

ESPRESSIF
DevCon23

ESP-IDF Getting Started

Beginner's Guide to Key Concepts and Resources

Darian Leung

Who am I ?

Software Platforms Team

- 6 years
- ESP-IDF Development
- Core subsystems
- Protocols



Darian Leung
Senior Embedded Software Engineer



r/embedded • 4 mo. ago
by [\[REDACTED\]](#)

Join

...

What is the best source to learn ESP IDF ?



r/esp32 • 2 yr. ago
by [\[REDACTED\]](#)

Join

Need for advice on learning to program with ESP IDF



r/esp32 • 5 yr. ago
by [\[REDACTED\]](#)

Join

...

How long to learn ESP-IDF, and where to start?



r/esp32 • 1 yr. ago
by [\[REDACTED\]](#)



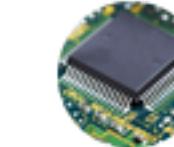
r/esp32 • 8 mo. ago
by [\[REDACTED\]](#)

Is ESP-IDF really worth it? Help with ESP-IDF transition.



r/esp32 • 4 yr. ago
by [\[REDACTED\]](#)

How to learn ESP-IDF?

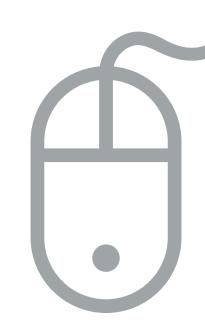


r/embedded • 1 yr. ago
by [\[REDACTED\]](#)

How to escape arduino hell?

CONTENTS

- Why ESP-IDF
- Project Workflow
- Programming Model
- Features & API
- Build System & Components
- More Resources



[Links](#) to Resources

ESP-IDF Programming Guide

ESPRESSIF

ESP32

stable (v5.1)

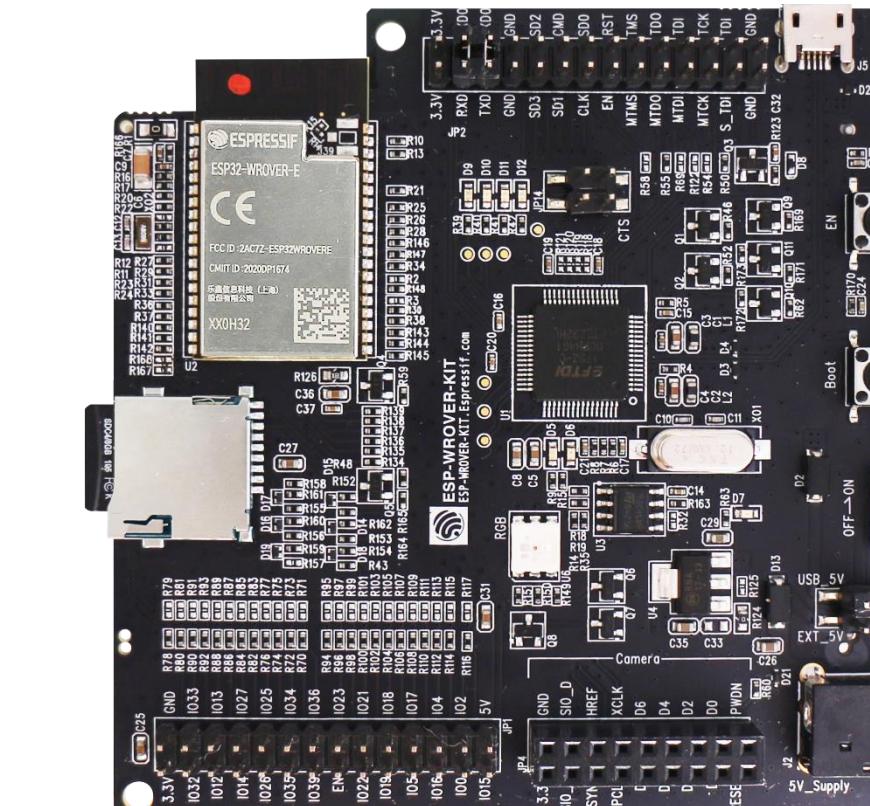
ESP-IDF v5.1



Ubuntu (Command Line)



Windows (Eclipse IDE)



ESP-WROVER-KIT
(ESP-32)

01

Why ESP-IDF?

What is ESP-IDF & why use it?

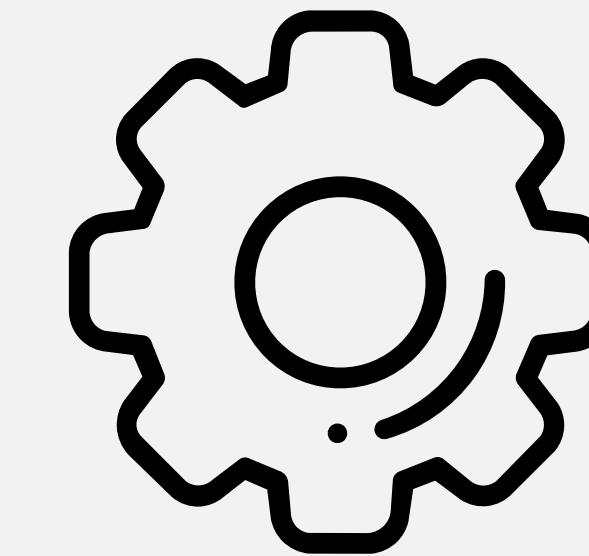
SDK Considerations



Language



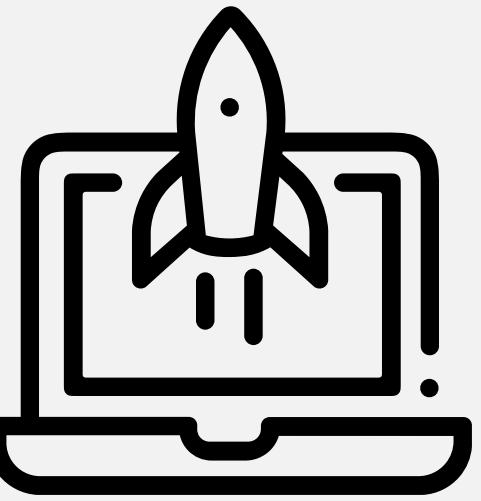
Features



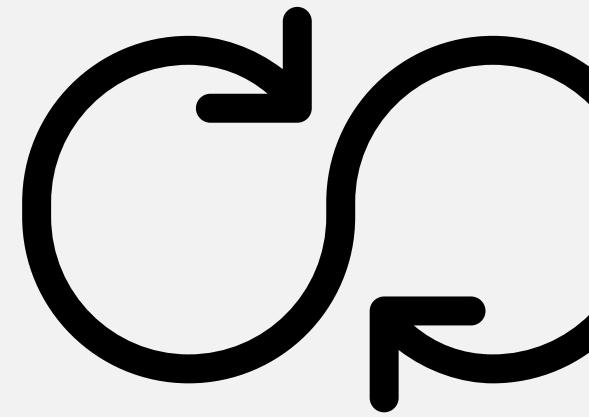
Configuration



Support &
Maintenance



Ease of Use



Debugging &
Testing

What is ESP-IDF?



[espressif/esp-idf](#) Public

Code Issues 1.4k Pull requests 122 Actions Projects Security 1 Insights

v5.1 17 branches 121 tags Go to file Code

File / Commit	Description	Date
.github	Merge branch 'contrib/github_pr_9273' into 'master'	last year
.gitlab	Merge branch 'refactor/driver_ut_to_test_app_v5.1' into 'release/v5.1'	3 months ago
components	Merge branch 'bugfix/mac_ext_order_for_c6_h2_v5.1' into 'release/v5.1'	2 months ago
docs	docs: update for esp32c6	3 months ago
examples	Merge branch 'bugfix/mac_ext_order_for_c6_h2_v5.1' into 'release/v5.1'	2 months ago
tools	build_template_app: Remove extra backslash	2 months ago
.editorconfig	refactor(editorconfig): Removed FreeRTOS tab rule	last year
.flake8	tinyusb: Use TinyUSB from component registry	10 months ago
.gitignore	feat(idf_monitor): move idf_monitor to separate repo	6 months ago
.gitlab-ci.yml	ci: rename target test jobs names	4 months ago
.gitmodules	esp_wifi: move coex part from esp_wifi to esp_coex	6 months ago
.mypy.ini	Add mypy check to pre-commit-config	2 years ago

About

Espressif IoT Development Framework. Official development framework for Espressif SoCs.

Readme Apache-2.0 license Security policy Activity 11k stars 491 watching 6.6k forks Report repository

Releases 98

ESP-IDF Release v5.1 Latest on Jun 30 + 97 releases

Why Use ESP-IDF?



Language

- C/C++
- Python tooling

Features

- Lots of components
- [Integrated networking stacks](#)
- [Component manager](#)

Configuration

- [Component level configuration](#)
- Supports Kconfig

Support & Maintenance

- Actively maintained
- Quickest to get new features/targets

Ease of Use

- Command line support
- [Optional IDE support](#)

Debugging & Testing

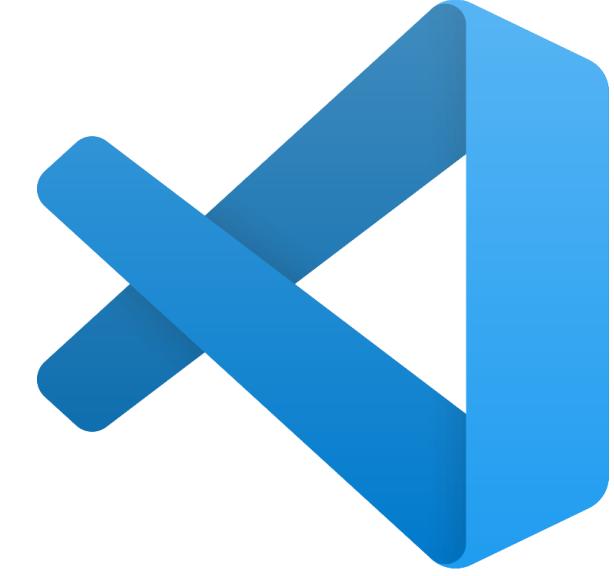
- [GDB Support](#)
- Lots of other debugging features
- [Unity & CMock](#)

02

Project Workflow

Basic workflow of an ESP-IDF Project

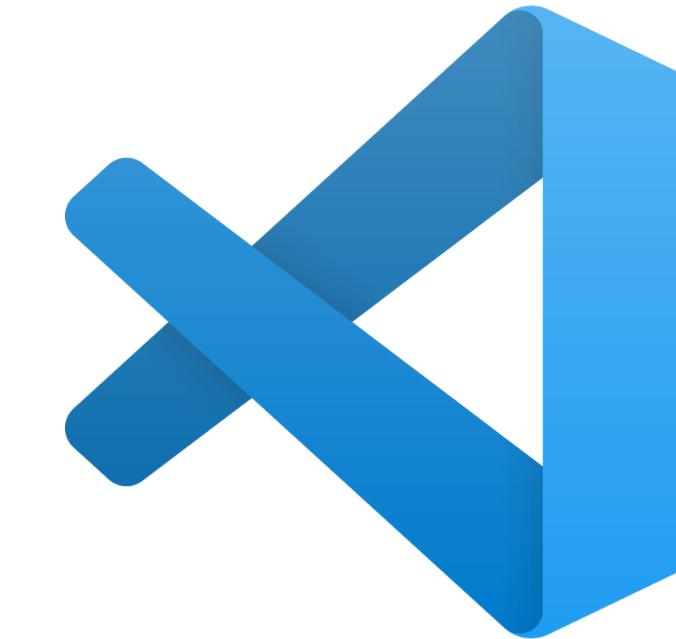
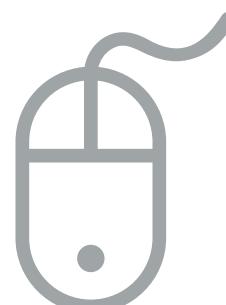
Select a Development Environment



Command Line & Editor



Eclipse Plugin & IDE



VSCODE Extension



Toolchain

GCC Compiler
Other GNU Tools
ESP-32 OpenOCD
...

Python Virtual Environment

ESP Tool
ESP eFuse
ESP Core Dump
...

System Packages

Python 3
Git
CMake
Ninja
...



`esp-idf/install.sh`

ESP-IDF

IDF Components
Examples
Tools
...

`esp-idf/export.sh`

Project

Application Code
Project Components
Project Configuration
Managed Components
...

IDF Component Manager

IDF Component Registry

ESP-IDF Installation

Command Line Install (Ubuntu)



```
user@pc:~$ sudo apt-get install git wget flex bison gperf  
python3 python3-venv cmake ninja-build ccache libffi-dev  
libssl-dev dfu-util libusb-1.0-0  
user@pc:~$ mkdir -p ~/esp  
user@pc:~$ cd ~/esp  
user@pc:~/esp$ git clone -b v5.1 -recursive  
https://github.com/espressif/esp-idf.git  
user@pc:~/esp$ cd ./esp-idf  
user@pc:~/esp/esp-idf$ ./install.sh esp32  
user@pc:~/esp/esp-idf$ . export.sh  
user@pc:~/esp/esp-idf$ idf.py -help
```

```
user@pc:~/esp/esp-idf$ idf.py -help
```

Usage: idf.py [OPTIONS] COMMAND1 [ARGS] ... [COMMAND2 [ARGS] ...] ...

