Escuela Técnica Superior de
Ingenieros de Telecomunicación

Guía de instalación de herramientas para compilación multiplataforma en C

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Grado en Ingeniería Biomédica

Escuela Técnica Superior de
Ingenieros de Telecomunicación

Guía de instalación de herramientas para compilación multiplataforma en C

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Grado en Ingeniería Biomédica

Autor:
Profesores Departamento de Ingeniería Electrónica
Coordinación y contacto: j.pagan@upm.es
Departamento de Ingeniería Electrónica

Índice general

Apéndices

Índice de figuras

Lista de acrónimos

API Application Programming Interface (Interfaz de Programación de Aplicaciones) 22

GDB GNU Debugger 17

HAL Hardware Abstraction Layer (Capa de Abstracción Hardware) 23

HW Hardware 16

IDE Integrated Development Environment (Entorno de Desarrollo Integrado) 5

JTAG Joint Test Action Group 16

GDB GNU Debugger 17

LED Light-Emitting Diode (Diodo Emisor de Luz) 27

ST STMicroelectronics 16

SW Software 17

SWD JTAG/serial Wire Debugging Software 16

WSL Windows Subsystem for Linux 2

Capítulo 1

Guía de instalación de VSCode y herramientas de desarrollo en C

Josué Pagán, Daniel Capellán, Pedro Malagón, Román Cárdenas

Este documento tiene como objetivo definir los pasos de instalación de las herramientas de trabajo Visual Studio Code (VSCode) y compilador, y configurar el entorno de proyecto para una experiencia de programación más cómoda. El documento está descrito para los sistemas operativos Windows, MacOS y Linux. Los pasos están probados y funcionan, pero pueden depender de las particularidades de la computadora de cada usuario. Lee con atención y adapta los pasos y rutas si lo consideras necesario.

Este documento es un resumen de los pasos que puede encontrar en el canal de Youtube de la asignatura SDG1: [SDG1-C] [0] Guía de instalación VSCode y compilador C. Puedes acceder a cada explicación con los accesos directos de la descripción del vídeo.

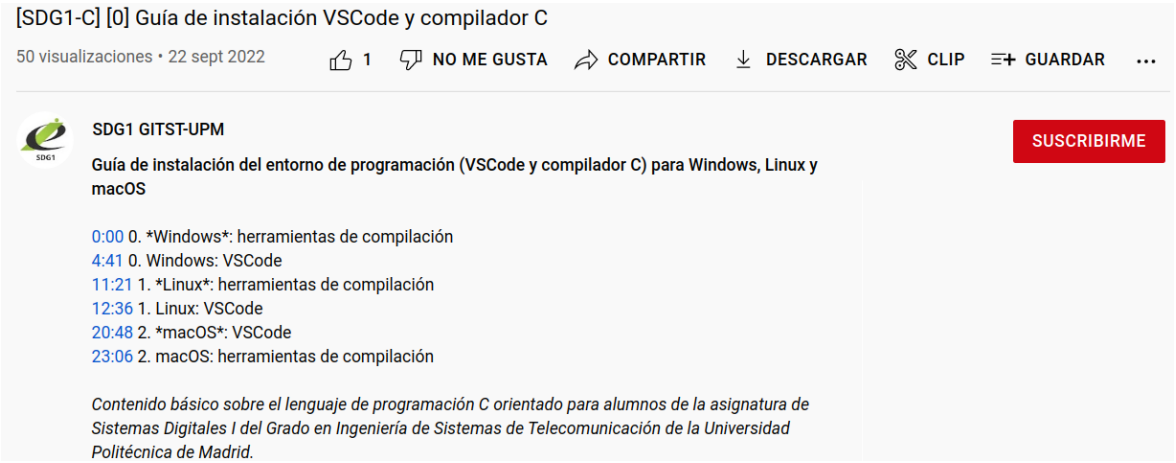


Figura 1.1: Vídeo [SDG1-C] [0] Guía de instalación VSCode y compilador C, disponible en el canal de Youtube de la asignatura SDG1.

1.1. Instalación de las herramientas de compilación

Lo primero que necesitamos instalar son las herramientas de compilación de C. El compilador es el encargado de convertir los códigos fuente de nuestro programa en ficheros binarios capaces de ser ejecutados en una máquina de iguales características a la nuestra (compilación nativa), o de distintas características (compilación cruzada). Nos centraremos en las herramientas de compilación nativa.

1.1.1. Instalación en Windows

En Windows, usaremos la herramienta Windows Subsystem for Linux (WSL) proporcionada por Microsoft. WSL permite a los desarrolladores instalar una distribución de Linux (como Ubuntu, OpenSUSE, Kali, Debian, Arch Linux, etc.) y utilizar aplicaciones, utilidades y herramientas de línea de comandos Bash de Linux directamente en Windows, sin modificar, sin la sobrecarga de una máquina virtual tradicional o una configuración de arranque dual.

1.1.1.1. Instalación de WSL

Abriremos una PowerShell en **modo de administrador haciendo clic con el botón derecho y seleccionando "Ejecutar como administrador"**. En la terminal, introduciremos el siguiente comando:

```
1 wsl --install -d Ubuntu-24.04
```

> **NOTA**
>
> Este paso puede tardar un tiempo considerable. Tened paciencia y no detengáis el proceso de instalación.

Veremos algo como lo que se muestra en la Figura 1.2.
A continuación, **reiniciaremos el ordenador**. Se abrirá automáticamente la aplicación Ubuntu para acabar con el proceso de instalación. Ya hemos habilitado las características necesarias para ejecutar WSL e instalar la distribución Ubuntu de Linux.

1.1.1.2. Configurar la información de usuario de Linux

Ahora, tenemos WSL con la última versión de Ubuntu instalada en nuestra máquina. Si buscas "Ubuntu" en tus aplicaciones, encontrarás un ejecutable que ejecuta un *shell* de Ubuntu sobre WSL. Puedes ver más información sobre cómo configurar tu entorno de desarrollo WSL aquí.

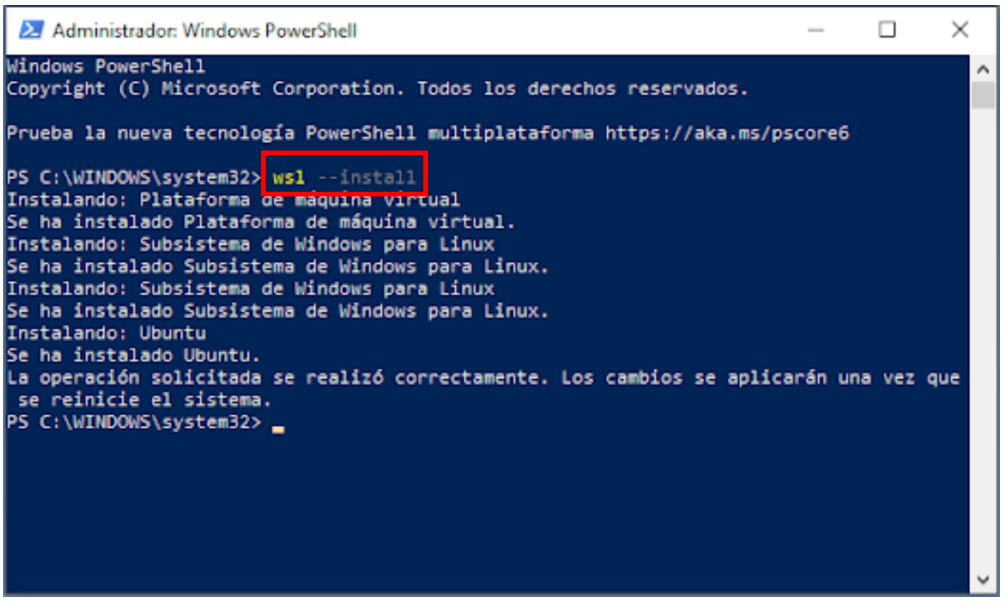


Figura 1.2: Instalación de WSL desde la Power Shell de Windows.

