Guía de instalación de ESP-IDF en Visual Studio Code

Yves Maillard Quiroz[1]
[1]Laboratorio de PDS, Facultad de Ingeniería UNAM, CDMX, México

Abril 2025

Abstract

El presente trabajo muestra una guía integrada para la instalación del entorno de desarrollo de Espressif, ESP-IDF, en conjunto y usado a través de Visual Code Studio (VSC). La instalación y uso del software mostrado en esta guía se realizó y aplica a sistemas operativos basados en Linux, en específico, Ubuntu 22.

El documento se divide en siete secciones y abarca desde la instalación de VSC hasta la creación y transferencia de un proyecto a una tarjeta ESP32. El uso de las herramientas especificas de ESP para el desarrollo de software empleado en las tarjetas va más allá del alcance de este documento.

Nota Importante: La instalación del software y ejecución de los comandos presentados a lo largo de este trabajo se realizó desde una cuenta de usuario con permisos de super usuario.

1 Instalación de Visual Studio Code en Ubuntu

Existen diferentes formas de instalar VSC, una de ellas es empleando la interfaz de línea de comandos de Linux y siguiendo lo que se muestra en seguida:

1. Entrar al sitio web de VSC.
2. Descargar el archivo .deb de acuerdo al sistema operativo empleado y a la versión deseada (code_1.99.0-1743632463_amd64.deb).
3. Una vez descargado el archivo en algún subdirectorio, abrir la terminal de Linux (CLI) e instalar la versión descargada de VSC mediante el comando:

```
sudo apt install ./<srchivo>.deb
```

En caso que la distribución de Linux no sea reciente, VSC se puede instalar de manera alternativa con los comandos:

```
sudo dpkg -i <archivo>.deb
sudo apt-get install -f
```

2 Instalación de los prerrequisitos

El ambiente de desarrollo de Espressif (ESP-IDF) requiere de algunos paquetes de software para poder ser utilizado, estos paquetes abarcan:

- Las herramientas necesarias para compilar código para ESP32 (ccache, libffi, libssl, python y python-venv).
- Las herramientas para descargar paquetes de ESP y generar una aplicación para ESP32 (git, wget, flex, bison, gperf, CMake y Ninja).

- Las herramientas para transferir la aplicación a la memoria de la tarjeta (dfu, libusb)
- Las bibliotecas de ESP-IDF (APIs).

En Ubuntu se pueden instalar las herramientas necesarias empleando un comando en la terminal de Linux. Para realizar esto, abrir la terminal de Linux y ejecutar el comando:

```
sudo apt-get install git wget flex bison gperf python3 python3-pip \
python3-venv cmake ninja-build ccache libffi-dev libssl-dev \
dfu-util libusb-1.0-0
```

En caso que se requiera mayor información sobre la instalación de los prerrequisitos se recomienda visitar los enlaces: 1, 2 y 3 mostrados en la *sección Enlaces* de este documento.

2.1 Bibliotecas de ESP-IDF

Las bibliotecas de ESP-IDF permiten y facilitan el desarrollo de aplicaciones para ESP32. Estas bibliotecas sirven como interfaces de software (API) que permiten utilizar el hardware de las tarjetas.

Las bibliotecas de ESP se encuentran almacenadas en un repositorio de Git, por lo que se necesitan descargar empleando éste último. Una vez descargadas, la última versión de ellas queda almacenada en una carpeta generada con nombre *esp-idf*.

En un principio se crea el directorio de espressif y posteriormente se realiza la descarga del entorno de desarrollo con Git. Esto se puede realizar mediante los siguientes comandos en la terminal de Linux:

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

Una vez descargado ESP-IDF es necesario instalar las herramientas usadas por ESP-IDF. Para esto, Espressif proporciona un script en múltiples lenguajes que se encarga de realizar la instalación. En Bash, éste se llama *install.sh* y se encuentra dentro de la carpeta *esp-idf*. Para correrlo se ejecuta lo siguiente en la terminal de Linux:

```
cd ~/esp/esp-idf
./install.sh esp32
```

En caso que se requieran otras tarjetas además de las que cuentan con ESP32, se ejecuta:

```
cd ~/esp/esp-idf
./install.sh all
```

Al realizar lo anterior se descargará la última versión de ESP-IDF, sin embargo, se recomienda utilizar la versión estable actual, ésta se puede consultar en el menú mostrado a la izquierda dentro del sitio web de la documentación de espressif https://docs.espressif.com/projects/esp-idf/en/latest/esp32/ y marcada como *stable*.

Para actualizar a la versión estable deseada se pueden utilizar los siguientes comandos en la terminal de Linux:

```
cd $IDF_PATH
git fetch
git checkout vX.Y.Z
git submodule update --init --recursive
```

Donde X.Y.Z corresponden a la versión estable que se desee obtener.

Alternativamente se pueden obtener las bibliotecas de ESP-IDF desde la extensión de Espressif en VSC como se muestra en la *sección 4* o en el sitio web https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/installation.html.

3 Configurar las variable de entorno

Si se realiza todo lo descrito en las secciones anteriores, en este punto se pueden crear, cargar y correr aplicaciones de ESP32 en las tarjetas empleando un editor de textos y la terminal de Linux. Pero como se mostrará en la siguiente sección esto se puede hacer empleando el entorno de VSC.

Para poder emplear las herramientas de Espressif, éstas se deben añadir a la variable de entorno PATH y se debe comprobar que los scripts de Python necesarios para generar el proyecto se encuentren. Espressif provee un archivo de Bash nombrado *export.sh* que permite realizar esto, el cual se encuentra dentro de la carpeta *esp-idf*. Una forma de correrlo es mediante el siguiente comando en la terminal de Linux:

```
. $HOME/esp/esp-idf/export.sh
```

4 Instalación de la extensión de VSC

Espressif desarrolló una extensión para VSC que permite descargar, instalar y utilizar las bibliotecas de desarrollo de ESP (ESP-IDF) dentro de VSC. Las siguientes secciones describen como descargar la extensión y emplear ESP-IDF en conjunto con la extensión.