ESP-IDF CLI build management tool. For commands that are not known to idf.py an attempt to execute it as a build system target will be made. Selected target: None

Options:

--version Show IDF version and exit.

...

-v, --verbose Verbose build output.

-p, --port TEXT Serial port.

--help Show this message and exit.

Commands:

...

all Aliases: build. Build the project.

clean Delete build output files from the build directory.

create-project Create a new project.

flash Flash the project.

fullclean Delete the entire build directory contents.

menuconfig Run "menuconfig" project configuration tool.

monitor Display serial output.

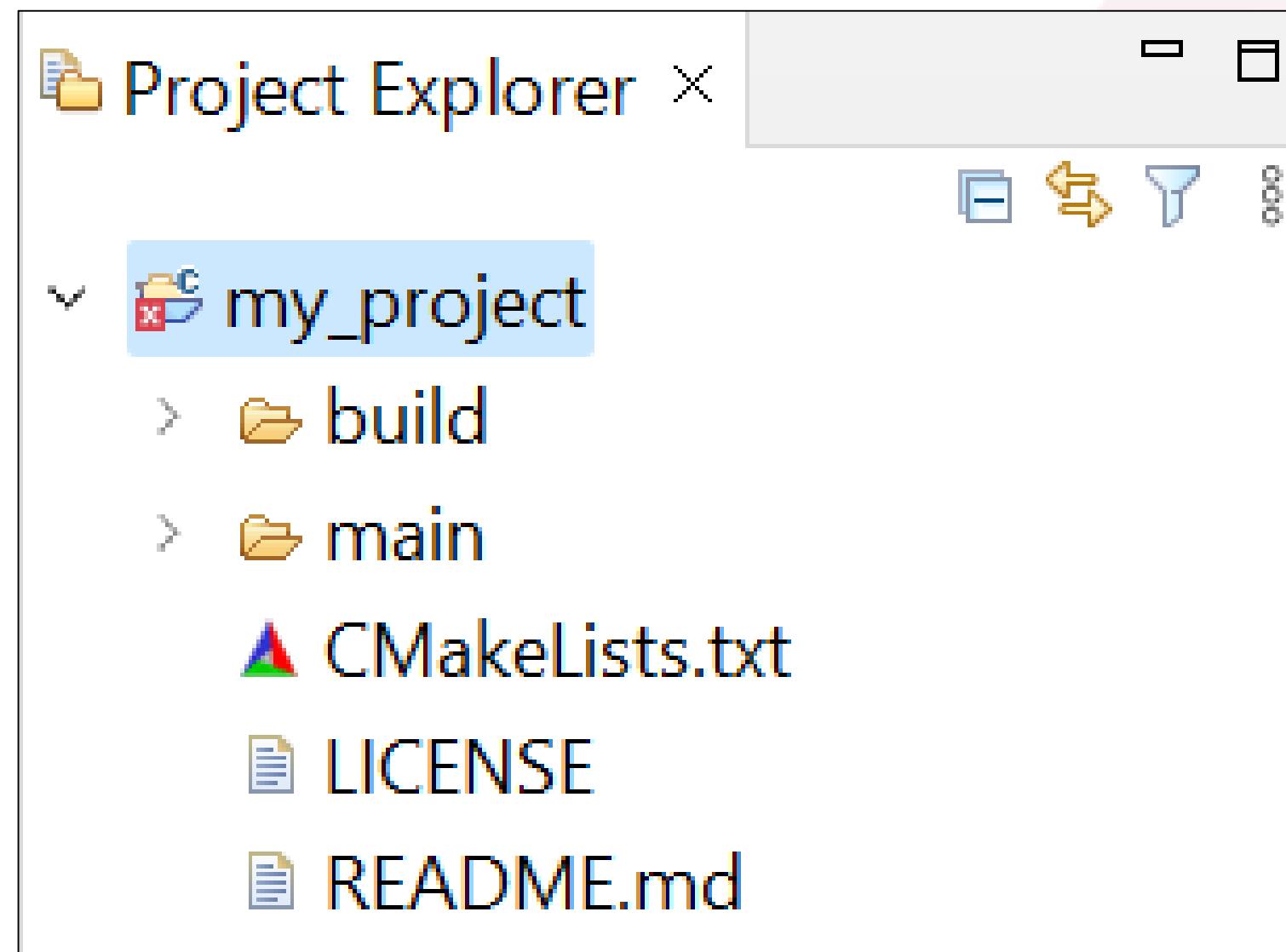
set-target Set the chip target to build.

Eclipse IDE Install (Windows)



Project Creation

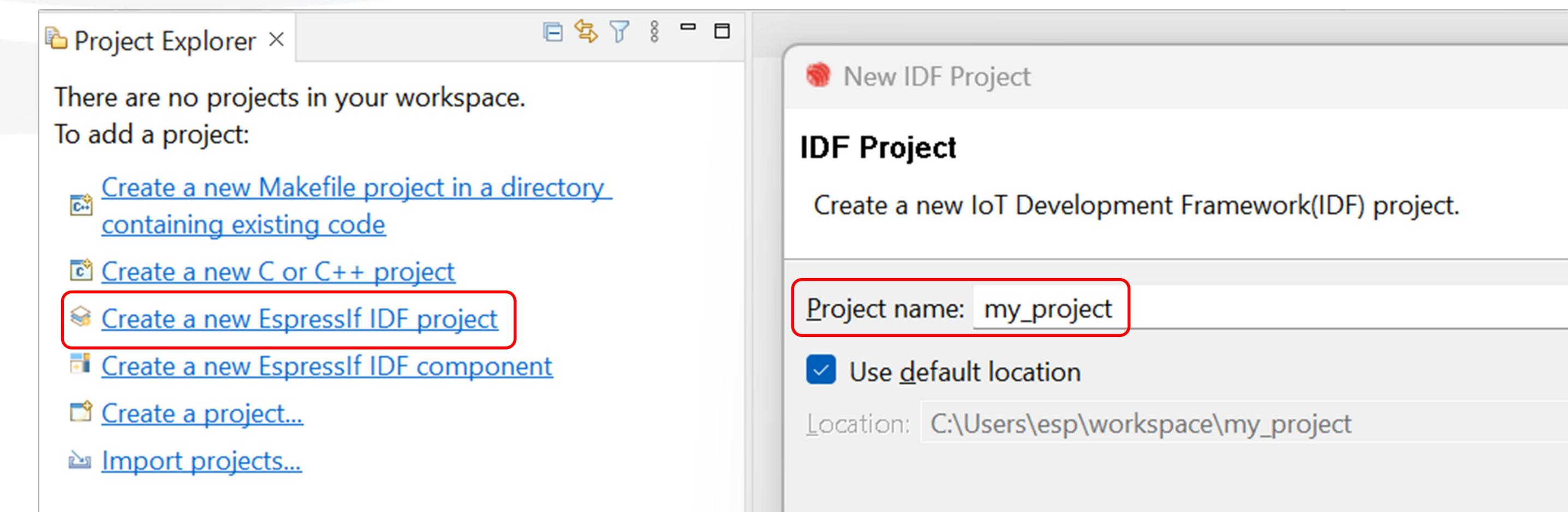
- my_project/
 - main/
 - CMakeLists.txt
 - my_project.c
 - CMakeLists.txt



Command Line

```
user@pc:~/esp$ idf.py create-project
my_project
user@pc:~/esp$ cd ./create-project
user@pc:~/esp/my_project$
```

Eclipse IDE



```
- my_project
  - main/
    - CMakeLists.txt
    - my_project.c
  - CMakeLists.txt
```

```
# In main/CMakeLists.txt
idf_component_register(
  SRCS "my_project.c"
  INCLUDE_DIRS ".")
```

```
# In CMakeLists.txt
cmake_minimum_required(VERSION 3.16)

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(my_project)
```

```
// In main/my_project.c
#include <stdio.h>

void app_main(void)
{
    printf("Hello World\n");
}
```

Project Workflow (Overview)

01 Configure

- Set configuration options from each component
- Generate ***sdkconfig*** file

02 Build

- Compile source files from all components.
- Link and generate **project binary**

03 Flash

- Reset the ESP-32 into **Firmware Download mode**
- Write the binary into **Flash memory**

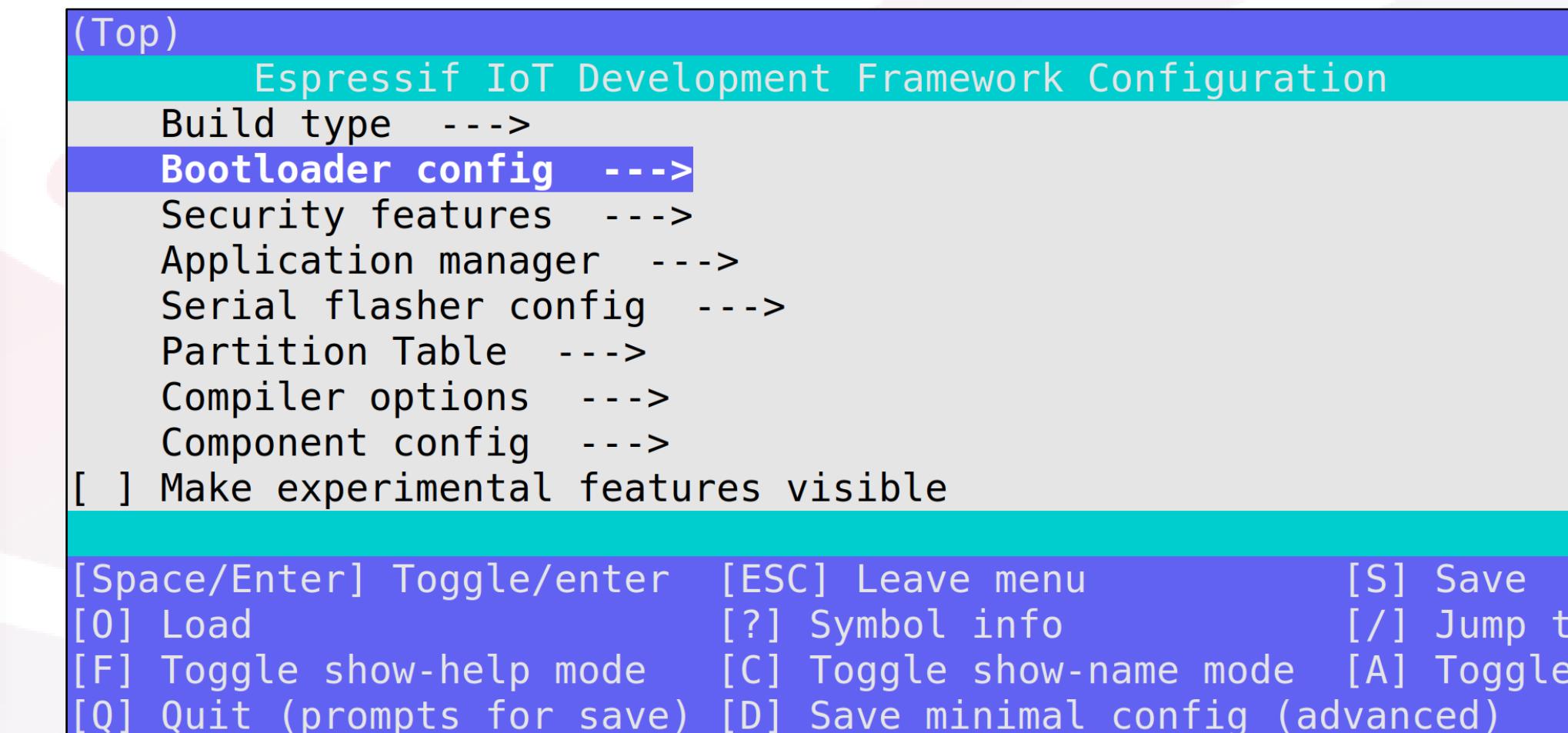
04 Monitor

- Reset the ESP-32 into **execution mode**
- Monitor the log output using **idf.py monitor**

