Architectures for Embedded Systems

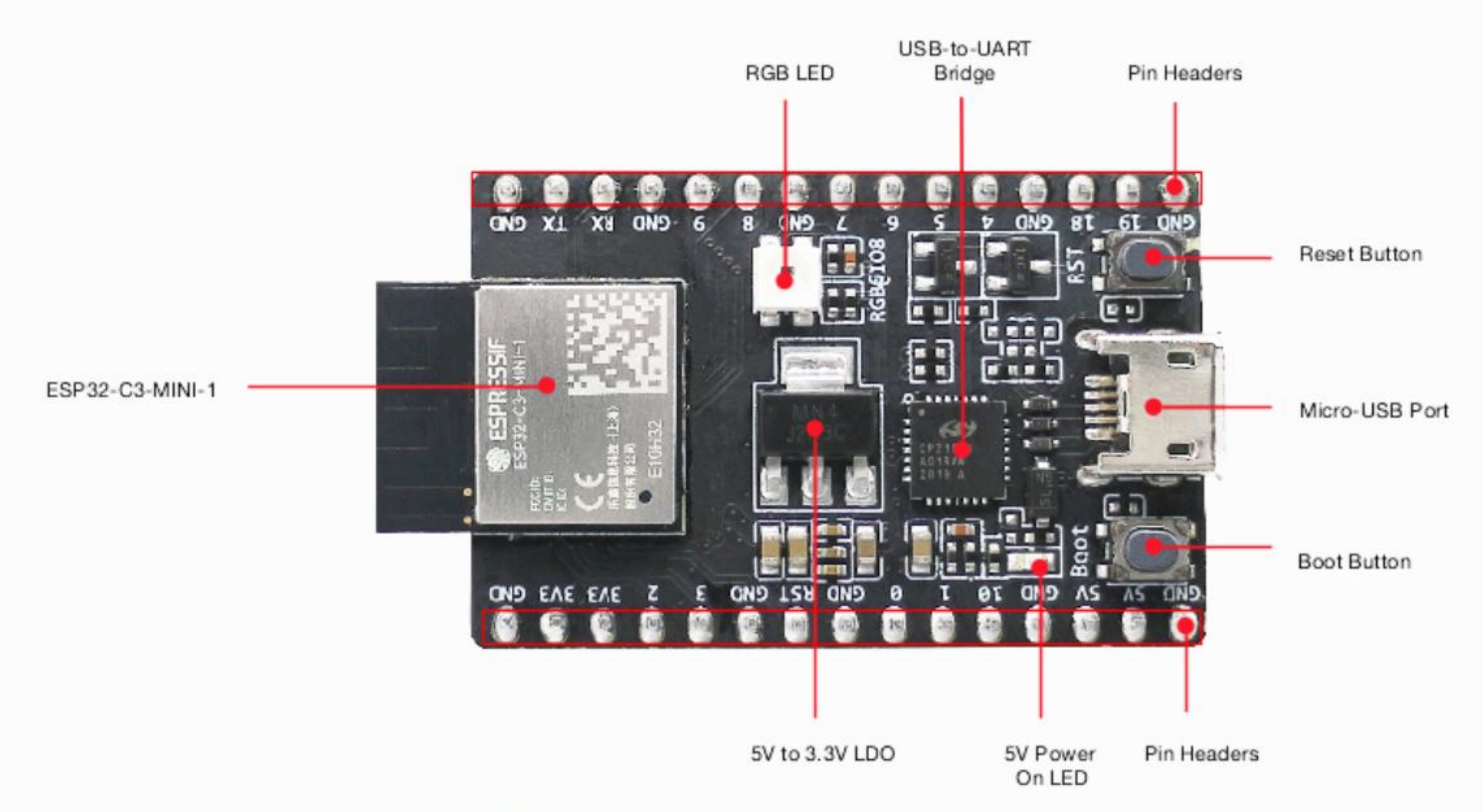
Introduction to ESP32, kit, toolchain and examples Laboratory assignments

Arnaldo S. R. Oliveira
16/02/2025 - Academic year 2024/25
Universidade de Aveiro – Dep. de Eletrónica, Telecomunicações e Informática

Outline

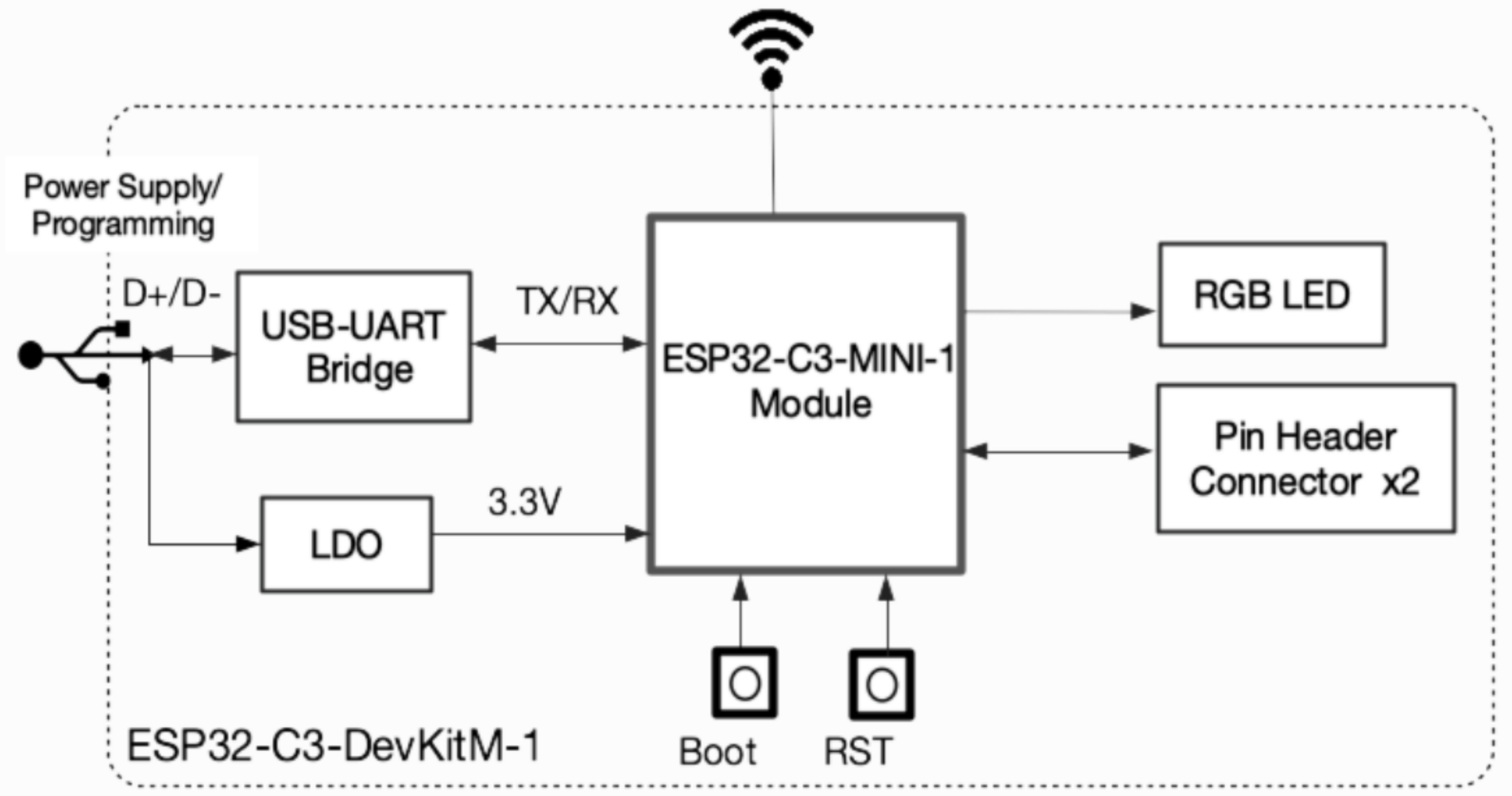
- ESP32-C3 - the kit, the module and the SoC
- Compilation toolchain
- Get started examples
- Lab assignments

