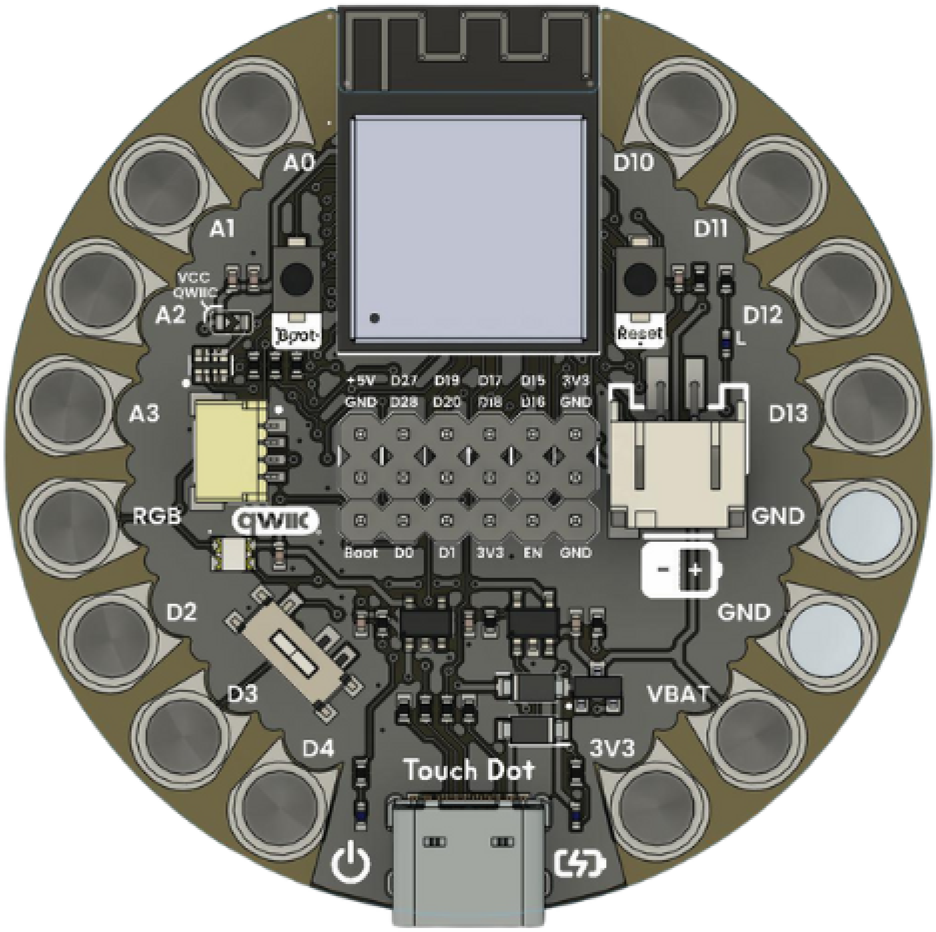

TouchDot User Guide and Technical Reference

Release 0.0.1

Department of Research, Innovation, and Development

Jun 10, 2025

Contents

Note: This documentation is actively evolving. For the latest updates and revisions, please visit the project's GitHub repository.

Leveraging the ESP32-S3 chip, the Touchdot S3 is a versatile development board crafted for creative wearables, IoT implementations, and smart devices. Inspired by the Lilypad aesthetic but delivering modern functionality, it marries a compact form factor with robust connectivity and power management features for seamless prototyping.

Microcontroller: ESP32-S3 Mini

- **Energy Efficient:** Optimized for low power consumption.
- **3.3 V Power Rail:** Compatible with wearable sensors and peripherals like QWIIC modules.

Power Supply & Battery Management

- **USB-C Charging & Communication:** Ensures reliable power delivery and straightforward programming.
- **Integrated LiPo Battery Management:** Streamlines power safety and efficiency without extra circuitry.
- **Distributed Power Pads:** Magnetic connectors deliver **GND** and **3.3 V** for simple, reliable wiring to sensors and actuators.

Key Features

Feature	Description
Wi-Fi & Bluetooth LE	Dual wireless connectivity for seamless IoT and mobile interactions.
Built-in LiPo Charging	Reliable battery charging integrated into the board design.
Power & Reset Controls	Direct access to board power management with dedicated buttons for power and reset.
Sewable Pads & Magnetic Connectors	Ideal for wearable projects and rapid prototyping with flexible module integration.
Multiple Solder Points for GND & 3.3 V	Facilitates easy wiring to external sensors or actuators without complex setup.
Standard QWIIC Connector	Supports quick connection of I^2C peripherals such as sensors, displays, and expansion modules.

CHAPTER ONE

HARDWARE LICENSE (MIT)

CHAPTER
TWO

PINOUT DISTRIBUTION

2.1 Development Board Description

This development board is built around the powerful ESP32-S3 microcontroller, offering a comprehensive range of features for advanced embedded projects. It integrates multiple interfaces including digital I/O, analog inputs, and dedicated communication channels, making it ideal for a variety of applications such as IoT, wearable devices, and rapid prototyping.

Key features include: - Detailed main pin map for the ESP32-S3 with clearly defined GPIO functions. - QWIIC-compatible JST connector, simplifying sensor and peripheral integrations. - Multiple debugging and programming interfaces including a JTAG test port and a serial programming header. - On-board schematic and pinout images embedded for easy reference via HTML elements.