4.1 Instalar la extensión ESP-IDF

Abrir VSC. Ir a la sección de extensiones dentro de la barra de actividades de VSC o mostrarla con la pestaña **View**>Extensions. Buscar la extensión *ESP-IDF* en la barra de búsqueda e instalarla.

4.2 Configurar la extensión ESP-IDF

Una vez instalada la extensión ESP-IDF, se agrega una nueva sección de Espressif a la barra de actividades de VSC. Ésta sección provee la lista de comandos de Espressif con los que cuenta la extensión.

Una forma de configurar la extensión de ESP-IDF es a través del comando *"Configure ESP-IDF Extension"* dentro de la lista previamente mencionada. Al seleccionar éste comando se despliega una ventana en VSC que permite seleccionar entre tres tipos de configuración como se muestra en la *Figura 1*

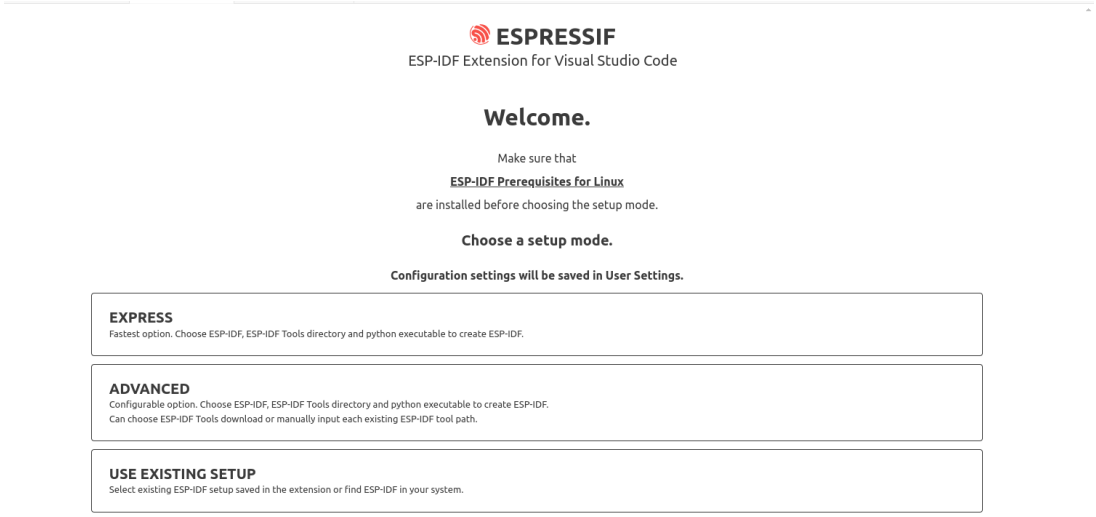


Fig. 1: Opciones de configuración de ESP-IDF en VSCode.

- La opción *"ADVANCED"* permite elegir cada una de las herramientas que se desean utilizar proporcionando la ruta de cada una.
- La opción *"USE EXISTING SETUP"* permite cargar una configuración previamente utilizada (se pueden tener múltiples configuraciones).

- La opción *"EXPRESS"* carga todas las herramientas de ESP-IDF con las que se cuenten.

En este documento se presentará la configuración *"EXPRESS"*.

Configuración EXPRESS

Al elegir la configuración *"EXPRESS"* se despliega un menú como el mostrado en la *Figura 2*. Si se siguió el procedimiento descrito en la *sección 2.1* ya se tienen descargadas e instaladas las herramientas de ESP-IDF, por lo que solo se necesita proveer el directorio donde se encuentran. Esto se realiza siguiendo lo listado en seguida. Alternativamente, si no tiene descargado e instalado ESP-IDF, la extensión permite descargarlo como se muestra en el sitio `https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/installation.html`.

La diferencia entre los procedimientos radica en la selección de ESP-IDF en el sistema propio o descargado desde GitHub.

1. Para la sección *"Select ESP-IDF version"* elegir la opción *Find ESP-IDF in your system.*
2. En la sección *"Enter ESP-IDF directory"* proporcionar la ruta donde se encuentran almacenado ESP-IDF, usualmente es `/home/<user>/esp/esp-idf`.
3. Para la sección *"Enter ESP-IDF Tools directory"* proporcionar la ruta donde se encuentran almacenadas las herramientas de ESP-IDF, usualmente es `/home/<user>/esp/.espressif`.
4. En *Select Python version* proporcionar la ruta donde se encuentra el ejecutable para correr scripts de Python, usualmente `/usr/bin/python3`.

Realizado lo anterior, dar click sobre el botón *Install* y monitorear el progreso de la configuración. Finalmente, si no se genera ningún error durante la configuración, aparece un mensaje sobre todos los ajustes configurados correctamente y que es posible emplear las herramientas de ESP-IDF en VSC.