ESP32-C3-DevKitM-1 - Kit front view



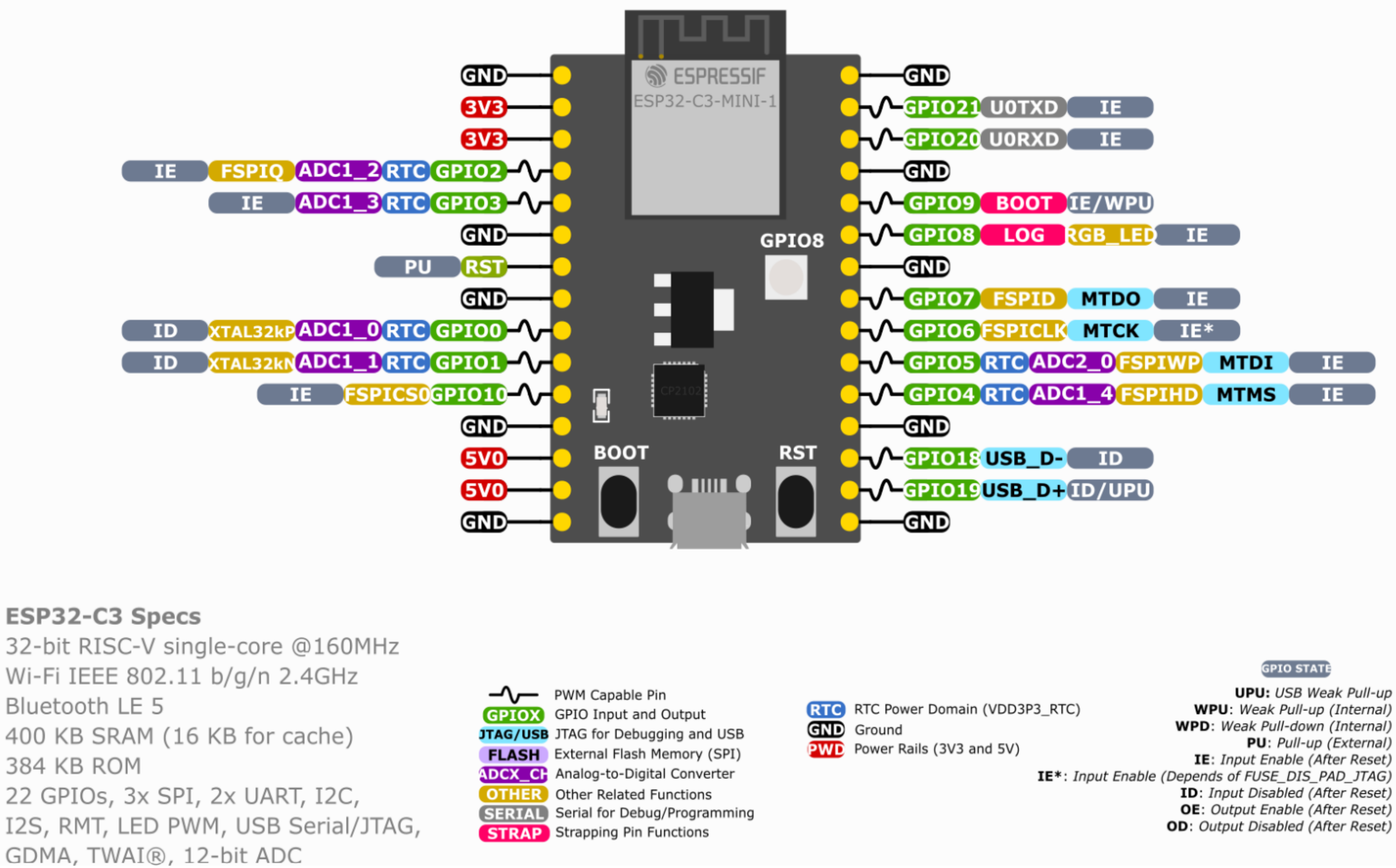
ESP32-C3-DevKitM-1 - front

ESP32-C3-DevKitM-1 - Kit block diagram

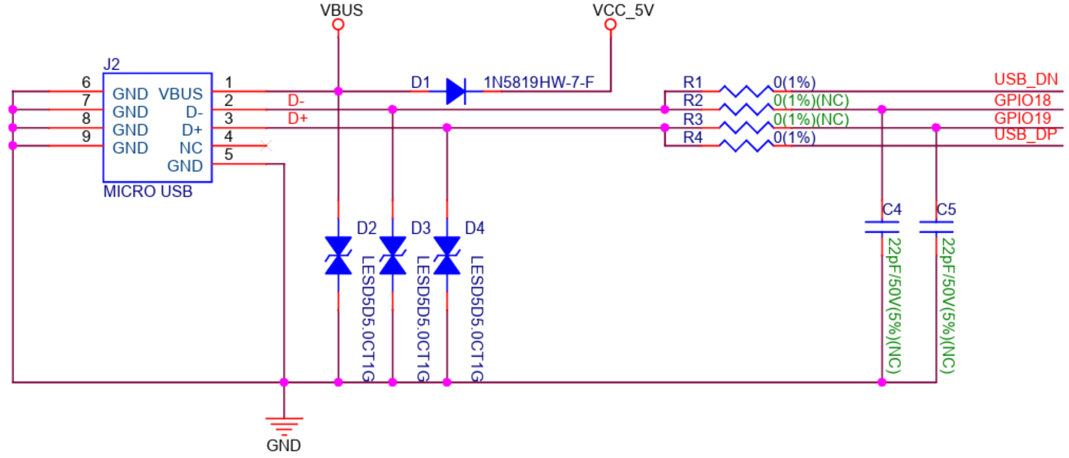


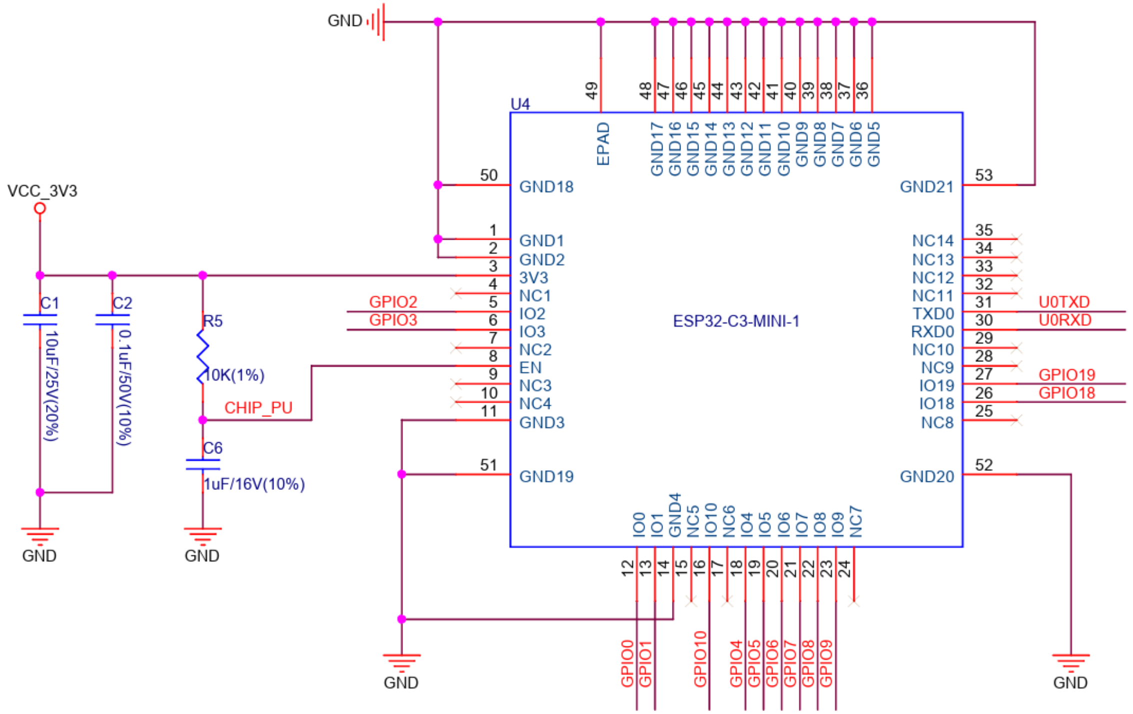
ESP32-C3-DevKitM-1 (click to enlarge)

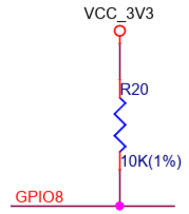
ESP32-C3-DevKitM-1 - Mux’ed pin functions



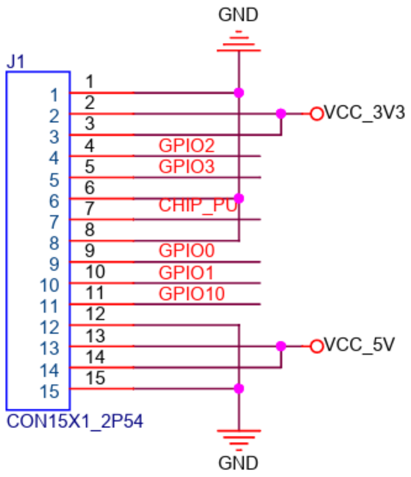
ESP32-C3-DevKitM-1 – Kit schematics (1/3)

