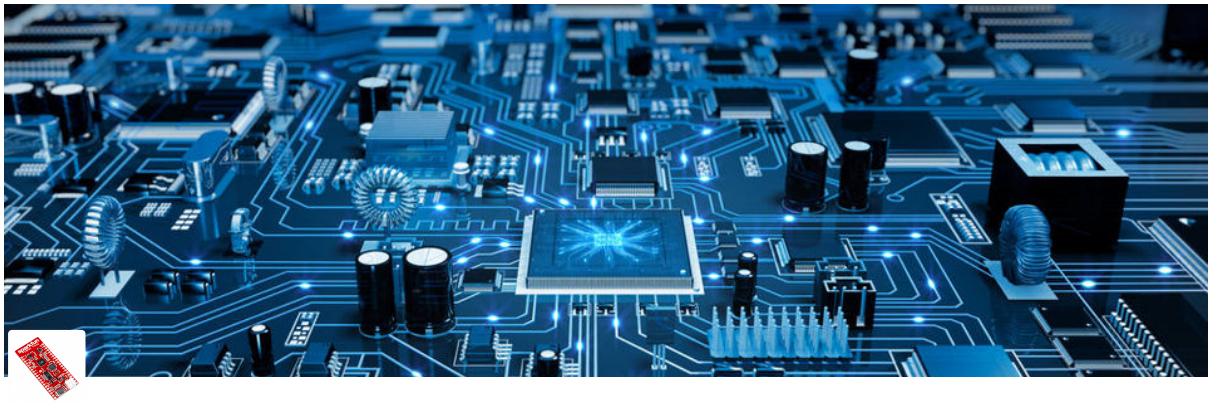


ESP32 development SW installation

Get started with ESP32

Authors: ITS8050 Group G (**Marco Hanisch, Romain Thollot, Antoine Pinon**)

Version: 1.0, reviewed last on 09/04/2020

Part I: Setting up Development Environment

Step 1: Install the prerequisite

▼ For **Linux**

```
# CentOS/RedHat/Fedora
sudo yum install git wget flex bison gperf python cmake ninja-build ccache
# Debian & Ubuntu
sudo apt-get install git wget flex bison gperf python python-pip python-setuptools cmake ninja-build ccache libffi-dev libs
# Arch
sudo pacman -S --needed gcc git make flex bison gperf python-pip cmake ninja ccache
```

```
# Install Python 3 and make it the default version
sudo apt-get install python3 python3-pip python3-setuptools
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 10
```

▼ For **Windows**: https://dl.espressif.com/dl/esp-idf-tools-setup-2.3.exe

This will install:

- Cross-compilers: OpenCOD, cmake, Ninja
- Python 3.7
- Git for Windows

Step 2: Get ESP-IDF

Linux, [Windows]: Create an 'esp' folder at `$HOME` or `%userprofile%` then enter these commands:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

⇒ ESP-IDF will be downloaded into *~/esp/esp-idf*.

Step 3: Set up the tools

▼ For **Windows**

- ESP-IDF Tools Installer creates an ESP-IDF Command Prompt shortcut which can set up all the required environment variables.

ESP-IDF Command Prompt (cmd.exe)

App

- Alternatively, you can use one of the following scripts:

```
cd %userprofile%\esp\esp-idf
install.bat
./install.ps1
```

▼ For **Linux**

```
cd $HOME\esp\esp-idf
./install.sh
```

⇒ The script will create a folder `$HOME/.espressif` and install every tools it needs.

Step 4: Set up the environment variables

▼ For **Windows**

- ESP-IDF Tools Installer introduced in Step 1 installs all the required tools.
- Another way to install these tools:

```
%userprofile%\esp\esp-idf\export.bat
.$HOME/esp/esp-idf/export.ps1
```

▼ For **Linux**

```
. $HOME/esp/esp-idf/export.sh
#echo ". $HOME/esp/esp-idf/export.sh" >> $HOME/.bash_profile
```

Tips: Adding this line to your `.profile` or `.bash_profile` script will automate this step, making ESP-IDF tools available in every terminal.

