ESPRESSIF

DevCon24

# Practical guide about Arduino as IDF component

Rodrigo Garcia

# CONTENTS

# 01 ESP32 Arduino

What is it? IDF relationship?

# Arduino overview

## Origin

Created in 2005 at the Interaction Design Institute Ivrea, Italy

Key Targets:

- ➔ Practical IDE for any O.S.
- ➔ Simple and easy to learn
- ➔ Complete Tool Chain - transparent to the user
- ➔ Integrated Bootloader
- ➔ Serial Monitor for Inspecting
- ➔ High abstraction level with Libraries and Examples

# Arduino overview

## API Documentation

https://www.arduino.cc/reference/

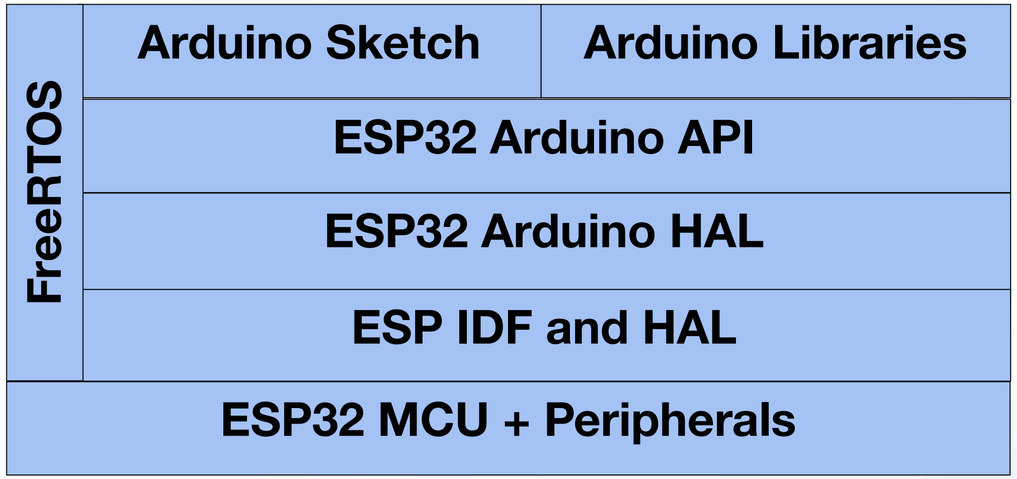
- ➔ Digital pins, Analog ADC and PWM abstraction
- ➔ Timers, GPIO Interrupt and time measuring
- ➔ USB HID & CDC
- ➔ SPI, I2C (Wire), Serial
- ➔ Generic API for Stream & Print
- ➔ A sketch is a C++ code.

- ➔ Many examples, community libraries and tutorials.

# Arduino & ESP IDF

ESP32 Arduino is an IDF layer

https://docs.espressif.com/projects/arduino-esp32/en/latest/getting_started.html



# Arduino as IDF Component

## Building basics

This is exactly the same as how an IDF project is built.

- ➔ Use IDF tools and toolchain
- ➔ Tune sdkconfig or copy it from ESP32 Arduino Libs (Arduino Core 3.x) or from Arduino Core 2.x release from Github
  https://github.com/espressif/esp32-arduino-libs/blob/idf-release/v5.1/esp32s3/sdkconfig
  https://github.com/espressif/arduino-esp32/blob/release/v2.x/tools/sdk/esp32s3/sdkconfig

- ➔ Use **`idf.py menuconfig`**, if necessary. Partition, Flash Size and Mode, etc.
- ➔ Set the target, add Sketch Code and Arduino Libraries, build, flash and monitor using the UART.

# Arduino as IDF Component

## Arduino and IDF versions

IDF version must match correspondent Arduino Core version
This can be verified in the Github Release information
https://github.com/espressif/arduino-esp32/releases

- ➔ Arduino Core 3.0.0 to 3.0.4 use IDF version 5.1.4
- ➔ Arduino Core 2.0.15 to 2.0.17 use IDF version 4.4.7
- ➔ Arduino Core 2.0.14 uses IDF version 4.4.6
- ➔ Arduino Core 2.0.10 to 2.0.13 use IDF version 4.4.5
- ➔ Arduino Core 2.0.7 to 2.0.9 use IDF version 4.4.4

# ESP Component Registry

ESPRESSIF DevCon24

## Managed Components and Libraries

This is central repository for components that can be used with ESP-IDF framwork.
https://components.espressif.com/

- ➔ Espressif and Community maintained Components (Libraries)
- ➔ Use **`idf_component.yml`** file to declare any component dependency
- ➔ ESP32 Arduino is a Managed Component from ESP Component Registry
  https://components.espressif.com/components/espressif/arduino-esp32/