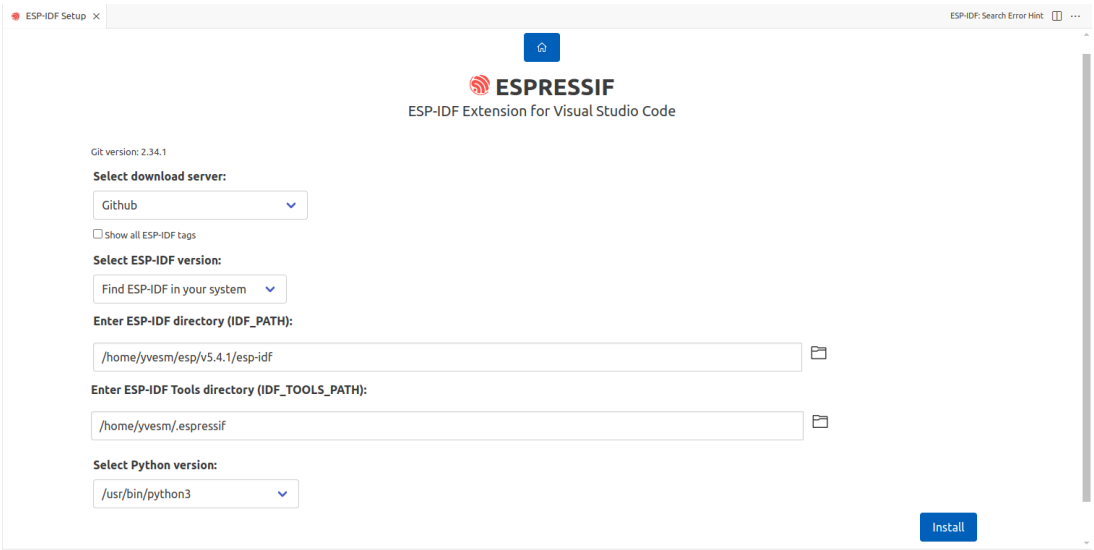


Fig. 2: Menú de configuración Express.

4.3 Permisos y uso del puerto USB

Los equipos donde se crea el código que ejecutarán las tarjetas de ESP y las tarjetas mismas pueden establecer comunicación en serie a través del puerto USB. Para establecer la comunicación con la tarjeta se necesita conocer el nombre del puerto de conexión y otorgar permiso de uso del puerto al usuario que desee emplearlo. Una vez que se cuenta con el permiso, la comunicación y transferencia de información a la memoria Flash de las tarjetas queda establecida.

En Linux existe un comando en la terminal que permite visualizar el nombre de los puertos a través de los cuales se conectan los dispositivos, este es:

```
ls /dev/tty*
```

Para conocer el nombre del puerto a través del cual se comunica la tarjeta basta con ejecutar este comando dos veces, la primera vez antes de conectar la tarjeta y la segunda después de conectarla. El nombre del puerto que aparece la segunda vez es el que se requiere para establecer la conexión. Usualmente el nombre del puerto para las tarjetas ESP32 es `/dev/ttyUSB0`.

Otorgar permiso al usuario

Una vez que se conoce el nombre del puerto con el que se comunica la tarjeta, se necesita otorgar permiso de lectura y escritura sobre USB al usuario que requiera emplearlo. En Linux, este permiso se puede conceder siguiendo los siguientes pasos:

1. Ejecutar uno de los siguientes comandos en la terminal:

```
sudo usermod -a - G dialout $USER
sudo usermod -a - G uucp $USER
```

2. Reingresar a la cuenta del usuario al que se le concedió el permiso de lectura/escritura.

En el enlace 4 se puede consultar una guía en caso de presentarse algún error durante la comunicación o que no sea posible establecerla.

5 Creación de un proyecto nuevo en VSC

Las secciones anteriores describieron como establecer un entorno para crear proyectos de ESP32 en VSC. La extensión de ESP en VSC ofrece diversas opciones para generar proyectos que se cargaran a las tarjetas ESP32. Se puede partir de los proyectos ejemplo incluidos en el ambiente ESP-IDF, importar proyectos existentes, partir de componentes de ESP o generar un proyecto nuevo vacío. En los proyectos vacíos se tiene lo necesario para compilar y cargar a la memoria Flash de la tarjeta objetivo, un código o códigos correspondientes a una aplicación ejecutada por la tarjeta. A continuación se describe un procedimiento para generar un proyecto nuevo vacío.

1. Conectar la tarjeta ESP32 a emplear.

2. En VSC hacer clic en la sección de ESP-IDF, esperar a que lance la pantalla de inicio de ESP y hacer clic en *New Project*. Alternativamente se puede abrir la paleta de comandos de VSC `view > Command Palette` (`Ctrl + Shift + P` ó `F1`) y buscar el comando `ESP-IDF: New Project`. Una vez hecho esto se mostrará un menú que solicita el entorno de ESP-IDF a utilizar, seleccionar el deseado del menú y continuar. Realizado lo anterior, se desplegará un menú como el mostrado en la *Figura 3*.

3. En el menú de la *Figura 3* establecer el nombre del proyecto nuevo y la carpeta contenedora del mismo. En caso que se desee, se puede crear una nueva carpeta contenedora del proyecto y al elegir la opción abrir, VSC establecerá la ruta de la nueva carpeta. En la opción *Choose ESP-IDF Board* elegir la tarjeta ESP32 a emplear, en caso que no aparezca en el listado, elegir *ESP32-chip (via ESP-PROG o via ESP-PROG2)*. Posteriormente, en el mismo menú elegir el nombre del puerto con el que se comunica la tarjeta, usualmente `/dev/ttyUSB0`. Finalmente establecer el nombre de la carpeta en caso que se desee agregar un componente de ESP previamente realizado o dejar en blanco si no se desea esto y hacer clic en el botón *Choose Template*.

4. El paso anterior conduce al menú mostrado en la *Figura 4*. En éste menú se puede elegir una plantilla y un conjunto de archivos con los que contará el proyecto de acuerdo a las necesidades de uso. Se tienen dos ramas para elegir el tipo de plantilla, *Extension* o *ESP-IDF*; la primer rama (*Extension*) permite escoger una de acuerdo al ambiente de trabajo que se requiera y la segunda rama (*ESP-IDF*) permite escoger una basada en los ejemplos incluidos en ESP-IDF, *Figura 5*. Para elegir una plantilla que contiene un solo archivo fuente en lenguaje C se elige la opción *template-app* dentro de la opción desplegable *Extension* como se muestra en la *Figura 6* y se hace clic en el botón *Create project using template-app*.