⇒ Result should look like the following:

```
C:\Users\      \esp\esp-idf>%userprofile%\esp\esp-idf\export.bat
Setting IDF_PATH: C:\Users\      \esp\esp-idf

Adding ESP-IDF tools to PATH...
    C:\Users\      \.espressif\tools\xtensa-esp32-elf\esp-2020r1-8.2.0\xtensa-esp32-elf\bin
    C:\Users\      \.espressif\tools\xtensa-esp32s2-elf\esp-2020r1-8.2.0\xtensa-esp32s2-elf\bin
    C:\Users\      \.espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf-binutils\bin
    C:\Users\      \.espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf-binutils\bin
    C:\Users\      \.espressif\tools\cmake\3.16.4\bin
    C:\Users\      \.espressif\tools\openocd-esp32\v0.10.0-esp32-20200309\openocd-esp32\bin
    C:\Users\      \.espressif\tools\ninja\1.10.0\
    C:\Users\      \.espressif\tools\idf-exe\1.0.1\
    C:\Users\      \.espressif\tools\ccache\3.7\
    C:\Users\      \.espressif\python_env\idf4.2_py3.7_env\Scripts
    C:\Users\      \esp\esp-idf\tools

Checking if Python packages are up to date...
Python requirements from C:\Users\      \esp\esp-idf\requirements.txt are satisfied.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

  idf.py build
```

*Note: `idf.py` script must be run in a project, not in `$HOME/esp/esp-idf`. Otherwise, you will get an error `idf.py command not found`.

Part II: Creating Your First Project

Our first project will consist in a simple hello_world program using the template given in the example folder of the esp-idf project.

Step 5: Start a Project

a) Copy get-started/hello_world to ~/esp directory

▼ For **Windows**

```
cd %userprofile%\esp
xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
```

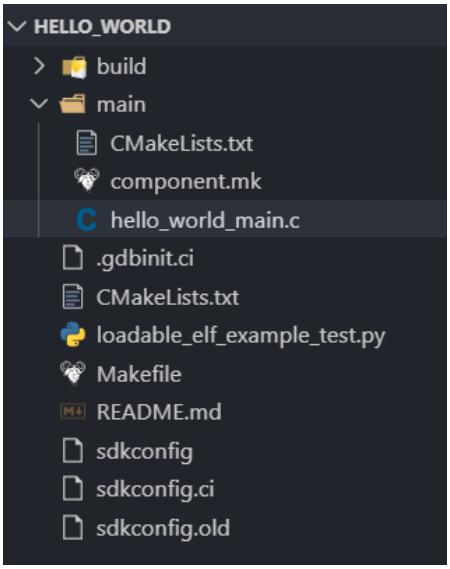
▼ For **Linux**

```
cd ~/esp
cp -r $IDF_PATH/examples/get-started/hello_world .
```

Important

The ESP-IDF build system does not support spaces in the paths to either ESP-IDF or to projects.

b) Project Structure



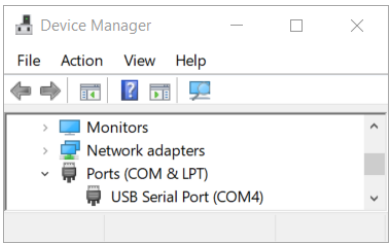
Step 6: Connect your device

Connecting our ESP32 board to the computer/raspberryPi and check under what serial port the board is visible.

Note: Serial ports have the following patterns in their names:

- **Windows**: names like `COM1`
- **Linux**: starting with `/dev/tty`

▼ For **Windows**: Check the device Manager after connecting the ESP-32 to a USB port.



▼ For **Linux**: /dev/ttyUSB0 should appears after connecting the device.

Troublshooting:

- Are the light on ESP32 working? Check the cable
- Gives propers authorizations to the user
- Install drivers from espressif.com.

Step 7: Configure

Here we need to set ESP32 chip as the target and run the project configuration utility menuconfig.

▼ For **Windows**

```
cd %userprofile%\esp\hello_world
idf.py set-target esp32
idf.py menuconfig
```

```
C:\Users\      \esp\hello_world>idf.py set-target esp32
Adding "set-target"'s dependency "fullclean" to list of commands with default set of options.

Executing action: fullclean
Executing action: set-target
Set Target to: esp32, new sdkconfig created. Existing sdkconfig renamed to sdkconfig.old.
Running cmake in directory c:\users\      \esp\hello_world\build
Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1 --warn-uninitialized -DIDF
_TARGET=esp32 -DCCACHE_ENABLE=1 c:\users\antoi\esp\hello_world"...
Warn about uninitialized values.
-- Found Git: C:/Program Files/Git/cmd/git.exe (found version "2.26.0.windows.1")
```