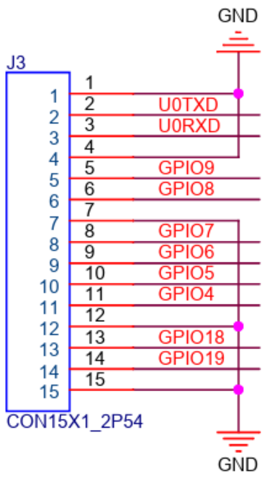


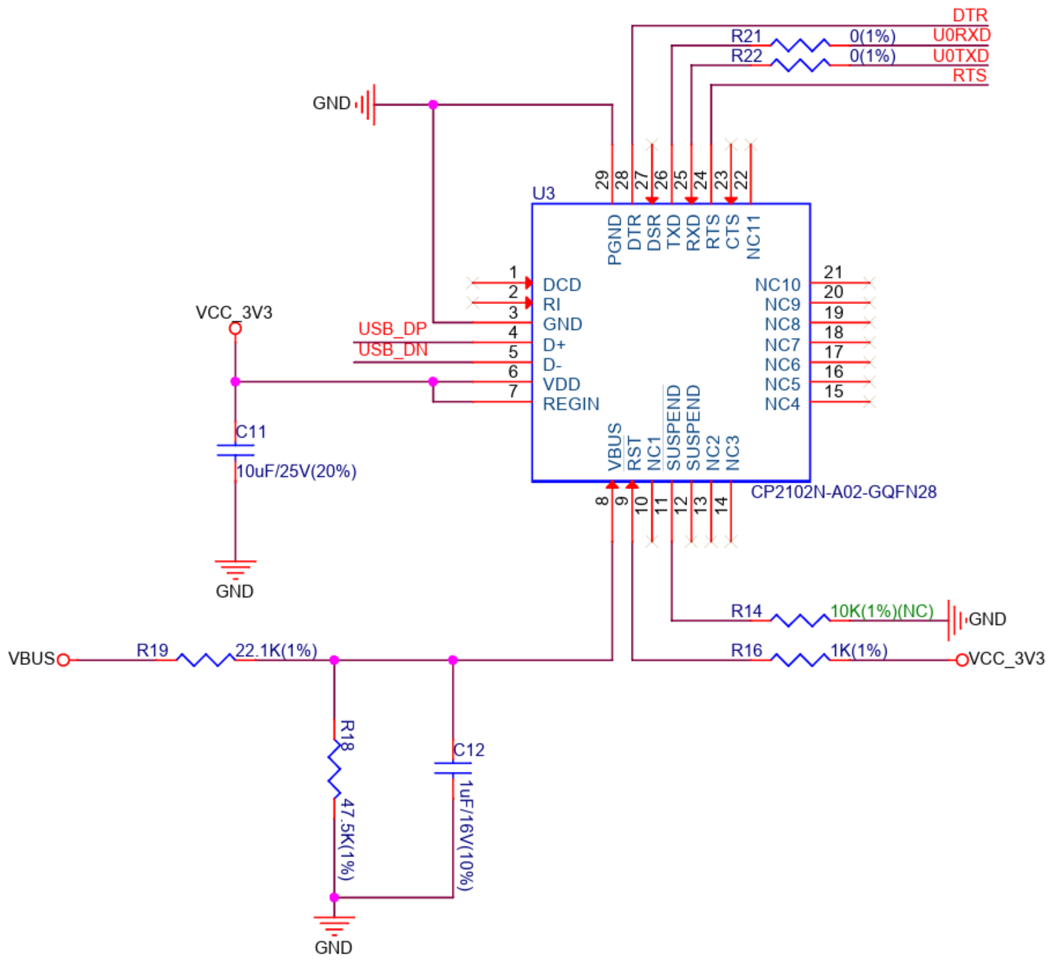




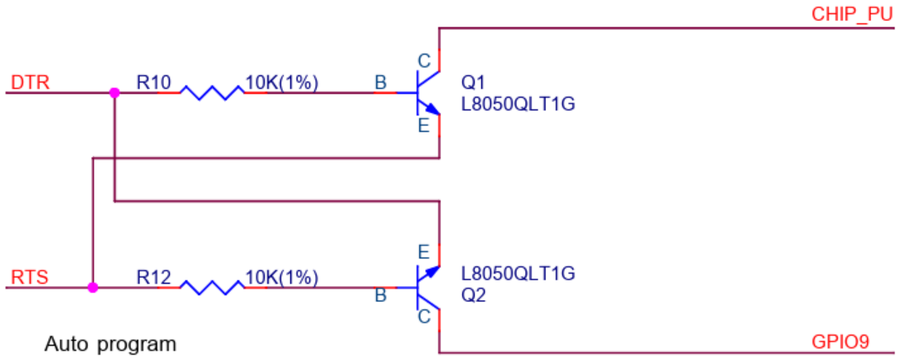
ESP32-C3-DevKitM-1 – Kit schematics (2/3)





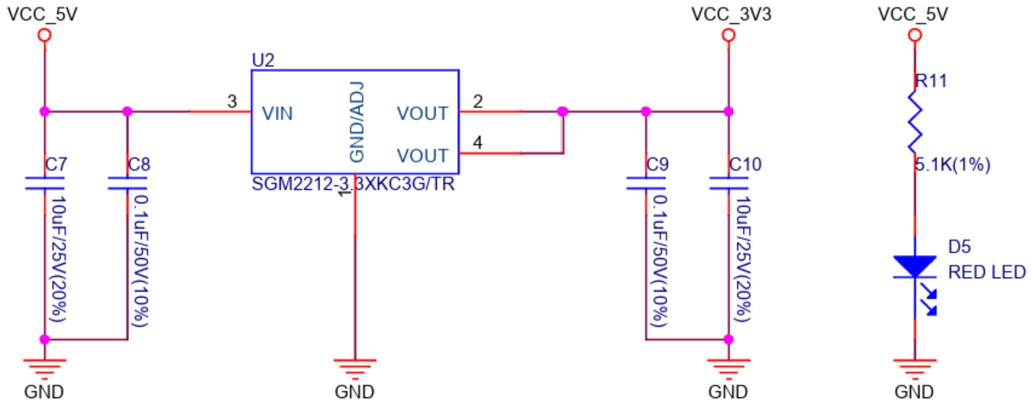


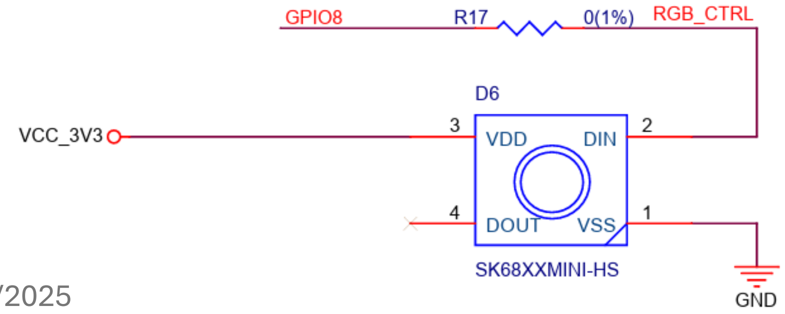
ESP32-C3-DevKitM-1 – Kit schematics (3/3)