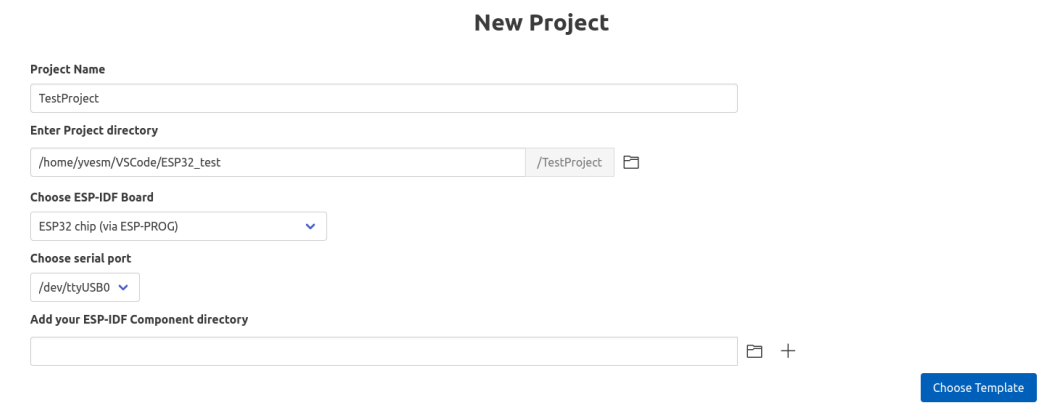


Fig. 3: Menú de configuración de un proyecto nuevo.

Fig. 4: Menú de selección de plantilla para el nuevo proyecto.

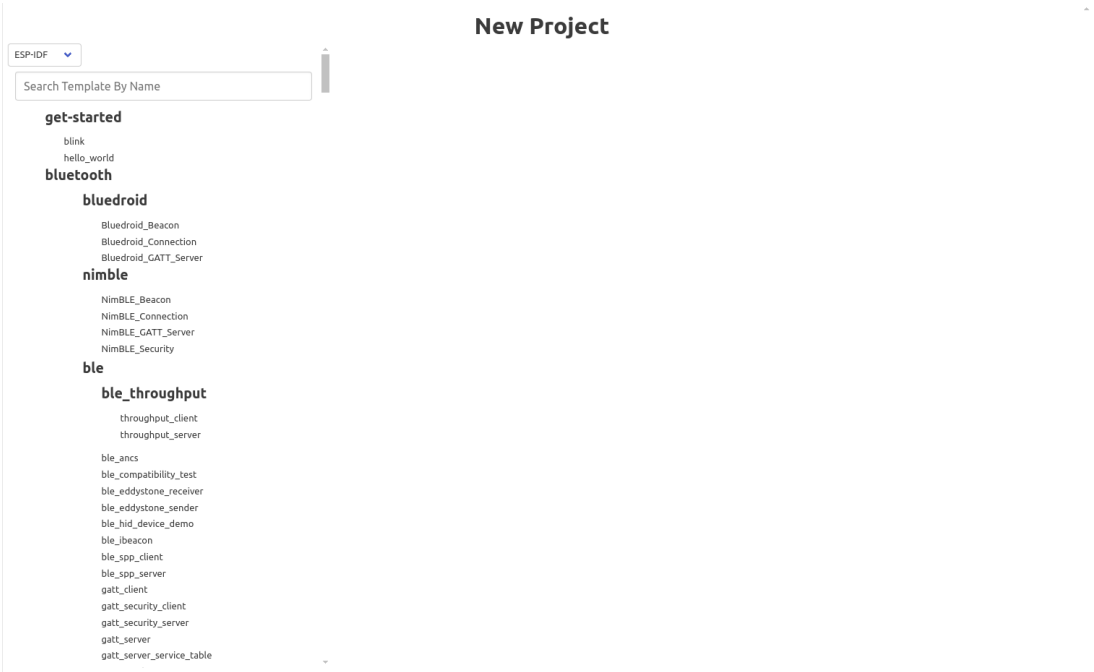


Fig. 5: Menú de selección de plantilla para el nuevo proyecto.

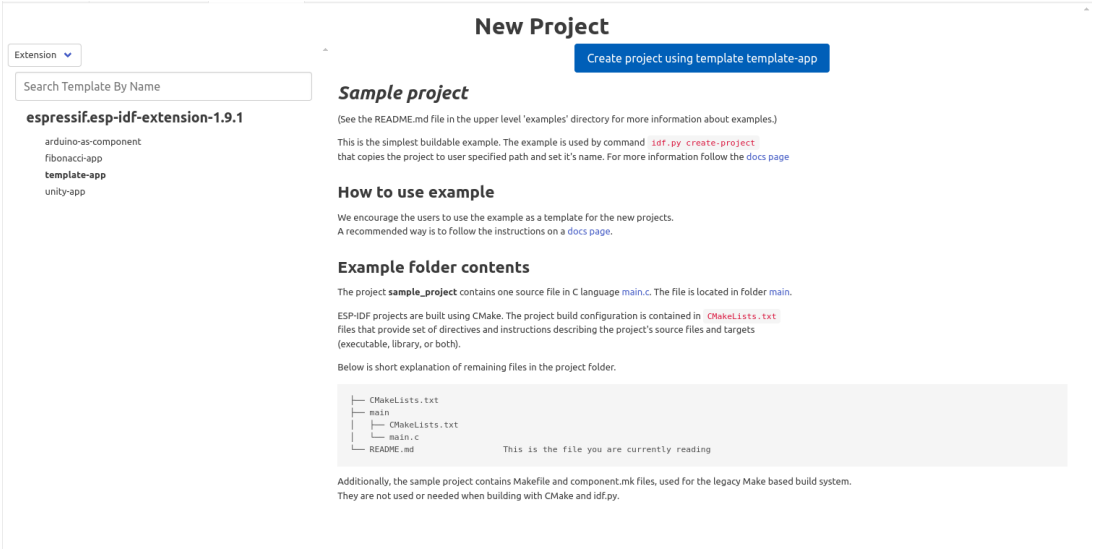


Fig. 6: Plantilla simple para el nuevo proyecto.