This versatile design provides users with extensive documentation and hardware accessibility, enabling efficient and effective development workflows.

2.2 Squematic Diagram

2.3 Pinout distribution

The following simplified tables present key pinout details.

2.3.1 Main Pin Map – ESP32-S3 TouchDot

Table 2.1: Main Pin Map – ESP32-S3 TouchDot (Simplified)

Pin (Arduino/Unit)	ESP32-S3 GPIO	Function
D13 (SCK) / D13-SCK/T7	GPIO7	RTC_GPIO7, TOUCH7
3.3V / 3.3V	3.3V	Power
AREF / -	•	•
A0 (Analog) / A0/T1	GPIO1	RTC_GPIO1, TOUCH1
A1 (Analog) / A1/T2	GPIO2	RTC_GPIO2, TOUCH2
A2 (Analog) / A2/T3	GPIO3	RTC_GPIO3, TOUCH3
A3 (Analog) / A3/T4	GPIO4	RTC_GPIO4, TOUCH4
A4 (SDA) / A4-SDA/T5	GPIO5	RTC_GPIO5, TOUCH5
A5 (SCL) / A5-SCL/T6	GPIO6	RTC_GPIO6, TOUCH6
5V / 5V	5V	Power
RESET / RST	EN	Enable/Disable
GND / GND	GND	Ground
D0 (RX) / D0-RX	GPIO44	U0RXD
D1 (TX) / D1-TX	GPIO43	U0TXD
D2 / D2-T11	GPIO11	ADC2_CH0
D3 / D3-T12	GPIO12	ADC2_CH1
D4 / D4-T13	GPIO13	ADC2_CH2
D5 / D5-T14	GPIO14	ADC2_CH3
D6	GPIO15	U0RTS
D7	GPIO16	U0CTS
D3 / D5-TX1	GPIO17	U1TXD
D4 / D6-RX1	GPIO18	U1RXD
D10 (SS) / D10-SS/T10	GPIO10	TOUCH10
D11 (MOSI) / D11-MOSI/T9	GPIO9	TOUCH9
D12 (MISO) / D12-MISO/T8	GPIO8	TOUCH8

2.3.2 QWIIC-Compatible JST Connector

Table 2.2: QWIIC-Compatible JST Connector (Simplified)

Pin / JST	ESP32-S3 GPIO	Function
1: GND	GND	Ground
2: 3.3V	3.3V	3.3V
3: SDA/MUX (A4)	GPIO5	RTC_GPIO5, TOUCH5
4: SCL/MUX (A5)	GPIO6	RTC_GPIO6, TOUCH6

2.3.3 JTAG Test Points

Table 2.3: JTAG Test Points (Simplified)

Function (Pin)	ESP32-S3 GPIO	Description
MTCK (D21)	GPIO39	MTCK, CLK_OUT3
MTDO (D22)	GPIO40	MTDO, CLK_OUT2
MTDI (D23)	GPIO41	MTDI, CLK_OUT1
MTMS (D24)	GPIO42	MTMS
GND1 (GND)	GND	Ground
TP_3V3 (3.3V)	Power	3.3V

2.3.4 Serial Programming Header (1x6)

Table 2.4: Serial Programming Header (Simplified)

Pin (JST)	ESP32-S3 GPIO	Function
1: GND	GND	Ground
2: EN/RESET	EN	Enable/Reset
3: 3.3V	3.3V	3.3V
4: TX0 (D1)	GPIO43	U0TXD
5: RX0 (D0)	GPIO44	U0RXD
6: BOOT (D29)	GPIO0	RTC_GPIO0

2.3.5 Expansion Header (2x6)

Table 2.5: Expansion Header (Simplified)

Pin / Function	ESP32-S3 GPIO	Description
1: 3.3V	•	Power
2: GND	•	Ground
3: GPIO33 (D15)	GPIO33	SPIIO4, FSPIHD
4: GPIO34 (D16)	GPIO34	SPIIO5, FSPICS0
5: GPIO35 (D17)	GPIO35	SPIIO6, FSPID
6: GPIO36 (D18)	GPIO36	SPIIO7, FSPI-CLK
7: GPIO37 (D19)	GPIO37	SPIDQS, FSPIQ
8: GPIO38 (D20)	GPIO38	FSPIWP
9: GPIO47/PDM_D (D27)	GPIO47	PDM_DATA
10: GPIO48/PDM_C (D28)	GPIO48	PDM_CLK
11: 5V	•	Power
12: GND	•	Ground

DESKTOP ENVIRONMENT

The environment setup is the first step to start working with the TouchDot S3 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

Fig. 3.1: Add python to PATH

3.1 Install the required software

The following software is required to start working with the TouchDot S3 board:

1. **Python 3.7 or later**: Python is required to run the scripts and tools provided by the TouchDot S3 board.
2. **Git**: Git is required to clone the TouchDot S3 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:

```
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.

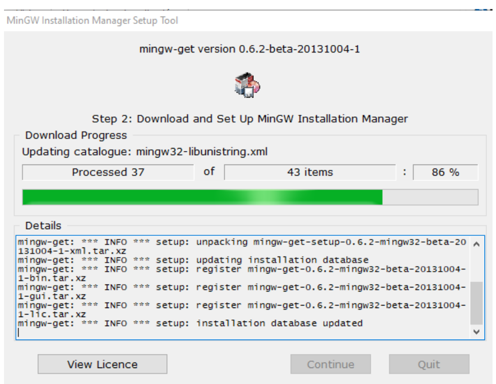

Fig. 3.2: MinGW installer

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

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

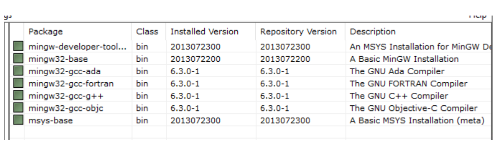

Fig. 3.3: MinGW installation

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

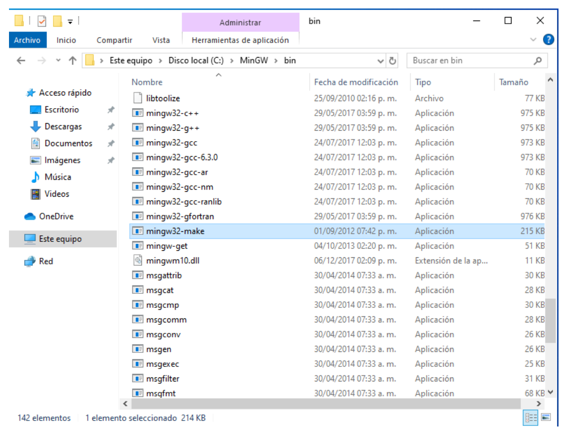

Fig. 3.4: Locating the *mingw32-make.exe* file

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*.

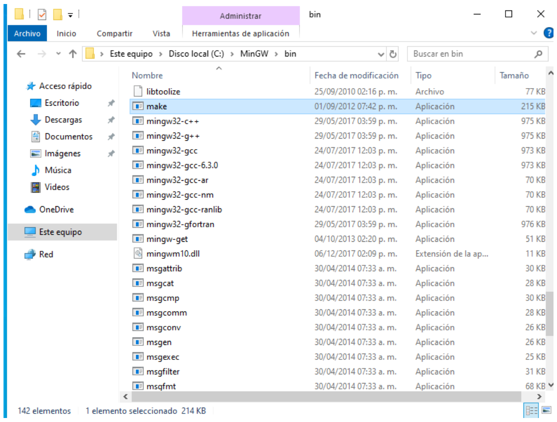

Fig. 3.5: Renaming *mingw32-make.exe* to *make.exe*

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
```

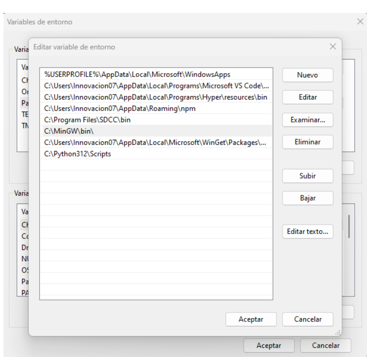

Fig. 3.6: Adding MinGW bin directory to environment variables

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.

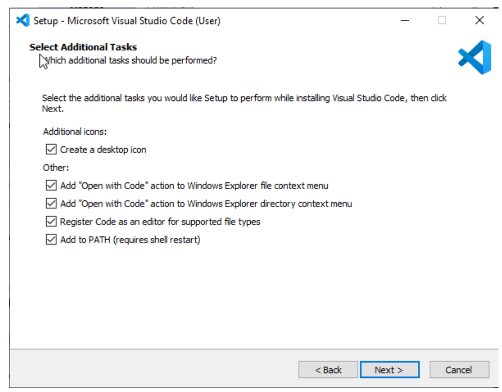

Fig. 3.7: Visual Studio Code installer

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:

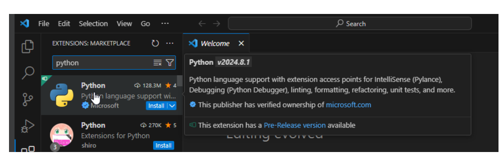

Fig. 3.8: Visual Studio Code extensions

3.6 Arduino IDE Installation

The Arduino IDE is a popular open-source platform for building and programming microcontroller-based projects. It provides a user-friendly interface and a wide range of libraries to simplify the development process.

To install the Arduino IDE, follow the instructions for your operating system in the

3.7 Thonny IDE Installation

Thonny is a Python IDE that is designed for beginners. It provides a simple interface and built-in support for MicroPython, making it an excellent choice for programming the TouchDot S3 board.

Follow the instructions for your operating system in the

CHAPTER
FOUR

INSTALLING PACKAGES - MICROPYTHON

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

4.1 Installation Guide Using MIP Library

Note: The *mip* library is utilized to install other libraries for MicroPython on the ESP32-S3. It simplifies the process of managing and installing packages directly from the internet.