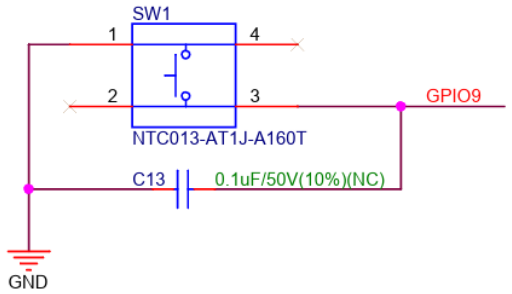


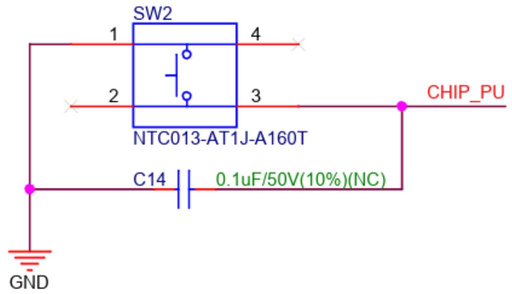
Auto program

DTR	RTS	-->EN	IO9
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0









ESP32-C3-MINI-1 - Module block diagram

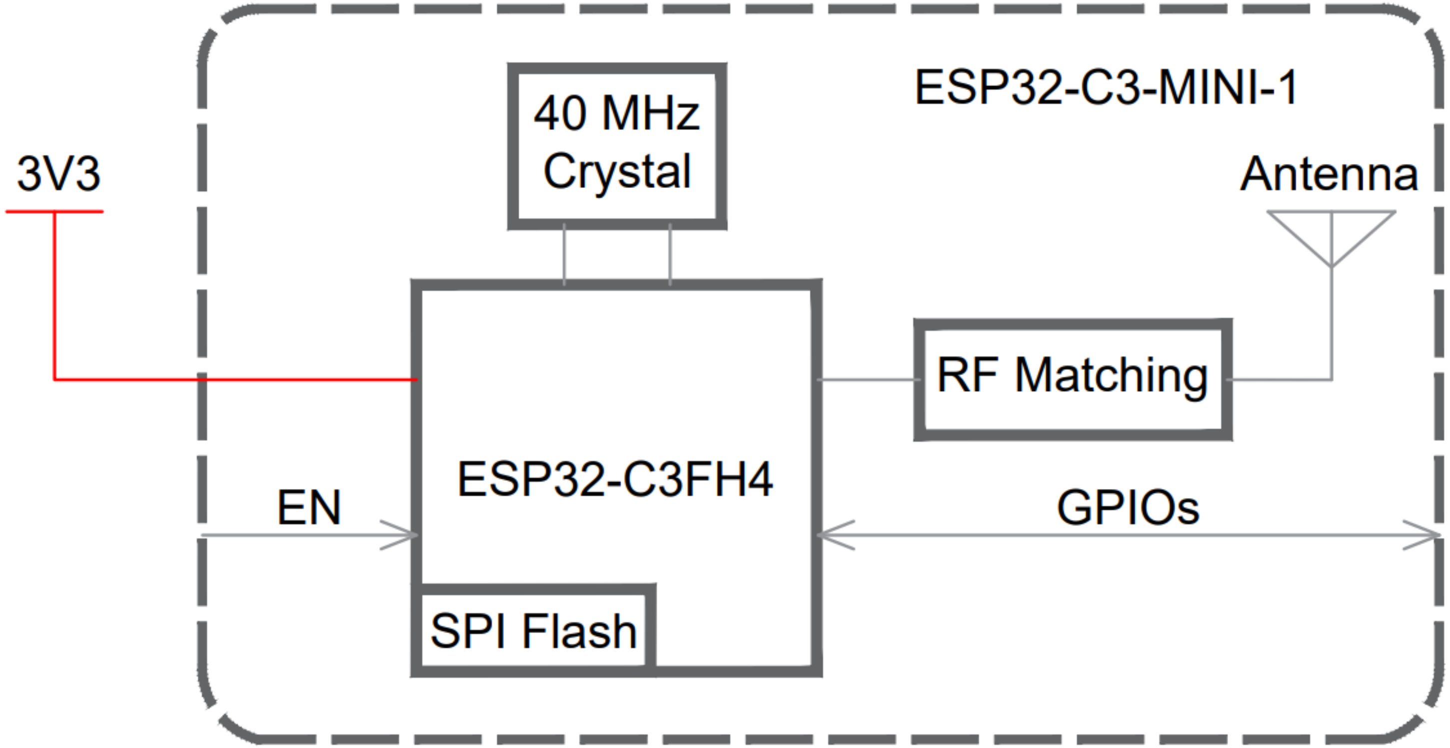
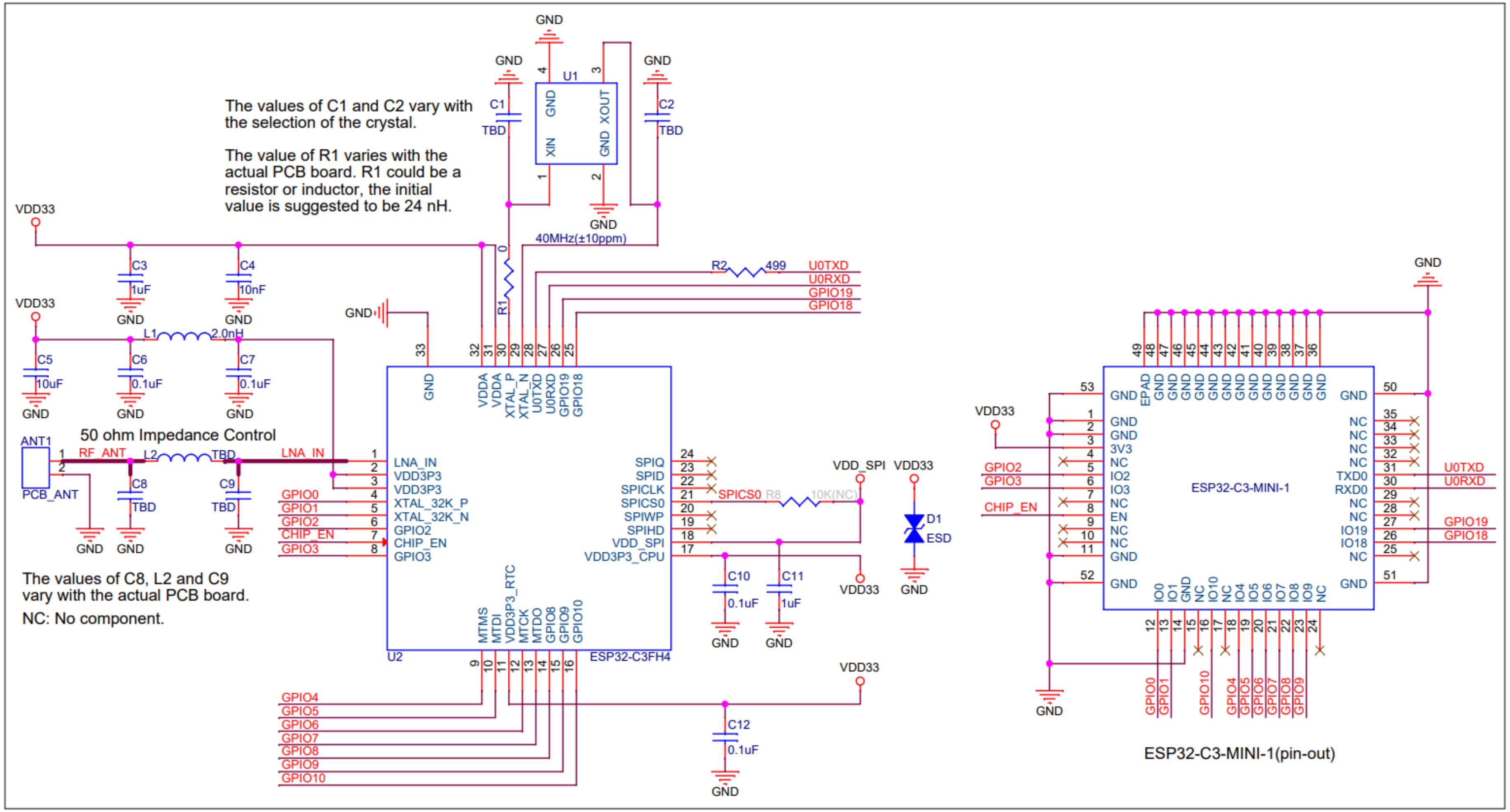
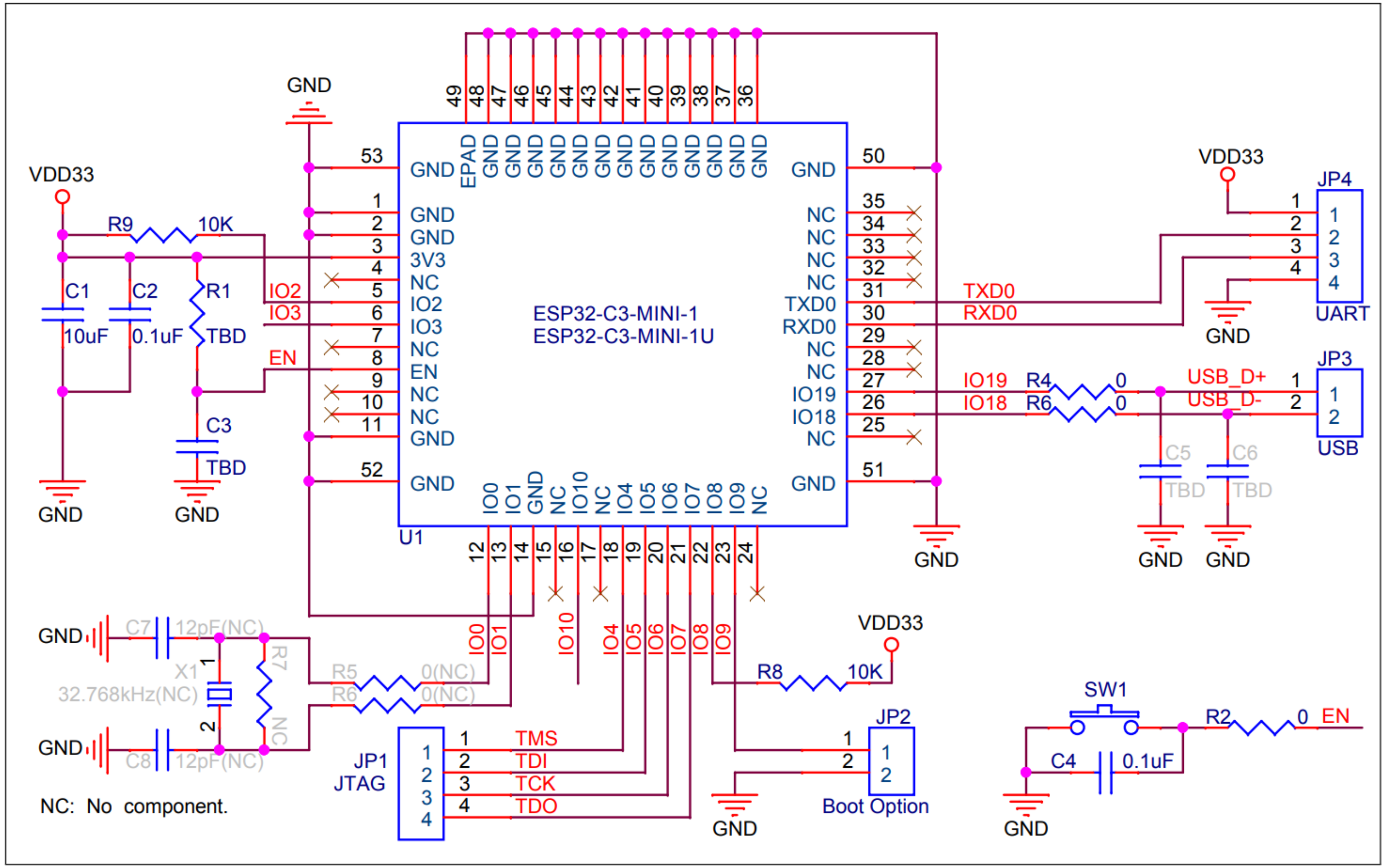


Figure 1: ESP32-C3-MINI-1 Block Diagram

ESP32-C3-MINI-1 - Module schematics (1/2)



ESP32-C3-MINI-1 - Module schematics (2/2)



