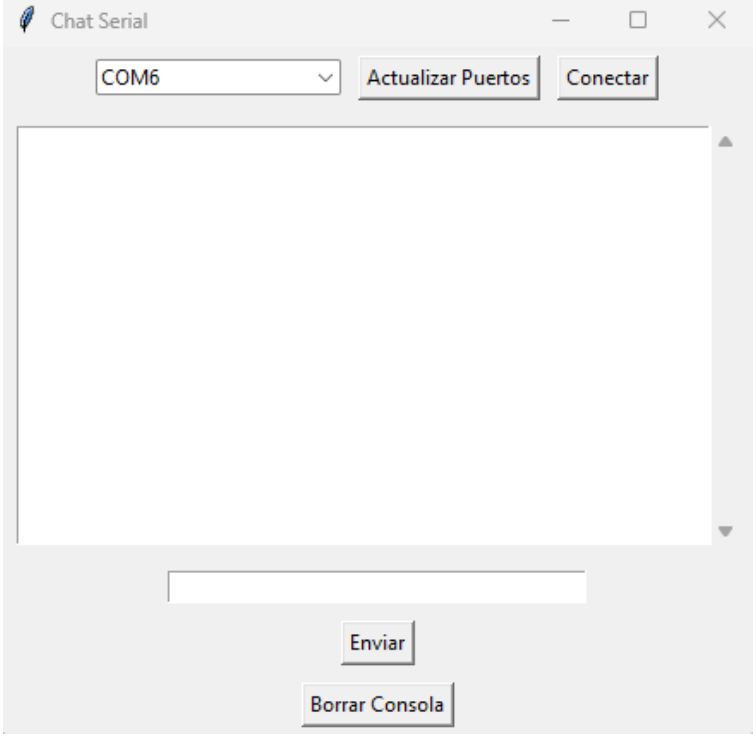


Fig. 3.1: Chatos Library Successfully Installed

3.2.2 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.

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:

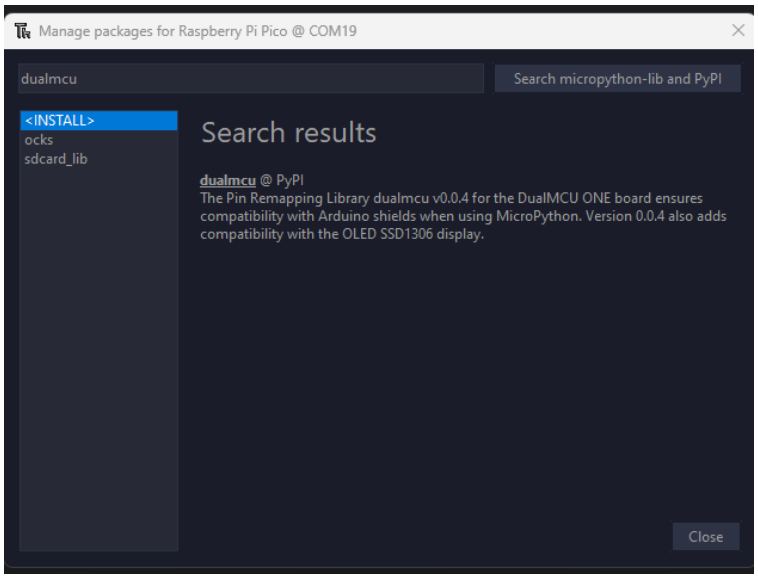


Fig. 3.2: DualMCU Library

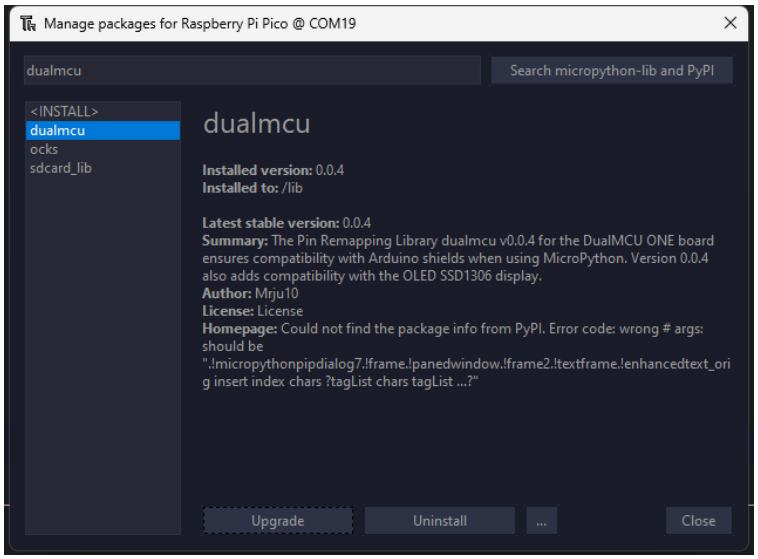


Fig. 3.3: DualMCU Library Successfully Installed

```
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()
```

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

FOUR

DEVELOPMENT BOARD

4.1 Squematic Diagram

4.2 Pinout distribution

The following table provides the pinout details for the UNIT Nano C6 and ESP32 C6 boards.

Arduino Nano Pin	Arduino Nano Description	UNIT Nano C6	ESP32 C6
1	D13 (SCK/LED)	D13/SCK	GPIO6/A6/(SCK)
2	3.3V	3.3V	3.3V
3	AREF	•	GPIO14
4	A0 (Analog)/D14	A0 /D14	GPIO0
5	A1 (Analog)/D15	A1/D15	GPIO1
6	A2 (Analog)/D16	A2/D16	GPIO3
7	A3 (Analog)/D17	A3/D17	GPIO4
8	A4 (SDA)/D18	A4 (SDA)/D18	GPIO22
9	A5 (SCL)/D19	A5 (SCL)/D19	GPIO23
10	A6 (Analog)	•	•
11	A7 (Analog)	A7	GPIO5
12	5V	5V	5V
13	RESET	RST	RST
14	GND	GND	GND
15	VIN	VIN	VIN
16	D0 (RX)	D0/RX	GPIO17
17	D1 (TX)	D1/TX	GPIO16
18	RESET	RST	RST
19	GND	GND	GND
20	D2	D2	GPIO8
21	D3 (PWM)	D3/NEOP	GPIO9
22	D4	D4	GPIO15
23	D5 (PWM)	D5	GPIO19
24	D6 (PWM)	D6	GPIO20
25	D7	D7	GPIO21
26	D8	D8	GPIO12
27	D9 (PWM)	D9	GPIO13

continues on next page

Table 4.1 – continued from previous page

Arduino Nano Pin	Arduino Nano Description	UNIT Nano C6	ESP32 C6
28	D10 (PWM/SS)	D10/SS	GPIO18
29	D11 (PWM/MOSI)	D11/MOSI	GPIO7/(MOSI)
30	D12 (MISO)	D12/MISO	GPIO2/A2/(MISO)

CHAPTER

FIVE

GENERAL PURPOSE INPUT/OUTPUT (GPIO) PINS

The General Purpose Input/Output (GPIO) pins on the UNIT NANO C6 development board are used to connect external devices to the microcontroller. These pins can be configured as either input or output. In this section, we will explore how to work with GPIO pins on the UNIT NANO C6 development board using both MicroPython and C++.

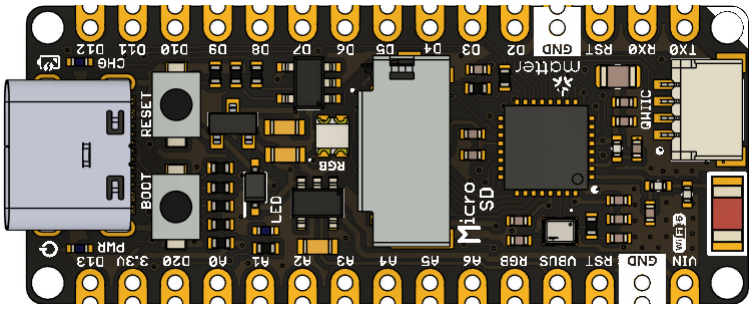


Fig. 5.1: UNIT NANO C6 Development Board

Let's begin with a simple example: blinking an LED. This example demonstrates how to control GPIO pins on the UNIT NANO C6 development board using both MicroPython and C++.