4.1.1 Requirements

- ESP32-S3 device
- Thonny IDE
- Wi-Fi credentials (SSID and Password)

4.1.2 Installation Instructions

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

4.1.3 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):
```

(continues on next page)

(continued from previous page)

```
        if wlan.isconnected():
            print('Connected to the Wi-Fi network')
            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')
```

4.2 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.

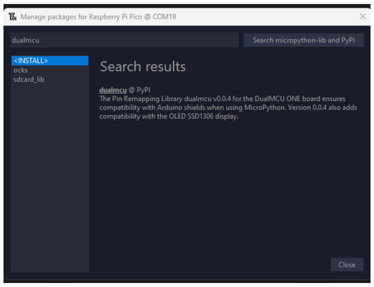

Fig. 4.1: DualMCU Library

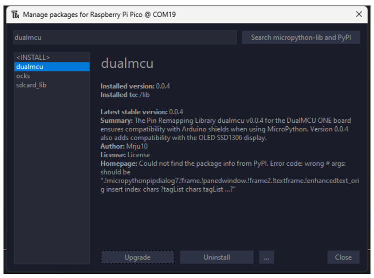

Fig. 4.2: DualMCU Library Successfully Installed

4.2.1 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:

```python
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()
```

(continues on next page)

(continued from previous page)

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

oled.show()
```

4.3 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.

CHAPTER

FIVE

ESP-IDF GETTING STARTED

The ESP-IDF (Espressif IoT Development Framework) is the official development framework for ESP32 series chips. It provides a comprehensive suite of tools, libraries, and APIs to facilitate application development for ESP32 devices.

This section offers a step-by-step guide to setting up the ESP-IDF environment for the ESP32-S3 chip, including installation instructions and basic usage examples. While the focus is on the ESP32-S3, the guidelines are generally applicable to other ESP32 chips.

Supported Environment: Ubuntu 20.04 or later.

For users on other operating systems, please consult the official ESP-IDF documentation for platform-specific instructions.

Note: **ESP-IDF** is compatible with Windows and macOS, but the installation process may differ. Refer to the official documentation for detailed instructions.

Attention: A stable internet connection is required during installation, as some steps involve downloading necessary files.

5.1 Installation Steps

1. **Install Prerequisites** Ensure all required dependencies are installed. Execute the following commands in a terminal:

```
sudo apt-get update
sudo apt-get install git wget flex␣
↪bison gperf python3 python3-pip␣
↪python3-setuptools python3-venv␣
↪cmake ninja-build ccache libffi-dev␣
↪libssl-dev dfu-util device-tree-
↪compiler
```

2. **Clone the ESP-IDF Repository** Clone the ESP-IDF repository from GitHub. Optionally, specify a particular version or branch.

```
git clone https://github.com/espressif/
↪esp-idf.git
```

3. **Set Up the Environment** Navigate to the cloned ESP-IDF directory and execute the setup script to configure environment variables.

```
cd esp-idf
./install.sh
. ./export.sh
```

Note: To install tools for all supported chips, use the following command:

```
./install.sh --all
```

---

4. **Install Additional Tools** For ESP32-S3-specific tools, run:

```
./install.sh --esp32s3
```

Note: The *install.sh* script downloads and installs the required tools and dependencies for the ESP32-S3 chip. The duration depends on your internet speed.

---

5. **Verify Installation** Confirm the installation by checking the ESP-IDF version:

```
idf.py --version
```

5.2 Customizing the Installation Path

To customize the installation path of ESP-IDF, set the *IDF_PATH* environment variable. For example:

```
export IDF_PATH=/path/to/your/esp-idf
. $IDF_PATH/export.sh
. $IDF_PATH/install.sh
```

Note: Replace */path/to/your/esp-idf* with the desired installation directory. This ensures the *IDF_PATH* variable points to the correct location, and the *export.sh* and *install.sh* scripts are executed from there.

5.3 First Steps with ESP-IDF

1. **Create a New Project** Create a directory for your ESP-IDF project and navigate to it:

   ```
   mkdir my_project
   cd my_project
   ```

2. **Generate a Basic Application** Use the *idf.py* tool to create a basic application template:

   ```
   idf.py create-project my_app
   ```

3. **Build the Project** Navigate to the project directory and build the application:

   ```
   cd my_app
   idf.py build
   ```

4. **Flash the Application** Connect your ESP32-S3 board to your computer and flash the application:

   ```
   idf.py -p /dev/ttyUSB0 flash
   ```

5. **Monitor the Output** Monitor the output from the ESP32-S3 board:

   ```
   idf.py -p /dev/ttyUSB0 monitor
   ```

6. **Modify the Code** Edit the code in the *main* directory of your project. The main application file is typically named *main.c* or *main.cpp*. After making changes, rebuild and flash the project.

7. **Clean the Project** To remove all build artifacts, run:

   ```
   idf.py fullclean
   ```

8. **Update ESP-IDF** To update ESP-IDF to the latest version, navigate to the ESP-IDF directory and execute:

   ```
   git pull
   ./install.sh
   . ./export.sh
   ```

9. **Uninstall ESP-IDF** To uninstall ESP-IDF, delete the cloned repository and unset related environment variables:

   ```
   rm -rf esp-idf
   unset IDF_PATH
   unset PATH
   unset LD_LIBRARY_PATH
   unset PYTHONPATH
   unset CMAKE_PREFIX_PATH
   ```

10. **Explore ESP-IDF Examples** The ESP-IDF repository includes numerous example projects demonstrating various features. These can be found in the *examples* directory. Copy and modify any example project as needed.