ESP32-C3 SoC Components

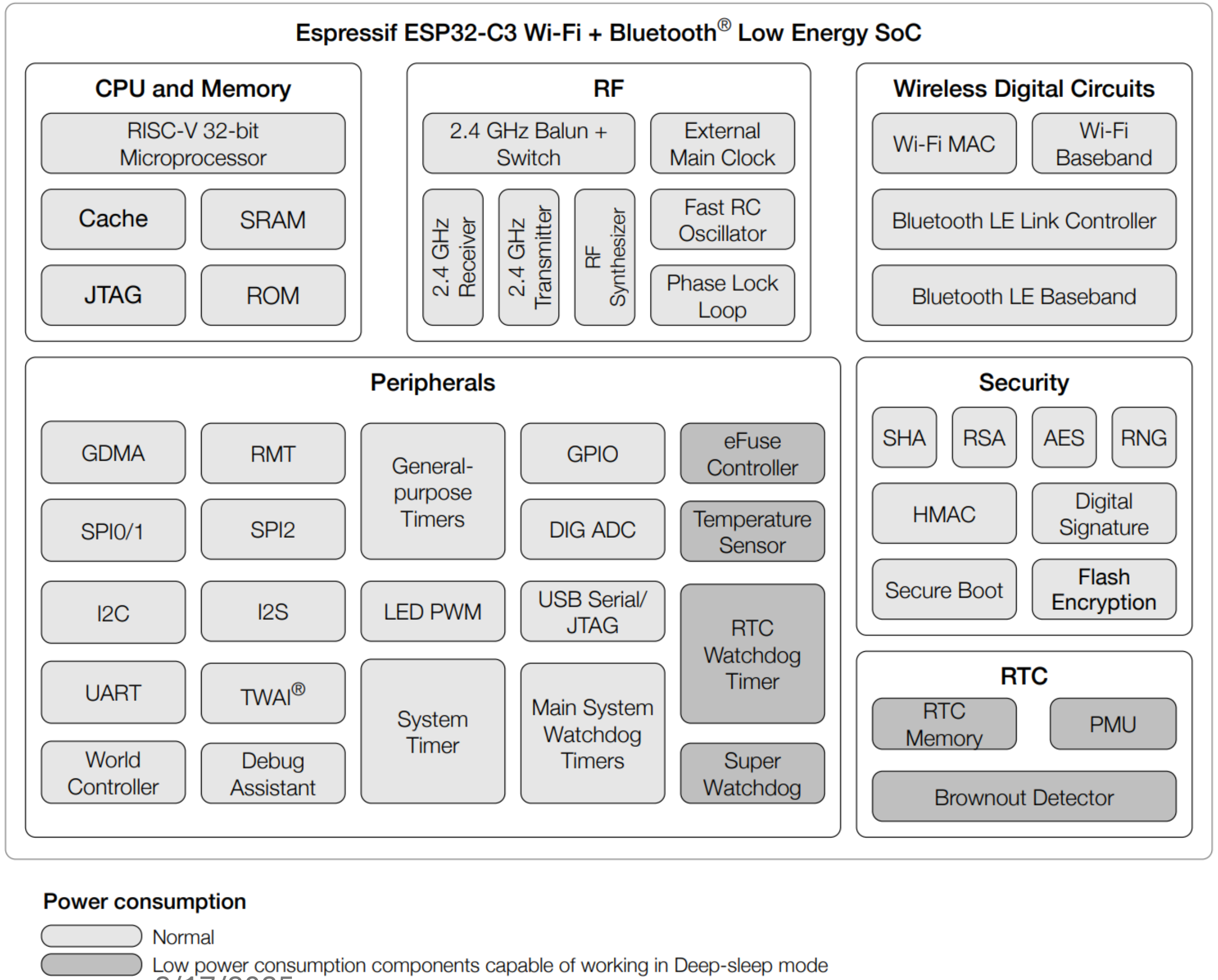


Table 4-1. Components and Power Domains

Power Domain / Power Mode	RTC	Digital	CPU	Optional Digital Periph	Wireless Digital Circuits	Analog	FOSC_CLK	XTAL_CLK	PLL	RF Circuits
Active	ON	ON	ON	ON	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON[1]	ON	ON	ON	ON	OFF[2]
Light-sleep	ON	ON	OFF[1]	ON[1]	OFF[1]	ON	OFF	OFF	OFF	OFF[2]
Deep-sleep	ON	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF

[1] Configurable, see the TRM.

[2] If Wireless Digital Circuits are on, RF circuits are periodically switched on when required by internal operation to keep active wireless connections running.

ESP32-C3 Memory Address Space

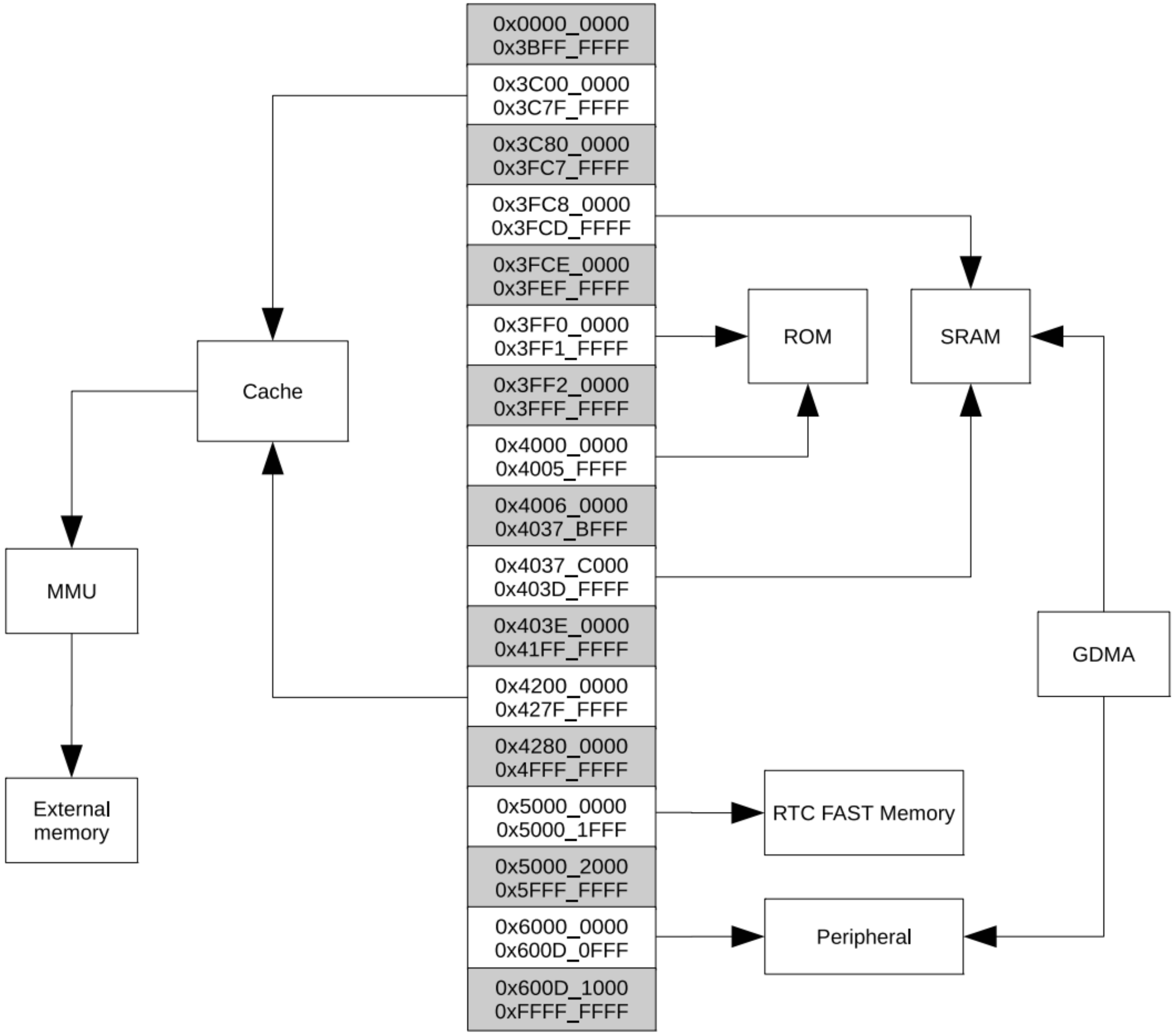
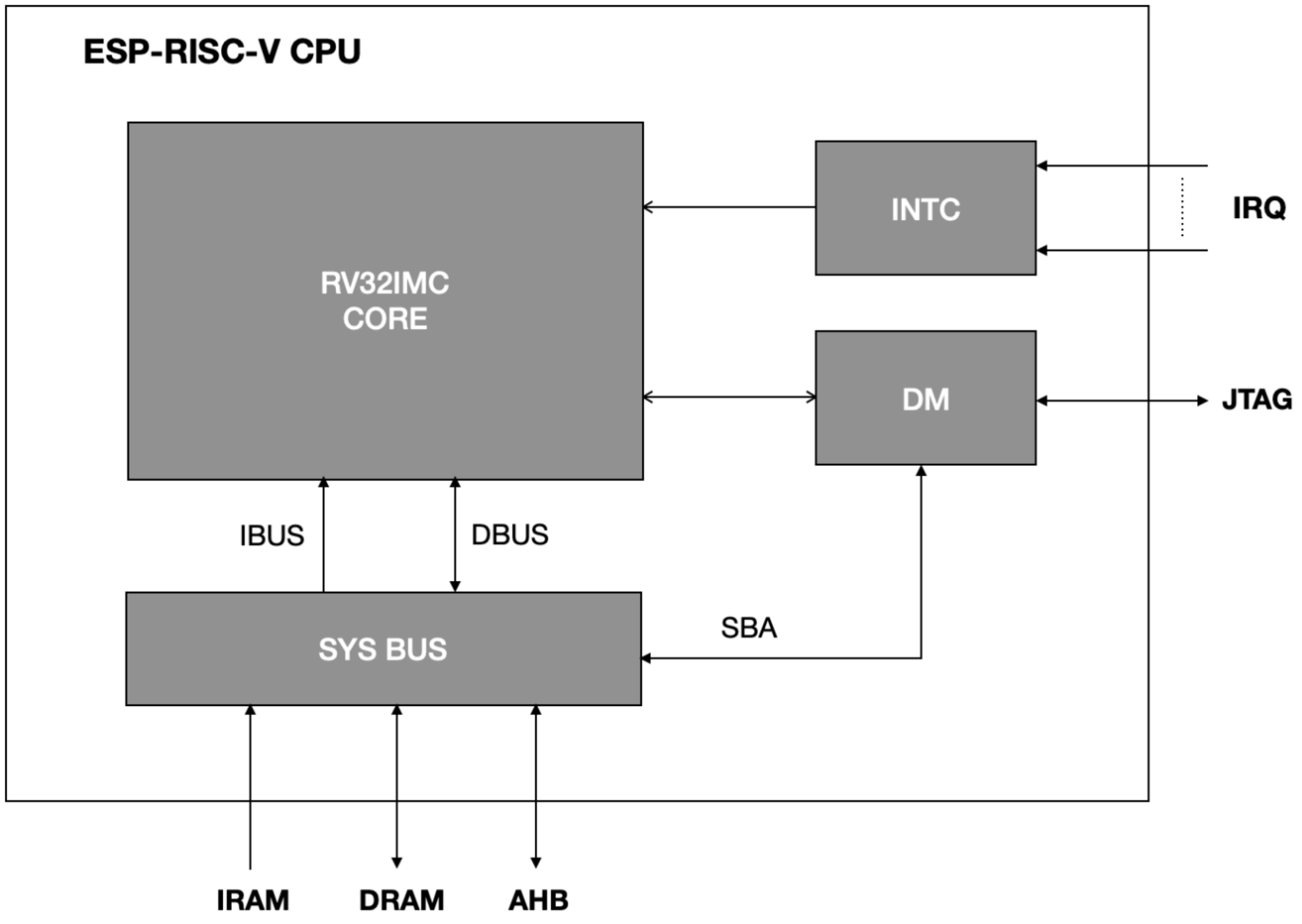


Figure 4-1. Address Mapping Structure

Table 1.3-1. CPU Address Map

Name	Description	Starting Address	Ending Address	Access
IRAM	Instruction Address Map	0x4000_0000	0x47FF_FFFF	R/W
DRAM	Data Address Map	0x3800_0000	0x3FFF_FFFF	R/W
DM	Debug Address Map	0x2000_0000	0x27FF_FFFF	R/W
AHB	AHB Address Map	*default	*default	R/W

