

Getting started with ESP32-C3-DevKitM-1

Software Design for Embedded Systems

Additional course materials

Author: Hrvoje Džapo

Faculty of electrical engineering and Computing

University of Zagreb, 2022

Contents

| | | |
|-----|--|----|
| 1 | Development tools installation..... | 3 |
| 1.1 | Espressif IDE Installation (Windows)..... | 3 |
| 1.2 | ESP32-C3-DevKitM-1 / ESP PROG FTDI drivers installation | 3 |
| 1.3 | Setting up correct driver versions for ESP32-C3 | 4 |
| 2 | Setting up ESP32-C3-DevKitM-1 for debugging via ESP PROG | 7 |
| 2.1 | Notes about ESP32-C3-DevKitM-1 hardware configuration..... | 7 |
| 2.2 | Enabling internal JTAG support for ESP32-C3-DevKitM-1 | 8 |
| 2.3 | Enabling external JTAG support for ESP32-C3-DevKitM-1..... | 10 |
| 3 | Creating project in Espressif IDE | 12 |
| 3.1 | Starting new project from the scratch..... | 12 |

1 Development tools installation

This guide will assume the following development environment:

- Espressif IDE with ESP-IDF framework¹
- Windows operating system²
- hardware: ESP32-C3-DevKitM-1 development kit
- programming and debugging: ESP PROG³

Device drivers need to be installed (FTDI) in addition to installation of IDE and libraries.

1.1 Espressif IDE Installation (Windows)

Download installer:

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html#ide>

espressif-ide-setup-espressif-ide-2.6.0-with-esp-idf-4.4.2.exe

Note: If necessary, uninstall previously installed version and manually delete what is left over in installation folder (e.g. C:/Espressif).

1.2 ESP32-C3-DevKitM-1 / ESP PROG FTDI drivers installation

After successfull installation of Espressif IDE, run installation of VCP FTDI drivers:

<https://ftdichip.com/drivers/vcp-drivers/>

| Operating System | Release Date | X86 (32-Bit) | X64 (64-Bit) | Comments |
|--------------------|--------------|---------------------------|---------------------------|--|
| Windows (Desktop)* | 2021-07-15 | 2.12.36.4 | 2.12.36.4 | WHQL Certified. Includes VCP and D2XX. Available as a setup executable Please read the Release Notes and Installation Guides . |

¹ Other development environments (e.g. VSCode) will be added in future document revisions

² Linux guidelines will be added in future document revisions

³ Although it is possible to program flash of ESP32-C3-DevKitM-1 without ESP PROG via USB cable and bootloader, such approach does not offer debugging capabilities; it is important to use external hardware debuggers such as ESP PROG to provide debugging capabilities; other debuggers (e.g. JLINK) may be covered in future revisions of this document

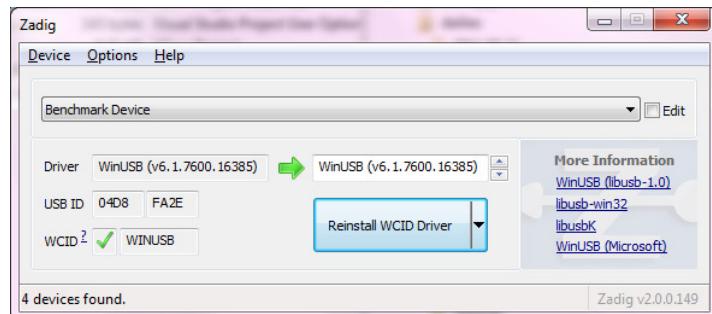
Espressif IDE will install **WinUSB drivers** during the installation process but it is important also to install **original FTDI drivers** used both for ESP32-C3-DevKitM-1 and ESP PROG.

IMPORTANT: Development environment is sensitive to combination which WinUSB / FTDI device drivers versions are used – if the right combination is not set up properly (manually!) hardware debugging will not work! Detailed description how to set up right versions of device drivers (WinUSB vs FTDI) will be described in the following section.

1.3 Setting up correct driver versions for ESP32-C3

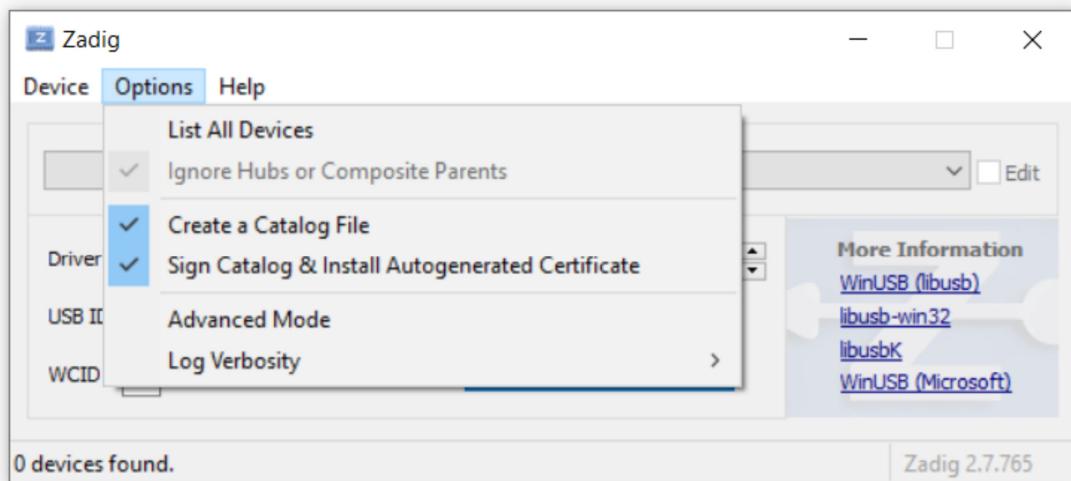
Install USB drivers management Zadig tool from:

<https://zadig.akeo.ie/>

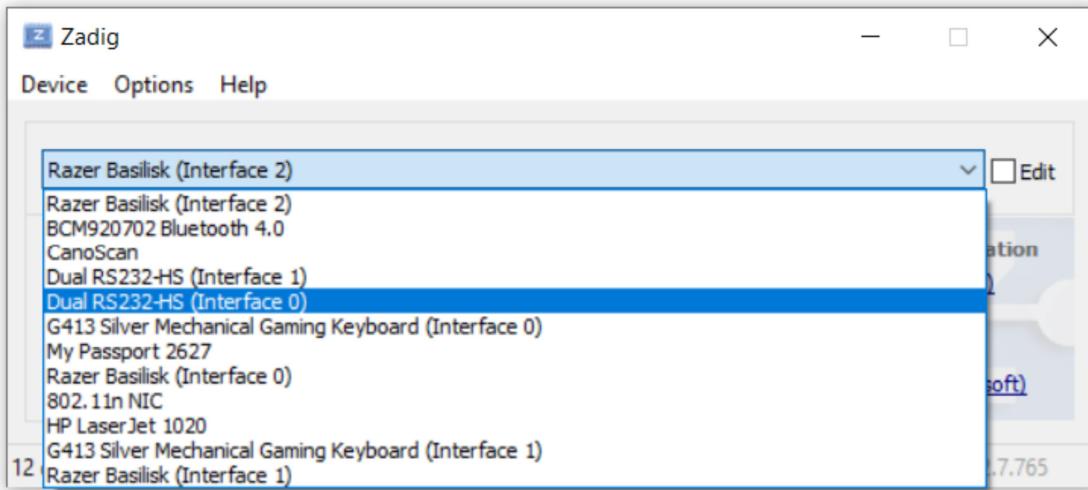


The following steps are very important to enable ESP PROG to work correctly:

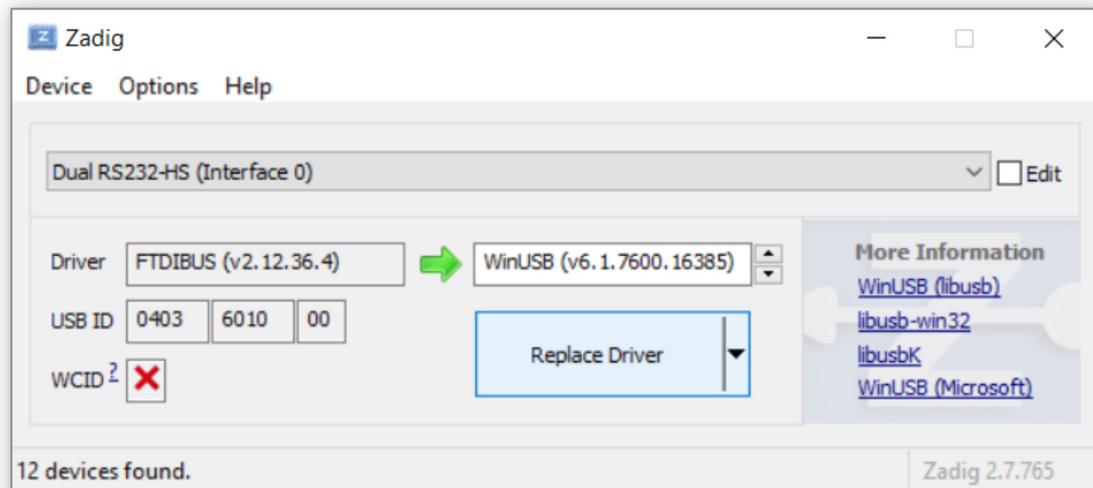
1. Plug ESP PROG into PC and wait for Windows to install FTDI device drivers
2. Run Zadig tool; select *List All Devices*:

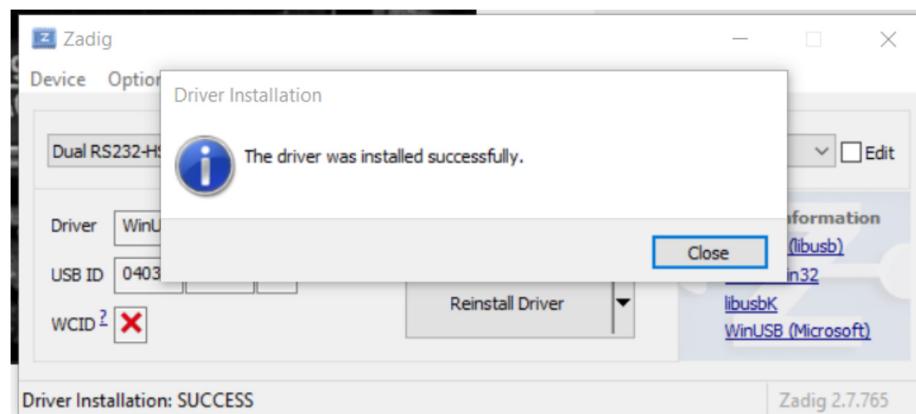
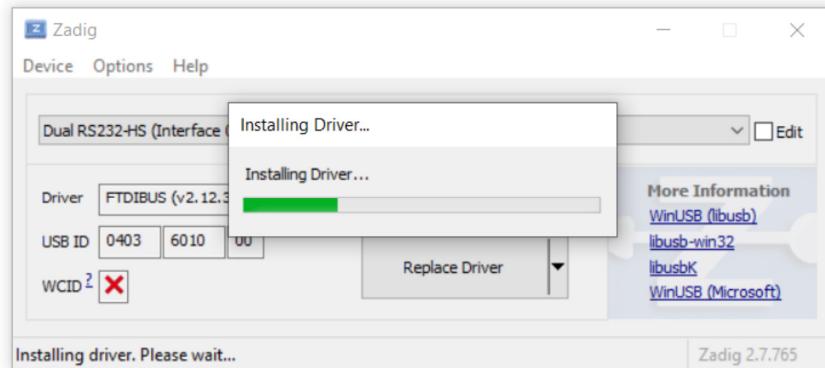


3. Find Dual RS232-HS **interface 0**:

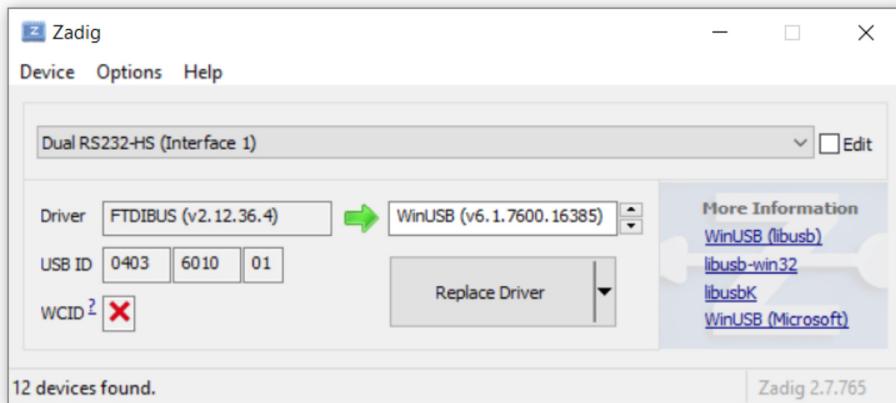


4. Replace existing FTDI driver for **Dual RS232-HS interface 0** with WinUSB driver:





5. **VERY IMPORTANT:** DO NOT REPLACE existing FTDI driver for Dual RS232-HS interface 1 with WinUSB driver:

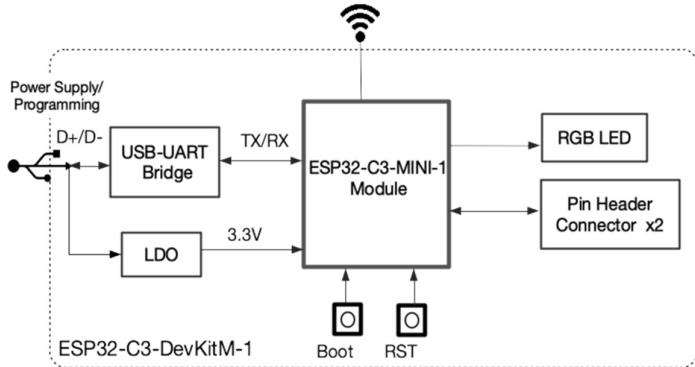


IMPORTANT: Connect ESP PROG before running Zadig, otherwise, drivers (interface 0/1) will not be displayed on a list!

2 Setting up ESP32-C3-DevKitM-1 for debugging via ESP PROG

2.1 Notes about ESP32-C3-DevKitM-1 hardware configuration

Out of the box [ESP32-C3-DevKitM-1](#) IS NOT READY FOR IN-CIRCUIT debugging!



The initial HW setup has the following configuration for programming/debugging:

- USB is connected to ESP32-C3 UART port for with bootloader programming support and diagnostics output console
- USB is connected to the [internal JTAG](#) debugger BUT please keep in mind:
 - o internal JTAG debugger is [not functional](#) since it [physically shares USB interface with UART](#); it is necessary to resolder two 0402 resistors to make hardware change to choose internal JTAG instead of UART for bootloader and diagnostics
 - o [external JTAG is not accessible](#) via TMS, TCK, TDI, TDI pins because [eFuse](#) is initially programmed that [only internal JTAG is accessible](#)
- external JTAG interface via TMS, TCK, TDI, TDI is not accessible because of eFuse factory default settings

This factory default does not allow the use of external JTAG programming/debugging capabilities. One has two choices:

- modify hardware to enable internal JTAG instead of bootloader UART
 - o PROBLEMS:
 - UART debug output provided by ESP-IDF is very useful tool for monitoring runtime state and to provide fallback option in case of any problems with JTAG programmer/debugger
 - Windows drivers for internal JTAG support are not stable – it is better to avoid internal JTAG programmer approach altogether for both said reasons!