11. **Refer to ESP-IDF Documentation** For comprehensive information, including API references and guides, visit the official ESP-IDF documentation: ESP-IDF Documentation.

12. **Join the ESP-IDF Community** For assistance or discussions, join the ESP-IDF community on GitHub or the Espressif Community Forum. The community is active and provides support for various ESP32 development topics.

CHAPTER
SIX

GENERAL PURPOSE INPUT/OUTPUT (GPIO) PINS

The General Purpose Input/Output (GPIO) pins on the **TouchDot S3** 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 **TouchDot S3** development board using both MicroPython and C++.

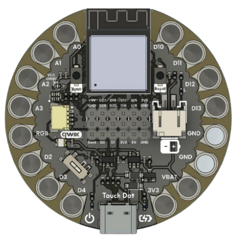

Fig. 6.1: **TouchDot S3** Development Board

Let's begin with a simple example: blinking an LED. This example demonstrates how to control GPIO pins on the **TouchDot S3** development board using both MicroPython and C++.

6.1 Working with LEDs on ESP32-S3

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.

6.1.1 LED Blinking Example

Tip: The following example demonstrates how to blink an LED connected to GPIO pin 6 on the **TouchDot S3** development board. The LED will turn on for 1 second and then turn off for 1 second, repeating this pattern indefinitely.

MicroPython

```python
import machine
import time

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

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++

```cpp
#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
}
```

esp-idf

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
```

(continues on next page)

(continued from previous page)

```c
#include "freertos/task.h"
#include "driver/gpio.h"

#define BLINK_GPIO GPIO_NUM_6  // Puedes␣
↪cambiarlo según tu hardware

void app_main(void)
{
    // Configura el GPIO como salida
    gpio_reset_pin(BLINK_GPIO);
    gpio_set_direction(BLINK_GPIO, GPIO_
↪MODE_OUTPUT);

    while (1) {
        // Enciende el LED
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(pdMS_TO_TICKS(500)); //␣
↪500 ms

        // Apaga el LED
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(pdMS_TO_TICKS(500)); //␣
↪500 ms
    }
}
```

CHAPTER SEVEN

ANALOG TO DIGITAL CONVERSION

Learn how to read analog sensor values using the ADC module on the **TouchDot S3** development board with the ESP32-S3. This section will cover the basics of analog input and conversion techniques.

7.1 ADC Definition

Analog-to-digital conversion (ADC) is a process that converts analog signals into digital values. The ESP32-S3, 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.

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, 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.

7.2 ADC Pin Mapping

Below is a table showing the distribution of ADC pins on the **TouchDot S3** board and their corresponding GPIO pins on the ESP32-S3.

Table 7.1: ADC Pin Mapping

Pin Number	**TouchDot S3**	ESP32-S3
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

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 0 // GPIO0 for A0
```

7.5 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);
```

7.6 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++

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

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

(continues on next page)

(continued from previous page)

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

esp-idf

```
#include <stdio.h>
#include "esp_log.h"
#include "esp_err.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_adc/adc_oneshot.h"

static const char *TAG = "ADC_MIN";

void app_main(void)
{
    adc_oneshot_unit_handle_t adc_handle;
    adc_oneshot_unit_init_cfg_t init_cfg =
↪{
        .unit_id = ADC_UNIT_1,
    };
    ESP_ERROR_CHECK(adc_oneshot_new_unit(&
↪init_cfg, &adc_handle));

    adc_oneshot_chan_cfg_t chan_cfg = {
        .bitwidth = ADC_BITWIDTH_DEFAULT,
        .atten = ADC_ATTEN_DB_12,  // <-␣
↪Usa el recomendado
    };
    ESP_ERROR_CHECK(adc_oneshot_config_
↪channel(adc_handle, ADC_CHANNEL_2, &chan_
↪cfg)); // GPIO2

    int adc_raw;
    while (1) {
        ESP_ERROR_CHECK(adc_oneshot_
↪read(adc_handle, ADC_CHANNEL_2, &adc_
↪raw));
        ESP_LOGI(TAG, "Lectura ADC␣
↪(GPIO2): %d", adc_raw);
        vTaskDelay(pdMS_TO_TICKS(1000));  /
↪/ <- Necesitabas incluir FreeRTOS
    }
}
```

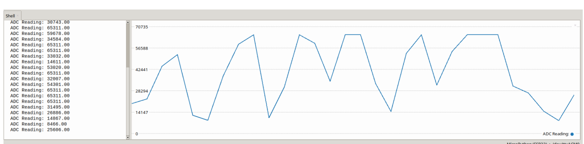

Fig. 7.1: Example of input ADC0 on the **TouchDot S3** board.

I2C (INTER-INTEGRATED CIRCUIT)

Discover the I2C communication protocol and learn how to communicate with I2C devices using the PULSAR C6 board. This section will cover I2C bus setup and communication with I2C peripherals.

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 PULSAR C6 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 PULSAR C6, detailing the pin assignments for SDA and SCL.