ESP32-C3 RISC-V CPU core



ESP32 I/O Pin Structure

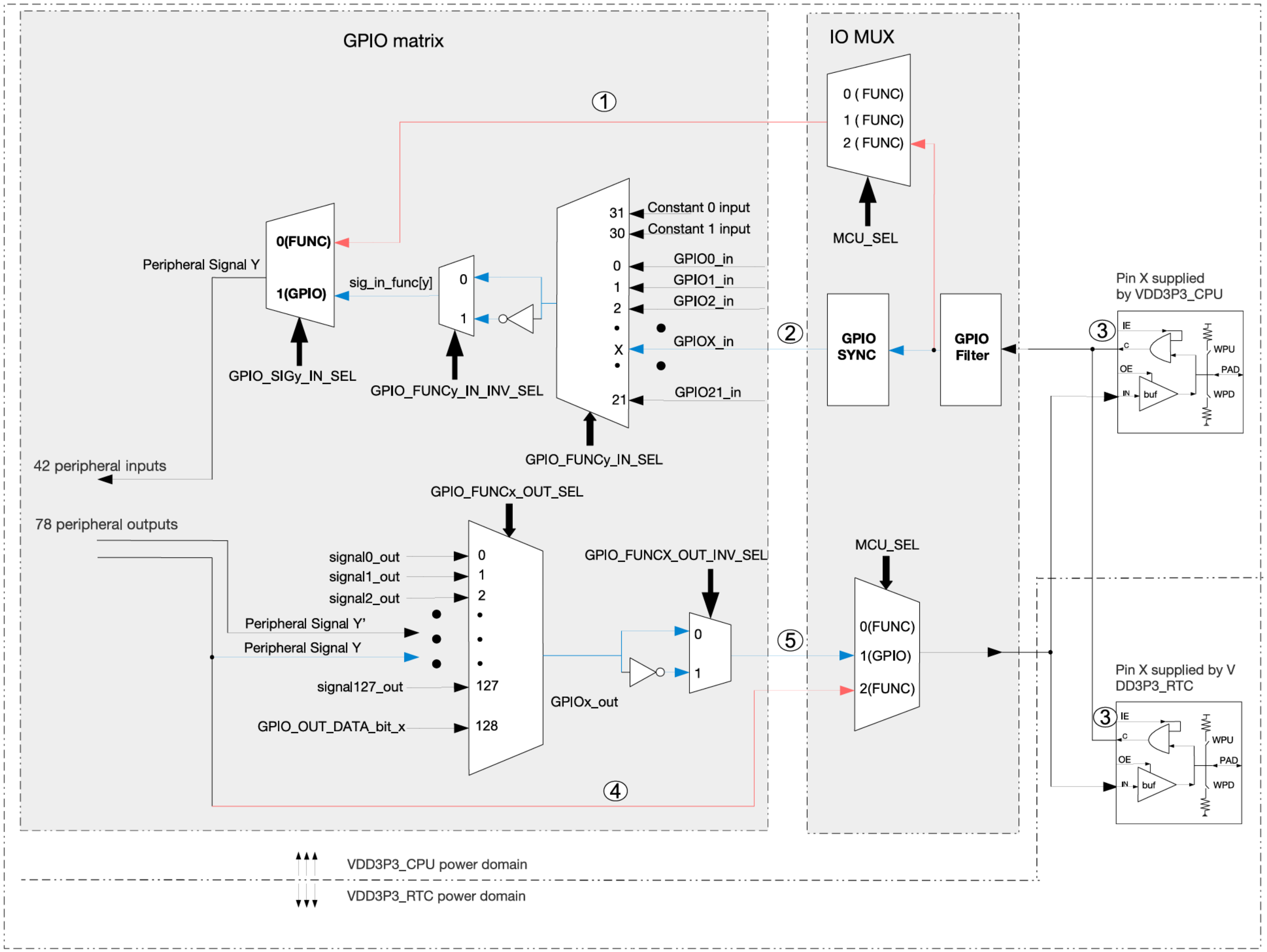
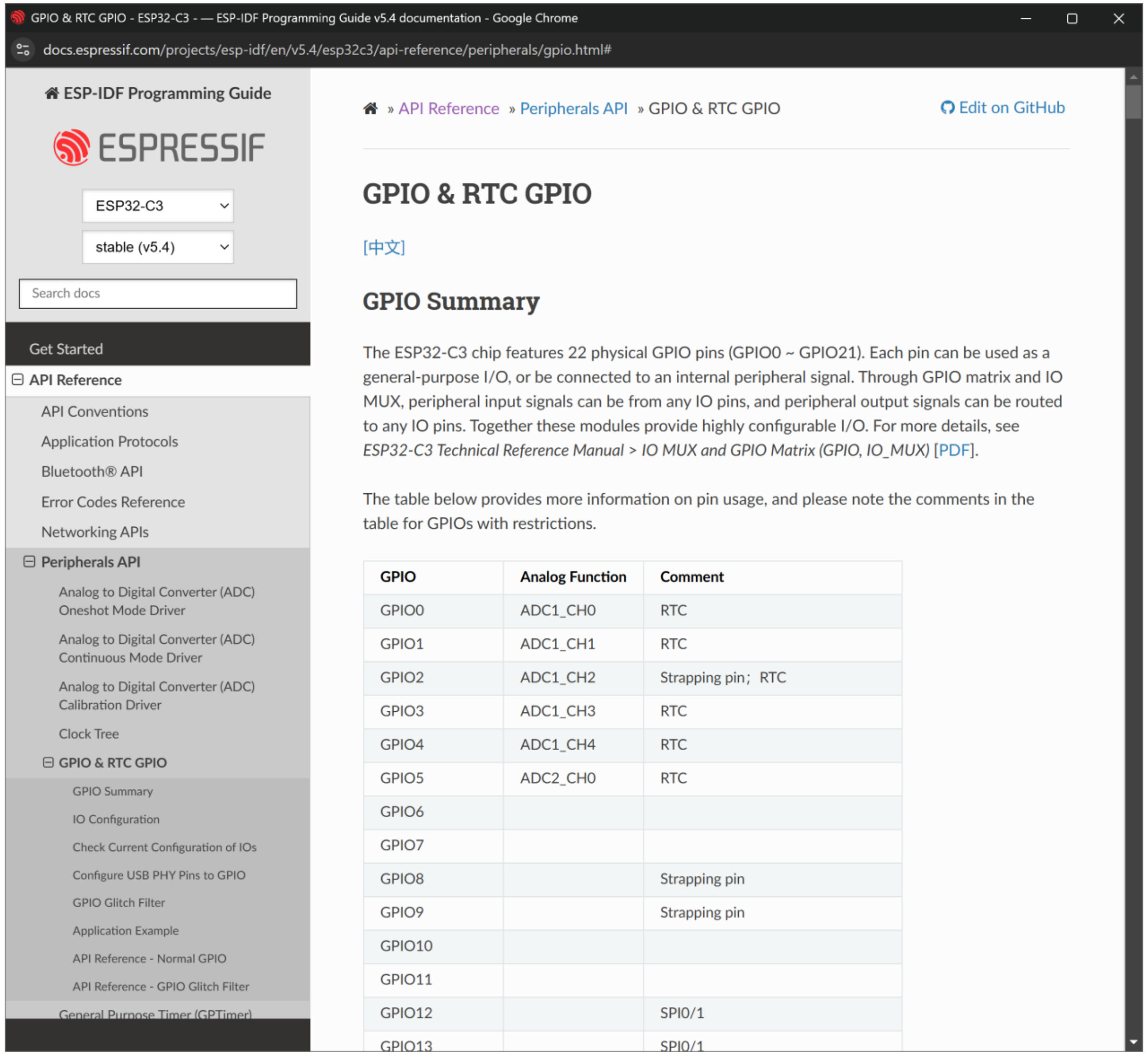


Figure 5.3-2. Architecture of IO MUX and GPIO Matrix

API Reference – General Purpose I/O



GPIO & RTC GPIO - ESP32-C3 - — ESP-IDF Programming Guide v5.4 documentation - Google Chrome

docs.espressif.com/projects/esp-idf/en/v5.4/esp32c3/api-reference/peripherals/gpio.html#

ESP-IDF Programming Guide

ESPRESSIF

ESP32-C3

stable (v5.4)

Search docs

- Get Started
- API Reference
  - API Conventions
  - Application Protocols
  - Bluetooth® API
  - Error Codes Reference
  - Networking APIs
  - Peripherals API
    - Analog to Digital Converter (ADC) Oneshot Mode Driver
    - Analog to Digital Converter (ADC) Continuous Mode Driver
    - Analog to Digital Converter (ADC) Calibration Driver
    - Clock Tree
    - GPIO & RTC GPIO
      - GPIO Summary
      - IO Configuration
      - Check Current Configuration of IOs
      - Configure USB PHY Pins to GPIO
      - GPIO Glitch Filter
      - Application Example
      - API Reference - Normal GPIO
      - API Reference - GPIO Glitch Filter
    - General Purpose Timer (GPTimer)

» API Reference » Peripherals API » GPIO & RTC GPIO

Edit on GitHub

GPIO & RTC GPIO

[中文]