# 02 Hello World!

Basic Example

# HelloWorld! Example

## Folder Structure

This is the basic example.

```
Proj
├── CMakeLists.txt
├── sdkconfig.defaults
└── main
    ├── CMakeLists.txt
    ├── idf_component.yml
    └── main.cpp
```

# HelloWorld! Example

ESPRESSIF DevCon24

## Proj / CMakeLists.txt

Global Project IDF CMake file.
https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/build-system.html

```
cmake_minimum_required(VERSION 3.16)


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

# HelloWorld! Example

## Proj / sdkconfig.defaults

It contains all modified **sdkconfig** settings.

Can be manually created with **`idf.py save-defconfig`**

```
# Arduino ESP32
CONFIG_AUTOSTART_ARDUINO=y


# FREERTOS
CONFIG_FREERTOS_HZ=1000
```

# HelloWorld! Example

ESPRESSIF DevCon24

## Proj / main / idf_component.yml

It contains all necessary IDF components from ESP Registry.
Arduino Core is one Managed Component that can be included here.

```
## IDF Component Manager Manifest File
dependencies:
  espressif/arduino-esp32:
    version: "*"
```

# HelloWorld! Example

ESPRESSIF DevCon24

## Proj / main / CMakeLists.txt

This is the CMake setting for building the Arduino Code

```
idf_component_register(
  SRCS         "main.cpp"
  INCLUDE_DIRS "."
)
```

# HelloWorld! Example

ESPRESSIF DevCon24

## Proj / main / main.cpp

Arduino Source Code in C++. **It must include “Arduino.h”**

```
#include "Arduino.h"

void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.println("Hello world!");
  delay(1000);
}
```

# HelloWorld! Example

## Building the Arduino as IDF Component project

This is a regular IDF project, therefore use IDF tools.

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/

```
idf.py --version
idf.py --help
idf.py --list-targets

idf.py fullclean  || rmdir /s/q build || rm -rf build
idf.py set-target esp32s3 || del or rm sdkconfig
idf.py menuconfig
idf.py -p <COM Port | DEV File> flash monitor
```

# 03 Adding Libraries

to the Arduino Project

# Arduino Core Libraries

ESPRESSIF DevCon24

## Libraries and Examples included in Arduino Core

Any library and related example that are already in the Arduino repository can be used with no special configuration.

https://github.com/espressif/arduino-esp32/tree/master/libraries

**ArduinoOTA, AsyncUDP, Update, WiFi, BLE, Bluetooth SPI, Wire, USB, ESP_I2S, BluetoothSerial, EEPROM DNSServer, ESPmDNS, OpenThread, NetBIOS, Insights RainMaker, Preferences, WiFiProv, WebServer, Ethernet, HTTPClient, HTTPUpdate, HTTPUpdateServer, ESP_NOW, FS, FFat, LittleFS, SPIFS etc.**

# External Arduino Libraries

## Arduino Libraries

Any external Arduino Library shall be included as a local component.

It demands proper `CMakeLists.txt` to declare at least the Library source code files and or folders.

It can be manually cloned directly from its repository into `components` folder.

# ESP Component Registry

## IDF Managed Components

Any Managed Component can be used within an *Arduino as IDF Component Project*.

It demands proper **`idf_component.yml`** declaration in order to allow the building system to manage and include it.

It MUST NOT be manually cloned or modified. The code will be placed into **`managed_components`** folder. All other dependencies will also be placed there.

# Adding Arduino Libraries

## Library Folder Structure

It may contain Arduino Libraries and settings.

Arduino Libraries shall be included as Local Components

```
├── CMakeLists.txt          Arduino Project Description and Settings
├── sdkconfig.defaults      Arduino and IDF Project Settings
├── main
│   ├── CMakeLists.txt      Arduino Sketch Description
│   ├── idf_component.yml   ESP32 Arduino Core version and necessary components
│   └── main.cpp            Main Sketch, equivalent to a .ino file
├── components
│   ├── user_library_1
│   │   ├── CMakeLists.txt   This will describe the Lib_1 source code files
│   │   ├── ...              Regular Library_1 files
│   ├── user_library_2
│   │   ├── CMakeLists.txt   This will describe the Lib_2 source code files
│   │   ├── ...              Regular Library_2 files
```

# Adding Arduino Libraries

## Example: WiFi Manager Library

This is one of the top used Libraries for ESP32

https://github.com/tzapu/WiFiManager

```
cd myProj
mkdir components
cd components
git clone https://github.com/tzapu/WiFiManager
```

```
This project already contains a CMakeLists.txt file!
```

# Example WiFi Manager Library

## CMakeLists.txt Library File

This shall declare at least the source files and included files necessary to compile the library.