5.1 Working with LEDs on ESP32-C6

In this section, we will learn how to control a single LED using a microcontroller. The LED will be connected to a GPIO pin, and we will control its on/off states using a simple program.

5.1.1 LED Blinking Example

Tip: The following example demonstrates how to blink an LED connected to GPIO pin 15 on the UNIT NANO C6 development board. The LED will turn on for 1 second and then turn off for 1 second, repeating this pattern indefinitely.

MicroPython

```
import machine
import time

led = machine.Pin(6, machine.Pin.OUT)
```

(continues on next page)

(continued from previous page)

```
def loop():
    while True:
        led.on()  # Turn the LED on
        time.sleep(1)  # Wait for 1 second
        led.off()  # Turn the LED off
        time.sleep(1)  # Wait for 1 second

loop()
```

C++

```
#define LED 6

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

// The loop function runs continuously
void loop() {
    digitalWrite(LED, HIGH);   // Turn the LED on (HIGH is the voltage level)
    delay(1000);               // Wait for 1 second
    digitalWrite(LED, LOW);    // Turn the LED off (LOW is the voltage level)
    delay(1000);               // Wait for 1 second
}
```

CHAPTER

SIX

ANALOG TO DIGITAL CONVERSION

Learn how to read analog sensor values using the ADC module on the UNIT Nano C6 development board with the ESP32-C6. This section will cover the basics of analog input and conversion techniques.

6.1 ADC Definition

Analog-to-digital conversion (ADC) is a process that converts analog signals into digital values. The ESP32-C6, equipped with multiple ADC channels, provides flexible options for reading analog voltages and converting them into digital values. Below, you will find the details on how to utilize these pins for ADC operations.

6.1.1 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, a 12-bit ADC can divide the analog signal into 4096 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.

6.2 ADC Pin Mapping

Below is a table showing the distribution of ADC pins on the UNIT Nano C6 board and their corresponding GPIO pins on the ESP32-C6.

Table 6.1: ADC Pin Mapping

Pin Number	UNIT Nano C6	ESP32-C6
1	A0/D14	GPIO0
2	A1/D15	GPIO1
3	A2/D16	GPIO3
4	A3/D17	GPIO4
5	A4/D18	GPIO22
6	A5/D19	GPIO23
7	A7	GPIO5

6.3 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.

6.4 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 0 // GPIO0 for A0
```

6.5 Reading Values

To read the analog value converted to a digital format:

MicroPython

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

C++

```
voltage = analogRead(ADC0);
```

6.6 Example Code

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

MicroPython

```
import machine
import time

# Setup
adc = machine.ADC(machine.Pin(0))  # Initialize pin GPIO0 for ADC

# 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++

```cpp
const int adcPin = 0; // GPIO0 (A0)
int adcValue = 0;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12); // Set resolution to 12-bit
  delay(1000);
}

void loop() {
  // Reading ADC value
  adcValue = analogRead(adcPin);
  Serial.println(adcValue);
  delay(500);
}
```