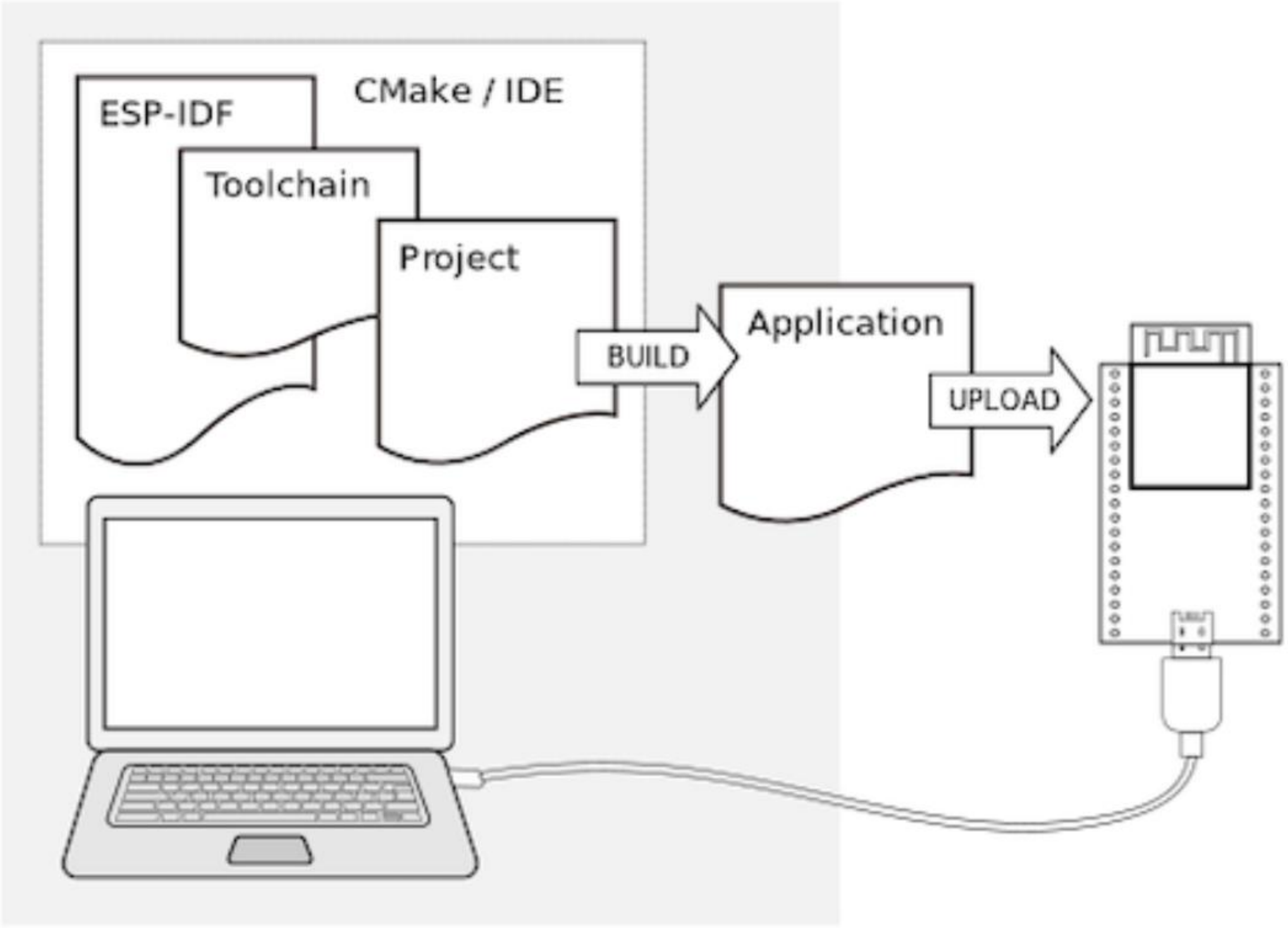
GPIO Summary

The ESP32-C3 chip features 22 physical GPIO pins (GPIO0 ~ GPIO21). Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Through GPIO matrix and IO MUX, peripheral input signals can be from any IO pins, and peripheral output signals can be routed to any IO pins. Together these modules provide highly configurable I/O. For more details, see *ESP32-C3 Technical Reference Manual > IO MUX and GPIO Matrix (GPIO, IO_MUX)* [PDF].

The table below provides more information on pin usage, and please note the comments in the table for GPIOs with restrictions.

GPIO	Analog Function	Comment
GPIO0	ADC1_CH0	RTC
GPIO1	ADC1_CH1	RTC
GPIO2	ADC1_CH2	Strapping pin； RTC
GPIO3	ADC1_CH3	RTC
GPIO4	ADC1_CH4	RTC
GPIO5	ADC2_CH0	RTC
GPIO6		
GPIO7		
GPIO8		Strapping pin
GPIO9		Strapping pin
GPIO10		
GPIO11		
GPIO12		SPI0/1
GPIO13		SPI0/1

ESP32-C3 Compilation Flow



ESP-IDF (command line tools)

Mostly used commands

```
idf.py set-target esp32c3

idf.py menuconfig

idf.py build

idf.py -p PORT flash

idf.py -p PORT flash monitor
```

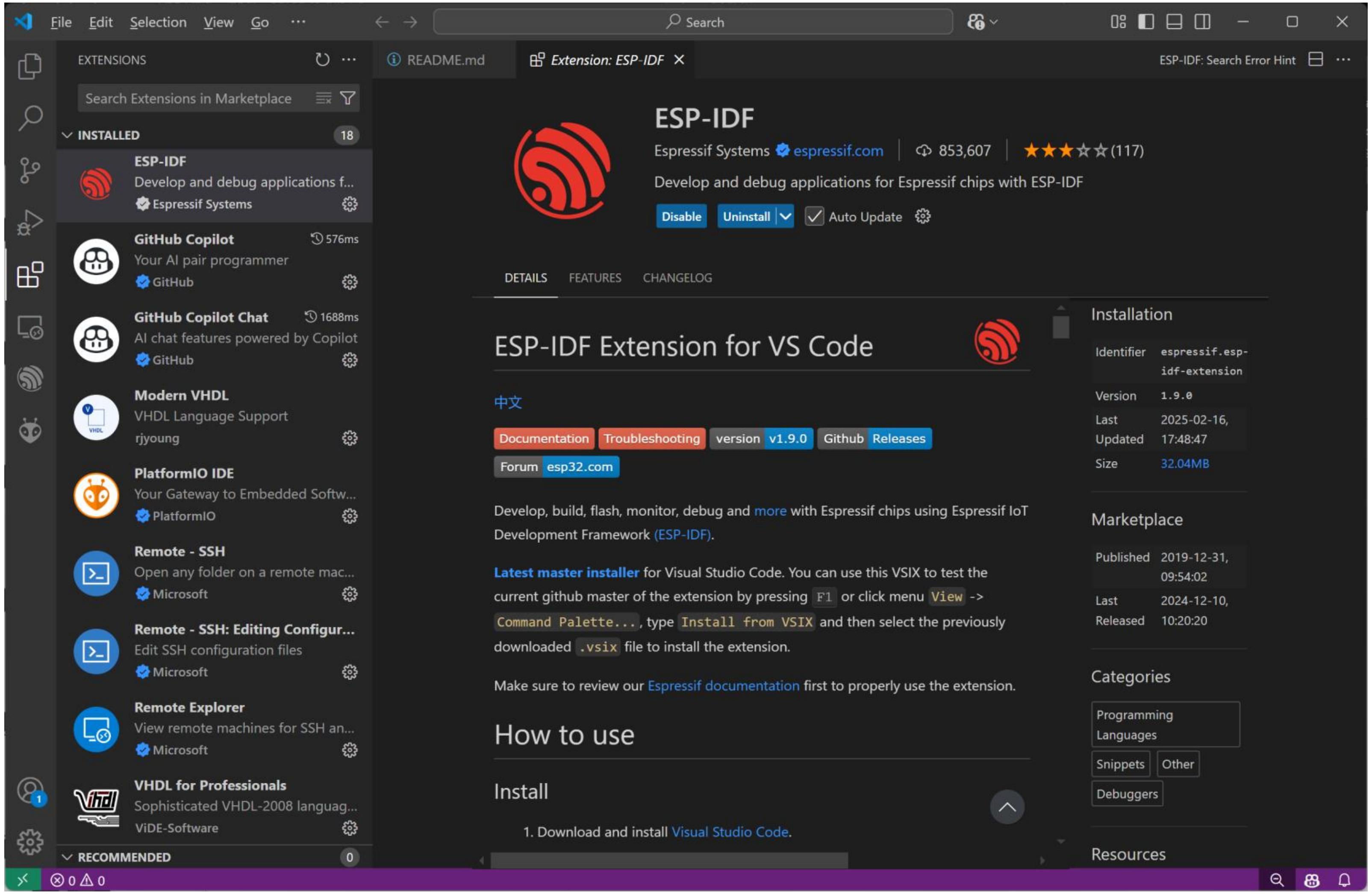
ESP-IDF (`idf.py menuconfig`)

```
ESP-IDF 5.4 PowerShell

(Top)
                    Espressif IoT Development Framework Configuration
    Build type  --->
    Bootloader config  --->
    Security features  --->
    Application manager  --->
    Boot ROM Behavior  --->
    Serial flasher config  --->
    Partition Table  --->
    Example Configuration  --->
    Compiler options  --->
    Component config  --->
[ ] Make experimental features visible

[Space/Enter] Toggle/enter   [ESC] Leave menu           [S] Save
[O] Load                     [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode    [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save)  [D] Save minimal config (advanced)
```

Visual Studio Code - based environment



What is the purpose of the CMake tool? (1/2)

CMake is an open-source tool designed to manage the build process of software in a platform-independent manner. It is primarily used for generating build files that can be used with different native build systems such as **Makefiles, Ninja, Visual Studio project files, Xcode project files**, and others.

Purpose of CMake:

1. **Cross-Platform Build Configuration**
   - CMake allows developers to write build scripts that work across different operating systems (Windows, macOS, Linux) and compilers (GCC, Clang, MSVC).
2. **Automatic Build System Generation**
   - Instead of manually writing complex Makefiles or project files, CMake generates them based on a simple CMake configuration file (`CMakeLists.txt`).
3. **Dependency Management**
   - It can find and configure external libraries, ensuring that all required dependencies are correctly located and linked.
4. **Out-of-Source Builds**
   - Supports building software in a separate directory from the source code, keeping the source clean and preventing conflicts.
5. **Integration with IDEs**
   - Works with popular Integrated Development Environments (IDEs) like Visual Studio, CLion, and Xcode.
6. **Build Customization**
   - Allows defining custom build options and configurations using `CMakeLists.txt` and `CMakeCache.txt`.
7. **Support for Multiple Build Types**
   - Easily configure different build types such as Debug, Release, RelWithDebInfo, and MinSizeRel.