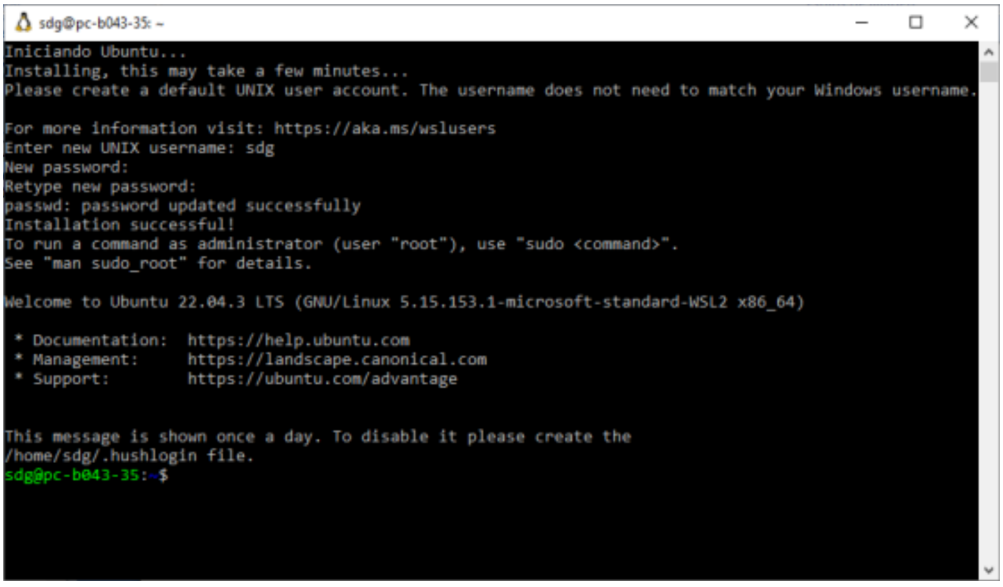


Figura 1.3: Vista de la primera vez que que se inicia la aplicación Ubuntu.

A continuación, abre Ubuntu desde tus aplicaciones (se te abrirá una terminal de Ubuntu). La primera vez, tardará algún tiempo en instalarse el sistema operativo. La primera vez debemos crear un nuevo usuario para el subsistema Ubuntu (ver Figura 1.3).

ATENCIÓN

Este paso sólo se realiza una vez, así que **asegúrate de recordar el nombre de usuario y la contraseña**, ya que tendrás que proporcionarlos cada vez que abras un nuevo terminal de Ubuntu. **No tiene por qué ser el mismo usuario y contraseña de tu sistema Windows.**

1.1.1.3. Instalación de las dependencias en el subsistema Ubuntu

Ahora, sigue las instrucciones de instalación para Ubuntu **desde el terminal de Ubuntu**, **¡no desde Windows PowerShell!** Para ello, ve a la Sección 1.1.2, pues el proceso es igual que para la instalación en Linux.

1.1.2. Instalación en Linux (Ubuntu)

Para instalar las herramientas abrimos una terminal. Podemos acceder buscándola en nuestro listado de aplicaciones o pulsando Ctrl+Alt+T.

NOTA

Si eres usuario de Windows y estás usando WSL, la aplicación Ubuntu es ya el terminal Linux. No es necesario abrir nada más.

Es posible que ya tengamos instalado el compilador, bien porque venga por defecto en nuestra distribución de Linux, o porque lo hayamos instalado con anterioridad. No obstante, escribiremos en la terminal los siguientes comandos que utilizan el instalador de paquetes *apt* **uno a uno**:

```
sudo apt update
sudo apt upgrade
sudo apt install gcc gdb make
```

Ya tenemos instaladas las herramientas de compilación y depuración en Linux. Solo nos falta el editor de código. Para ello, iremos a la sección "Instalación de VSCode" (Sección 1.2).

1.1.3. Instalación en MacOS

Tenemos que instalar las herramientas del paquete de desarrollo para Mac XCode. Incluye, por supuesto, las herramientas de compilación (el compilador CLang) y depuración que necesitamos. Para ello necesitamos abrir una terminal: en el *Spotlight* de nuestro Mac escribimos "`terminal`". Se nos abrirá una terminal.

Escribiremos en la terminal el siguiente comando:

```
1 xcode-select --install
```

NOTA

Este proceso puede llevar un tiempo considerable. Tened paciencia y no detengáis el proceso de instalación.

Ya tenemos instaladas las herramientas de compilación y depuración en MacOS. Solo nos falta el editor de código. Para ello, ve a la sección "Instalación de VSCode" (Sección 1.2)

1.2. Instalación de VSCode

1.2.1. Instalación de VSCode y extensiones

Visual Studio Code (VSCode) es un editor de texto que permite la instalación de *extensiones* para el procesamiento de ficheros de programación en distintos lenguajes. **Importante: VSCode no es el Integrated Development Environment (Entorno de Desarrollo Integrado) (IDE) de ningún lenguaje, y los compiladores o intérpretes deben ser instalados aparte.** Las mencionadas extensiones hacen que la herramienta sea muy versátil para programadores que trabajen en varios lenguajes (C, Python, Java, LaTeX, R, Rust...) y no quieran depender de distintos IDEs.

Para instalar la herramienta solo tenemos que acceder a su web y descargarla. Podemos llegar a su página de descargas desde el navegador buscando "Visual Studio Code Download", o directamente desde esta URL: https://code.visualstudio.com/Download. Descargamos el instalador de nuestro sistema operativo y lo ejecutamos de la manera habitual con la que instalamos programas.