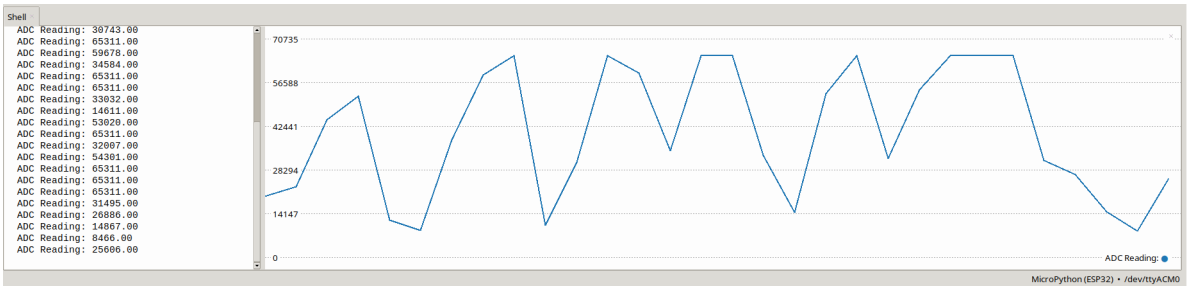


Fig. 6.1: Example of input ADC0 on the UNIT Nano C6 board.

CHAPTER

SEVEN

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.

7.1 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.

7.2 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 7.1: I2C Pinout

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

7.3 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++

```ino
#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();

    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
}
```

7.4 SSD1306 Display

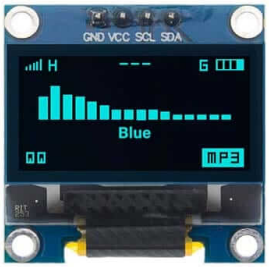


Fig. 7.1: SSD1306 Display

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 7.2: SSD1306 Display Pinout

Pin	Connection
1	GND
2	VCC
3	SDA
4	SCL

7.4.1 Library Support

MicroPython

The *ocks.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 `ocks` and click **Install**.

Alternatively, download the library from ocks.py.

Microcontroller Configuration

class 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 ocks import SSD1306_I2C

i2c = machine.SoftI2C(freq=400000, timeout=50000, sda=machine.Pin(*), scl=machine.Pin(*))

oled = SSD1306_I2C(128, 64, i2c)

# Fill the screen with white and display
oled.fill(1)
oled.show()

# Clear the screen (fill with black)
oled.fill(0)
oled.show()

# Display text
oled.text('UNIT', 50, 10)
oled.text('ELECTRONICS', 25, 20)
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
  }

  // 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
```

(continues on next page)

(continued from previous page)

```
  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

EIGHT

SPI (SERIAL PERIPHERAL INTERFACE)

8.1 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.

8.2 SDCard SPI

Warning: Ensure that the Micro SD contain data. We recommend saving multiple files beforehand to facilitate the use. Format the Micro SD card to FAT32 before using it with the ESP32-C6.

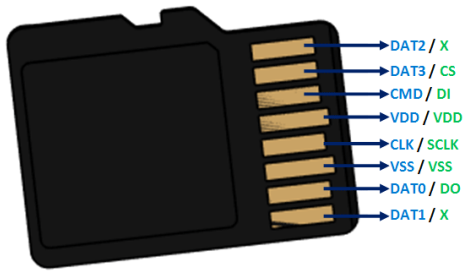


Fig. 8.1: Micro SD Card Pinout

The conections are as follows:

This table illustrates the connections between the SD card and the GPIO pins on the ESP32-C6