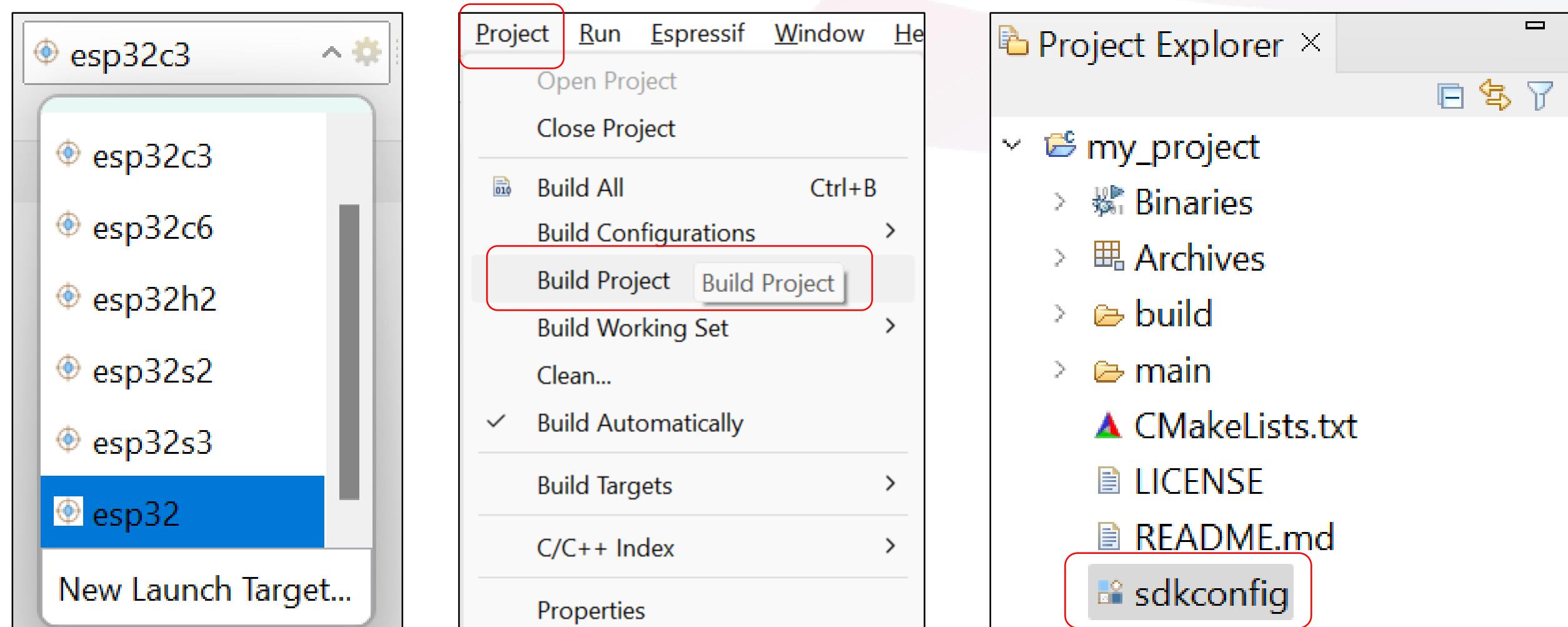
Project Workflow (Configuration)

Command Line

```
user@pc:~/esp/my_project$ idf.py
set-target esp32
user@pc:~/esp/my_project$ idf.py
menuconfig
```



Eclipse IDE



The screenshot shows the Eclipse IDE interface with the following components:

- Project View:** Shows available targets: esp32c3, esp32c6, esp32h2, esp32s2, esp32s3, esp32c3, and esp32. The esp32 target is selected.
- Run Menu:** Shows options like Open Project, Close Project, Build All, Build Configurations, Build Project (highlighted), Build Working Set, Clean..., Build Automatically, Build Targets, C/C++ Index, and Properties.
- Project Explorer:** Shows the project structure for 'my_project' including Binaries, Archives, build, main, CMakeLists.txt, LICENSE, README.md, and sdkconfig (highlighted).
- SDK Configuration:** A detailed configuration view for the esp32 target, listing various hardware components and their configuration parameters such as configTICK_RATE_HZ, configCHECK_FOR_STACK_OVERFLOW, configNUM_THREAD_LOCAL_STORAGE_POINTERS, configMINIMAL_STACK_SIZE, configUSE_IDLE_HOOK, configMAX_TASK_NAME_LEN, configENABLE_BACKWARD_COMPATIBILITY, configTIMER_TASK_PRIORITY, configTIMER_TASK_STACK_DEPTH, configTIMER_QUEUE_LENGTH, configQUEUE_REGISTRY_SIZE, configUSE_TRACE_FACILITY, and configGENERATE_RUN_TIME_STATS.

Project Workflow (Configuration) (2)

Generated Files

- my_project
 - main/
 - CMakeLists.txt
 - my_project.c
 - CMakeLists.txt
 - build/
 - sdkconfig

- build stores all build files
- *sdkconfig* stores generated configuration

sdkconfig

```
...  
#  
# Wi-Fi  
#  
CONFIG_ESP_WIFI_ENABLED=y  
CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM=10  
...  
# end of Wi-Fi  
...  
#  
# FreeRTOS  
#  
CONFIG_FREERTOS_HZ=100  
# CONFIG_FREERTOS_UNICORE is not set  
...  
# end of FreeRTOS
```

```
Executing action: all (aliases: build)
Running ninja in directory /home/User/esp/my_project/build
[0/1] Re-running CMake...
-- Project sdkconfig file
/home/User/esp/my_project/sdkconfig
...
Partition table binary generated.
...
[57/103] Building C object esp-
idf/bootloader_support/.../bootloader_init.c.obj
...
Successfully created esp32 image.
Generated
/home/User/esp/my_project/my_project/build/bootloader/boot
loader.bin
...
[370/884] Building C object esp-
idf/freertos/.../tasks.c.obj
...
[708/884] Building C object esp-idf/driver/.../i2c.c.obj
...
Successfully created esp32 image.
Generated /home/User/esp/my_project/build/my_project.bin
...
Project build complete. To flash, run this command:
```

Command Line

```
user@pc:~/esp/my_project$  
idf.py all
```

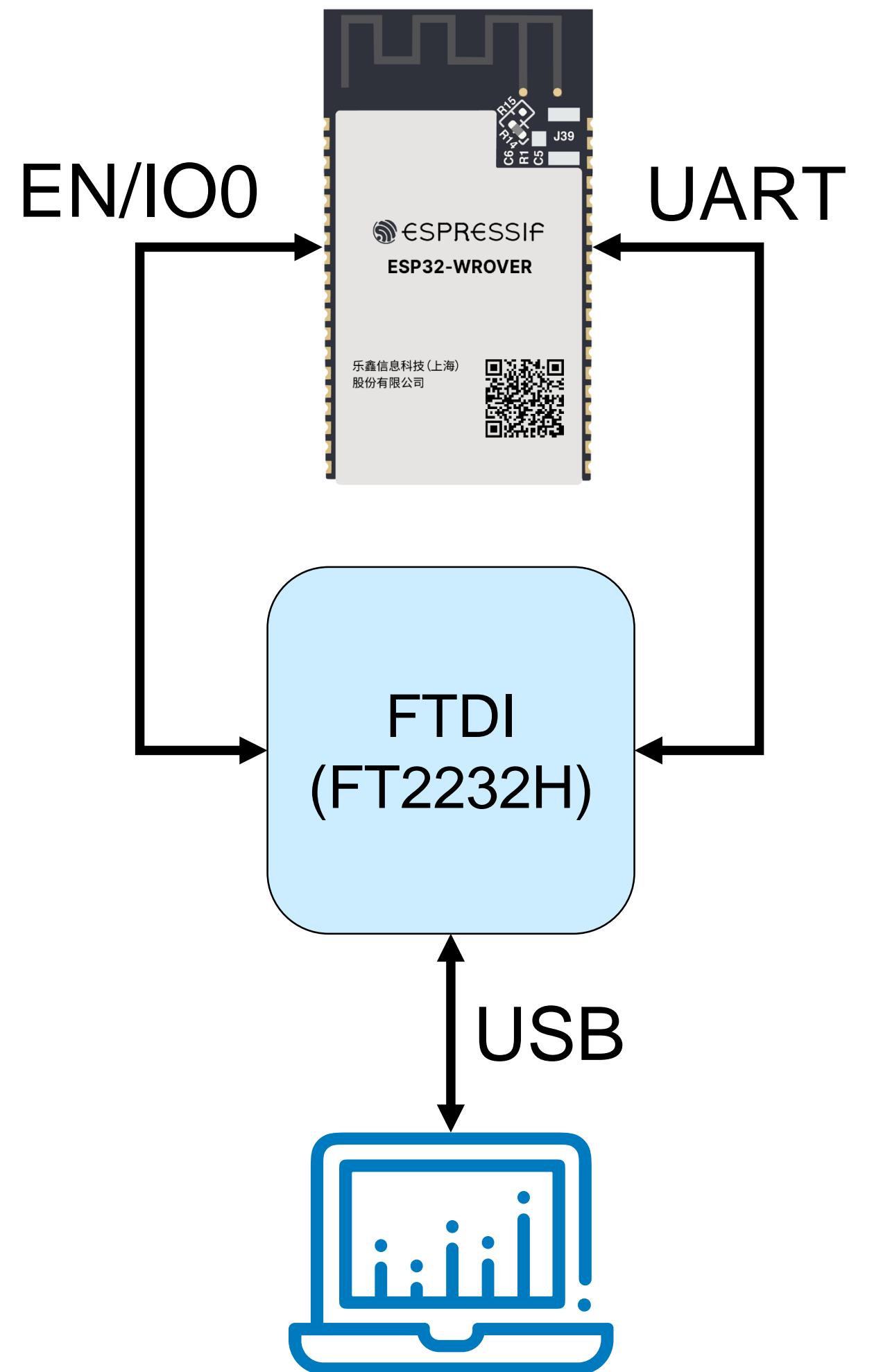
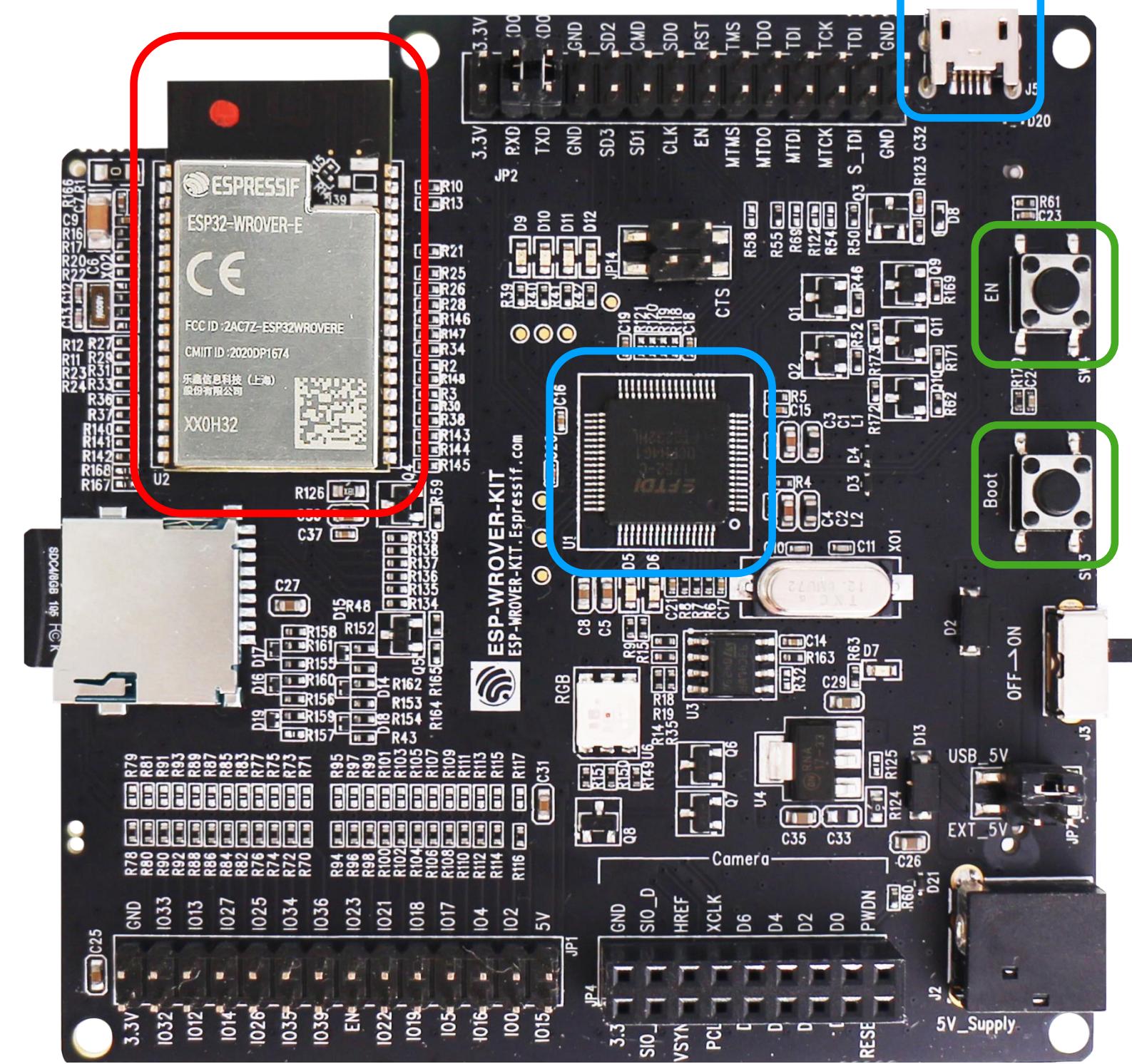
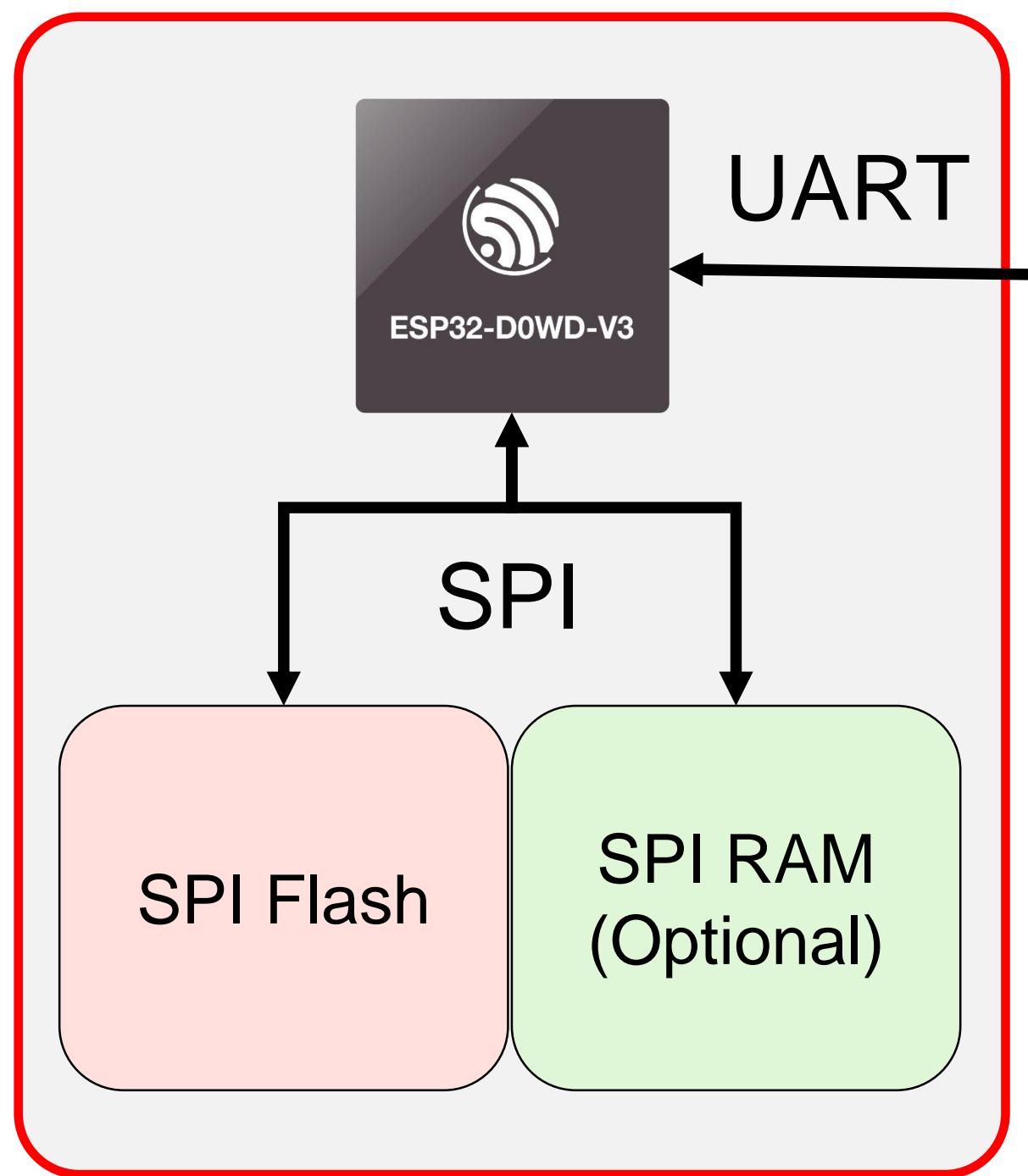
Eclipse IDE