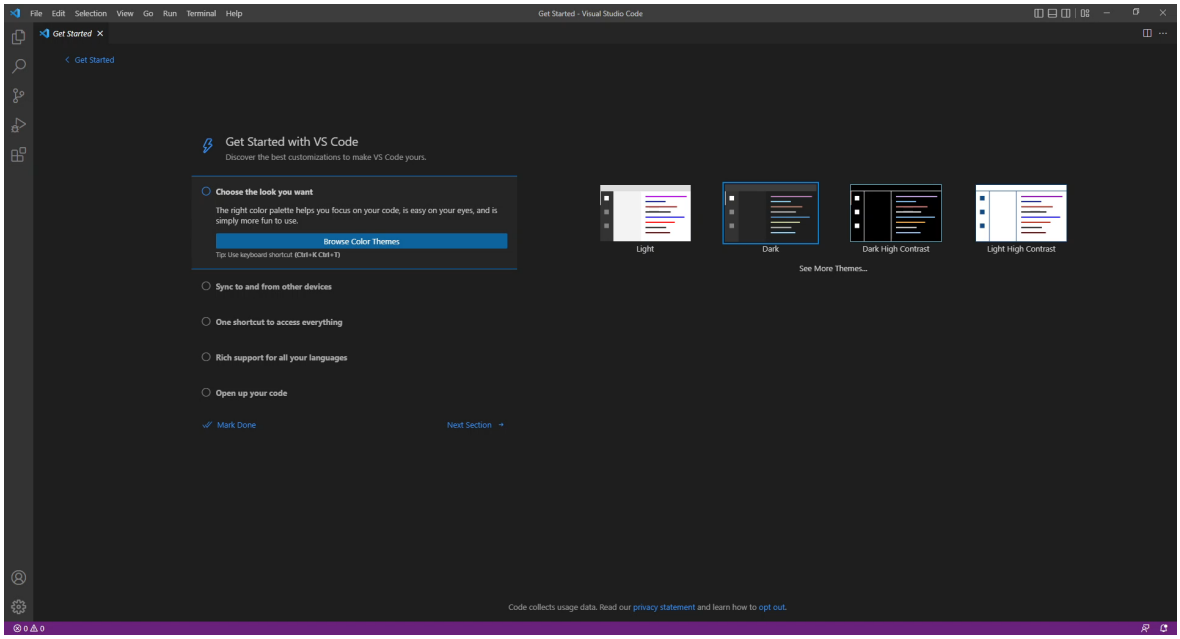


Figura 1.4: Pantalla inicial de Visual Studio Code.

NOTA

Si eres usuario de Linux:
Se entiende que conoces el manejo típico de un sistema operativo Linux. No obstante, te recordamos que, para instalar el paquete descargado de VSCode, puedes hacerlo ejecutando desde la terminal, en el directorio donde haya descargado el paquete: `sudo dpkg -i code_x.y.z` (donde `code_x.y.z` es el nombre del ejecutable descargado).

La instalación no requiere ningún paso especial. **Se recomienda dejar las rutas de instalación por defecto**.

ATENCIÓN

Ningún directorio de la ruta debe contener espacios, caracteres especiales, tildes, ni ñ.

Una vez instalado, la aplicación tendrá un aspecto como en la Figura 1.4.
A continuación instalaremos unas extensiones que nos facilitarán la tarea de programación para esta asignatura. Desde la barra lateral de VSCode accedemos al menú lateral, accedemos a las Extensiones  . En la cajetilla del buscador de la esquina superior izquierda buscamos e instalamos:

- 

La extensión **"C/C++ Extension Pack"** que incluye muchos paquetes como: C/C++, C/C++ Themes, CMake, CMakeTools, ...

- La extensión **"C/C++ Project Generator"**, que nos permitirá crear una estructura de proyecto base sobre la que trabajar.
- Una extensión útil para ver binarios es *"Hex Editor"*.
- **Si eres usuario de MacOS:** Instala la extensión de ayuda a la depuración *"codeLLDB"*.
- **Si eres usuario de Windows:** Instala la extensión *"WSL"* para habilitar la comunicación con la aplicación Ubuntu en WSL, donde están instaladas las herramientas de compilación y depuración.

1.2.2. Creación de un proyecto base *hello*

Para comprobar que la instalación ha funcionado correctamente, vamos a crear un sencillo programa utilizando la extensión C/C++ Project Generator. Para ello, tenemos que ir a la "Paleta de Comandos" (*Command Palette*), a la que podemos acceder de 4 maneras y escribir: "Create C project" y damos *enter* (ver Figura 1.5).

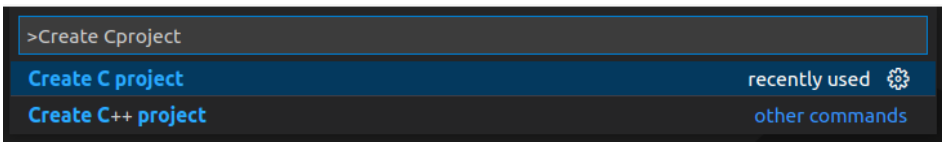
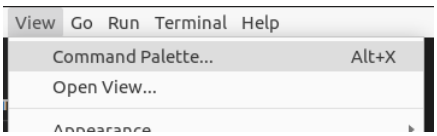


Figura 1.5: Escribir en la Paleta de Comandos: "Create C Project" y dar enter, o pinchar con el ratón.

La paleta de comandos es una ventana de 1 línea que se abre en el centro arriba y que nos permite ejecutar órdenes de VSCode. Para acceder a ella podemos hacerlo de cualquiera de las siguientes formas:

- Desde la pestaña "View → Command Palette":



- Presionando Alt+X
- Presionando F1
- Presionando Ctrl+Shift+P

Seguidamente se nos abrirá una ventana para seleccionar la carpeta en la que queremos guardar el proyecto. Le damos el nombre ***"hello"*** **y lo guardamos en un directorio cuya ruta no contenga espacios, caracteres especiales, tildes, ni ñ**.

NOTA

Recomendación: por ser ordenados, podríamos crear un directorio llamado "projects" guardado a su vez en una carpeta llamada con el nombre de la asignatura (por ejemplo "sdg" o "selc"). Ahí podemos guardar los proyectos que tengamos en la asignatura, pero puede elegir otro. El árbol de directorios sería para "sdg":

Se nos habrá creado un "workspace" con un aspecto como este (ver Figura 1.6):

Figura 1.6: Estructura del proyecto hello creado-

Quizás nos aparezca una ventana emergente como la de la Figura 1.7 preguntando si nos fiamos del propietario del directorio. En tal caso, le diremos que sí.