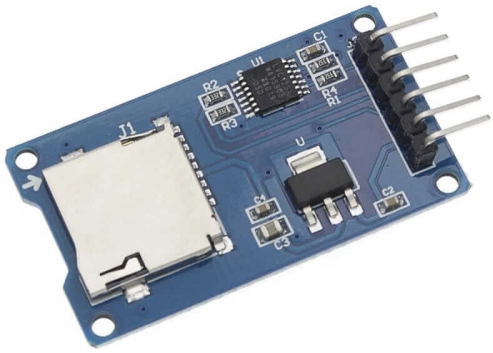


Fig. 8.2: Micro SD Card external reader

Table 8.1: HSPI Connections

SD Card	ESP32C6	PIN
D2		
D3	SS (Slave Select)	19
CMD	MOSI	7
VSS	GND	
VDD	3.3V	
CLK	SCK (Serial Clock)	6
VSS	GND	
D0	MISO	2
D1		

```
import machine
import os
from sdcard import SDCard

# Definir pines para SPI y SD
MOSI_PIN = 7
MISO_PIN = 2
SCK_PIN = 6
CS_PIN = 19

# Inicializar SPI
spi = machine.SPI(1, baudrate=500000, polarity=0, phase=0,
                  sck=machine.Pin(SCK_PIN),
                  mosi=machine.Pin(MOSI_PIN),
                  miso=machine.Pin(MISO_PIN))

# Inicializar tarjeta SD
sd = SDCard(spi, machine.Pin(CS_PIN))

# Montar la SD en el sistema de archivos
```

(continues on next page)

(continued from previous page)

```
os.mount(sd, "/sd")

# Listar archivos y directorios en la SD
print("Archivos en la SD:")
print(os.listdir("/sd"))

# Crear y escribir en un archivo
file_path = "/sd/test.txt"
with open(file_path, "w") as file:
    file.write("Hola, MicroPython en SD!\n")
    file.write("Esto es una prueba de escritura.\n")

# Leer el archivo
with open(file_path, "r") as file:
    print("\nContenido del archivo:")
    print(file.read())

# Confirmar que el archivo se ha creado correctamente
print("\nArchivos en la SD después de la escritura:")
print(os.listdir("/sd"))
```

CHAPTER

NINE

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.

Table 9.1: Pin Mapping for WS2812

PIN	GPIO ESP32C6
DIN	8

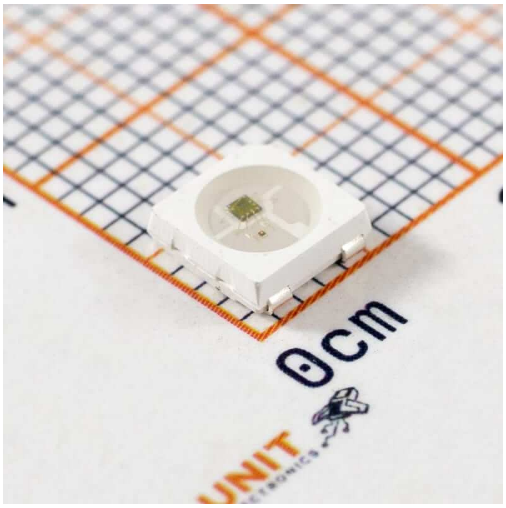


Fig. 9.1: WS2812 LED Strip

9.1 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(8), 1)
np[0] = (255, 128, 0) # set to red, full brightness
```

(continues on next page)

(continued from previous page)

```
np.write()
```

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

CHAPTER

TEN

COMMUNICATION

Unlock the full communication potential of the NANO ESP32C6 board with various communication protocols and interfaces. Learn how to set up and use Wi-Fi, Bluetooth, and serial communication to connect with other devices and networks.

10.1 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())
```

10.2 Bluetooth

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

scan sniffer Code

```
import bluetooth
import time

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

# Helper function to convert memoryview to MAC address string
```

(continues on next page)

(continued from previous page)

```
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

# Global counter for devices found
devices_found = 0
max_devices = 10  # Limit to 10 devices

# Callback function to handle advertising reports
def bt_irq(event, data):
    global devices_found
    if event == 5:  # event 5 is for advertising reports
        if devices_found >= max_devices:
            ble.gap_scan(None)  # Stop scanning
            print("Scan stopped, limit reached.")
            return

        addr_type, addr, adv_type, rssi, adv_data = data
        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")

        devices_found += 1  # Increment counter

        if devices_found >= max_devices:
            ble.gap_scan(None)  # Stop scanning
            print("Scan stopped, limit reached.")

# 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
```

(continues on next page)

(continued from previous page)

```
while True:
    time.sleep(1)
```

10.3 Serial

Learn about serial communication and how to communicate with other devices via serial ports.

CHAPTER

ELEVEN

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

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

M

S