Table 8.1: QWIIC-Compatible JST Connector

Pin	JST Function	Arduino Compatibility	ESP S3 GPI(	GPIO Function
1	GND	GND	GNE	GND
2	3.3V	3.3V	3.3V	3.3V
3	SDA / MUX IO	A4	GPI(	RTC_GPIO5, GPIO5, TOUCH5, ADC1_CH4
4	SCL / MUX IO	A5	GPI(	RTC_GPIO6, GPIO6, TOUCH6, ADC1_CH5

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.I2C(0, scl=machine.Pin(6),␣
↪sda=machine.Pin(5))
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(5);
  Wire.setSCL(6);
  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␣
```

(continues on next page)

(continued from previous page)

```
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
}
```

8.4 SSD1306 Display

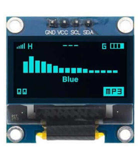

Fig. 8.1: SSD1306 Display

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 8.2: SSD1306 Display Pinout

Pin	Connection
1	GND
2	VCC
3	SDA
4	SCL

8.4.1 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

```
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(*))
```

(continues on next page)

(continued from previous page)

```
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()
```

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

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**.

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        5   // SDA pin
#define SCL_PIN        6   // 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);
```

(continues on next page)

(continued from previous page)

```
  // Initialize I2C
  Wire.setSDA(SDA_PIN);
  Wire.setSCL(SCL_PIN);
  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
  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);
}
```

esp-idf

```c
#include "ssd1306.h"
#include "driver/i2c.h"
#include "esp_log.h"

#define I2C_MASTER_NUM I2C_NUM_0
#define I2C_MASTER_SDA_IO 5
#define I2C_MASTER_SCL_IO 6
#define I2C_MASTER_FREQ_HZ 100000

static const char *TAG = "MAIN";

void scan_i2c_bus(void) {
    ESP_LOGI(TAG, "Scanning I2C bus...");
    for (uint8_t addr = 1; addr < 127; addr++) {
        i2c_cmd_handle_t cmd = i2c_cmd_link_create();
        i2c_master_start(cmd);
        i2c_master_write_byte(cmd, (addr << 1) | I2C_MASTER_WRITE, true);
        i2c_master_stop(cmd);
        esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 100 / portTICK_PERIOD_MS);
        i2c_cmd_link_delete(cmd);
        if (ret == ESP_OK) {
          ESP_LOGI(TAG, "Found device at 0x%02X", addr);
        }
    }
    ESP_LOGI(TAG, "Scan complete.");
}

void app_main(void) {
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };

    i2c_param_config(I2C_MASTER_NUM, &conf);
    i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);

    scan_i2c_bus(); // Optional

    ssd1306_init(I2C_MASTER_NUM);
    ssd1306_clear(I2C_MASTER_NUM);
    ssd1306_draw_text(I2C_MASTER_NUM, 0,
```

(continues on next page)

(continued from previous page)

```c
"ESP32-S3 ");
    ssd1306_draw_text(I2C_MASTER_NUM, 2, "I2C Scan + OLED");
    ssd1306_draw_text(I2C_MASTER_NUM, 4, "Monosaurio");
}
```

CHAPTER

NINE

SPI (SERIAL PERIPHERAL INTERFACE)

9.1 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 TouchDot development board features SPI communication capabilities, allowing you to interface with a wide range of SPI devices.

9.2 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-S3.

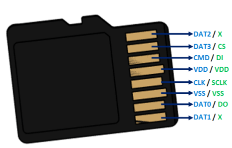

Fig. 9.1: Micro SD Card Pinout

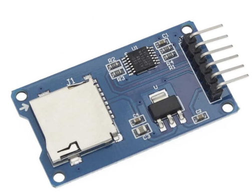

Fig. 9.2: Micro SD Card external reader

The conections are as follows:

This table illustrates the connections between the SD card and the GPIO pins on the ESP32-S3

Table 9.1: microSD Connector (SPI Mode)

P	microSD Pin Name	SPI Function	ESP S3 GPI(	GPIO Function	Type
1	DAT2	Not used in SPI	•	•	•
2	CD / DAT3	CS (Chip Select)	GPI(	RTC_GPIO2, GPIO21	I/O/T
3	CMD	MOSI	GPI(	RTC_GPIO9, TOUCH9, ADC1_CH8, FSPIHD, SUBSPIHD	I/O/T
4	VDD	3.3V	3.3V	Power Supply	P
5	CLK	SCLK	GPI(	RTC_GPIO7, TOUCH7, ADC1_CH6	I/O/T
6	VSS	GND	GNI	Ground	GND
7	DAT0	MISO	GPI(	RTC_GPIO8, TOUCH8, ADC1_CH7, SUBSPICS1	I/O/T
8	DAT1	Not used in SPI	•	•	•

MicroPython

```
import machine
import os
from sdcard import SDCard

# Definir pines para SPI y SD
MOSI_PIN = 9
MISO_PIN = 8
SCK_PIN = 7
CS_PIN = 21
```

(continues on next page)

(continued from previous page)

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

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

C++

```
#include <SPI.h>
#include <SD.h>

// Pines SPI (ajusta según tu placa si es
    necesario)
#define MOSI_PIN 9
#define MISO_PIN 8
#define SCK_PIN 7
#define CS_PIN 21

File myFile;

void setup() {
  Serial.begin(115200);
  while (!Serial) ; // Esperar a que el
    puerto serie esté listo

  // Configurar los pines SPI manualmente
    si tu placa lo requiere
  SPI.begin(SCK_PIN, MISO_PIN, MOSI_PIN,
    CS_PIN);

  Serial.println("Inicializando tarjeta SD.
    ..");

  if (!SD.begin(CS_PIN)) {
    Serial.println("Error al inicializar
    la tarjeta SD.");
    return;
  }
```