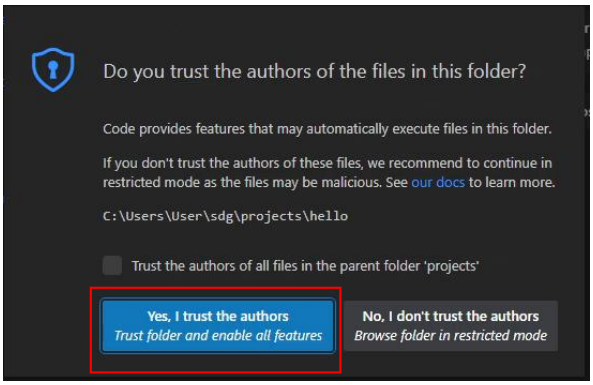
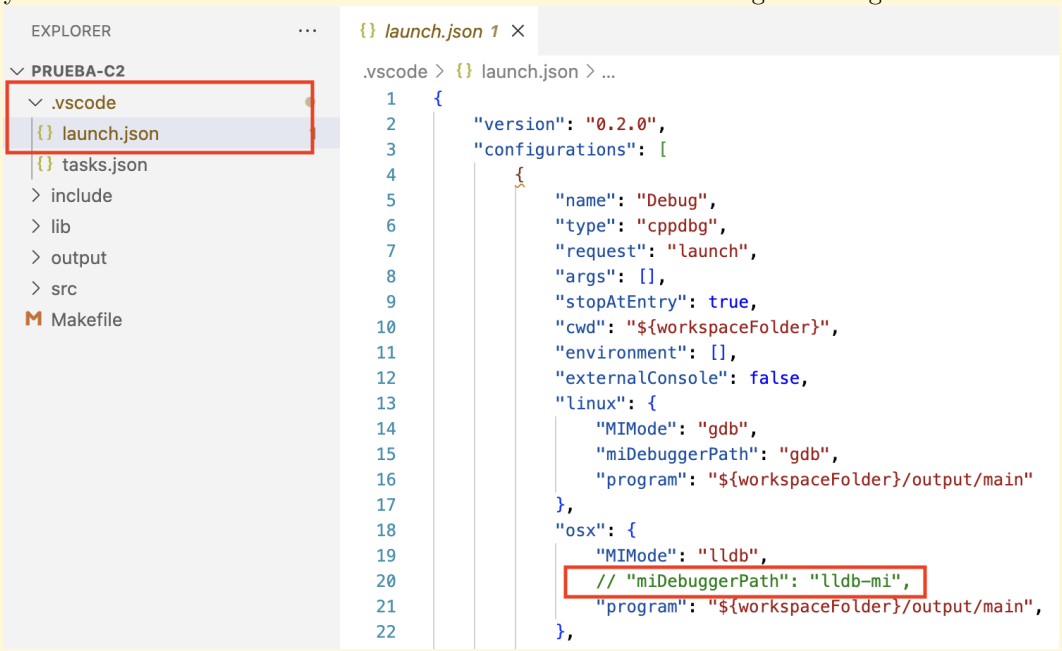


Figura 1.7: Confiar en la autoría de los ficheros del directorio creado.

ATENCIÓN

Si eres usuario de MacOS:

¡Antes de continuar! Abre el fichero ***"launch.json"*** que se encuentra en la carpeta **".vscode"**. Ve a la `línea 20` donde se encuentra la opción "miDebuggerPath" para "osx"' y comenta la línea con una doble barra como se muestra en la siguiente Figura:



1.2.3. Conectar VSCode con WSL (solo para Windows)

Como has observado, hemos creado el proyecto desde Windows, y lo hemos guardado en una ruta de nuestro sistema. Lo haremos así siempre porque WSL tiene acceso a esa ruta de Windows, pero no al revés.

Siempre que abramos un proyecto de C, haremos lo siguiente.

Las herramientas de compilación y depuración no las tenemos instaladas en Windows, sino en el subsistema Ubuntu que hemos instalado con WSL. Es por ello que tenemos que conectar VSCode con WSL para que ejecute todo desde allí. Se puede hacer de hasta tres formas distintas como nos dice la web de VSCode (Developing in WSL). La más sencilla es:

1. Abre el proyecto en VSCode si no lo tenías ya abierto.

2. Pulsa F1 (o Ctrl+Shift+P) para que se abra la *Command Palette*, y selecciona el comando **WSL: Reabrir carpeta en WSL** (**WSL: Reopen Folder in WSL**

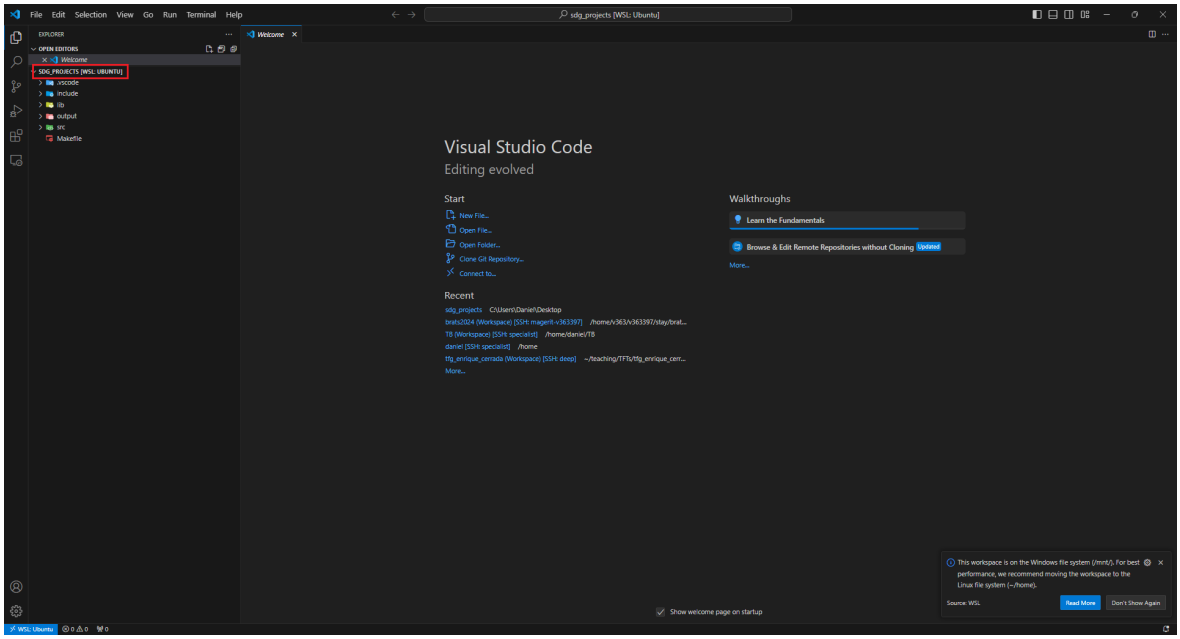


Figura 1.8: Carpeta de proyecto abierta en WSL desde VSCode.

en inglés). Se te preguntará qué distribución desea utilizar; selecciona Ubuntu. Se te abrirá tu carpeta de Windows en el subsistema Ubuntu como muestra la Figura 1.8.

Únicamente la primera vez que abrimos un proyecto en WSL.
Como ya hemos mencionado, Ubuntu y Windows son subsistemas diferentes que están interconectados. De hecho, cuando nos conectamos a WSL por VSCode, realmente dejamos de usar el VSCode que hemos instalado en Windows y empezamos a usar un VSCode instalado dentro del subsistema Ubuntu. Debido a que son VSCodes distintos, en Ubuntu no dispondremos de todos los plugins de VSCode que nos hemos descargado previamente. Por eso, si vamos a la pestaña de plugins, veremos que VSCode ahora los clasifica por plugins instalados en Windows y plugins instalados en WSL.
Debemos asegurarnos de volver a instalar **en WSL**, al menos, las siguientes extensiones de VSCode. Se verá algo como lo de la Figura 1.9:

- La extensión **"C/C++ Extension Pack"**
- Una extensión útil para ver binarios es ***"Hex Editor"***.

1.2.4. Prueba de depuración

Vamos a comprobar que el proyecto *hello* compila y depura correctamente. Para ello:

- Hacemos doble clic sobre el fichero *"main.c"* que está dentro del directorio *"src"*.