```
idf_component_register(SRCS "WiFiManager.cpp"
                       INCLUDE_DIRS "."
                       REQUIRES espressif__arduino-esp32
                       )
```

**`espressif__arduino-esp32`** shall be replaced by the folder name used for ESP32 Arduino (managed) component.

# Example WiFi Manager Library

ESPRESSIF DevCon24

## sdkconfig.defaults Project File

For ESP32 and ESP32-S3 which have 2 cores, it is necessary to configure the WatchDog Timer settings to avoid problems with WiFi:

```
# Execute the Arduino Sketch from setup() and loop()
CONFIG_AUTOSTART_ARDUINO=y

# Disable checking CPU1 IDLE Task in Dual Core SoC
CONFIG_ESP_TASK_WDT_CHECK_IDLE_TASK_CPU1=n

# 1 tick = 1ms
CONFIG_FREERTOS_HZ=1000
```

# 04 Enabling USB CDC

Serial USB port

# Hardware Serial

## USB-Serial-JTAG Controller

Some ESP32 SoC have a Hardware implementation for USB CDC Class.
ESP32-C3, ESP32-C6, ESP32-S3, ESP32-H2, ESP32-P4

ESP32 Ardudino Framework will enable it and redefine **`Serial`** symbol to be attached to the HW CDC port.

This is done by adding two **defined** symbols:

```
ARDUINO_CDC_ON_BOOT = 1 //will make Serial to be USB CDC
ARDUINO_USB_MODE = 1 //1 for HW Serial and 0 for USB OTG
```

# Hardware Serial

## Adding necessary global “defines”

This is done in the **`CMakeLists.txt`** global project file.

```
# Adds necessary definitions for compiling it
# using Serial symbol attached to the HW USB CDC port

list(APPEND compile_definitions "ARDUINO_USB_CDC_ON_BOOT=1")
list(APPEND compile_definitions "ARDUINO_USB_MODE=1")
```

# Hardware Serial

## Arduino usage

For all ESP32 SoC that supports Hardware Serial interface, **`HWCDCSerial`** object from **`HWCDC`** class will be available.

If **`ARDUINO_CDC_ON_BOOT=1`** is set, Serial will be defined as **`HWCDCSerial`** symbol, therefore, it will behave accordingly.

For the S3 and P4, this also depends on **`ARDUINO_USB_MODE = 1`** because the SoC has both USB modes, HW Serial and USB OTG (based on TinyUSB driver).
All other HW Serial compatible SoC shall also set **`ARDUINO_USB_MODE = 1`** in order to have Serial as the HW USB CDC port.

# Hardware Serial

ESPRESSIF DevCon24

## Uploading the firmware

After finishing building, uploading will require the SoC to enter in Download Mode before starting the upload.

For that, just hold BOOT button and pulse RESET/EN button. After that release BOOT button and the SoC will be ready for uploading using USB port.

At the end of the uploading process, just pulse RESET/EN to exit the Download Mode and start the execution of the firmware.

# USB OTG + CDC

ESPRESSIF DevCon24

## Only Available for ESP32-S2 and ESP32 -S3

This is done in the **`CMakeLists.txt`** global project file.

```
# Adds necessary definitions for compiling it
# using Serial symbol attached to the HW USB CDC port

list(APPEND compile_definitions "ARDUINO_USB_CDC_ON_BOOT=1")
list(APPEND compile_definitions "ARDUINO_USB_MODE=0")

set(EXTRA_COMPONENT_DIRS
esp32-arduino-lib-builder/components/arduino_tinyusb)
```

# USB OTG + CDC

## Necessary additional Local components

Clone extra components into the project folder.

https://docs.espressif.com/projects/arduino-esp32/en/latest/esp-idf_component.html#manual-installation-of-arduino-framework

```
git clone
https://github.com/espressif/esp32-arduino-lib-builder.git
esp32-arduino-lib-builder