Siguiendo estos cuatro pasos se crea un proyecto nuevo vacío. Este proyecto se puede abrir en una nueva ventana y queda almacenado dentro de una nueva carpeta con el nombre elegido para el proyecto. Dentro de la carpeta del proyecto se genera un conjunto de subdirectorios. Dentro del subdirectorio *main* se encuentra el archivo *main.c*, en el cual se añade el código correspondiente a la aplicación a desarrollar y que se cargará a la tarjeta ESP32.

6 Configuración, construcción y transferencia de un proyecto

La extensión de Espressif agrega un conjunto de comandos ejecutables a la paleta de VSC y una lista de íconos en la barra de estatus, como se muestra en la *Figura 7*, que sirven de atajo para ejecutar comandos empleados con frecuencia al configurar y cargar un proyecto a una tarjeta ESP32. Entre estos comandos se encuentran: selección del puerto de comunicación, establecimiento del dispositivo o tarjeta a emplear, configuración de la tarjeta (editor de configuración), construcción del proyecto, transferencia a la memoria flash de la tarjeta, monitor y debug.

Una vez creado un nuevo proyecto, como se mostró en la sección 5, se puede compilar y cargar a la tarjeta objetivo de acuerdo a lo siguiente:

1. Conectar y seleccionar puerto: Conectar tarjeta en caso de que no se haya realizado. Ejecutar el comando de la paleta de VSC `ESP-IDF: Select Port to Use` y elegir el nombre del puerto correspondiente a la tarjeta, usualmente `/dev/ttyUSB0`. Alternativamente se puede seleccionar el ícono de conector.

2. Configurar proyecto: Para crear y abrir los archivos de configuración del proyecto se ejecutan los siguientes comandos de la paleta de VSC. `ESP-IDF: Set Espressif Device Target` para establecer la tarjeta objetivo (ESP32 PROG1 para el caso mostrado) o ícono de circuito integrado (chip), después `ESP-IDF: Select OpenOCD Board Configuration` para elegir los archivos de configuración openOCD y finalmente el comando `ESP-IDF: SDK Configuration Editor` o ícono de engrane con el que se despliega el menú de configuración del proyecto y el cual generará el archivo de configuración del proyecto y la carpeta *build*. Si se siguió lo mostrado en la *sección 5* el puerto y la tarjeta objetivo ya se encuentran establecidos.

3. Opcional: Las extensiones de C/C++ requieren el archivo *compile_commands.json* el cual se coloca en el directorio *build*. Para generar este archivo ejecutar `ESP-IDF: Run idf.py reconfigure task` o hacer clic en notificación *Generate compile_commands.json*, *Figura 8*.

4. Crear aplicación: Escribir y guardar el código correspondiente a la aplicación en el archivo *main.c* dentro de la carpeta *main* del proyecto.

Fig. 7: Lista de íconos de ESP agregados.

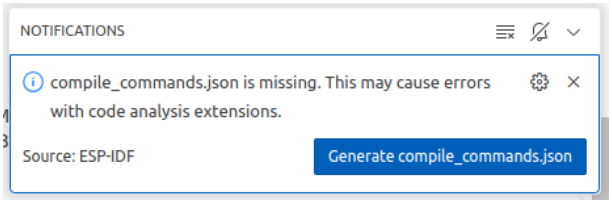


Fig. 8: Notificación para generar compile_commands.

5. Construir el proyecto: Ejecutar el comando de la paleta de VSC `ESP-IDF: Build Your Project` o ícono de herramienta. Esto compilará el proyecto y mostrará el resultado de la compilación en la consola incluida en VSC, también mostrará el tamaño o memoria requerida para la aplicación.

6. Transferir a memoria Flash del dispositivo: Una vez que el proyecto se compila exitosamente se puede transferir a la tarjeta objetivo si se ejecuta el comando `ESP-IDF: Flash Your Project` o se elige el ícono del rayo y posteriormente se elige el método de transferencia (JTAG, UART o DFU), por defecto el método configurado es UART.

7. Monitorear salida: Para ver mensajes y resultados de procesos dentro de la aplicación desarrollada se puede ver el monitor serial de la aplicación con el comando `ESP-IDF: Monitor Device` o el ícono de pantalla.

Las páginas web de los enlaces 5-9 proveen información adicional sobre el proceso de configuración y transferencia del proyecto.

6.1 Proyecto ejemplo

El siguiente ejemplo muestra el funcionamiento de lo descrito previamente. El ejemplo considera un proyecto en el que la aplicación cargada a la tarjeta ESP32-WROOM se encarga de encender y apagar cuatro LEDs conectados a ésta última en forma secuencial. Esto es, se enciende cada uno de los LEDs manteniendo los LEDs anteriores encendidos; una vez que todos los LEDs se encuentran encendidos, apaga éstos uno por uno manteniendo los anteriores en el mismo estado hasta que todos se apagan.