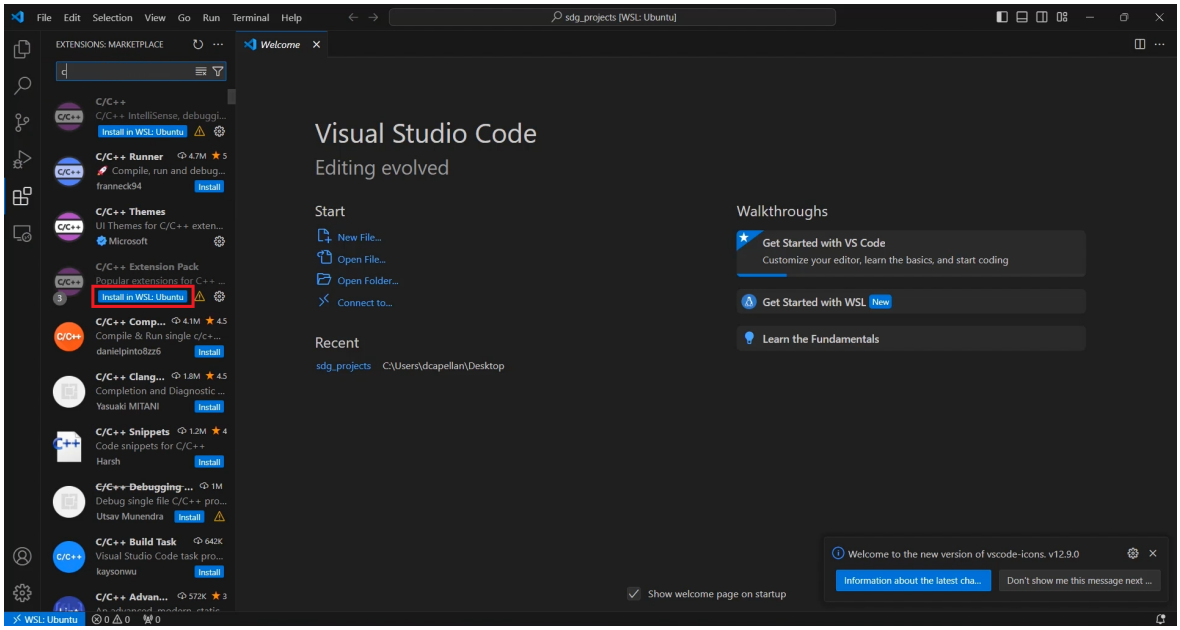


Figura 1.9: Reinstalar extensiones de VSCode en WSL.

- Colocamos un punto de parada haciendo clic junto al número de la línea de código 7, como muestra la Figura 1.10.

Figura 1.10: Incluir un punto de parada haciendo clic junto al número de la línea de código 7

- Abrimos el depurador, que aparece en la barra lateral izquierda con el icono 
- Al abrirlo nos cambia la vista al entorno de depuración, y nos aparecen las ventanas de variables y los puntos de parada, entre otras cosas.
- Pinchamos sobre la flecha "verde"  que compilará el proyecto, y ejecutará el programa, haciendo que se pare en la línea 7.
    - Otras formas que tenemos de lanzar la depuración son:
        - Desde la pestaña "Run → Start Debugging":

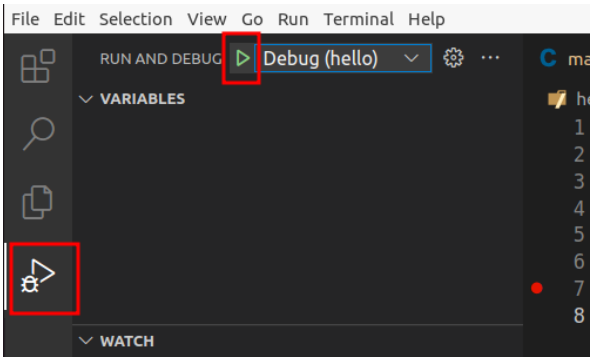


Figura 1.11: Entorno de depuración

- Presionando F5

- Al iniciarse la depuración, habrá aparecido la barra con las acciones del depurador. La barra es móvil, y contiene -en este orden- las acciones de: 
  - Continuar: Continúa la ejecución hasta el siguiente punto de parada, si lo hay.
  - Saltar: Salta de una línea a la siguiente. Si implica una llamada a función, la ejecuta.
  - Entrar: Entra a la función sobre la que estemos.
  - Salir: Sale de la función en la que estemos.
  - Reiniciar: relanza el depurador.
  - Parar: detiene la depuración.

- En la pestaña *"Terminal"* habrá aparecido el mensaje "Hello World!" como se muestra en la Figura 1.12. En esta misma figura también se observa, a la derecha, que podemos tener acceso a la terminal del sistema operativo (PowerShell en el caso de Windows, por ejemplo) si pinchamos en *"build (hello)"*, y de vuelta a la terminal de la ejecución/depuración del programa si pinchamos en "cppdbg: hello".

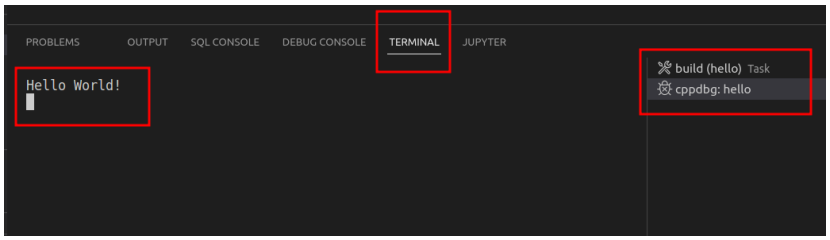


Figura 1.12: Impresión por la ventana de terminal tras la ejecución de la función printf

Si usas MacOS: en MacOS, la salida del programa **NO** aparece en la pestaña *"Terminal"*. En nuestro caso, la salida está en la pestaña *"Debug Console"*, como se muestra en la Figura 1.13. Además, de la salida del programa, veremos más *logs* del depurador en color amarillo. Podemos ignorarlos.

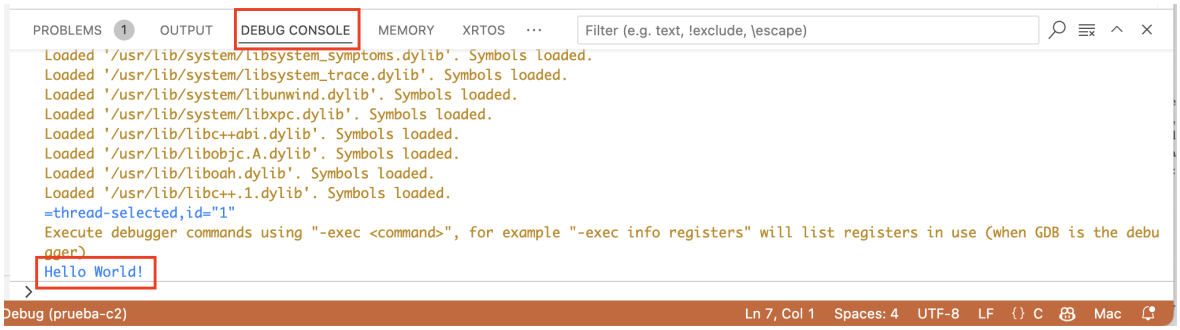


Figura 1.13: Impresión por la ventana de depuración en MacOS.