git clone https://github.com/hathach/tinyusb.git
esp32-arduino-lib-builder/components/arduino_tinyusb/tinyusb
```

# 05 Debugging

Enabling debug level

# Debug Messages

ESPRESSIF DevCon24

## There are 3 possible places to activate debugging

Those shall be changed using **`idf.py menuconfig`**

1. Boot 2nd stage debug messages - default is INFO
   *(Top) → Bootloader config → Bootloader log verbosity*

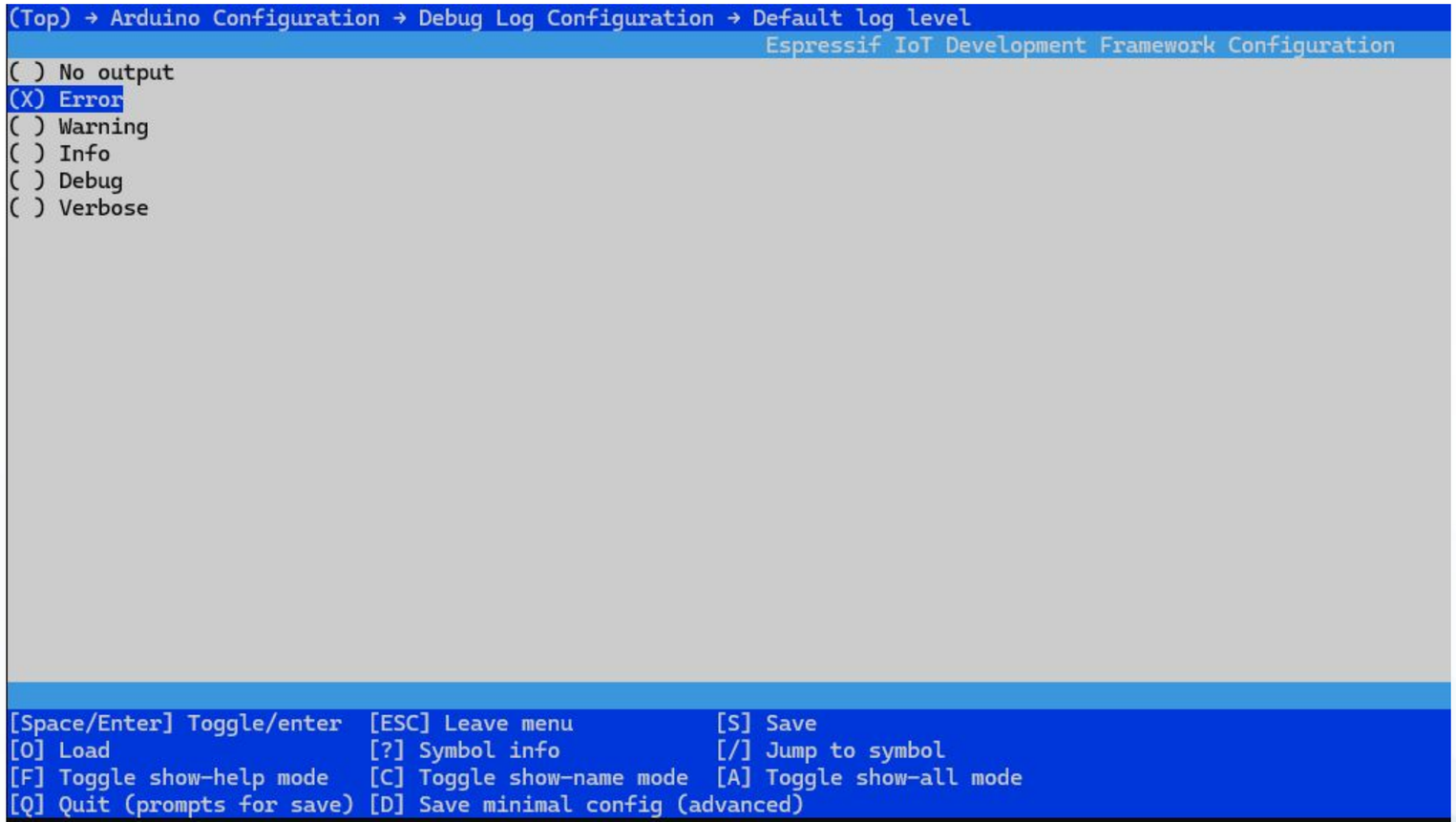
2. IDF components debug messages - default is INFO
   *(Top) → Component config → Log output → Default log verbosity*

3. Arduino LOG level - default is ERROR
   *(Top) → Arduino Configuration → Debug Log Configuration → Default log level*

# Debug Messages

## Screenshot for the Arduino Log Level

It be changed using **`idf.py menuconfig`**



# Arduino Sketch Debugging

ESPRESSIF DevCon24

## ESP32 Arduino has its own way to enable Debug

First, pick which Serial interface will be used to print the logs.

```
Serial.setDebugOutput(true); // Log output goes to UART0 or USB CDC

Serial1.setDebugOutput(true); // Log output goes to UART1
```

Then print the logs using the proper Log Level.

```
log_i("Log Message"); // logs a INFO message to the chosen output
// log_e() | log_w() | log_i() | log_d() | log_v()
```

# Final Information

## Get the presentation and Examples

https://github.com/suglider/devcon2024/

ESPRESSIF

DevCon24

# Thanks for watching !