(or CTRL + B)

Project Workflow (Build)

Hardware Overview



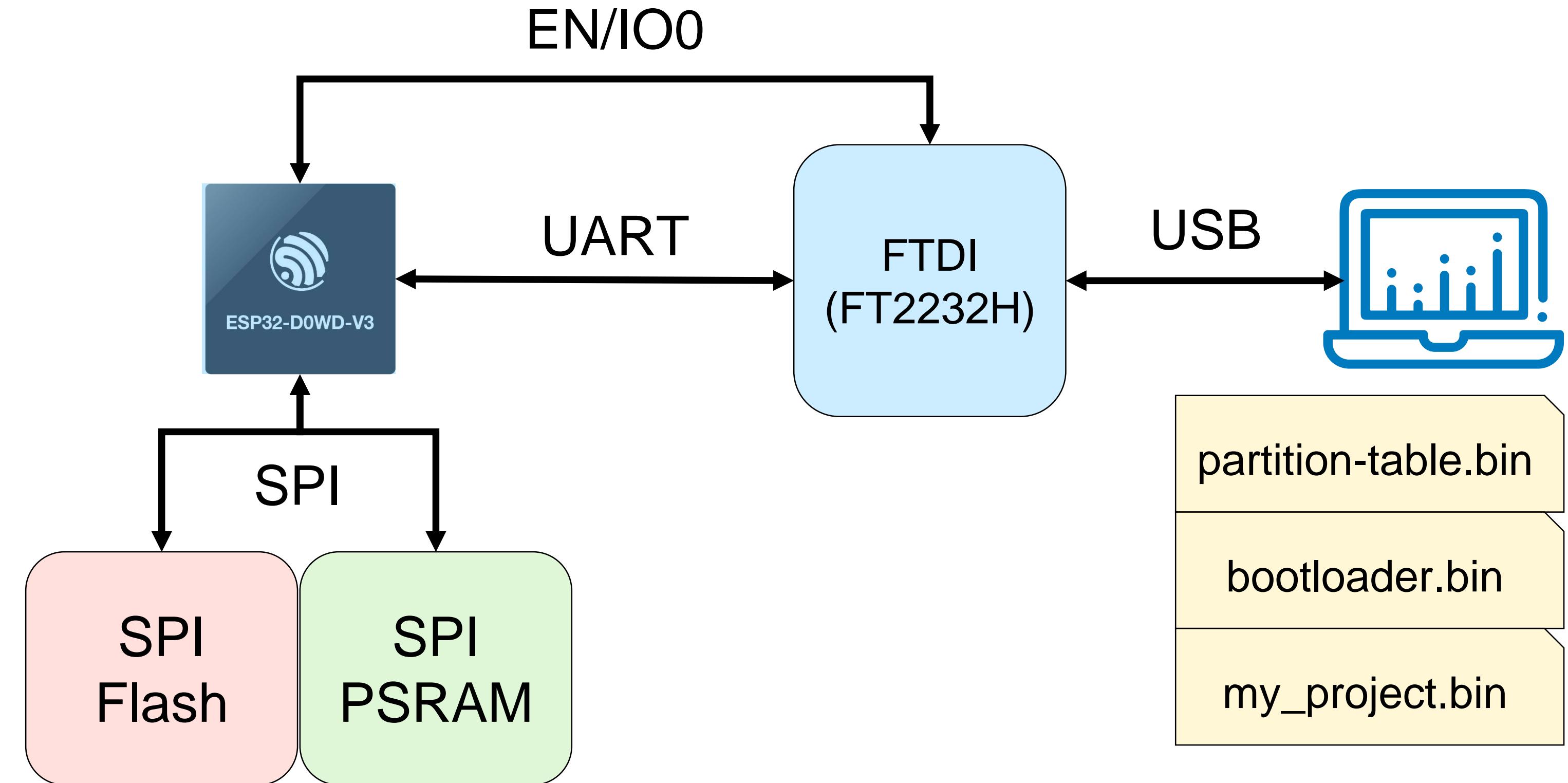
Project Workflow (Flash)

Download Mode

Reset ESP-32 while holding IO0 is low

Flashing Binaries

Write binaries to SPI Flash via ESP-32



Project Workflow (Flash) (2)

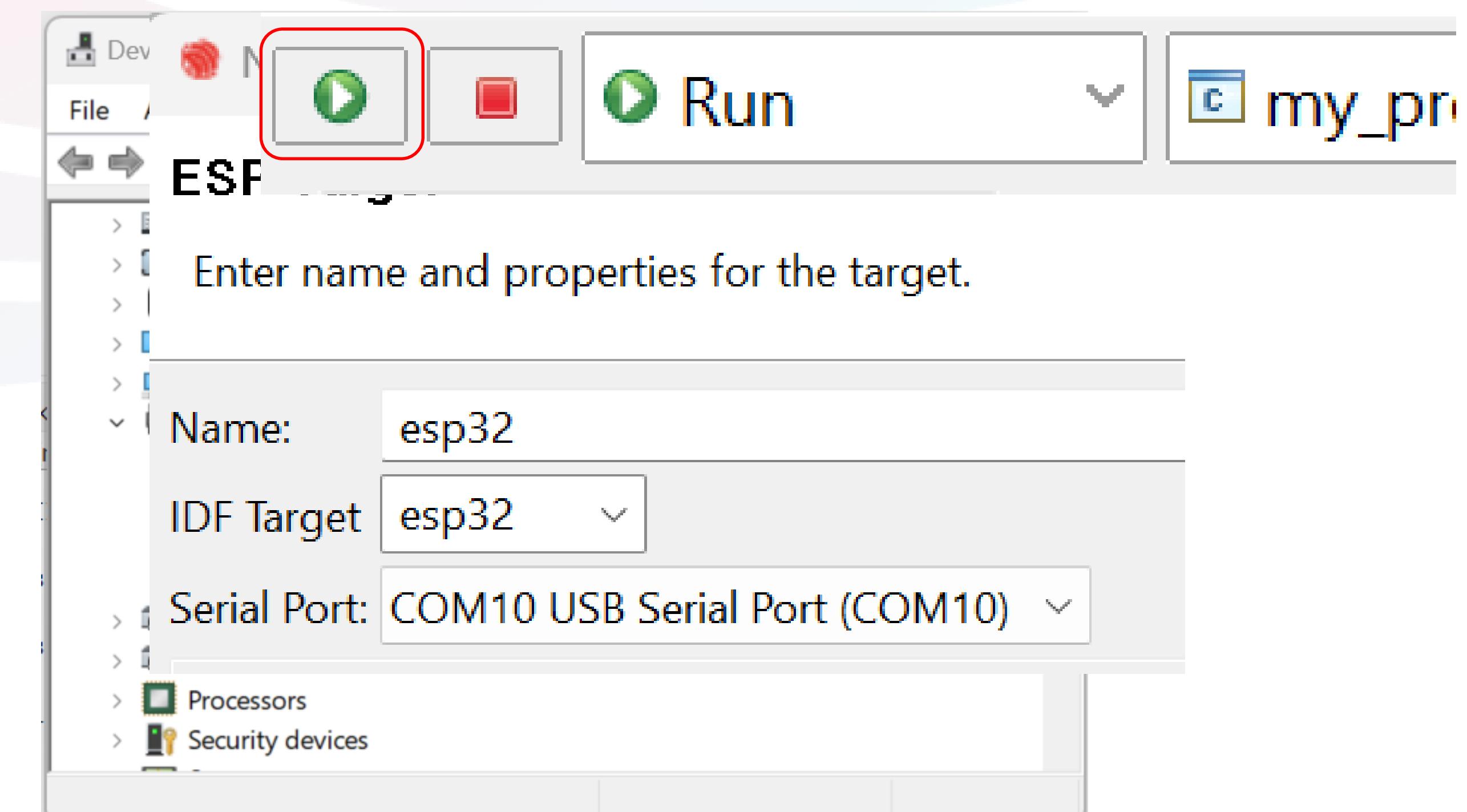
Command Line

- Find correct serial port
- Call **idf.py flash**

```
user@pc:~/esp/my_project$  
ls /dev/ttyUSB*  
  
user@pc:~/esp/my_project$  
idf.py -p /dev/ttyUSB1  
flash
```

Eclipse IDE

- Find correct serial port (via Device Manager)
- Create a new launch target
- Flash via **Run** button



```
user@pc:~/esp/esp-idf$ idf.py -p dev/ttyUSB1 flash  
Executing action: flash  
...  
esptool esp32 -p /dev/ttyUSB1 -b 460800 --before=default_reset --  
after=hard_reset write_flash --flash_mode dio --flash_freq 40m --  
flash_size 2MB 0x1000 bootloader/bootloader.bin 0x10000  
my_project.bin 0x8000 partition_table/partition-table.bin  
...  
Leaving...  
Hard resetting via RTS pin...  
Done
```

Project Workflow (Monitor)

Running the IDF Monitor tool

Command Line

- Call `idf.py monitor`

```
user@pc:~/esp/my_project$  
idf.py -p /dev/ttyUSB1  
monitor
```