- Pulsamos continuar para finalizar el programa.

Capítulo 2

Guía de instalación de herramientas para compilación cruzada en C

Román Cárdenas, Amadeo de Gracia, Sergio Esteban, Josué Pagán

El presente capítulo tiene como objetivo definir los pasos de instalación de (i) las herramientas de trabajo VSCode, (ii) de las herramientas de compilación cruzada para trabajar con la placa **Nucleo-STM32F446RE** (y similares), y (iii) configuración del entorno de proyecto para una experiencia de programación más cómoda. El documento está descrito para los sistemas operativos Windows, MacOS y Linux. Los pasos están probados y funcionan, pero pueden depender de las particularidades de la computadora de cada usuario. Lea con atención y adapte los pasos y rutas si lo considera necesario.

Esta guía de instalación requiere de la configuración previa presentada en el Capítulo 1 "Guía de instalación de VSCode y herramientas de desarrollo en C".

Vamos a instalar una serie de herramientas, y a trabajar con una estructura de proyecto que nos permita trabajar de forma cómoda y eficiente, y que será útil para múltiples plataformas y microcontroladores. Este entorno que vamos a crear se llama **MatrixMCU**.

NOTA

MatrixMCU es el acrónimo de *"Microcontroller Application Toolkit Redefining Integrated Xperience for MCUs"*, y es el nombre que se le ha dado al conjunto de herramientas y entorno de desarrollo (*toolkit*) para programación en máquina desnuda (*baremetal*) en dispositivos empotrados con la que vamos a trabajar.

2.1. Instalación de las herramientas de desarrollo

Vamos a necesitar instalar (i) la toolchain de ARM para C, (ii) los drivers para comunicarnos con el programador/ depurador in-circuit ST-LINK/V2, y el software para interactuar con el depurador.

Lo primero que necesitamos instalar es la toolchain para la arquitectura ARM (Arm GNU Toolchain). La toolchain —típicamente denominada "el compilador cruzado"— es un conjunto de herramientas que contiene, entre otros, el compilador de `C`, el *linker*, las librerías y el depurador. La compilación cruzada se encarga de convertir los códigos fuente de nuestro programa en ficheros binarios capaces de ser ejecutados en una plataforma con arquitectura ARM, que es la arquitectura del microcontrolador **STM32F446RE** que usaremos.

La *toolchain* que instalaremos se llama "`arm-none-eabi`". Los nombres de las *toolchains* tienen típicamente la forma de: **arch** [**-vendor**] [**-os**] **- eabi**, donde:

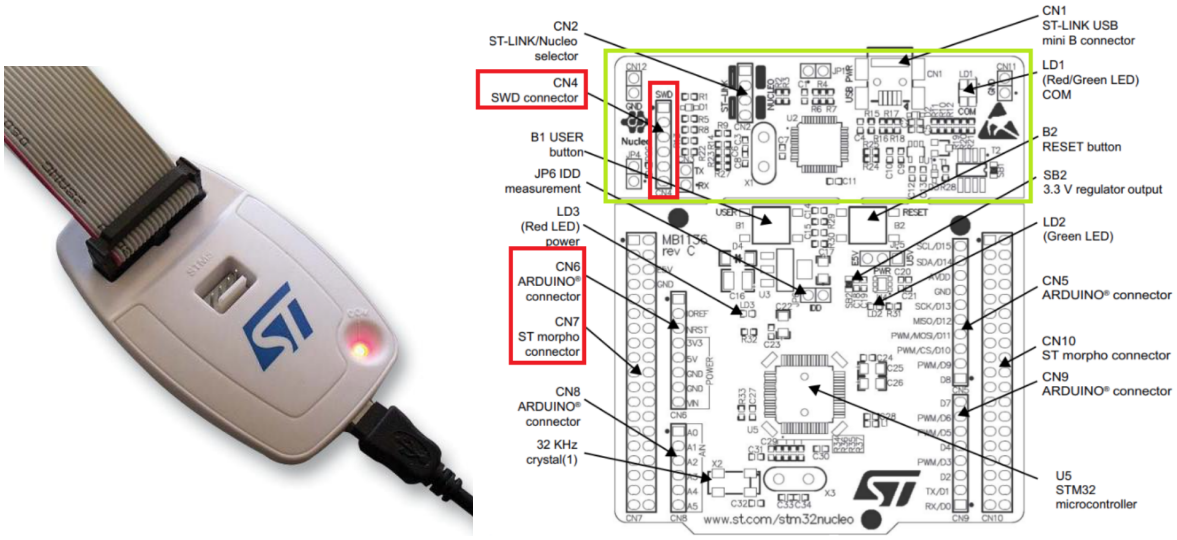
- **arch**: la arquitectura de la plataforma destino
- **vendor**: (opcional) quién proporciona la toolchain
- **os**: (opcional) sistema operativo de la plataforma destino
- **eabi**: significa Embedded Application Binary Interface

Por lo que "`arm-none-eabi`" es la toolchain para una arquitectura `ARM`, `sin proveedor`, el sistema operativo destino es "ninguno" (`none`) y compila con la `ARM EABI`.

Lo segundo que necesitamos instalar son los *drivers* para comunicarnos con el programador/ depurador *in-circuit* ST-LINK/V2, y el software `OpenOCD` para interactuar con el depurador Hardware (HW) ST-LINK/V2. ST-LINK/V2 es la segunda versión de un programador/ depurador de la empresa STMicroelectronics (ST). El ST-LINK/V2 (ver Figura 2.1a) permite programar nuestro microcontrolador y depurar el código sobre el propio chip. El ST-LINK utiliza el estándar/ interfaz de comunicación Joint Test Action Group (JTAG) (o su adaptación para ARM JTAG/serial Wire Debugging Software (SWD)).

En nuestra placa **Nucleo-STM32F446RE** no necesitamos el dispositivo de la Figura 2.1a, pues ya está integrado en la placa; es todo el submódulo marcado en la Figura 2.1b. Esto es muy común en las placas de desarrollo como la nuestra, pero no en las placas desplegadas en producción. Esta parte contiene la alimentación de la placa y el ST-LINK.

A modo informativo, decir que, si la recortásemos —por ejemplo para reducir espacio— necesitaríamos alimentar el microcontrolador mediante V_{IN}, $E5V$ y `3.3V` a través de



(a) Programador/ depurador ST-LINK/V2 de ST

(b) Programador/ depurador ST-LINK/V2 en placa.

Figura 2.1: Programador/ depurador ST-LINK.

los conectores `CN7` o `CN6` marcados también en la Figura 2.1b. En tal caso, para poder seguir programando el microcontrolador sí necesitaremos conectar el ST-LINK/V2 con cables SWD al conector `CN7`.
Para que nuestro PC pueda comunicarse con ST-LINK/V2 necesitamos un *driver*, y para que podamos depurar necesitamos un programa que se comunique con el depurador Software (SW) GNU Debugger (GDB) (instalado con la toolchain) y el depurador HW que es el ST-LINK/V2. Dicho programa será `OpenOCD`.
Elija su sistema operativo y vaya a la sección adecuada a continuación.