Una vez creado el proyecto ejemplo siguiendo lo mostrado en la *sección 5* se realizó lo mostrado en la sección anterior inmediata. A continuación se enumeran los pasos para la configuración, construcción y transferencia del proyecto ejemplo y los resultados obtenidos de esto.

1. Se conectó la tarjeta ESP32-WROOM y se ejecutó el comando en VSC `ESP-IDF: Select Port to Use`. El nombre del puerto elegido y correspondiente a la tarjeta fue el `/dev/ttyUSB0`.

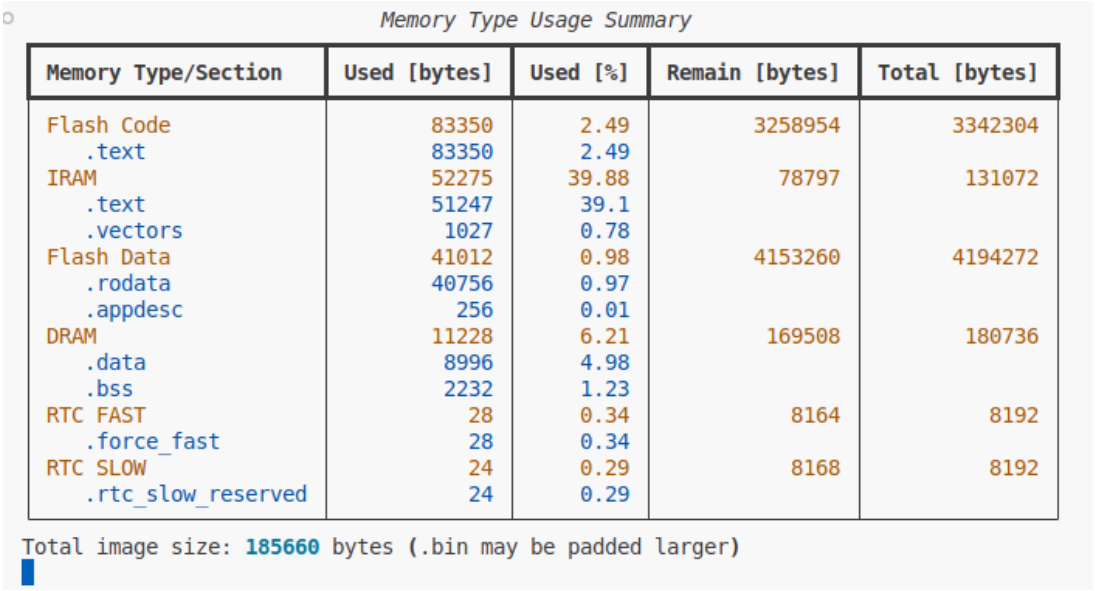
2. Se ejecutó el comando de VSC `ESP-IDF: Set Espressif Device Target` y se estableció ESP32-PROG1 como tarjeta objetivo. Posteriormente se eligieron los archivos de configuración con el comando `ESP-IDF: Select OpenOCD Board Configuration` y finalmente se ejecutó el comando `ESP-IDF: SDK Configuration Editor` para abrir el menú de configuración de las opciones del proyecto. Como opciones de configuración se eligieron y guardaron las que se tenían por defecto.

3. Se generó el archivo *compile_commands.json* mediante el comando `ESP-IDF: Run idf.py reconfigure task`.

4. Se escribió el código de la aplicación, mostrado en el *Apéndice A*, que encendía y apagaba los LEDs en forma secuencial en el archivo *main.c* dentro de la carpeta *main* del proyecto. Se puede modificar el nombre del archivo *main.c*, pero el nombre se debe conservar una vez que se compila el proyecto.

5. Se ejecutó el comando en VSC `ESP-IDF: Build Your Project` para construir el ejecutable del proyecto. Si la compilación no genera errores, el resultado de la compilación es similar al mostrado en el *Figura 9*, el cual corresponde al proyecto ejemplo. Además, la extensión de ESP genera una tabla que muestra la cantidad de memoria en la tarjeta que ocupa la aplicación, es decir, el tamaño y distribución de memoria de la aplicación, ver *Figura 10*.

Fig. 9: Resultado de la construcción del proyecto ejemplo.



Memory Type Usage Summary

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	83350	2.49	3258954	3342304
.text	83350	2.49		
IRAM	52275	39.88	78797	131072
.text	51247	39.1		
.vectors	1027	0.78		
Flash Data	41012	0.98	4153260	4194272
.rodata	40756	0.97		
.appdesc	256	0.01		
DRAM	11228	6.21	169508	180736
.data	8996	4.98		
.bss	2232	1.23		
RTC FAST	28	0.34	8164	8192
.force_fast	28	0.34		
RTC SLOW	24	0.29	8168	8192
.rtc_slow_reserved	24	0.29		

Total image size: **185660** bytes (.bin may be padded larger)

Fig. 10: Memoria empleada por el proyecto ejemplo.

6. Una vez construido exitosamente, se transfirió el proyecto a la memoria Flash de la tarjeta ESP32-WROM. Para ésto se ejecutó el comando `ESP-IDF: Flash Your Project` y se eligió como protocolo de transferencia a UART. La transferencia a la Flash es un proceso de mayor duración que los anteriores y se muestra la realización del mismo a través de la terminal de VSC. Si la transferencia se realiza exitosamente, la terminal muestra un mensaje como el expuesto en la *Figura 11*.

7. Se observaron los mensajes emitidos por la aplicación desarrollada a través del monitor serial con el comando `ESP-IDF: Monitor Device`, como se observa en la *Figura 12*. Las imágenes de las *Figuras* y muestran la secuencia de LEDs, es decir, el resultado de la aplicación corriendo en la tarjeta.