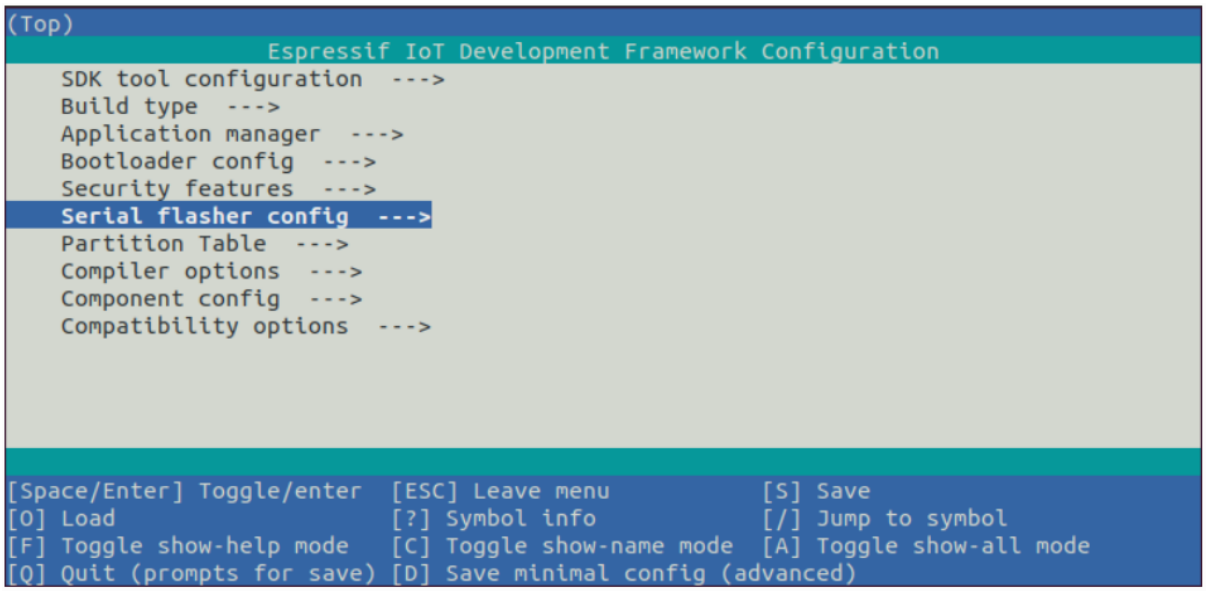
[...]

```
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/      /esp/hello_world/build
```

▼ For **Linux**

```
cd ~/esp/hello_world
idf.py set-target esp32
idf.py menuconfig
```

The command `idf.py menuconfig` will return the menu below. Here we can access and tweak pretty much everything on the board: processor, partition, bootloader, Wi-Fi, etc.



Project configuration - Home window

Step 8: Build the Project

Build the project by running idf.py build. This will compile the everything and generate bootloader, partition table and app. binaries. In other words, a **Firmware,** in the form of a *.bin* file.

```
idf.py build
```

⇒ Command Execution on a Windows machine:

```
[56/62] Building C object esp-idf/bootloader_support/CMakeFiles/__idf_bootloader
_support.dir/src/esp32/secure_boot.c.obj
[57/62] Linking C static library esp-idf\bootloader_support\libbootloader_suppor
t.a
[58/62] Linking C static library esp-idf\efuse\libefuse.a
[59/62] Linking C static library esp-idf\spi_flash\libspi_flash.a
[60/62] Linking C static library esp-idf\main\libmain.a
[61/62] Linking C executable bootloader.elf
[62/62] Generating binary image from built executable
esptool.py v2.8
Generated C:/Users/      /Desktop/esp-idf/hello_world/build/bootloader/bootloade
r.bin
[820/820] Generating binary image from built executable
esptool.py v2.8
Generated C:/Users/      /Desktop/esp-idf/hello_world/build/hello-world.bin

Project build complete. To flash, run this command:
C:\Users\      \.espressif\python_env\idf4.0_py3.7_env\Scripts\python.exe ..\com
ponents\esptool_py\esptool\esptool.py -p (PORT) -b 460800 --before default_reset
 --after hard_reset write_flash --flash_mode dio --flash_size detect --flash_fre
q 40m 0x1000 build\bootloader\bootloader.bin 0x8000 build\partition_table\partit
ion-table.bin 0x10000 build\hello-world.bin
or run 'idf.py -p (PORT) flash'
```

Step 9: Flash onto the Device

```
idf.py -p <PORT> [-b <bauds>] flash
# idf.py -p COM4 flash
```

```
C:\Users\      \esp\hello_world>idf.py -p COM4 flash
Executing action: flash
Running ninja in directory c:\users\      \esp\hello_world\build
Executing "ninja flash"...
[1/5] cmd.exe /C "cd /D C:\Users\      \esp\hello_world\build\esp-idf\partition_table...cho
**************************************************************************"
Partition table binary generated. Contents:
*************************************************************************
# ESP-IDF Partition Table
# Name, Type, SubType, Offset, Size, Flags
nvs,data,nvs,0x9000,24K,
phy_init,data,phy,0xf000,4K,
factory,app,factory,0x10000,1M,
*************************************************************************
[2/5] Performing build step for 'bootloader'
```