(continues on next page)

(continued from previous page)

```
  Serial.println("Tarjeta SD inicializada
    correctamente.");

  // Listar archivos
  Serial.println("Archivos en la SD:");
  listDir(SD, "/", 0);

  // Crear y escribir en el archivo
  myFile = SD.open("/test.txt", FILE_
    WRITE);
  if (myFile) {
    myFile.println("Hola, Arduino en SD!");
    myFile.println("Esto es una prueba de
    escritura.");
    myFile.close();
    Serial.println("Archivo escrito
    correctamente.");
  } else {
    Serial.println("Error al abrir test.
    txt para escribir.");
  }

  // Leer el archivo
  myFile = SD.open("/test.txt");
  if (myFile) {
    Serial.println("\nContenido del
    archivo:");
    while (myFile.available()) {
      Serial.write(myFile.read());
    }
    myFile.close();
  } else {
    Serial.println("Error al abrir test.
    txt para lectura.");
  }

  // Volver a listar archivos
  Serial.println("\nArchivos en la SD
    después de la escritura:");
  listDir(SD, "/", 0);
}

void loop() {
  // Nada en el loop
}

// Función para listar archivos y carpetas
void listDir(fs::FS &fs, const char *
    dirname, uint8_t levels) {
  File root = fs.open(dirname);
  if (!root) {
    Serial.println("Error al abrir el
```

(continues on next page)

(continued from previous page)

```
↪directorio");
    return;
  }
  if (!root.isDirectory()) {
    Serial.println("No es un directorio");
    return;
  }

  File file = root.openNextFile();
  while (file) {
    Serial.print("  ");
    Serial.print(file.name());
    if (file.isDirectory()) {
      Serial.println("/");
      if (levels) {
        listDir(fs, file.name(), levels -␣
↪1);
      }
    } else {
      Serial.print("\t\t");
      Serial.println(file.size());
    }
    file = root.openNextFile();
  }
}
```

esp-idf

```
#include <string.h>
#include <sys/stat.h>
#include "esp_log.h"
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"

#define MOUNT_POINT "/sdcard"

#define PIN_NUM_MISO  CONFIG_EXAMPLE_PIN_
↪MISO
#define PIN_NUM_MOSI  CONFIG_EXAMPLE_PIN_
↪MOSI
#define PIN_NUM_CLK   CONFIG_EXAMPLE_PIN_
↪CLK
#define PIN_NUM_CS    CONFIG_EXAMPLE_PIN_CS

static const char *TAG = "SDCARD";

void app_main(void)
{
    esp_err_t ret;
    sdmmc_card_t *card;

    ESP_LOGI(TAG, "Initializing SD card...
```

(continues on next page)

(continued from previous page)

```
↪");

    esp_vfs_fat_sdmmc_mount_config_t mount_
↪config = {
        .format_if_mount_failed = false,
        .max_files = 3,
        .allocation_unit_size = 16 * 1024
    };

    sdmmc_host_t host = SDSPI_HOST_
↪DEFAULT();

    spi_bus_config_t bus_cfg = {
        .mosi_io_num = PIN_NUM_MOSI,
        .miso_io_num = PIN_NUM_MISO,
        .sclk_io_num = PIN_NUM_CLK,
        .quadwp_io_num = -1,
        .quadhd_io_num = -1,
        .max_transfer_sz = 4000,
    };

    ret = spi_bus_initialize(host.slot, &
↪bus_cfg, SDSPI_DEFAULT_DMA);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to init SPI␣
↪bus.");
        return;
    }

    sdspi_device_config_t slot_config =␣
↪SDSPI_DEVICE_CONFIG_DEFAULT();
    slot_config.gpio_cs = PIN_NUM_CS;
    slot_config.host_id = host.slot;

    ret = esp_vfs_fat_sdspi_mount(MOUNT_
↪POINT, &host, &slot_config, &mount_
↪config, &card);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to mount␣
↪filesystem.");
        return;
    }

    ESP_LOGI(TAG, "Filesystem mounted.");

    const char *file_path = MOUNT_POINT"/
↪test.txt";
    FILE *f = fopen(file_path, "w");
    if (f == NULL) {
        ESP_LOGE(TAG, "Failed to open file␣
↪for writing.");
        return;
```

(continues on next page)

(continued from previous page)

```
    }

    fprintf(f, "Hello from ESP32!\n");
    fclose(f);
    ESP_LOGI(TAG, "File written.");

    f = fopen(file_path, "r");
    if (f) {
        char line[64];
        fgets(line, sizeof(line), f);
        fclose(f);
        ESP_LOGI(TAG, "Read from file: '%s'
↪", line);
    } else {
        ESP_LOGE(TAG, "Failed to read file.
↪");
    }

    esp_vfs_fat_sdcard_unmount(MOUNT_POINT,
↪ card);
    spi_bus_free(host.slot);
    ESP_LOGI(TAG, "Card unmounted.");
}
```

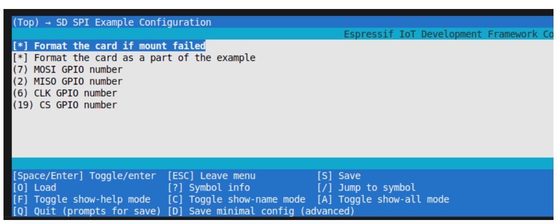

Fig. 9.3: ESP-IDF Menuconfig SD SPI Configuration

CHAPTER TEN

WS2812 CONTROL

Harness the power of WS1280 LED strips with the TouchDot S3 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 TouchDot S3 board. The TouchDot S3 board has a GPIO pin embebbed connected to the single WS2812 LED.