- leave hardware as is (i.e. no changes to two 0402 resistors) and burn eFuse to disable internal JTAG to enable external JTAG
 - o then connect ESP PROG or JLINK via external pins

In the next paragraphs both approaches will be documented in more details although option 1 is not recommended.

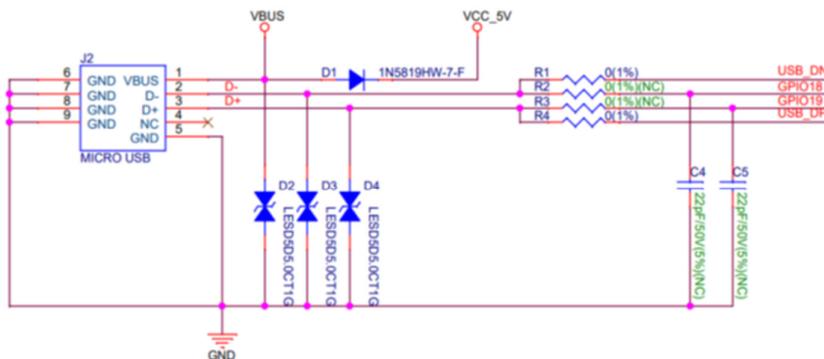
2.2 Enabling internal JTAG support for ESP32-C3-DevKitM-1

ESP32-C3 pins and USB signals:

| ESP32-C3 Pin | USB Signal |
|--------------|------------|
| GPIO18 | D- |
| GPIO19 | D+ |
| 5V | V_BUS |
| GND | Ground |

If internal JTAG debug is intended to be used, please verify that the ESP32-C3 pins used for USB communication are not connected to some other HW that may disturb the JTAG operation.

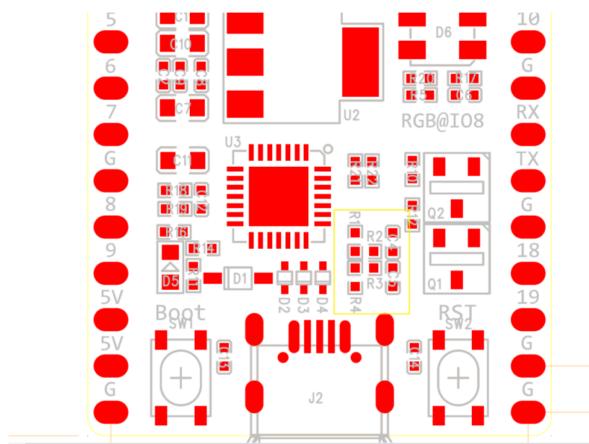
ESP32-C3-DevKitM-1 schematic:



If we want to use internal JTAG, GPIO18 and GPIO19 must be connected to D- and D+. Factory default for ESP32-C3-DevKitM-1 does not allow it – R2 and R3 are N.C.! R1 and R2 are O.R.O meaning that D+ and D- are connected to USB_DN and ESB_DP on a chip. For more information please refer to:

https://dl.espressif.com/dl/schematics/SCH_ESP32-C3-DEVKITC-02_V1_1_20210126A.pdf

https://dl.espressif.com/dl/schematics/PCB_ESP32-C3-DevKitC-02_V1_1_20210121AA.pdf



Note: changing the resistors will enable internal JTAG access, but disable UART bootloader support!
The operation is reversible but still not ideal solution, suitable approach only in case when external programmers (such as ESP PROG and JLINK) are not available.

Important:

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/jtag-debugging/index.html>

Debugging through the USB interface implemented in ESP32-C3 requires to have a **chip with revision 3 or newer**. Other debugging options (e.g. with ESP-Prog) are mandatory for chip revision 1 and 2. The easiest way to determine the chip revision is to look for the Chip is ESP32-C3 (revision 3) message near the end of a successful chip flashing done by `idf.py flash`.

```
Serial port COM8
Connecting...
Chip is ESP32-C3 (revision 3)
Features: Wi-Fi
Crystal is 40MHz
```

Another option:

```
C:\Espressif\frameworks\esp-idf-v4.4.2>esptool.py --port COM7 flash_id
esptool.py v3.3.2-dev
Serial port COM7
Connecting...
Detecting chip type... ESP32-C3
Chip is ESP32-C3 (revision 3)
Features: Wi-Fi
Crystal is 40MHz
MAC: 84:f7:03:40:2f:d0
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

2.3 Enabling external JTAG support for ESP32-C3-DevKitM-1

Using external JTAG tool is very easy but the important prerequisite is to disable internal JTAG on ESP32-C3. This is done by burning fuse DIS_USB_JTAG (changing it from 0 => 1)!

Some useful references:

<https://chowdera.com/2022/02/202202040556046742.html>

It is helpful first to examine the current state of all fuses. It may be done by issuing the command from CLI (ESP-IDF 4.4 CMD):

```
espefuse.py --port COM7 summary
```

where port is serial port on PC (ESP32 bootloader support comm port). The result:

```
C:\Espressif\frameworks\esp-idf-v4.4.2> espefuse.py --port COM6 summary
Connecting....
Detecting chip type... ESP32-C3
espefuse.py v3.3.2-dev

==== Run "summary" command ====
EFUSE_NAME (Block) Description      = [Meaningful Value] [Readable/Writeable] (Hex Value)
-----
Config fuses:
DIS_ICACHE (BLOCK0)                Disables ICache          =
False R/W (0b0)
...
Usb Config fuses:
DIS_USB_JTAG (BLOCK0)              Disables USB JTAG. JTAG access via pads is
control = False R/W (0b0)           led separately          =
DIS_USB_DEVICE (BLOCK0)            Disables USB DEVICE      =
False R/W (0b0)
USB_EXCHG_PINS (BLOCK0)           Exchanges USB D+ and D- pins =
False R/W (0b0)
DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE (BLOCK0)   Disables USB-Serial-JTAG download feature in
UART    = False R/W (0b0)           download boot mode
...

```

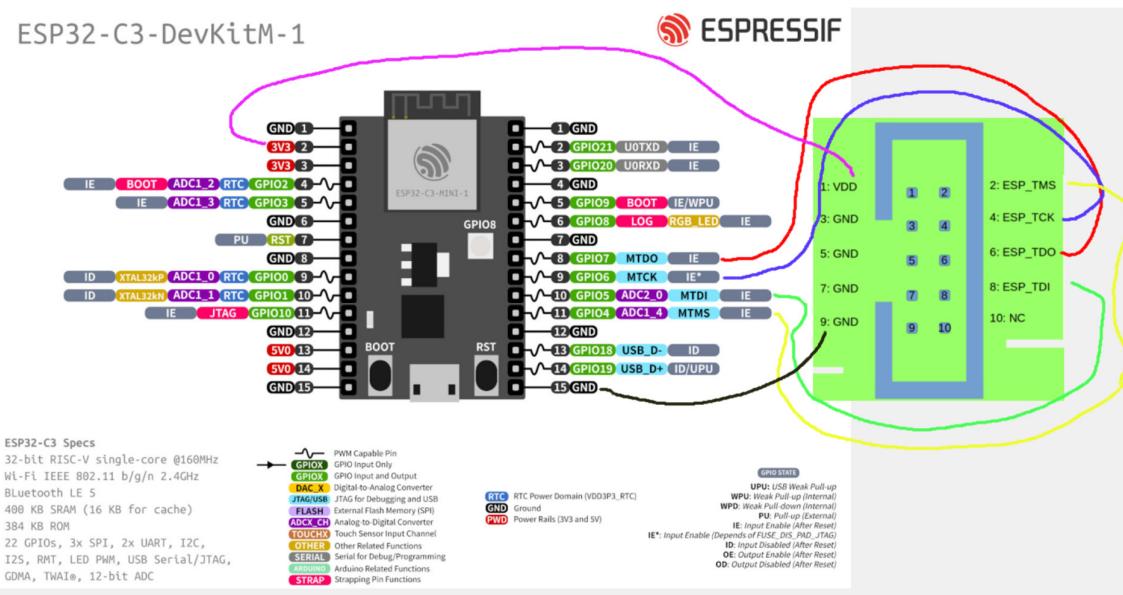
Now we have to burn DIS_USB_JTAG efuse - this is a one-time (irreversible) operation, afterwards JTAG will only be accessible over GPIOs. Built-in USB to Serial part of USB SERIAL JTAG peripheral will still work.

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-guides/jtag-debugging/configure-other-jtag.html>

This is what is needed to burn the fuse:

```
espefuse.py --port COM7 burn_efuse DIS_USB_JTAG
```

Now we can use external ESP PROG! Wiring:

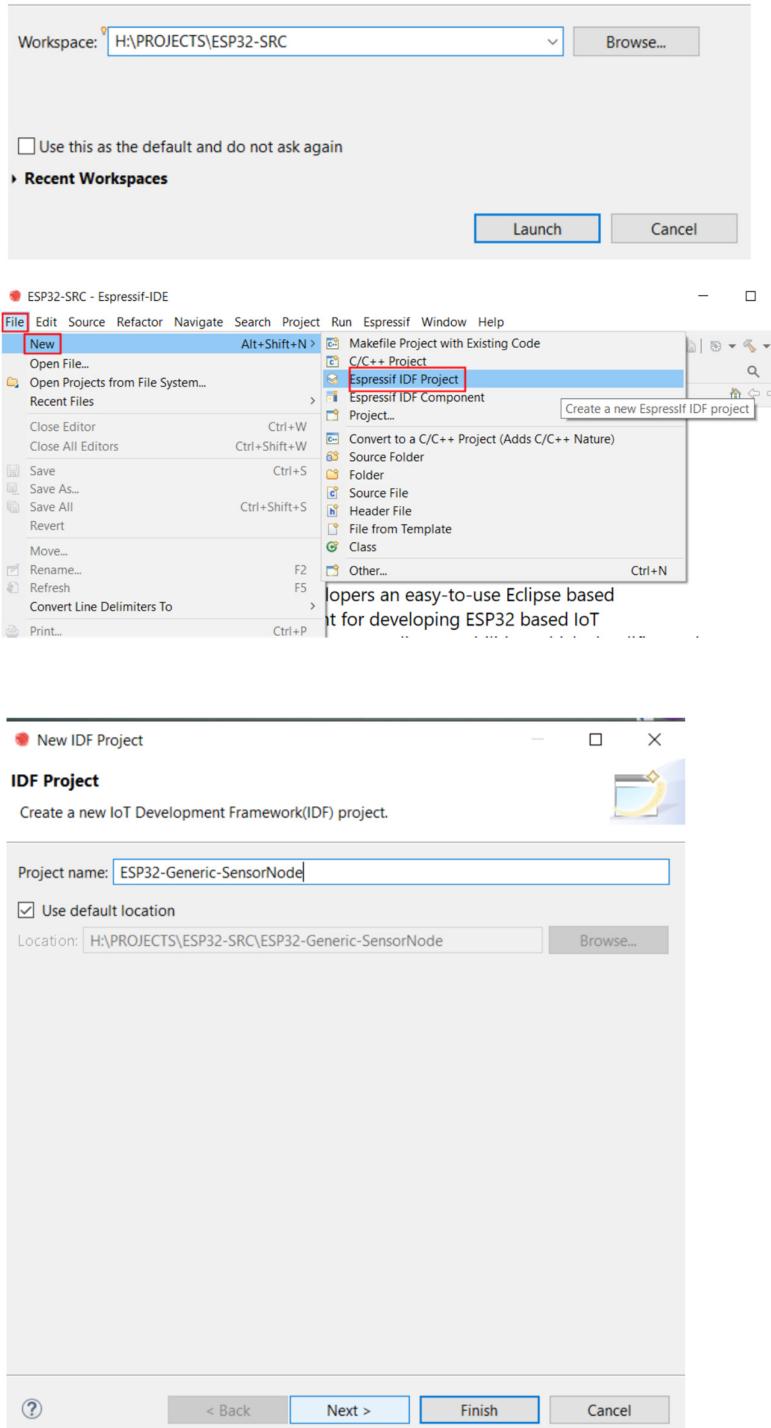


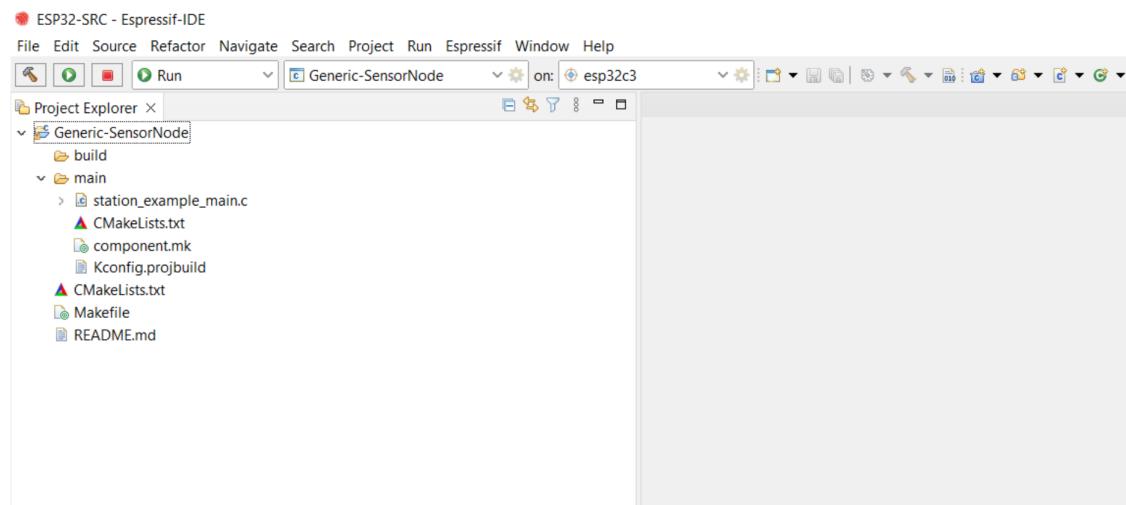
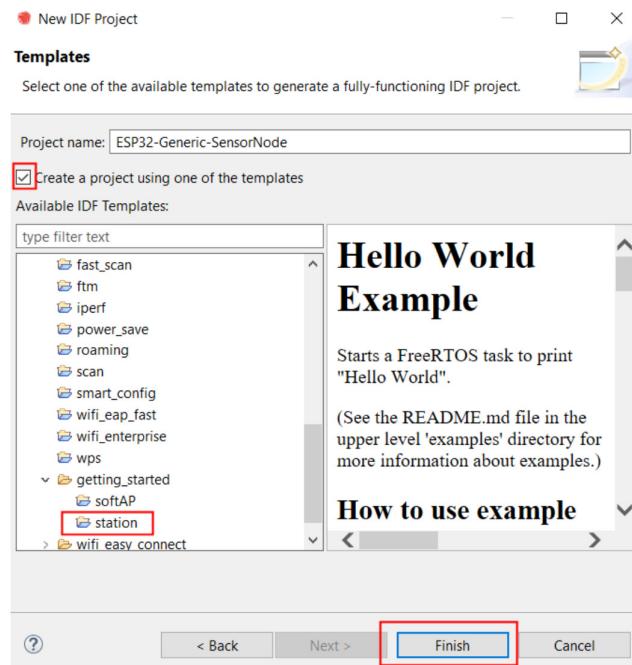
Note: when a physical port on PC to which ESP PROG is connected is changed (i.e. you simply connect USB cable to a different physical port), you need to re-run driver installation procedure!!! This introduces inconveniences in daily work so it is advisable to use one fixed USB port on a PC for ESP PROG connection to minimize the hassle! Each port change where you connect ESP PROG on PC requires another procedure of updating which USB driver you use! (WinUSB vs FT232!)!

3 Creating project in Espressif IDE

3.1 Starting new project from the scratch

Set workspace folder (even if it does not exist, to start with clean project).



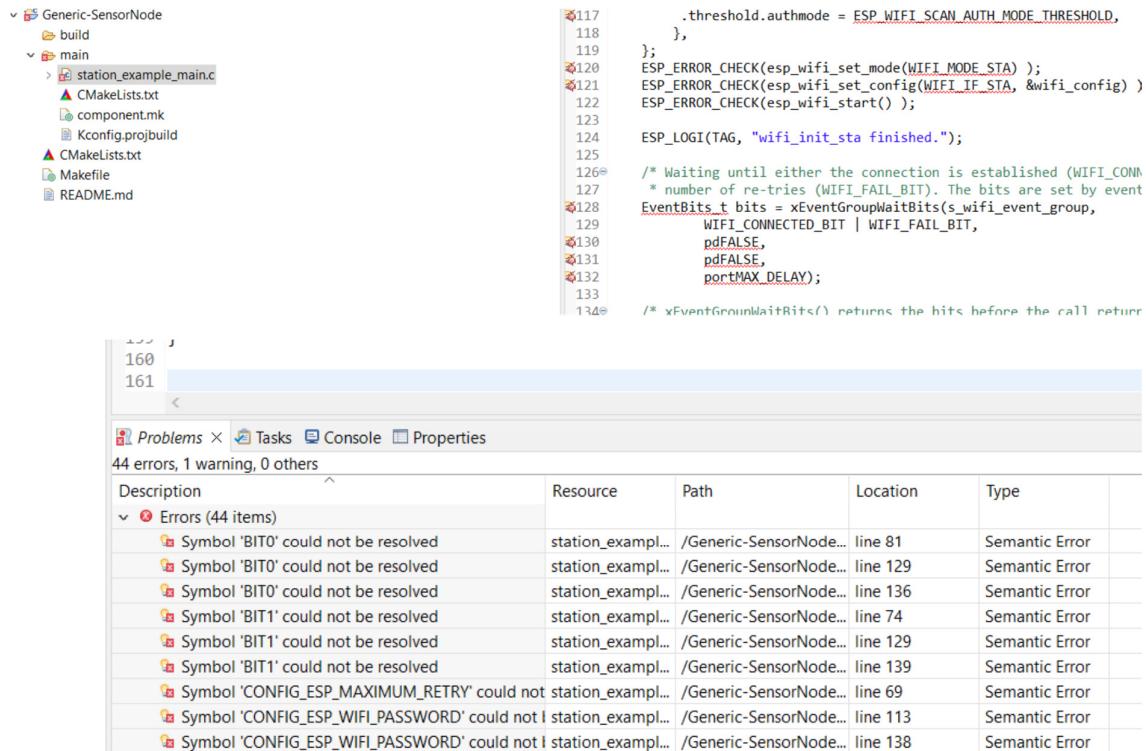


(note: you may rename project by right click on the Project Explorer root)

Then call *Project – Build Project*

NOTE: it takes some longer time to build the project for the first time, since all ESP-IDF components (libraries) are built from the source.

Before build some error indicators may be shown:



```

117     .threshold.authmode = ESP_WIFI_SCAN_AUTH_MODE_THRESHOLD,
118 },
119 };
120 ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
121 ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) )
122 ESP_ERROR_CHECK(esp_wifi_start() );
123
124 ESP_LOGI(TAG, "wifi_init_sta finished.");
125
126 /* Waiting until either the connection is established (WIFI_CONN
127 * number of re-tries (WIFI_FAIL_BIT). The bits are set by event
128 EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
129 WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
130 pdFALSE,
131 pdFALSE,
132 portMAX_DELAY);
133
134 */ xEventGroupWaitBits() returns the hits before the call return

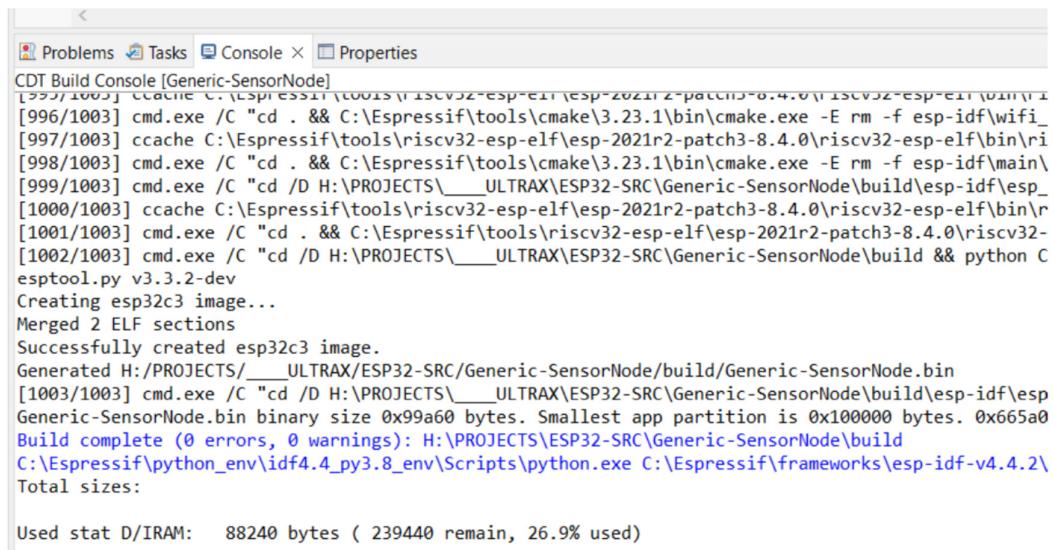
```

Problems X Tasks Console Properties

44 errors, 1 warning, 0 others

| Description | Resource | Path | Location | Type |
|---|-------------------|------------------------|----------|----------------|
| Errors (44 items) | | | | |
| Symbol 'BIT0' could not be resolved | station_exampl... | /Generic-SensorNode... | line 81 | Semantic Error |
| Symbol 'BIT0' could not be resolved | station_exampl... | /Generic-SensorNode... | line 129 | Semantic Error |
| Symbol 'BIT0' could not be resolved | station_exampl... | /Generic-SensorNode... | line 136 | Semantic Error |
| Symbol 'BIT1' could not be resolved | station_exampl... | /Generic-SensorNode... | line 74 | Semantic Error |
| Symbol 'BIT1' could not be resolved | station_exampl... | /Generic-SensorNode... | line 129 | Semantic Error |
| Symbol 'BIT1' could not be resolved | station_exampl... | /Generic-SensorNode... | line 139 | Semantic Error |
| Symbol 'CONFIG_ESP_MAXIMUM_RETRY' could not be resolved | station_exampl... | /Generic-SensorNode... | line 69 | Semantic Error |
| Symbol 'CONFIG_ESP_WIFI_PASSWORD' could not be resolved | station_exampl... | /Generic-SensorNode... | line 113 | Semantic Error |
| Symbol 'CONFIG_ESP_WIFI_PASSWORD' could not be resolved | station_exampl... | /Generic-SensorNode... | line 138 | Semantic Error |

After successful build (it may take several minutes) all red marks should be gone and the result of successful build should be displayed:



```

CDT Build Console [Generic-SensorNode]
[996/1003] cmd.exe /C "cd . && C:\Espressif\tools\cmake\3.23.1\bin\cmake.exe -E rm -f esp-idf\wifi_
[997/1003] ccache C:\Espressif\tools\riscv32-esp-elf\esp-2021r2-patch3-8.4.0\riscv32-esp-elf\bin\riscv32-esp-elf
[998/1003] cmd.exe /C "cd . && C:\Espressif\tools\cmake\3.23.1\bin\cmake.exe -E rm -f esp-idf\main\
[999/1003] cmd.exe /C "cd /D H:\PROJECTS\__ULTRAX\ESP32-SRC\Generic-SensorNode\build\esp-idf\esp_32s2\esp32c3
[1000/1003] ccache C:\Espressif\tools\riscv32-esp-elf\esp-2021r2-patch3-8.4.0\riscv32-esp-elf\bin\riscv32-esp-elf
[1001/1003] cmd.exe /C "cd . && C:\Espressif\tools\riscv32-esp-elf\esp-2021r2-patch3-8.4.0\riscv32-esp-elf
[1002/1003] cmd.exe /C "cd /D H:\PROJECTS\__ULTRAX\ESP32-SRC\Generic-SensorNode\build && python C:\Espressif\python_env\idf4.4_py3.8_env\Scripts\python.exe C:\Espressif\frameworks\esp-idf-v4.4.2\build\esp32c3\esp32c3.py
Creating esp32c3 image...
Merged 2 ELF sections
Successfully created esp32c3 image.
Generated H:/PROJECTS/__ULTRAX/ESP32-SRC/Generic-SensorNode/build/Generic-SensorNode.bin
[1003/1003] cmd.exe /C "cd /D H:/PROJECTS/__ULTRAX/ESP32-SRC/Generic-SensorNode/build\esp-idf\esp_32s2\esp32c3\esp32c3.py
Generic-SensorNode.bin binary size 0x99a60 bytes. Smallest app partition is 0x100000 bytes. 0x665a0
Build complete (0 errors, 0 warnings): H:/PROJECTS/ESP32-SRC/Generic-SensorNode/build
C:\Espressif\python_env\idf4.4_py3.8_env\Scripts\python.exe C:\Espressif\frameworks\esp-idf-v4.4.2\build\esp32c3\esp32c3.py
Total sizes:

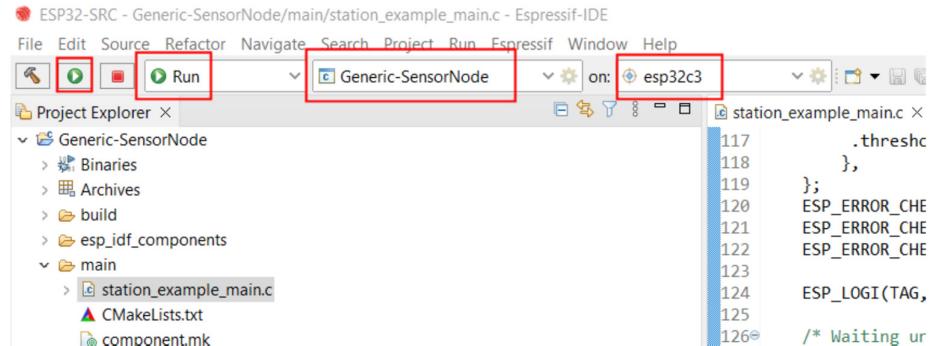
Used stat D/IRAM: 88240 bytes ( 239440 remain, 26.9% used)

```

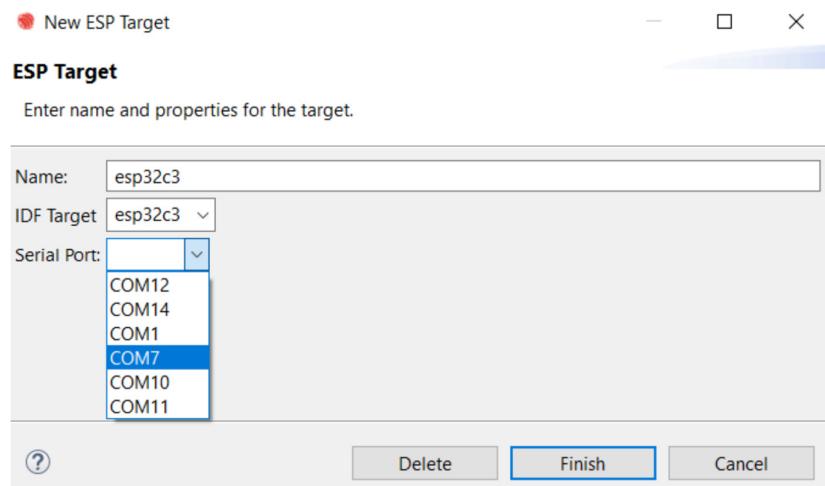
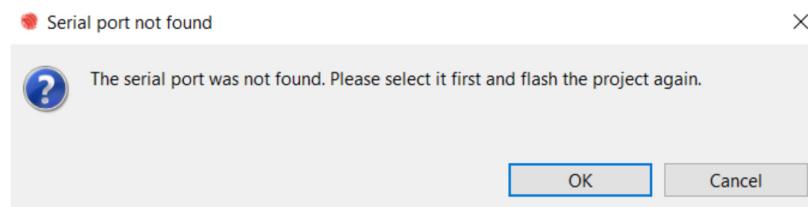
There are two ways to program the target⁴

- via serial port (via bootloader, no debugging)
- via JTAG (e.g. ESP PROG)

Let's run the example:

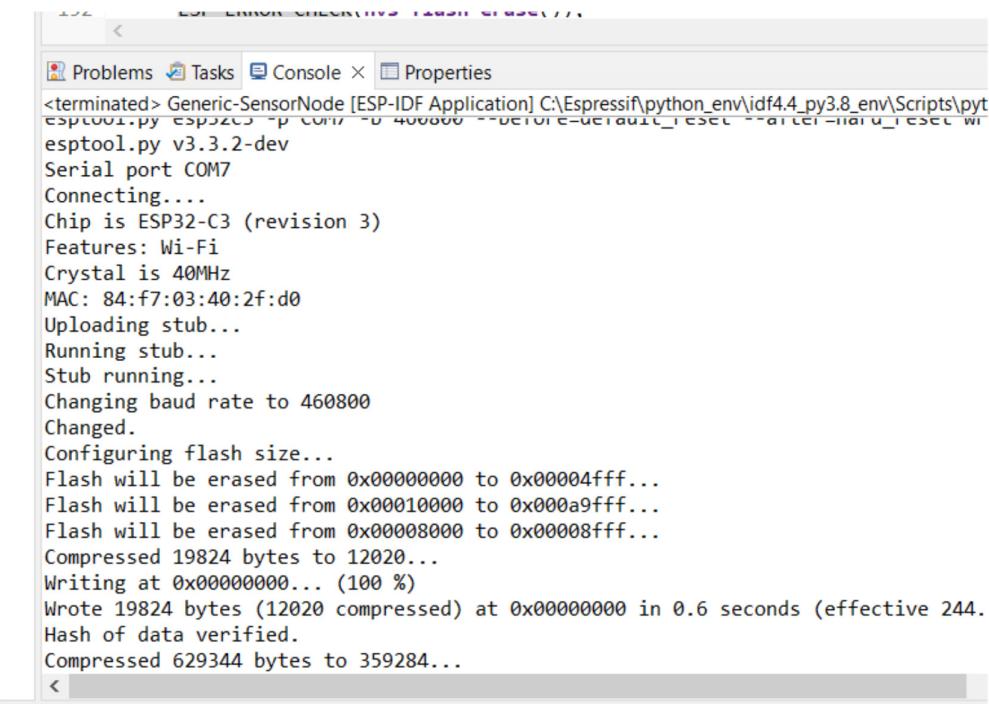


If this is a first run and serial port is not defined:

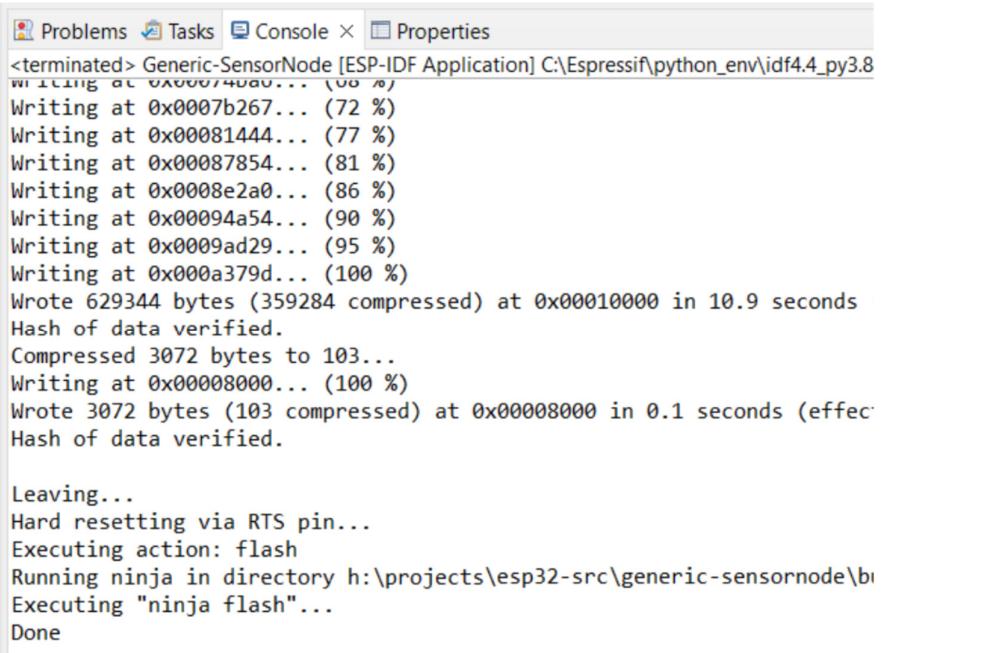


⁴ This guide assumes that ESP PROG is used for external debugging.

Run again:



```
<terminated> Generic-SensorNode [ESP-IDF Application] C:\Espressif\python_env\idf4.4_py3.8\Scripts\pyt
esp32\esp32.py com1 -b 400000 --before-default_reset --after-hard_reset wi
esptool.py v3.3.2-dev
Serial port COM7
Connecting....
Chip is ESP32-C3 (revision 3)
Features: Wi-Fi
Crystal is 40MHz
MAC: 84:f7:03:40:2f:d0
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00000000 to 0x00004fff...
Flash will be erased from 0x00010000 to 0x000a9fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 19824 bytes to 12020...
Writing at 0x00000000... (100 %)
Wrote 19824 bytes (12020 compressed) at 0x00000000 in 0.6 seconds (effective 244.
Hash of data verified.
Compressed 629344 bytes to 359284...
```

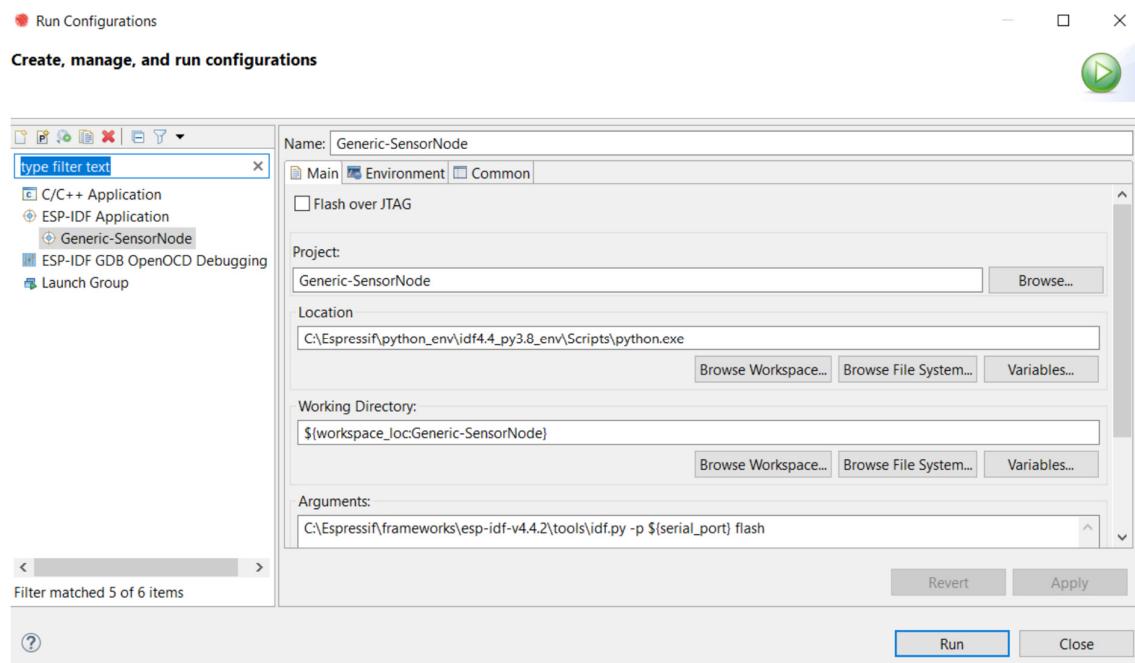
```
<terminated> Generic-SensorNode [ESP-IDF Application] C:\Espressif\python_env\idf4.4_py3.8
writing at 0x00007400... (00 %)
Writing at 0x0007b267... (72 %)
Writing at 0x00081444... (77 %)
Writing at 0x00087854... (81 %)
Writing at 0x0008e2a0... (86 %)
Writing at 0x00094a54... (90 %)
Writing at 0x0009ad29... (95 %)
Writing at 0x000a379d... (100 %)
Wrote 629344 bytes (359284 compressed) at 0x00010000 in 10.9 seconds
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effec
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Executing action: flash
Running ninja in directory h:\projects\esp32-src\generic-sensornode\b
Executing "ninja flash"...
Done
```

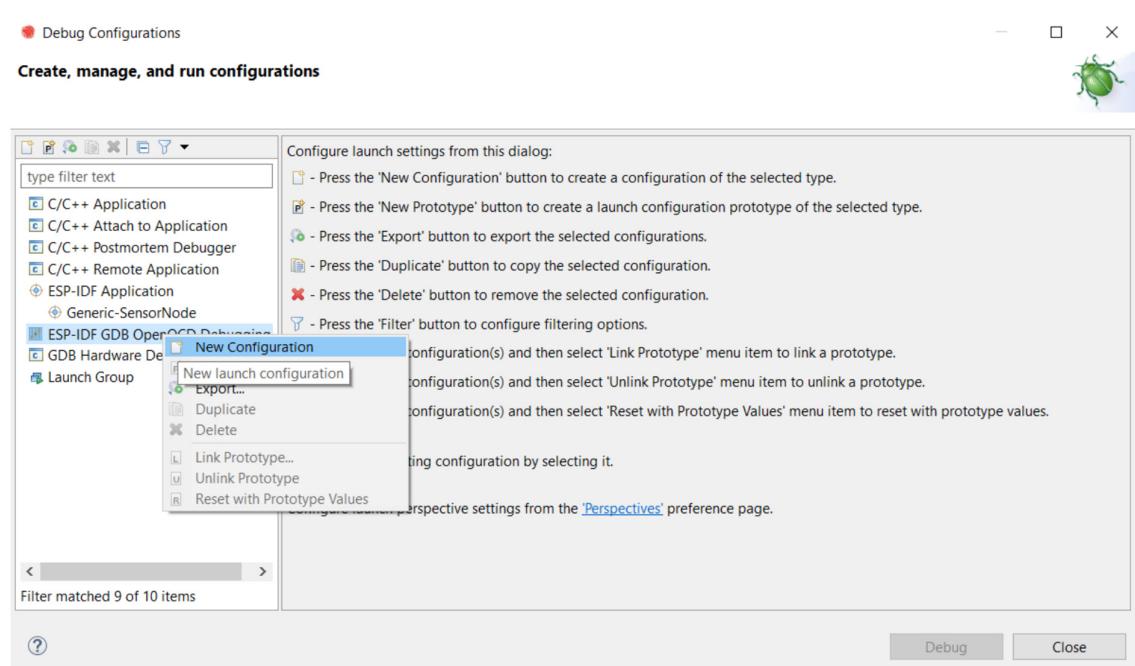
The content of Console output window indicates that target has been programmed.

This approach will upload last firmware build but debugging is not possible.

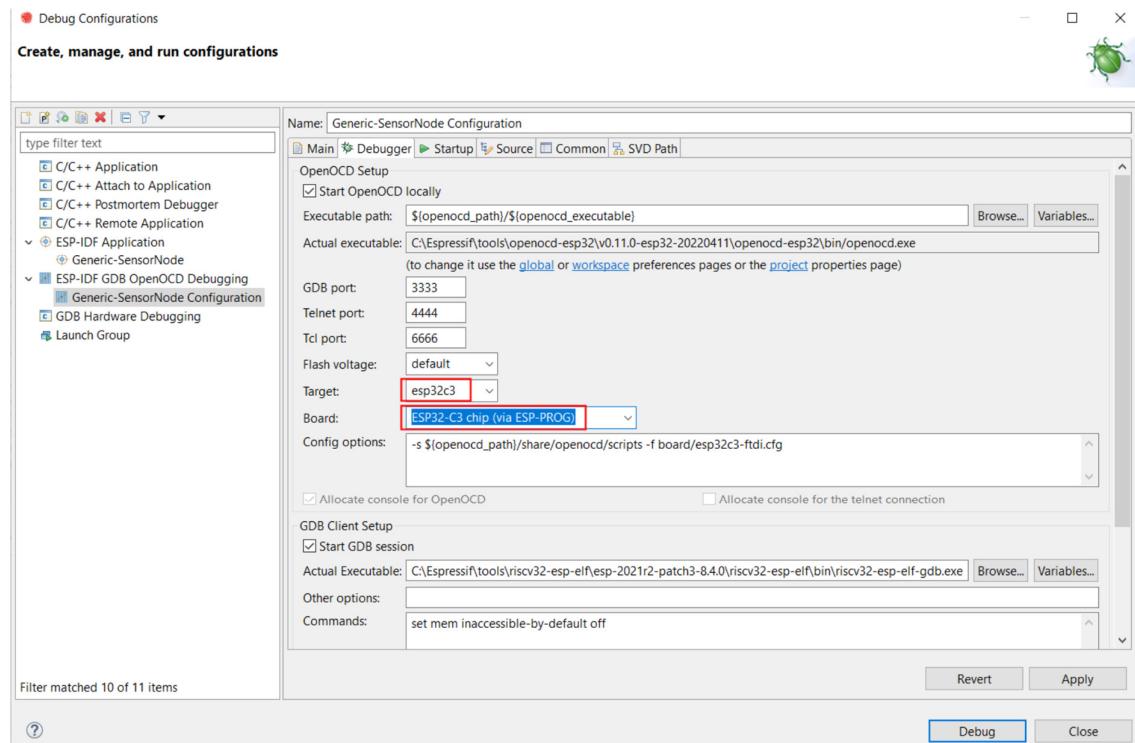