Eclipse IDE

- Open the integrated terminal

Project Workflow (Monitor) (2)

1st Stage Bootloader

```
user@pc:~/esp/esp-idf$ idf.py -p dev/ttyUSB1 monitor  
ets Jun  8 2016 00:22:57  
  
rst:0x1 (POWERON_RESET),boot:0x3e (SPI_FAST_FLASH_BOOT)  
...  
load:0x3fff0030,len:7076  
load:0x40078000,len:15576  
load:0x40080400,len:4  
0x40080400: __init at ???:?  
  
load:0x40080404,len:3876  
entry 0x4008064c
```

Project Workflow (Monitor) (3)

2nd Stage Bootloader

```
I (29) boot: ESP-IDF v5.1 2nd stage bootloader
...
I (56) boot: Partition Table:
I (60) boot: ## Label           Usage            Type ST Offset  Length
I (67) boot: 0 nvs             WiFi data        01 02 00009000 00006000
I (75) boot: 1 phy_init       RF data         01 01 0000f000 00001000
I (82) boot: 2 factory        factory app     00 00 00010000 00100000
I (90) boot: End of partition table
...
I (177) boot: Loaded app from partition at offset 0x10000
...
I (189) cpu_start: Starting app cpu, entry point is 0x400810d8
0x400810d8: call_start_cpul at /home/User/esp/esp-
idf/components/esp_system/port/cpu_start.c:154
```

Project Workflow (Monitor) (3)

App Startup

```
I (0) cpu_start: App cpu up.  
I (207) cpu_start: Pro cpu start user code  
...  
I (207) cpu_start: cpu freq: 16000000 Hz  
I (207) cpu_start: Application information:  
I (212) cpu_start: Project name:      my_project  
...  
I (313) app_start: Starting scheduler on CPU0  
I (317) app_start: Starting scheduler on CPU1  
I (317) main_task: Started on CPU0  
I (327) main_task: Calling app_main()
```

Project Workflow (Monitor) (4)



User Entry Point

Hello World

I (327) main_task: Returned from app_main()

// In main/my_project.c

```
#include <stdio.h>

void app_main(void)
{
    printf("Hello World\n");
}
```

See [Boot Process Documentation](#)
for more details

03

Programming Model

ESP-IDF Programming Model

Where do we go from here?

Arduino?

```
// In main/my_project.c

#include <stdio.h>

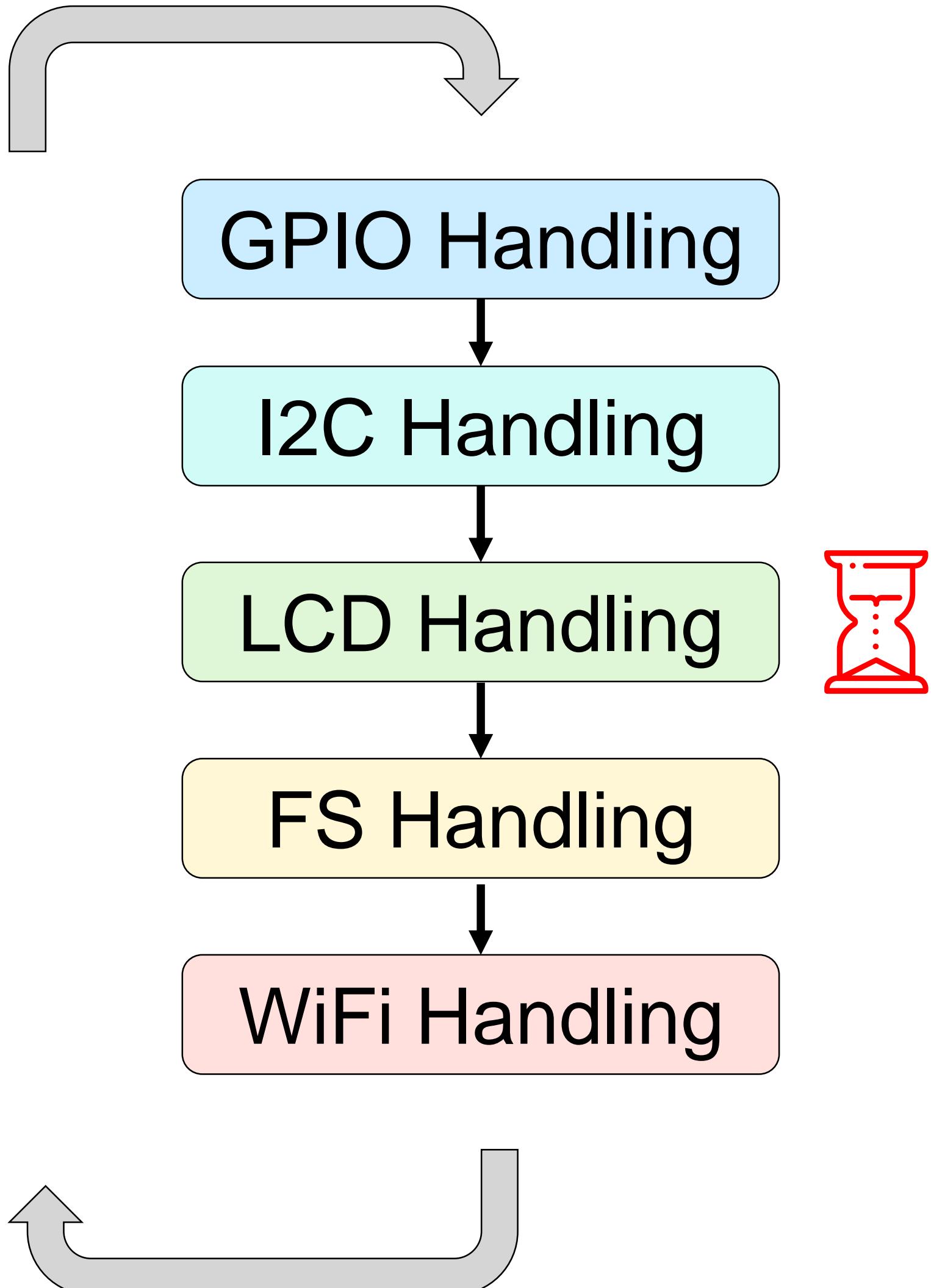
void app_main(void)
{
    printf("Hello World\n");
    setup();
    while (1) {
        loop();
    }
}
```

STM32 CubeMX?

```
// In main/my_project.c
#include <stdio.h>

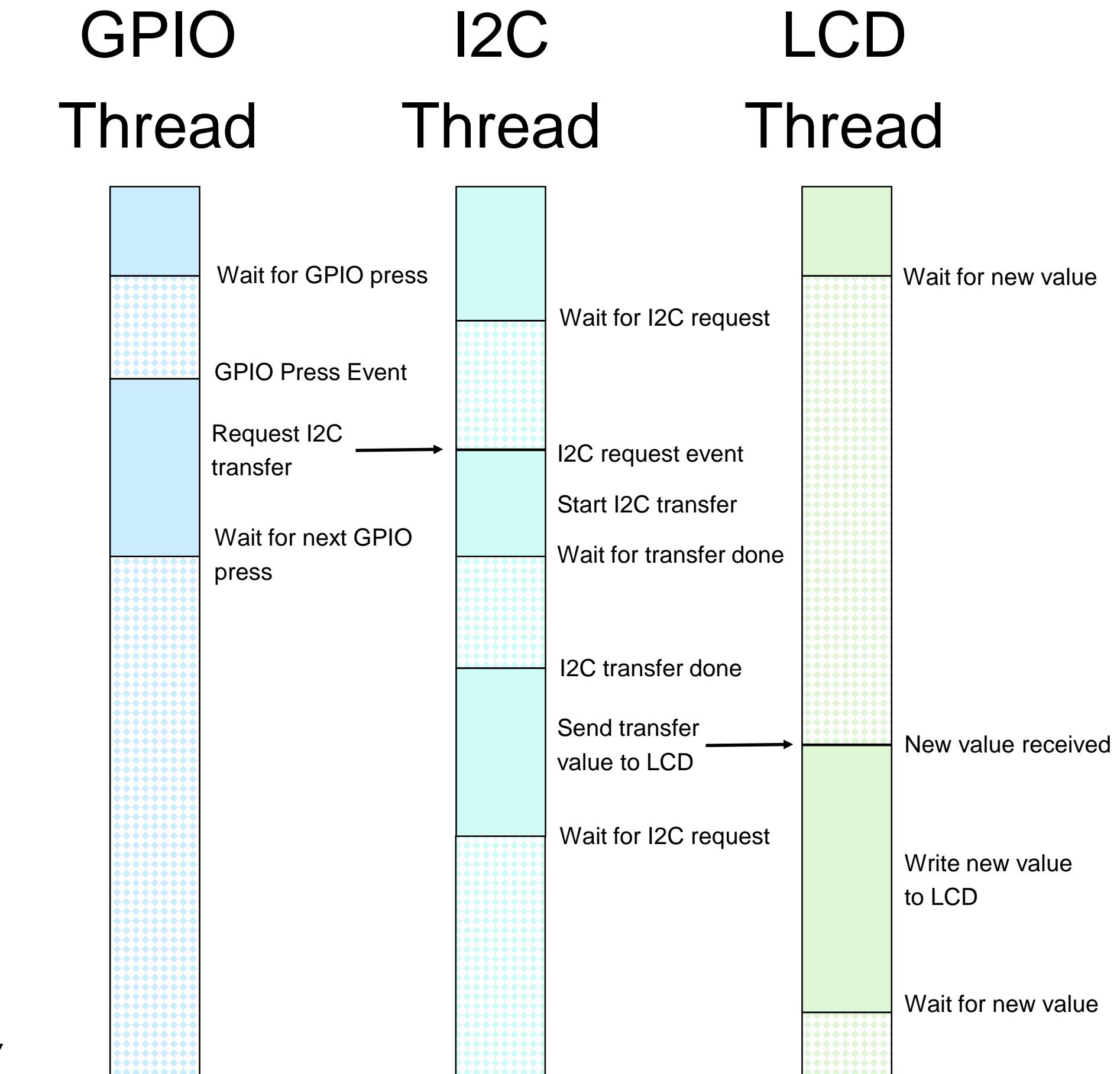
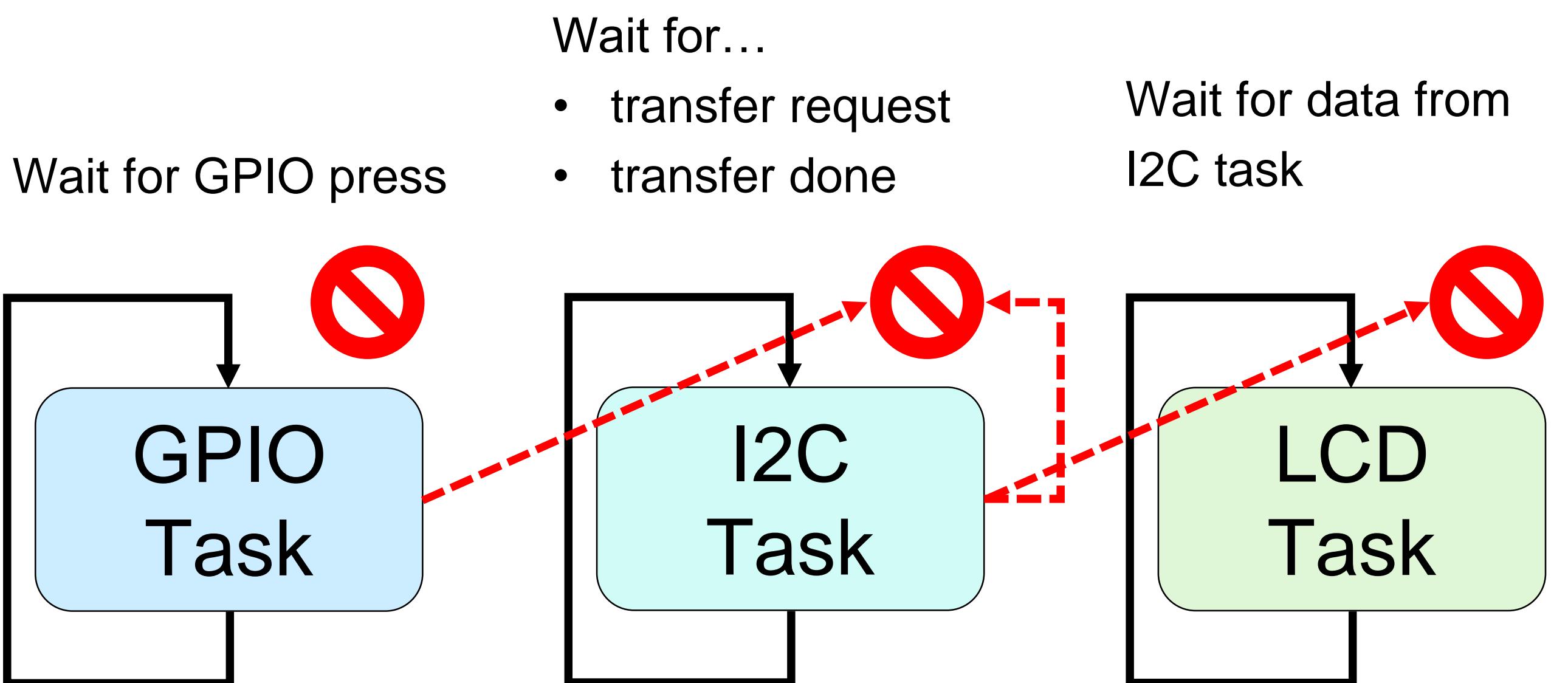
void app_main(void)
{
    printf("Hello World\n");
    /* USER CODE BEGIN 2 */
    user_setup();
    /* USER CODE END 2 */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
        user_loop();
    }
    /* USER CODE BEGIN 3 */
}
```

Issues with Superloop



```
static void loop(void)
{
    /*
    Button press triggers I2C sensor read.
    Ready value gets output to LCD
    */
    if (check_gpio_press()) {
        int val;
        // Handle debounce
        handle_gpio();
        // Blocking read to I2C
        val = handle_i2c();
        // Output value to LCD
        handle_lcd(val);
    }
}
```

Multi-threading via an OS



FreeRTOS (Overview)

What is FreeRTOS?