2.1.1. Instalación en Windows

Para la instalación en Windows es suficiente con que siga los pasos del `README` del repositorio de la *toolkit* **MatrixMCU** `https://github.com/sdg2DieUpm/install-MatrixMCU` para Windows. No obstante, aquí se detallan los pasos igualmente.

Pre-requisitos

Asegúrese de haber seguido los pasos del Capítulo 1 donde se instalan las herramientas necesarias para trabajar utilizando WSL.
Es importante que compruebe que en WSL está instalada la distribución **"Ubuntu 24.04"**. Puede comprobarlo abriendo el programa "Ubuntu" en el buscador de Windows y ejecutando la siguiente instrucción donde debería ver algo similar a lo mostrado en la Figura 2.2 en caso de estar todo instalado correctamente:

```
sdg2@sdg2$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=24.04
DISTRIB_CODENAME=noble
DISTRIB_DESCRIPTION="Ubuntu 24.04.1 LTS"
sdg2@sdg2$
```

Figura 2.2: Salida por pantalla que muestra la distribución instalada en WSL.

```
1 cat /etc/lsb-release
```

IMPORTANTE

En caso de que su versión de Ubuntu sea diferente a la de `Ubuntu-24.04`, vaya al Apéndice A y siga los pasos indicados para obtener la versión adecuada para el correcto desarrollo de esta guía.

USBIPD

Para hacer funcionar nuestro proyecto será necesario conectar nuestra placa STM32 con WSL (que es una máquina virtual) a través del sistema *host* que es Windows. Esto no está habilitado por defecto en WSL, por lo que será necesario instalar **usbipd-win** primero. Para ello, siga los siguientes pasos:

1. Abra una ventana de `Windows Powershell` haciendo clic derecho sobre ella y seleccione la opción de **"Ejecutar como administrador"**. El sistema nos preguntará si queremos permitir que la aplicación haga cambios en nuestro dispositivo. Seleccione **"Sí"**.

2. Ejecute el siguiente comando en `Windows Powershell`:

```
1 winget install usbipd
```

   Es probable que el gestor de paquetes **winget** pregunte si quiere aceptar los términos de uso. Inserte **"Y"** en la ventana de terminal de Powershell para aceptarlos. Reinicie el PC para aplicar los cambios realizados hasta el momento.

Para comprobar si **usbipd-win** se ha instalado correctamente, siga los siguientes pasos:

3. Abra de nuevo una ventana de `Windows Powershell` haciendo clic derecho sobre ella, y seleccione la opción de **"Ejecutar como administrador"**

4. Ejecute el siguiente comando en `Windows Powershell`:

```
PS C:\WINDOWS\system32> usbipd --help
usbipd-win 4.3.0

Description:
  Shares locally connected USB devices to other machines, including Hyper-V guests and WSL 2.

Usage:
  usbipd [command] [options]

Options:
  --version       Show version information
  -?, -h, --help  Show help and usage information

Commands:
  attach   Attach a USB device to a client
  bind     Bind device
  detach   Detach a USB device from a client
  license  Display license information
  list     List USB devices
  policy   Manage policy rules
  server   Run the server on the console
  state    Output state in JSON
  unbind   Unbind device

PS C:\WINDOWS\system32>
```

Figura 2.3: Salida por pantalla que muestra la documentación de ayuda asociada al paquete usbipd.

```
1 usbipd --help
```

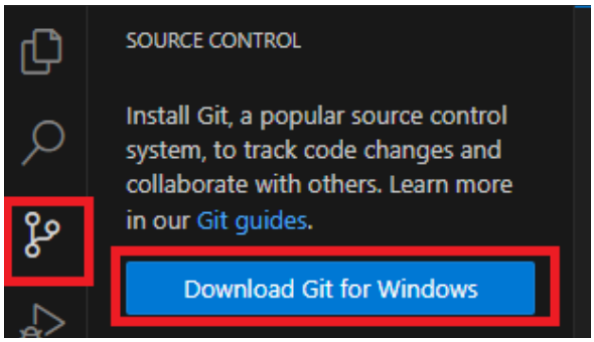
De estar correctamente instalado, debería ver algo similar a lo mostrado en la Figura 2.3.

Si quisiera información adicional acerca del paquete **usbipd-win**, la puede encontrar en su correspondiente Repositorio GitHub.

Git en Windows

Más adelante necesitaremos tener instalado `Git`. Puede hacerlo desde VSCode:

5. Desde la barra lateral acceda a `Source Control` .
6. Si no está instalado, dele a *Download Git...*.



7. Siga los pasos, con las opciones por defecto, para instalar `Git`. En Windows, asegúrese de descargar la versión *"Standalone Installer"* de 64 bits.
8. Cierre y relance VSCode.

De ahora en adelante, utilice la terminal de Ubuntu para instalar las dependencias de Linux que se le detallarán más adelante en esta guía. Siga los pasos indicados en la guía de instalación en Linux descritos en la Sección 2.1.2 sobre una **terminal de Ubuntu**, no sobre `Windows Powershell`.

2.1.2. Instalación en Linux

Para la instalación en Linux es suficiente con que siga los pasos del `README` del repositorio de la *toolkit* **MatrixMCU** `https://github.com/sdg2DieUpm/install-MatrixMCU`. No obstante, aquí se detallan los pasos igualmente.

Pre-requisitos

Asegúrese de tener instalado el gestor de repositorios `git`. En caso de no tenerlo instalado, abra una terminal y escriba los siguientes comandos:

- Para actualizar el repositorio de paquetes:

```
sudo apt update
```

- Para instalarlo:

```
sudo apt install git
```

A continuación, vamos a descargarnos (*clonar*) el repositorio de instalación de las herramientas de compilación necesarias. Para ello, en la terminal:

1. Clonaremos el repositorio de instalación (`git clone`), y entramos (`cd`):

```
git clone https://github.com/sdg2DieUpm/install-MatrixMCU.git
cd install-MatrixMCU
```

2. Ejecutamos el *script* para Linux. Para ello escriba:

```
source ./ubuntu24.sh
```

   Pulse *enter*; le pedirá su contraseña que puso en la instalación de Ubuntu.

3. Cuando haya acabado, reinicie la terminal, cerrándola y abriéndola de nuevo.

4. Comprobamos **uno por uno** que se han instalado correctamente todo lo necesario. Para ello, escribimos en la terminal, los siguientes comandos pulsando *enter* cada vez. Tras ejecutar cada uno, debería aparecer la versión de la herramienta instalada:

```
arm-none-eabi-gcc --version
arm-none-eabi-g++ --version
arm-none-eabi-gdb --version
arm-none-eabi-objcopy --version
arm-none-eabi-objdump --version
```

5. Tras la instalación, puede eliminar el repositorio clonado `install-MatrixMCU`, si lo desea:

```
cd ..
rm -rf install-MatrixMCU
```

A continuación, vamos a instalar `Doxygen` para generar la documentación de nuestro código. Para ello, vaya a la Sección 2.2.

2.1.3. Instalación en MacOS

Para la instalación en MacOS es suficiente con que siga los pasos del `README` del repositorio de la *toolkit* **MatrixMCU** `https://github.com/sdg2DieUpm/install-MatrixMCU`. No obstante, aquí se detallan los pasos igualmente.