What is the purpose of the CMake tool? (2/2)

Basic Workflow of CMake

1. **Write a `CMakeLists.txt` file** that defines the project, source files, dependencies, and build settings.
2. **Run `cmake`** to generate the build system files:

   css   Copy   Edit

   ```css
   cmake -S . -B build
   ```

3. **Use the generated build system** to compile the project:

   css   Copy   Edit

   ```css
   cmake --build build
   ```

CMake is widely used in modern C++ projects due to its flexibility, portability, and ease of integration with various tools and compilers.

What is the purpose of the Ninja tool? (1/2)

Purpose of the Ninja Build System

Ninja is a small, fast build system designed to efficiently handle incremental builds. It is primarily used to speed up the compilation of large projects by minimizing unnecessary work and maximizing parallelism.

Key Features and Purpose of Ninja:

1. **Optimized for Speed**
   - Ninja is significantly faster than traditional build systems like **Make** because it avoids unnecessary file checks and only rebuilds what is required.
2. **Incremental Builds**
   - It tracks dependencies efficiently, ensuring that only modified files are recompiled, reducing build times dramatically.
3. **Parallel Execution**
   - Ninja automatically maximizes CPU usage, making it much faster than `make -j` in large projects.
4. **Minimal Overhead**
   - The Ninja build system is lightweight and has fewer features than Make, focusing only on executing build commands as quickly as possible.
5. **Integration with CMake and Other Generators**
   - Unlike Make, Ninja does not have its own build configuration language. Instead, **CMake, GN (Google's build system), or Meson** are often used to generate Ninja build files.
6. **Cross-Platform Support**
   - Works on Windows, Linux, and macOS, making it a portable solution for fast builds.

Ninja vs. Other Build Systems

- **Make:** Ninja is significantly faster and better at handling dependencies.
- **CMake:** CMake is not a build system but a **build system generator**. It can generate Ninja files for fast compilation.
- **Bazel/Meson:** Ninja is simpler but does not include higher-level dependency management.

What is the purpose of the Ninja tool? (2/2)

Basic Usage of Ninja

1. **Generating Ninja Build Files using CMake**

css

```css
cmake -G Ninja -S . -B build
```

2. **Building the Project with Ninja**

mathematica

```mathematica
ninja -C build
```

or

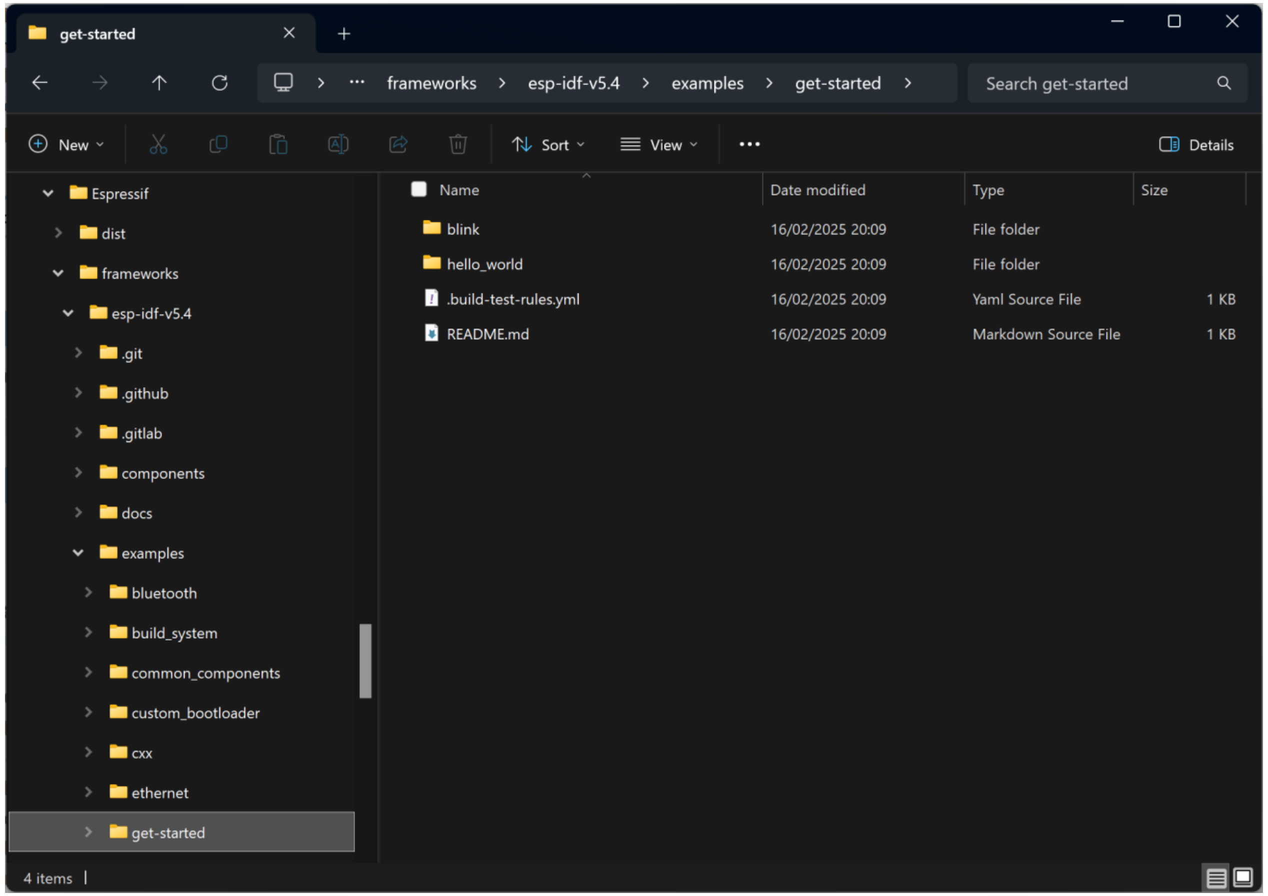
graphql

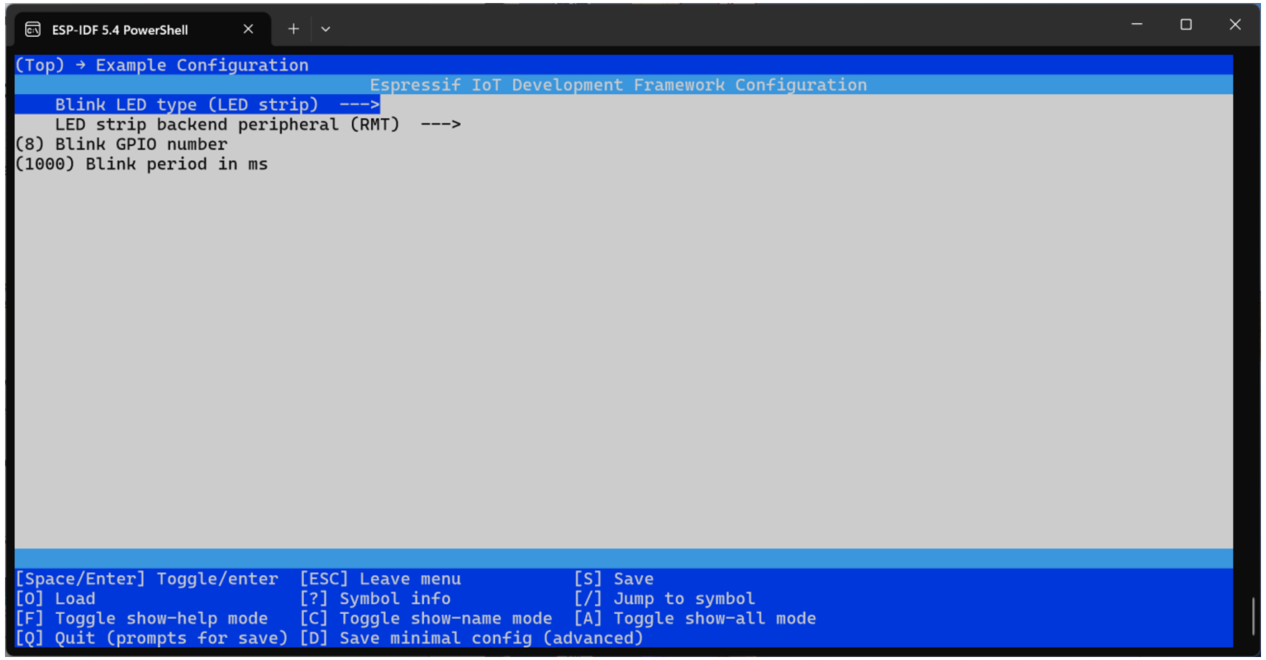
```graphql
cmake --build build -- -j8  # Using CMake with Ninja backend
```

Conclusion

Ninja is widely used in modern development workflows, especially in projects where **fast incremental builds** are essential, such as **Chromium, LLVM, and Android development**. If your project is large and relies on CMake, switching to Ninja can significantly improve build performance. 🚀

Get Started Examples (hello_world and blink)





Laboratory Assignment 1 – GPIO LED

- Create a copy of the “blink” project to your profile directory
- Connect a LED (+current limiting resistor) in the breadboard to a kit’s GPIO pin of your choice
- Configure the project settings accordingly
- Compile and test the project

Laboratory Assignment 2 – Read in / Write out

- Select a GPIO to be used as an input
- Create a copy of the previous project to your profile directory
- Develop a program that reads the GPIO input and writes the corresponding value to the GPIO output associated with the LED
- Compile and test the project

Laboratory Assignment 3 – Brightness control

- Create a copy of the previous project to your profile directory
- Develop a program that allows to set the brightness of the LED, based on a software-generated PWM signal with a varying duty-cycle controlled by the user
- Compile and test the project

Final Remarks

- You must be acquainted with the previous topics and complete the 3 lab assignments this week
- Always bring with you
  - Breadboard with the kit inserted and connected to the required components
  - USB-A to USB-micro cable
  - Cutting pliers
  - Wires (not jumpers!)
  - Oscilloscope probes (2 un.)