- Popular RTOS (Real Time Operating System)
- Integrated into ESP-IDF (central to programming model)
- Allows multi-threading (known as tasks)

FreeRTOS Features

- Small and simple
- Free and open source
- Multiple synchronization primitives

FreeRTOS (Tasks)

```
void example_task(void *pvParameters)
{
    // Initialize local variables here
    int local_variable = (int) pvParameters;

    while (1) {
        // Task main loop
        ...
        // Block on some event (e.g., 1000ms)
        xSemaphoreTake(event_sem, pdMS_TO_TICKS(1000));
        // Do some work
        handle_event();
        // Give some event
        xSemaphoreGive(other_event_sem);
        ...
        // Check for exit condition
        if (exit_condition_met) {
            break;
        }
    }

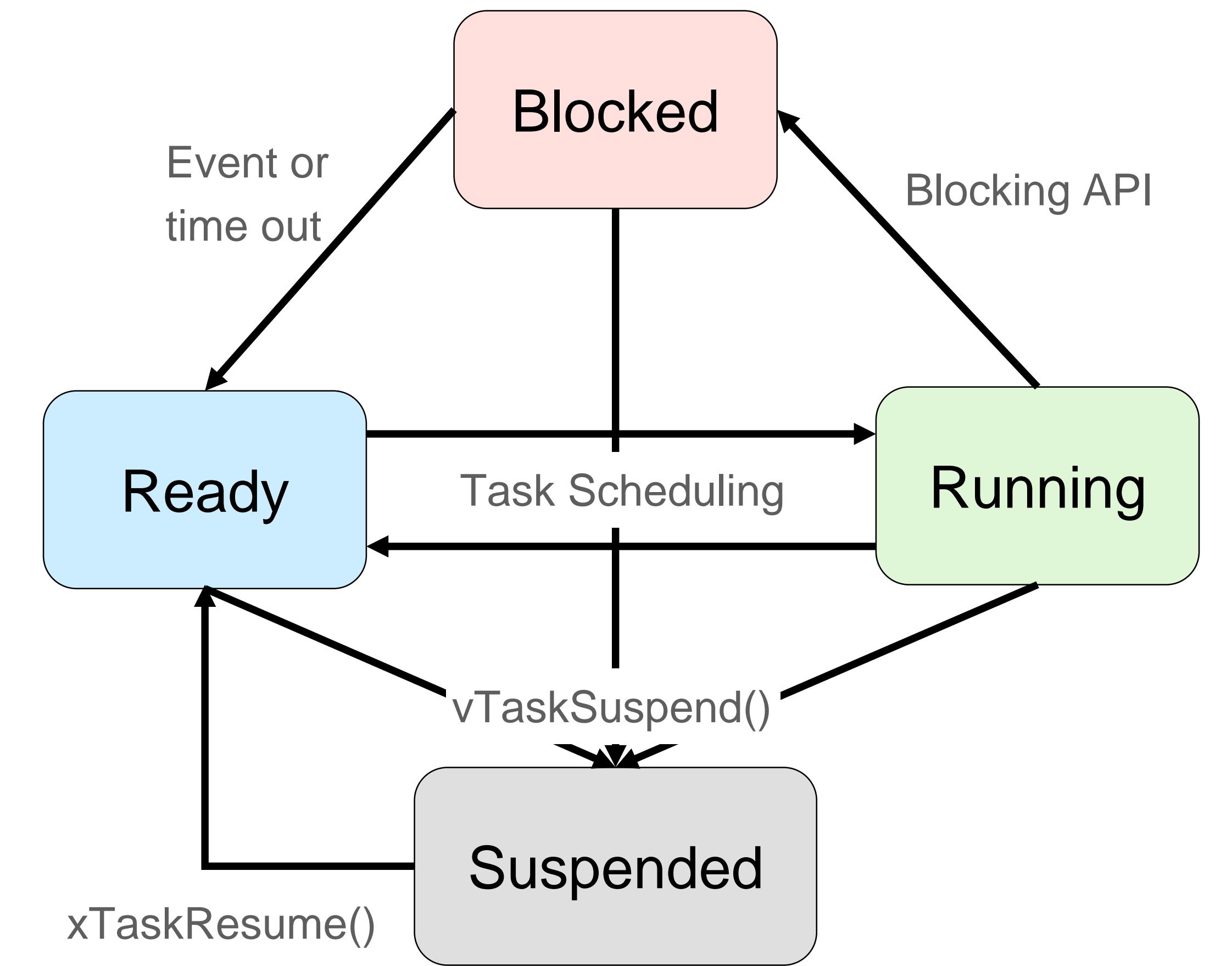
    // Delete task here
    vTaskDelete(NULL);
    // MUST NEVER RETURN!
}
```

Overview of FreeRTOS Task

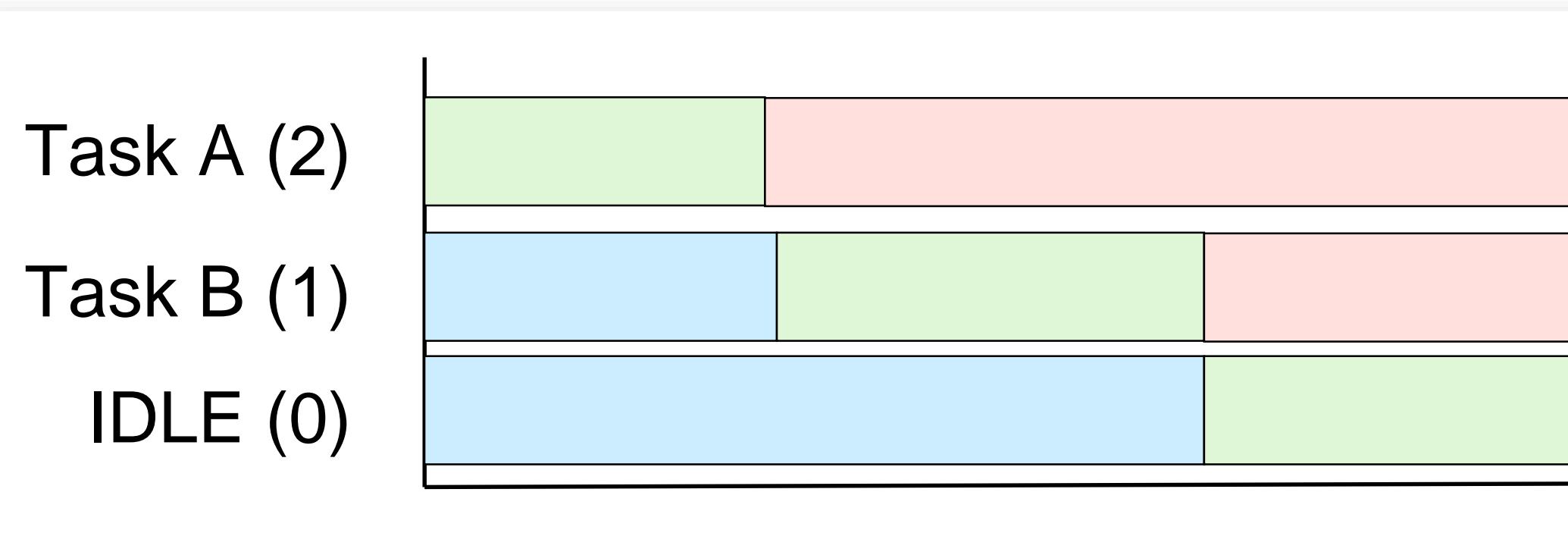
- Implemented as C function
- Usually implemented as an infinite loop
- Tasks can block on events (does not consume CPU time)
- Tasks can give events (unblock other tasks)
- Function must never return

FreeRTOS (Task States)

Task State	Description
Running	The task is currently be executed
Ready	The task is able to execute but is not currently executing
Blocked	The task is waiting for a temporal or external event. The task will not be scheduled until it is unblocked by the event or times out.
Suspended	Task will not be scheduled until explicitly unsuspended



FreeRTOS (Scheduling Algorithm)



Running Ready Blocked

Fixed Priority

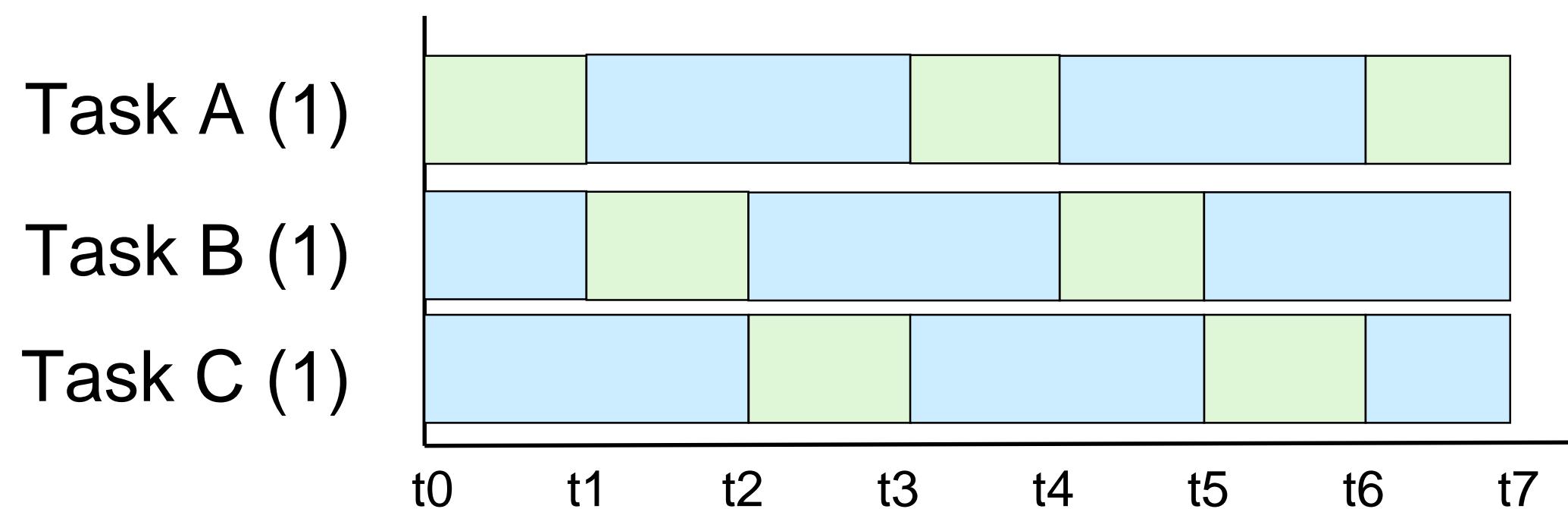
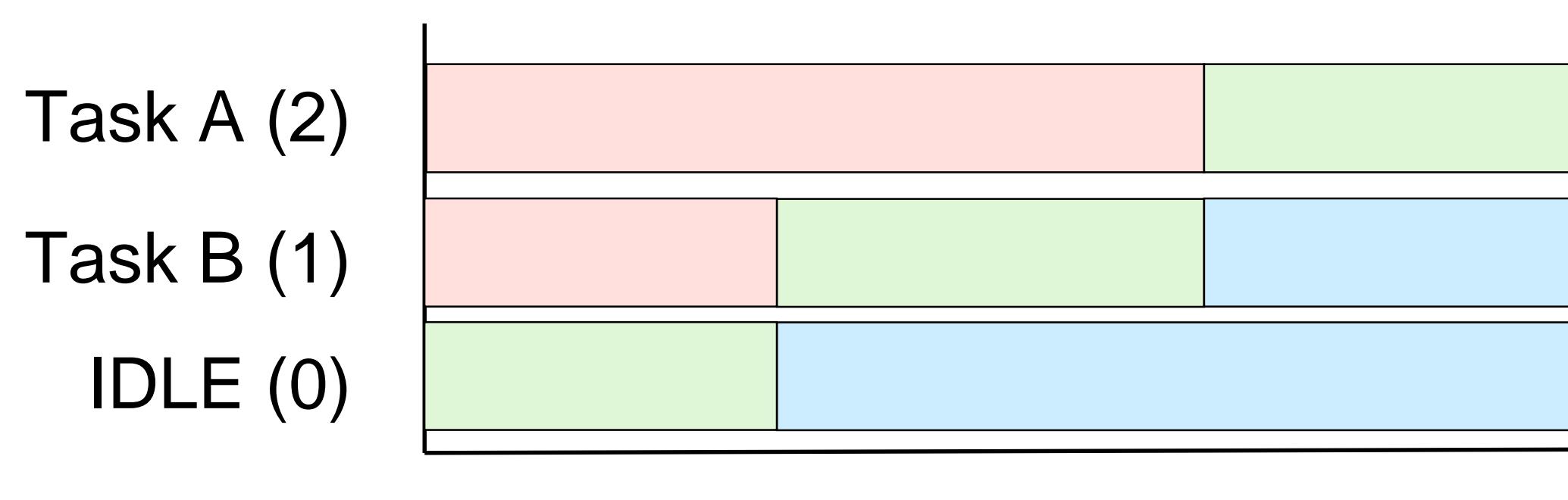
- Each task has a fixed priority
- Scheduler always executes the **highest priority ready state** task

Preemptive

- Scheduler can **switch out current task at any time** (without cooperation)
- Higher priority tasks will preempt current task.