[...]

```
Wrote 150752 bytes (79569 compressed) at 0x00010000 in 1.9 seconds (effective 638.5 kbit/s)
...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
```

Step 10: Monitor

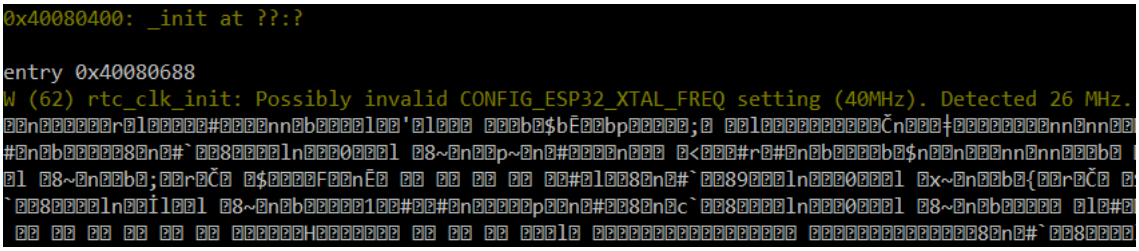
To check if “hello_world” is indeed running, type `idf.py -p <PORT> monitor`. Note that you can also use *putty*, *minicom*, etc with transmission parameters set to `115200-8-1-N`.

- **Normal behaviour**: **Boot** >> **Execution** `Hello world!` >> **Restarting**

```
I (31) boot: ESP-IDF v4.2-dev-1034-ge599b794b 2nd stage bootloader
I (31) boot: compile time 11:36:39
I (31) boot: chip revision: 1
I (35) boot_comm: chip revision: 1, min. bootloader chip revision: 0
I (51) boot.esp32: SPI Speed      : 40MHz
I (51) boot.esp32: SPI Mode       : DIO
I (52) boot.esp32: SPI Flash Size : 2MB
I (56) boot: Enabling RNG early entropy source...
I (62) boot: Partition Table:
I (65) boot: ## Label            Usage          Type ST Offset   Length
I (73) boot:  0 nvs              WiFi data        01 02 00009000 00006000
I (80) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (87) boot:  2 factory          factory app      00 00 00010000 00100000
```

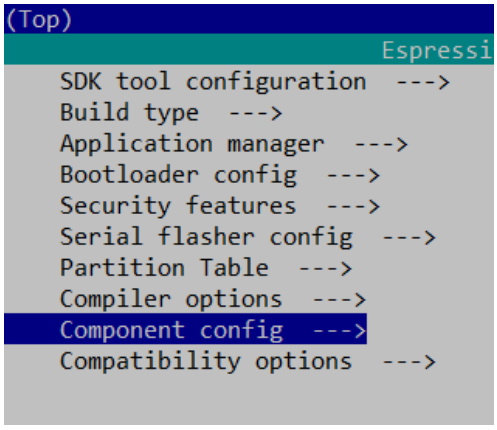
```
Hello world!
This is esp32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 2MB external flash
Free heap: 299328
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
```

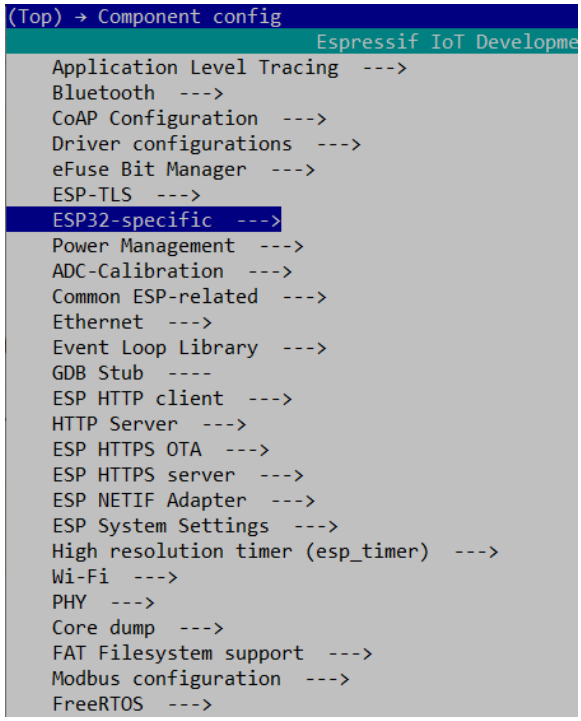
- During our 1st attempt, we encounter an issue regarding ESP-IDF Frequency. Apparently, the board we use (*Sparkfun ESP32 Thing*) runs with a 26MHz crystal. However, by default the frequency is 40MHz. This result in a strange behavior:

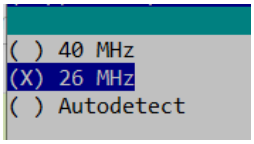


▼ **Troubleshoot** `??n????rl????#?`:

1. Go to the menuconfig : `idf.py menuconfig`
2. Go to *Component config* → *ESP32-specific* → *Main XTAL frequency*, then change CONFIG_ESP32_XTAL_FREQ_SEL to 26 MHz.







3. Build and flash the application again.

```
idf.py build
idf.py -p COM4 flash
```

Part III: Exercises

⇒ Source Code for this part is available at https://gitlab.cs.ttu.ee/rothol/its8050-2020.

Exercise 1: Create two FreeRTOS tasks to do the same thing, outputing two strings of "Hello" and "World" each from each tasks with separate delays.

- **Basic Libraries to use**:

```
/* Standard includes. */
#include <stdio.h>
#include <stdlib.h>

/* FreeRTOS kernel includes. */
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

/* ESP32 includes. */
#include "sdkconfig.h"
#include "esp_system.h"
#include "esp_spi_flash.h"
```

- **Creating a task**:

```
int main(void){

    xTaskCreate(
        taskOne,          /* Name of the Task function to call. */
        "TaskOne",        /* String containing the name of the task. */
        10000,            /* Stack size in bytes, calc. possible, hardcoded here. */
        NULL,             /* Parameter passed as input of the task, (void *) [2] */
        1,                /* Priority of the task. */
        NULL);            /* TaskHandle, return a handle used for reference of the task.
}
```

- **Task function**:

```
// Must have this signature (W: no return segment!)
void vTaskFunctionName( void *pvParameters ){
    for( ; ; )
    {
        //-- Task application code here. --
    }

    /* Tasks must not attempt to return from their implementing
    function or otherwise exit. If it is necessary for a task to
    exit then have the task call vTaskDelete( NULL ) to ensure
    its exit is clean. */
    vTaskDelete( NULL );
}
```

Once done build and flash your device. Eventually, we will need to re-do the steps 4 (export env vars like `idf.py`) and 7 (`set-target` & `menuconfig`) before build.

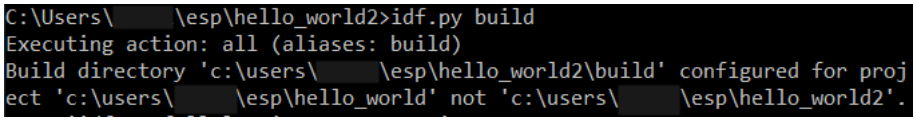
Exercise 2: Create two event loops to achieve the same result.

The task is the same but here we mix up task and loops.

```
xTaskCreate(vTaskHello, "vTaskHello", 10000, NULL, 1, NULL);

for(int i = 0; i < 4; i++){
    //-- Task application code here. --
   printf("World ");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
```

**Note: If the project is built on top of a demo folder, you might encounter configuration issue,*



Run `idf.py fullclean` before building.

Resources

- **Getting started (basic examples)**: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html
- **Standard setup of Toolchain for Windows**: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-setup.html
- **Establish serial connection**: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/establish-serial-connection.html
- **Build explained (select-target)**: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html#selecting-idf-target
- **FreeRTOS overview**: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html
- **Mastering the FreeRTOS Kernel** (book): https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

Annexe I: *hello_world_main.c* example source code

```
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"

void app_main(void)
{
    printf("Hello world!\n");

    /* Print chip information */
    esp_chip_info_t chip_info;
    esp_chip_info(&chip_info);
    printf("This is %s chip with %d CPU cores, WiFi%s%s, ",
            CONFIG_IDF_TARGET,
            chip_info.cores,
            (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
            (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");

    printf("silicon revision %d, ", chip_info.revision);

    printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
            (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");

    printf("Free heap: %d\n", esp_get_free_heap_size());

    for (int i = 10; i >= 0; i--) {
        printf("Restarting in %d seconds...\n", i);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    printf("Restarting now.\n");
    fflush(stdout);
    esp_restart();
}
```