Fig. 11: Resultado de la transferencia a Flash del proyecto ejemplo.

```
I (173) boot: Loaded app from partition at offset 0x10000
I (173) boot: Disabling RNG early entropy source...
I (183) cpu_start: Multicore app
I (191) cpu_start: Pro cpu start user code
I (191) cpu_start: cpu freq: 160000000 Hz
I (191) app_init: Application information:
I (192) app_init: Project name:     LEDs_Test
I (196) app_init: App version:      1
I (199) app_init: Compile time:     May 19 2025 17:16:16
I (204) app_init: ELF file SHA256:  0a892a27e...
I (208) app_init: ESP-IDF:          v5.4.1-dirty
I (213) efuse_init: Min chip rev:     v0.0
I (216) efuse_init: Max chip rev:     v3.99
I (220) efuse_init: Chip rev:         v3.0
I (225) heap_init: Initializing. RAM available for dynamic allocation:
I (231) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (236) heap_init: At 3FFB2BE0 len 0002D420 (181 KiB): DRAM
I (241) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (246) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (252) heap_init: At 4008CC34 len 000133CC (76 KiB): IRAM
I (259) spi_flash: detected chip: generic
I (261) spi_flash: flash io: dio
W (264) spi_flash: Detected size(4096k) larger than the size in the binary image header(2048k). Using the size in the binary image header.
I (277) main_task: Started on CPU0
I (287) main_task: Calling app_main()
I (287) Sample App: OUTPUT GPIO(s): 18
I (287) Sample App: OUTPUT GPIO(s): 19
I (287) Sample App: OUTPUT GPIO(s): 21
I (287) Sample App: OUTPUT GPIO(s): 22
I (287) Sample App: Mascara GPIOs: 7077888
I (297) Sample App: Configurando GPIO(s)...
I (297) gpio: GPIO[18]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (307) gpio: GPIO[19]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (317) gpio: GPIO[21]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (327) gpio: GPIO[22]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (337) Sample App: GPIO(s) configurados
I (337) Sample App: Iniciando secuencia LEDS
I (337) Sample App: Encendiendo LED: 1
I (1347) Sample App: Encendiendo LED: 2
I (2347) Sample App: Encendiendo LED: 3
I (3347) Sample App: Encendiendo LED: 4
I (4347) Sample App: Apagando LED: 4
I (5347) Sample App: Apagando LED: 3
I (6347) Sample App: Apagando LED: 2
I (7347) Sample App: Apagando LED: 1
I (8347) main_task: Returned from app_main()
```

Fig. 12: Terminal de salida mostrada para el proyecto ejemplo.

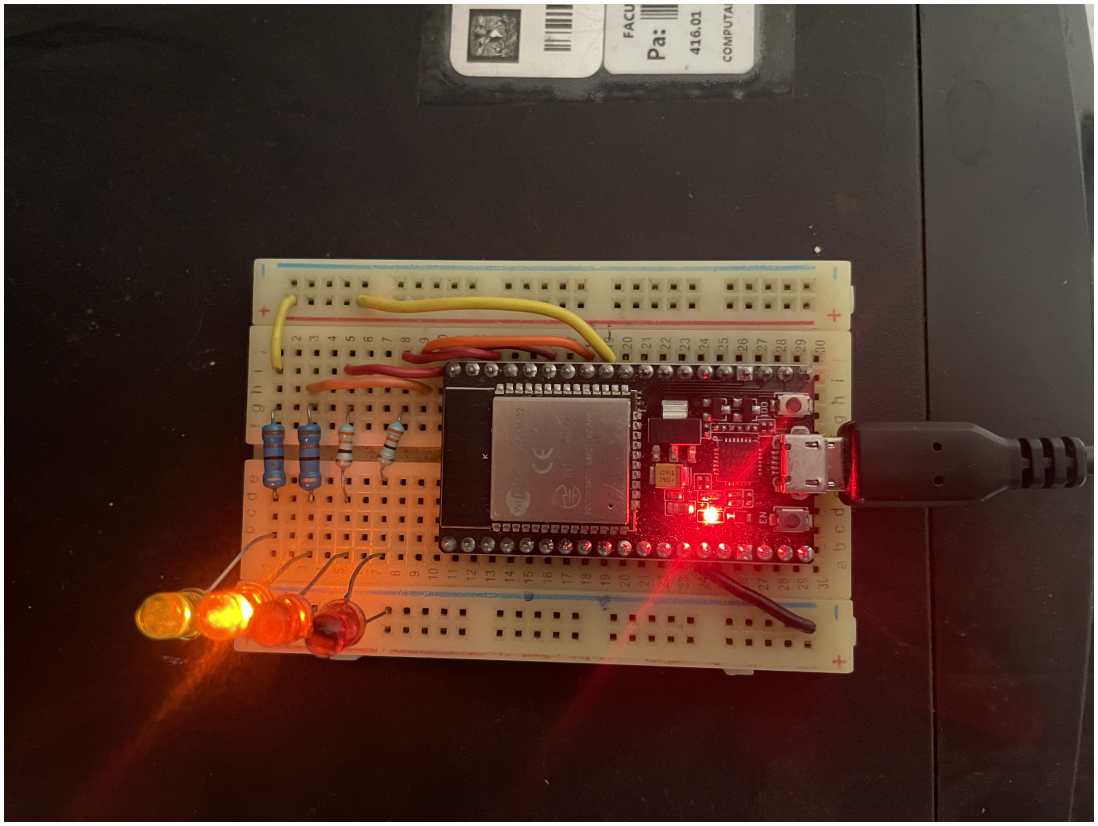


Fig. 13: Aplicación corriendo en ESP32-WROOM (1).