Time Slicing

- Periodic switching of **ready tasks of the same priority**, governed by tick interrupt.
- Switch in Round Robin fashion.



FreeRTOS (Further Resources)



FreeRTOS (Upstream)

- [RTOS Fundamentals](#)
- [Mastering the FreeRTOS Kernel](#)
- [FreeRTOS Configuration](#)



FreeRTOS (IDF Integration & SMP)

- [ESP-IDF FreeRTOS Integration](#)
- [ESP-IDF FreeRTOS SMP Changes](#)
- [ESP-IDF FreeRTOS Supplemental Features](#)
- [2022 Devcon FreeRTOS Presentation](#)

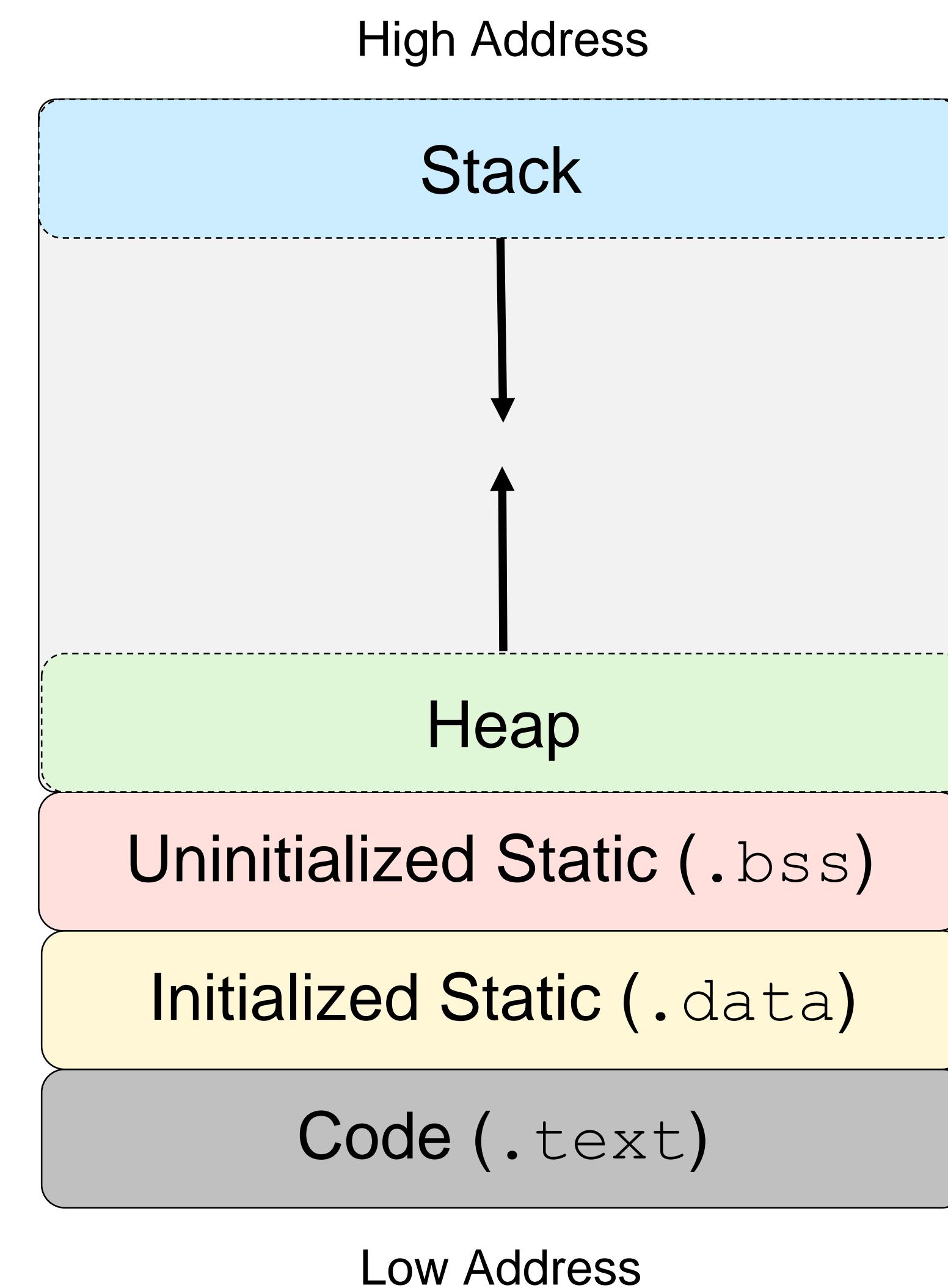
Memory Model

The ideal C memory model assumptions:

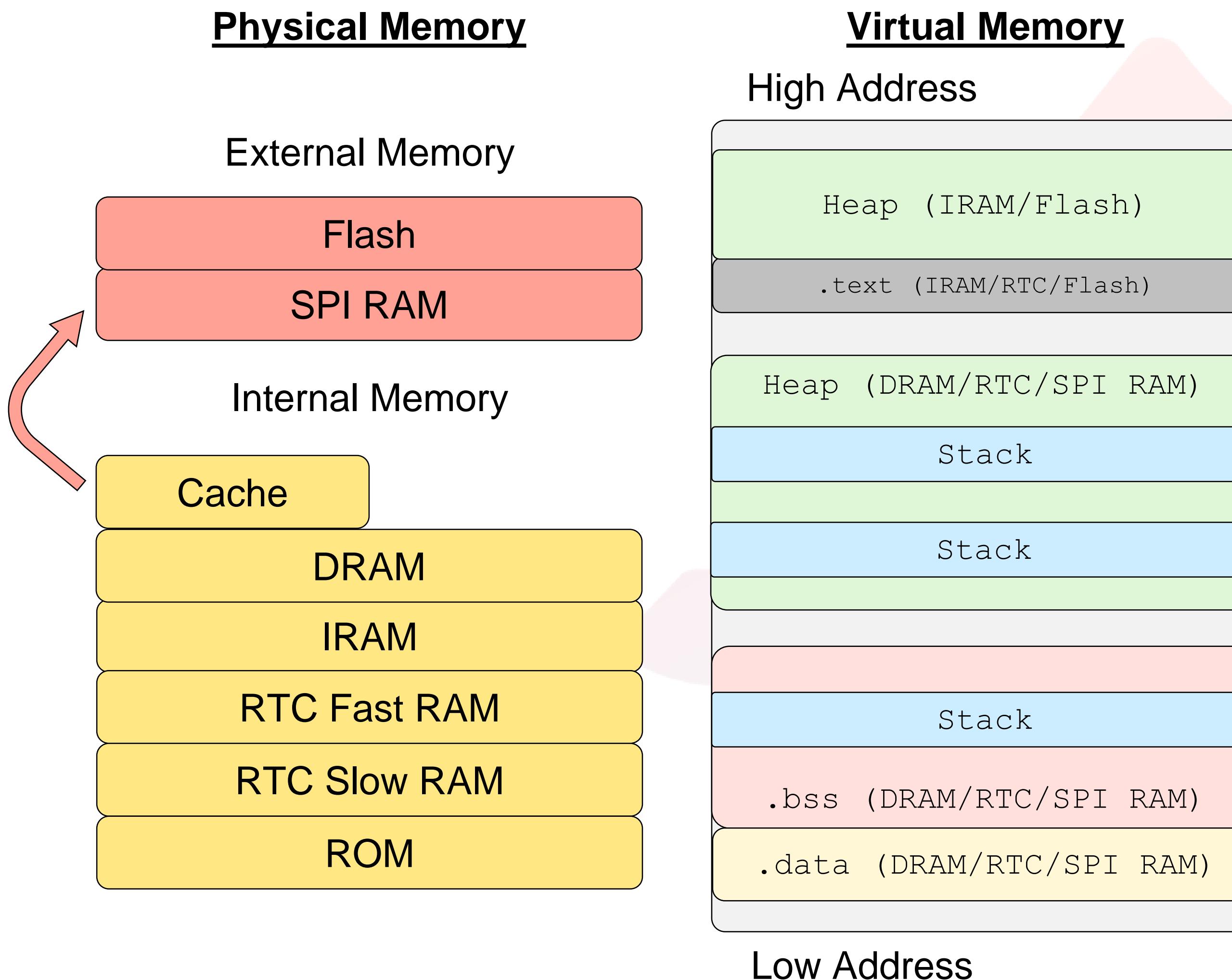
- All memory is physically identical
- Single stack
- Heap and stack grow in opposite directions

Issues:

- Multiple stacks
- Memory is not physically identical



Memory Model (2) [SIMPLIFIED]



Physical Memory

- Internal memory with different capabilities
- External Memory accessed via Cache

Virtual Memory

- Expanded static sections to cover different physical memories
- Heap can allocate from different physical memories
- Each stack is now allocated from heap or static memory

Memory Model (3)



Key Takeaways

- Memory is non-uniform
- Multiple tasks means multiple stacks
- Task stacks can be allocated from heap/.data/.bss

Memory: Further Reading

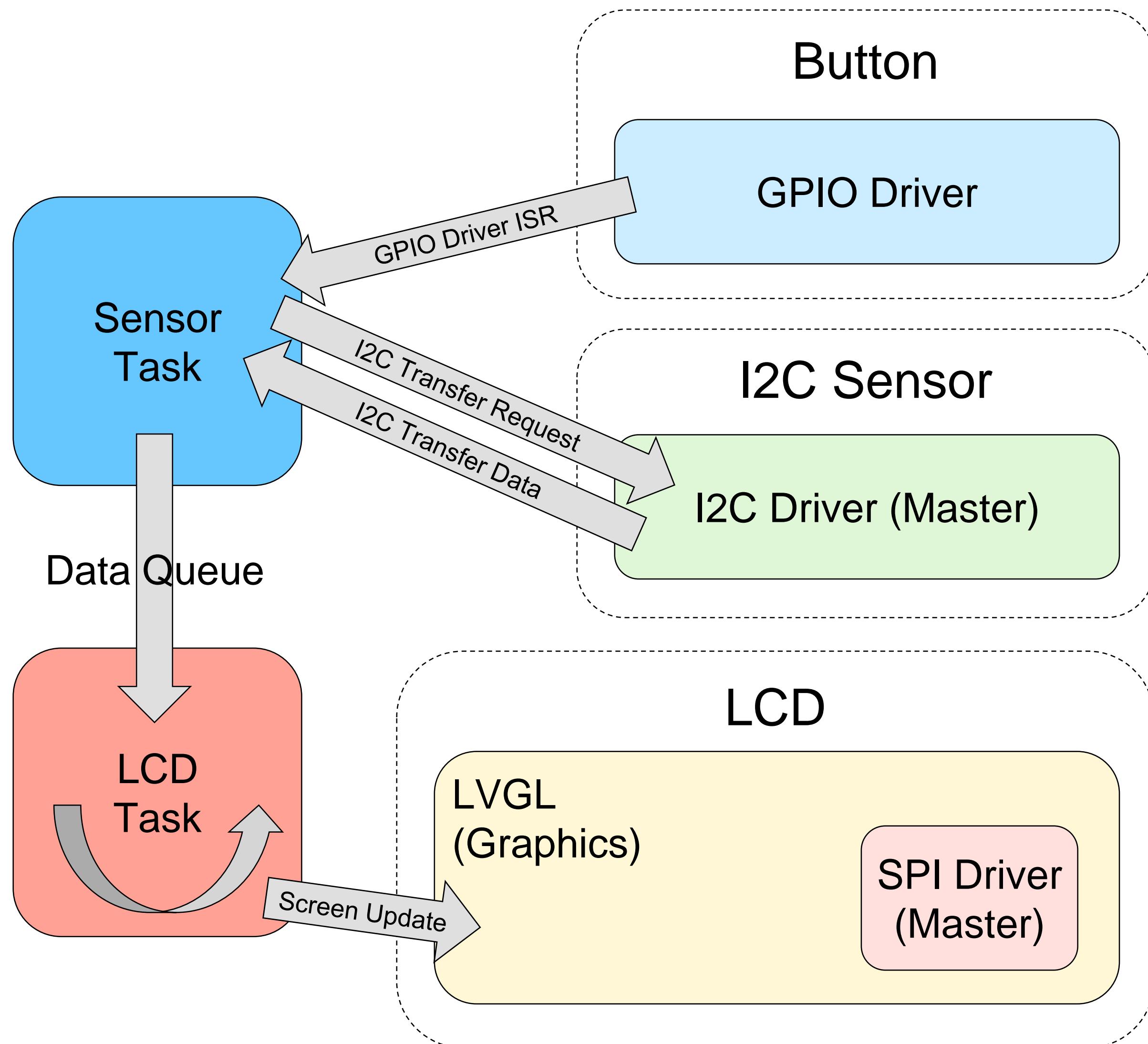
- [ESP-32 TRM \(System & Memory\)](#)
- [ESP-IDF Memory Types](#)
- [Heap Memory Allocator](#)
- [ESP-IDF SPI-RAM Guide](#)
- [Linker Script Generation](#) (How to place data/functions into particular type of memory)

04

ESP-IDF Features

Where to find ESP-IDF features

IDF Features In a Project



1. Breakdown project into subsystems
2. Find/write components and libraries required for each subsystem
3. Read API of each required component and library
4. Implement glue logic to connect subsystems

Where to Find Components?