If you click Run-Run Configurations:



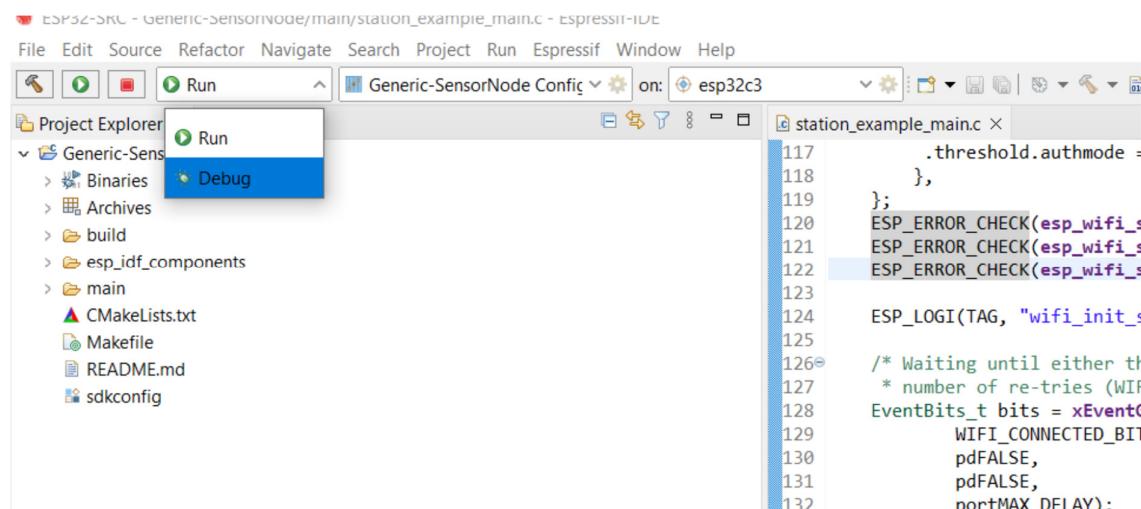
However, we want to have Debug session started over JTAG during development as a faster mean to upload firmware and have capability for in-circuit debugging. To do so, call option *Run-Debug Configurations*. Add new debug configuration:



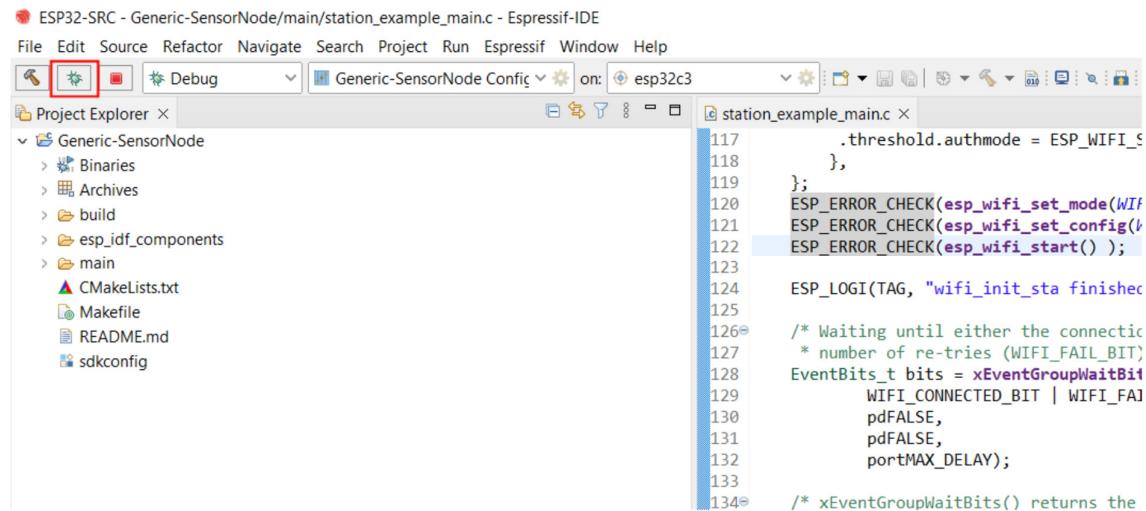
Ensure correct target and debug interface chosen:



Switch to Debug mode:



Start debug session:



The screenshot shows the Espressif-IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Espressif, Window, and Help. A toolbar with various icons is visible above the main workspace. The central area has tabs for 'station_example_main.c' and 'Generic-SensorNode Config'. The 'station_example_main.c' tab is active, showing code for an ESP32 project. The code includes functions like 'esp_wifi_set_mode()', 'esp_wifi_set_config()', and 'esp_wifi_start()'. A line of code at the bottom is highlighted with a blue selection bar. On the left, the 'Project Explorer' shows the project structure under 'Generic-SensorNode', including Binaries, Archives, build, esp_idf_components, main, CMakeLists.txt, Makefile, README.md, and sdkconfig. The 'Debug' button in the toolbar is highlighted with a red box.

Note: if you installed correctly drivers for ESP PROG, you may still experience difficulties with drivers:

If you change physical port where you connect ESP PROG, you need to re-run driver installation procedure!!! Each port change where you connect ESP PROG on PC requires another procedure of updating which USB driver you use! (WinUSB vs FTDI!)

Now you are ready to program and debug your application.