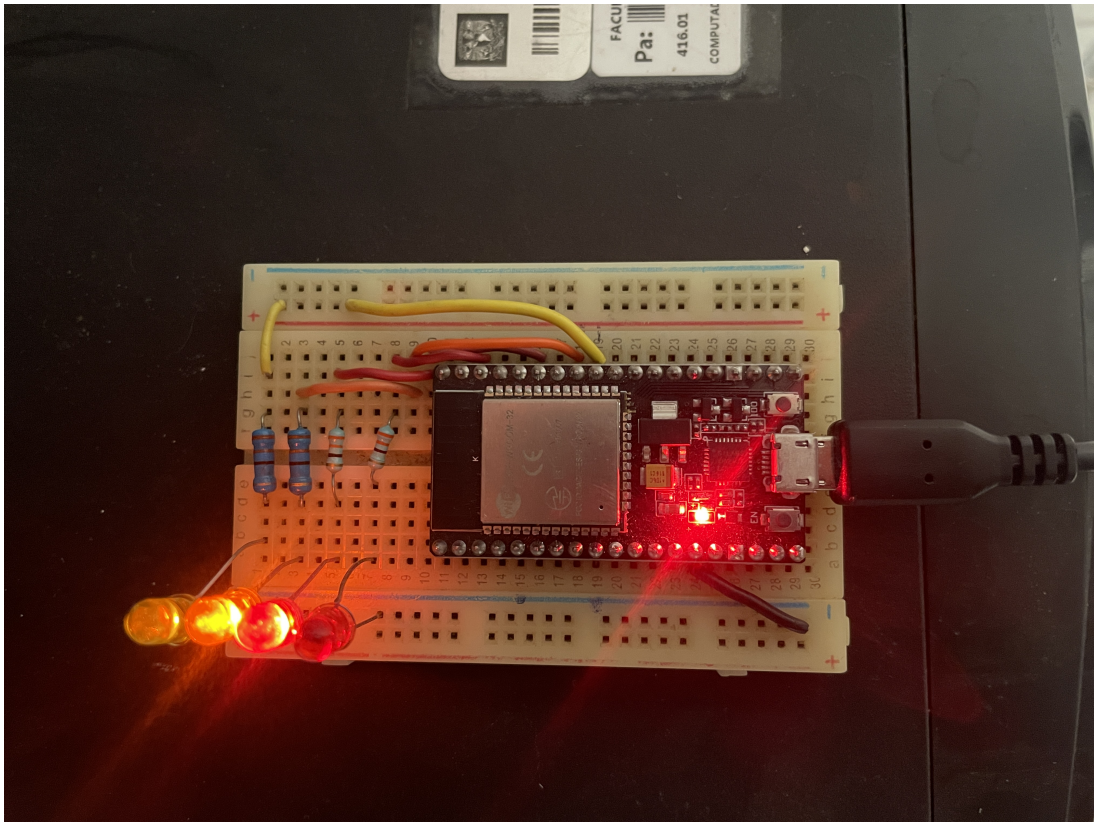


Fig. 14: Aplicación corriendo en ESP32-WROOM (2).

Contenido

A Código del proyecto ejemplo

```c
#include <stdio.h>
#include <driver/gpio.h>
#include <unistd.h>
#include "esp_log.h"

#define numLEDs 4

uint16_t gpio_out[numLEDs] = {18,19,21,22};

static const char* TAG = "Sample App";

void app_main(void)
{
    int ledIt;
    uint32_t GPIO_OUT_SEL = 0;
    gpio_config_t out_conf = {};

    for(ledIt = 0; ledIt < numLEDs; ledIt++){
        GPIO_OUT_SEL = GPIO_OUT_SEL|(1<<gpio_out[ledIt]);
        ESP_LOGI(TAG, "OUTPUT GPIO(s): %d",gpio_out[ledIt]);
    }

    ESP_LOGI(TAG, "Mascara GPIOs: %lu",GPIO_OUT_SEL);

    ESP_LOGI(TAG, "Configurando GPIO(s)...");

    out_conf.intr_type = GPIO_INTR_DISABLE;           //Desabilita interrupciones
    out_conf.mode = GPIO_MODE_OUTPUT;              //Establece GPIOs como salida
    out_conf.pin_bit_mask = GPIO_OUT_SEL;            //Mascara binaria de pines a usar
    out_conf.pull_down_en = 0;                 //Desabilita resistor de pull down
    out_conf.pull_up_en = 0;                 //Desabilita resistor de pull up
    gpio_config(&out_conf);                    //Configura GPIOs

    for(ledIt = 0; ledIt < numLEDs; ledIt++){
        gpio_set_level(gpio_out[ledIt],0);
    }

    ESP_LOGI(TAG, "GPIO(s) configurados");

    ESP_LOGI(TAG, "Iniciando secuencia LEDS");

    while(1){
        for(ledIt = 0; ledIt < numLEDs; ledIt++){
            gpio_set_level(gpio_out[ledIt],1);
            ESP_LOGI(TAG, "Encendiendo LED: %d",ledIt + 1);
            sleep(1);
        }

        for(ledIt = numLEDs -1; ledIt >= 0; ledIt--){
            gpio_set_level(gpio_out[ledIt],0);
            ESP_LOGI(TAG, "Apagando LED: %d",ledIt + 1);
            sleep(1);
        }
        break;
    }
}
```

Código 1: Enciende y apaga LEDs en secuencia ESP32

Enlaces

1. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html.
2. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/
3. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html#step-1-install-prerequisites.

4. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/establish-serial-connection.html.
5. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/connectdevice.html.
6. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/configureproject.html.
7. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/buildproject.html.
8. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/flashdevice.html.
9. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/monitoroutput.html.
10. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/
11. https://docs.espressif.com/projects/vscode-esp-idf-extension/en/latest/startproject.html
12. https://github.com/espressif/vscode-esp-idf-extension/blob/master/README.md