1. [ESP-IDF components](#)
2. [IDF Component Registry \(Component Manager\)](#)
3. Create your own

```
- my_project/
  - components/
    - my_i2c_sensor/
      - i2c_sensor.c
      - CMakeLists.txt
    ...
    - my_graphics_lib/
      - graphics_lib.c
      - CMakeLists.txt
    ...
  - main/
  - CMakeLists.txt
```

The screenshot shows the 'API Reference' section of the ESP-IDF Programming Guide. It includes a search bar, dropdown menus for 'ESP32' and 'stable (v5.1)', and a sidebar with links to various API categories and documentation pages.

- Networking APIs
 - Wi-Fi
 - Ethernet
 - Thread
 - ESP-NETIF
 - IP Network Layer
 - Application Layer
- 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
 - Digital To Analog Converter (DAC)
 - GPIO & RTC GPIO
 - General Purpose Timer (GPTimer)
 - Inter-Integrated Circuit (I2C)
 - Inter-IC Sound (I2S)
 - LCD
 - LED Control (LEDC)
 - Motor Control Pulse Width Modulator (MCPWM)
 - Pulse Counter (PCNT)
 - Remote Control Transceiver (RMT)
 - SD Pull-up Requirements
 - SDMMC Host Driver
- API Conventions
- Application Protocols
- Bluetooth API
- Error Codes Reference
- Networking APIs
- Peripherals API
- Project Configuration
- Provisioning API
- Storage API
- System API

The screenshot shows the ESP Registry homepage with a search bar and a link to 'Sign in'. The main heading reads 'Find the most exciting ESP-IDF components'.

Kickstart your next IoT Project with the open-source components, you can also easily integrate the components into your existing IDF projects.

Search components

Browse components

ALL Board Support Packages ESP32 ESP32-C2 ESP32-C3 ESP32-C6 ESP32-H2 ESP32-S2 ESP32-S3

05

Build System & Components

Overview of ESP-IDF build system
and what components are.

ESP-IDF Build System Overview



The ESP-IDF build system is based on CMake.

01

Components > Libraries

02

CMake wrapper commands to accommodate components

03

Integrate into idf.py frontend

04

Still possible to use pure CMake

Components vs Library

Component is a library but with extra stuff

- Source files, headers, include directories
- Linker Fragments
- Embedded binary/text files
- Other pre-compiled libraries
- Kconfig configuration
- Unit tests

```
some_component/
  - include/
    - some_component.h
    ...
  - src/
    - some_component.c
    ...
  - linker.lf
  - data.txt
  - some_other_lib.a
  - Kconfig
  - tests/
  ...
```

Component CMakeLists.txt

```
project(libX)
add_library(libX libX.c)
target_link_libraries(libX PRIVATE libY PRIVATE libZ)
```

```
# Register component
idf_component_register(
    # Source files for this component
    SRCS ./src/my_component.c ./src/algo.c
    # Include directories for this component
    INCLUDE_DIRS ./include
    # Other components required by this component
    REQUIRES esp_lcd
    PRIV_REQUIRES driver
    # Linker fragment for this component
    LDFRAGMENTS linker.lf)
```

Project CMakeLists.txt

```
project(my_project)
add_executable(my_app my_app.c)
target_link_libraries(my_app PUBLIC libA PRIVATE libX)
```

```
cmake_minimum_required(VERSION 3.16)

# Pull in IDF Cmake Wrappers
include(${ENV{IDF_PATH}}/tools/cmake/project.cmake)

# Default directories that are searched for components
#   - "esp-idf/components/"
#   - "my_project/components/"
# Declare any extra directories to search for components
set(EXTRA_COMPONENT_DIRS "${ENV{MY_LIBS}}/components")

# Declare project
project(my_project)
```

Build Systems (Further Resources)

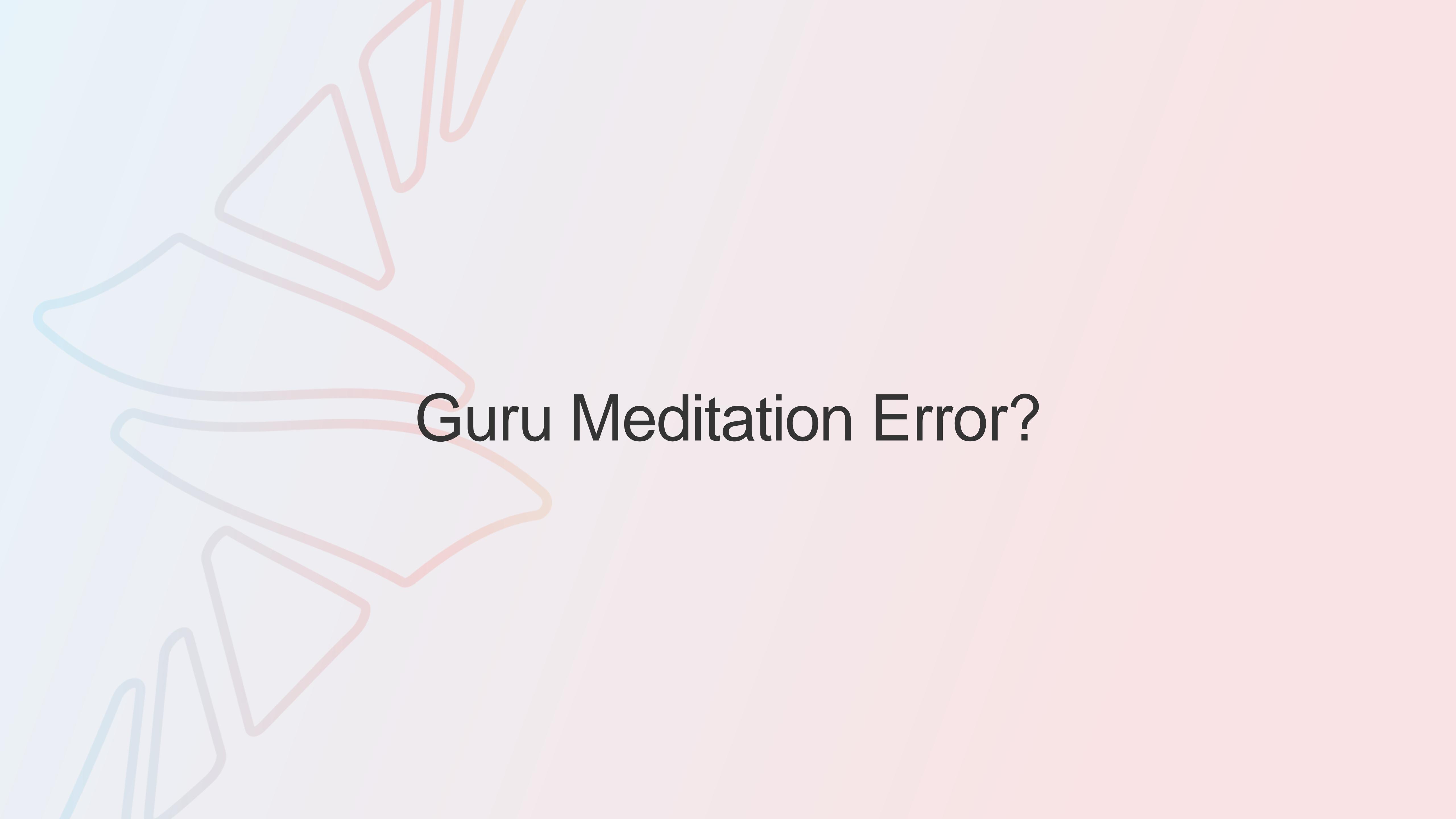


[ESP-IDF Build System Documentation](#)

06

More Resources

Where to get more help & resources



Guru Meditation Error?

```
#include <stdio.h>

static void some_func(int *some_ptr)
{
    *some_ptr = 1; ← NULL pointer access
}

void app_main(void)
{
    printf("Hello World\n");
    some_func((int *)NULL);
}
```

Guru Meditation Error: Core 0 panic'ed (StoreProhibited). Exception was unhandled.

Core 0 register dump:

PC	: 0x400e23f1	PS	: 0x00060a30	A0	: 0x800d50df	A1	: 0x3ffb4c20
A2	: 0x00000000	A3	: 0x3f4028fc	A4	: 0x3f40297c	A5	: 0x3ffb4c40
A6	: 0x3ffb4c20	A7	: 0x0000000c	A8	: 0x00000001	A9	: 0x3ffb4be0
A10	: 0x0000000a	A11	: 0x3ffaeb44	A12	: 0x3ffb4c20	A13	: 0x0000000c
A14	: 0x3ffb2934	A15	: 0xb33fffff	SAR	: 0x00000004	EXCCAUSE	: 0x0000001d
EXCVADDR	: 0x00000000	LBEG	: 0x400014fd	LEND	: 0x4000150d	LCOUNT	: 0xfffffffffc

Backtrace: 0x400e23ee:0x3ffb4c20 0x400d50dc:0x3ffb4c40 0x400e30b8:0x3ffb4c60
0x40087bad:0x3ffb4c90

Fatal Errors



- Fatal error cause
- Register Dump
- Backtrace
- Auto address decoding by IDF monitor

Guru Meditation Error: Core 0 panic'ed (StoreProhibited). Exception was unhandled.

Core 0 register dump:

```

PC      : 0x400e23f1  PS      : 0x00060a30  A0      : 0x800d50df  A1      : 0x3ffb4c20
0x400e23f1: some_func at /home/User/my_project/build/../main/my_project.c:5
A2      : 0x00000000  A3      : 0x3f4028fc  A4      : 0x3f40297c  A5      : 0x3ffb4c40
A6      : 0x3ffb4c20  A7      : 0x0000000c  A8      : 0x00000001  A9      : 0x3ffb4be0
A10     : 0x0000000a  A11     : 0x3ffaeb44  A12     : 0x3ffb4c20  A13     : 0x0000000c
A14     : 0x3ffb2934  A15     : 0xb33fffff  SAR    : 0x00000004  EXCCAUSE: 0x0000001d
EXCVADDR: 0x00000000 LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT   : 0xfffffffffc
  
```

Backtrace: 0x400e23ee:0x3ffb4c20 0x400d50dc:0x3ffb4c40 0x400e30b8:0x3ffb4c60 0x40087bad:0x3ffb4c90

0x400e23ee: some_func at /home/User/my_project/build/../main/my_project.c:4

0x400d50dc: app_main at /home/User/my_project/build/../main/my_project.c:12

0x400e30b8: main_task at /home/User/esp/esp-idf/components/freertos/app_startup.c:217 (discriminator 13)

0x40087bad: vPortTaskWrapper at /home/User/esp/esp-idf/components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:162

Debugging Tools



Runtime Debugging Tools

- [GDB & OpenOCD](#)
- [Application Tracing](#)
- [GDB Stub \(Runtime\)](#)

Post-Mortem Debugging Tools

- [Core Dump](#)
- [GDB Stub \(Panic\)](#)

Support

Github

- [Report issues or feature requests](#)
- Contribute [pull requests](#) (see [ESP-IDF Contributors Guide](#))

[ESP32 Forum](#)

- Ask questions
- Discuss ESP32/ESP-IDF



ESPRESSIF
DevCon23

Thanks for watching !