Table 10.1: Pin Mapping for WS2812

PIN	GPIO ESP32-S3
DIN	45

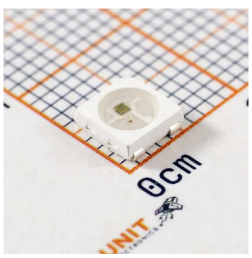

Fig. 10.1: WS2812 LED Strip

10.1 Code Example

Below is an example that demonstrates how to control WS1280 LED strips using the TouchDot S3 board

MicroPython

```
from machine import Pin
from neopixel import NeoPixel
np = NeoPixel(Pin(45), 1)
np[0] = (255, 128, 0) # set to red, full brightness

np.write()
```

C++

```
#include <Adafruit_NeoPixel.h>
#define PIN 45
Adafruit_NeoPixel strip = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ800);
void setup() {
   strip.begin();
   strip.setPixelColor(0, 255, 128, 0); // set to red, full brightness
   strip.show();
}
```

esp-idf

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/rmt_tx.h"
#include "esp_err.h"

void app_main(void) {
   rmt_channel_handle_t tx_channel = NULL;
   rmt_tx_channel_config_t tx_config = {
      .gpio_num = GPIO_NUM_45,
      .clk_src = RMT_CLK_SRC_DEFAULT,
      .resolution_hz = 10000000, // 10MHz resolution, 1 tick = 0.1us
      .mem_block_symbols = 64,
      .trans_queue_depth = 4,
      .flags.invert_out = false,
      .flags.with_dma = false,
   };
   ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_config, &tx_channel));
   ESP_ERROR_CHECK(rmt_enable(tx_channel));

   rmt_encoder_handle_t bytes_encoder = NULL;
   rmt_bytes_encoder_config_t bytes_encoder_config = {
      .bit0 = {.level0 = 1, .duration0 = 3, .level1 = 0, .duration1 = 9},  // 0: ~0.3us high, ~0.9us low
      .bit1 = {.level0 = 1, .duration0 = 9,
```

(continues on next page)

(continued from previous page)

```
↪ .level1 = 0, .duration1 = 3},  // 1: ~0.
↪9us high, ~0.3us low
      .flags.msb_first = true,
   };
   ESP_ERROR_CHECK(rmt_new_bytes_encoder(&
↪bytes_encoder_config, &bytes_encoder));

   rmt_transmit_config_t tx_trans_config =
↪{
      .loop_count = 0,
   };

   uint8_t r = 255, g = 0, b = 0;

   while (1) {
      if (r == 255 && g < 255 && b == 0) {
            g++;
      } else if (g == 255 && r > 0 && b ==␣
↪0) {
            r--;
      } else if (g == 255 && b < 255 && r␣
↪== 0) {
            b++;
      } else if (b == 255 && g > 0 && r ==␣
↪0) {
            g--;
      } else if (b == 255 && r < 255 && g␣
↪== 0) {
            r++;
      } else if (r == 255 && b > 0 && g ==␣
↪0) {
            b--;
      }
      uint8_t color_data[3] = {g, r, b};

      // printf("%d %d %d\n",r,g,b);

      ESP_ERROR_CHECK(rmt_transmit(tx_
↪channel, bytes_encoder, color_data,␣
↪sizeof(color_data), &tx_trans_config));
      ESP_ERROR_CHECK(rmt_tx_wait_all_
↪done(tx_channel, portMAX_DELAY));
      vTaskDelay(pdMS_TO_TICKS(10));
   }
}
```

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

CHAPTER

ELEVEN

COMMUNICATION

Unlock the full communication potential of the ESP32-S3 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.

11.1 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())
```

11.2 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␣
```

(continues on next page)

(continued from previous page)

```
↪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

# 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,␣
```

(continues on next page)

(continued from previous page)

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

11.3 Serial

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

CHAPTER

TWELVE

HOW TO GENERATE AN ERROR REPORT

This guide explains how to generate an error report using GitHub repositories.

12.1 Steps to Create an Error Report

1. **Access the GitHub Repository**
   Navigate to the GitHub repository where the project is hosted.
2. **Open the Issues Tab**
   Click on the "Issues" tab located in the repository menu.
3. **Create a New Issue**
   - Click the "New Issue" button.
   - Provide a clear and concise title for the issue.
   - Add a detailed description, including relevant information such as:
     - Steps to reproduce the error.
     - Expected and actual results.
     - Any related logs, screenshots, or files.
4. **Submit the Issue**
   Once the form is complete, click the "Submit" button.

12.2 Review and Follow-Up

The development team or maintainers will review the issue and take appropriate action to address it.

INDEX

M