Pre-requisitos

Asegúrese de tener instalado el gestor de paquetes Homebrew. Si no lo tiene, abra una terminal (en el `Spotlight` de nuestro Mac escribimos `terminal`). A continuación, escribiremos el siguiente comando:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

El *script* explica lo que hará y luego se detiene antes de hacerlo.
Para comprobar que la instalación ha sido correcta, **abra una nueva terminal** y escriba `brew --version` y pulse *enter*. Debería aparecer la versión de `brew` instalada.

Instalación de las herramientas

A continuación, vamos a descargarnos (*clonar*) el repositorio de instalación de las herramientas de compilación necesarias. Para ello, en la terminal:

1. Clonaremos el repositorio de instalación y ejecutaremos el instalador:

```
git clone https://github.com/sdg2DieUpm/install-MatrixMCU.git
cd install-MatrixMCU
./macos.sh
```

2. Comprobamos **uno por uno** que se han instalado correctamente todo lo necesario. Para ello, escribimos en la terminal, los siguientes comandos de uno en uno y pulsamos *enter* cada vez. Tras ejecutar cada uno, debería aparecer la versión de la herramienta instalada:

```
arm-none-eabi-gcc --version
arm-none-eabi-g++ --version
arm-none-eabi-gdb --version
arm-none-eabi-objcopy --version
arm-none-eabi-objdump --version
```

3. Tras la instalación, puede eliminar el repositorio clonado `install-MatrixMCU`, si lo desea:

```
cd ..
rm -rf install-MatrixMCU
```

A continuación, vamos a instalar `Doxygen` para generar la documentación de nuestro código. Para ello, vaya a la Sección 2.2.

2.2. Instalación de Doxygen

En el ámbito profesional de desarrollo de proyectos la documentación del código es fundamental. Por un lado, desde el punto de vista del cliente, porque éste tendrá constancia de todo lo que hace su código entregado. Por otro lado, desde el punto de vista interno de la empresa, porque el proyecto podrá tener continuidad fácil sin depender de quién lo desarrolló. Y desde el punto de vista personal, porque podremos saber qué hacía el código que escribimos.
Estamos acostumbrados a ver documentación de código cuando utilizamos librerías, "paquetes", "módulos"... de casi cualquier lenguaje o entorno de desarrollo: Python, Matlab, Java, y por supuesto, C. A esta documentación se le denomina comúnmente Application Programming Interface (Interfaz de Programación de Aplicaciones) (API): una interfaz nos especifica qué recibe, qué devuelve una función y, qué y cómo lo hace. Escribir estas APIs puede ser una tarea titánica, pero afortunadamente contamos con herramientas como Doxygen que interpretan ciertas palabras clave en los comentarios del código y lo compila generando una web o documento por el que podemos navegar y que constituye la API.
En los proyectos que hagamos deberemos documentar nuestro código (ficheros, funciones, *defines*, estructuras, *etc.*). Un ejemplo de API es la que se encuentra en el proyecto/ plantilla que usaremos más tarde `https://github.com/sdg2DieUpm/c_cross_template`. Cómo usar Doxygen para documentar está explicado en el libro de referencia [1].

2.2.1. Instalación en Windows

Como estamos trabajando sobre la máquina virtual WSL con Ubuntu, vaya a la Sección 2.2.2 y siga los pasos de instalación en Linux. Hágalo sobre la terminal de WSL.

2.2.2. Instalación en Linux

Abra un terminal y escriba el siguiente comando:

```
sudo apt install doxygen
```

Podemos comprobar que se ha instalado correctamente escribiendo el siguiente comando:

```
doxygen --version
```

Si nos imprime el número de versión de Doxygen instalado, es que ha ido bien.
Ya tenemos instaladas las herramientas. Ahora vaya a la Sección 2.3.

2.2.3. Instalación en MacOS

De nuevo, haremos uso del gestor de paquetes Homebrew para instalar Doxygen. Abra una terminal y ejecute el siguiente comando:

```
brew install doxygen
```

Podemos comprobar que se ha instalado correctamente escribiendo el siguiente comando:

```
doxygen --version
```

Si nos imprime el número de versión de Doxygen instalado, es que ha ido bien.
Ya tenemos instaladas las herramientas. Ahora vaya a la Sección 2.3.

2.3. Setup de desarrollo MatrixMCU y VSCode

En esta sección vamos a configurar el entorno de desarrollo con la *toolkit* **MatrixMCU** para trabajar con la placa **Nucleo-STM32F446RE**. Nuestro editor de proyectos con la *toolkit* será VSCode.
Primero descargaremos el *workspace* (entorno de trabajo de **MatrixMCU**). Después, instalaremos unas extensiones que nos facilitarán la tarea de programación y depuración en VSCode. Finalmente, configuraremos todo para que compile y cargue en la placa nuestro primer proyecto.

2.3.1. Descarga del workspace de MatrixMCU

Como se ha comentado, el *workspace* es el entorno de trabajo de la *toolkit* **MatrixMCU** que se ha preparado para trabajar con los proyectos en los que usaremos la **Nucleo-STM32F446RE**, pero que también permite otras placas. El *workspace* contiene los ficheros de configuración de VSCode, CMake, librerías, la Hardware Abstraction Layer (Capa de Abstracción Hardware) (HAL) de ST, y una carpeta para alojar los proyectos. Para ello vamos a seguir los siguientes pasos, aunque se puede conseguir el mismo resultado de otras formas:

1. Creamos una carpeta donde queramos instalar el entorno de trabajo. Esta carpeta contendrá el *workspace* de **MatrixMCU**. Dele a la carpeta un nombre representativo para usted, como `entornos`, o `nombre_de_asignatura`. Un lugar posible puede ser:

   - En Windows: `C:\Users\$USER\Documentos\entornos\`, por ejemplo, donde `$USER` es su nombre de usuario.
   - En Linux y MacOS: `home/$USER/entornos/`, por ejemplo, donde `$USER` es su nombre de usuario.

¡IMPORTANTE! Ni la ruta donde se guarde ni el nombre de la carpeta creada puede contener espacios, caracteres especiales, tildes, ni ñ.

ATENCIÓN

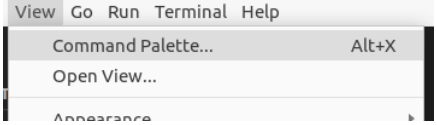
Si está utilizando **Windows**, **tenga en cuenta que los ficheros del proyecto (el código), estará en Windows, aunque la edición, compilación y depuración se haga desde WSL.**
Hágase la imagen mental: VSCode está instalado en Windows, pero se conectará a WSL donde están las herramientas de compilación, para trabajar con ficheros que, a su vez, están en Windows.
¿Por qué hacemos esto? Lo hacemos porque es más fácil instalar y mantener el entorno en Linux que en Windows.

Ahora vamos a descargarnos el *workspace* de **MatrixMCU** desde el repositorio de GitHub: `https://github.com/sdg2DieUpm/MatrixMCU`. En realidad, lo que haremos será *clonar* el repositorio de manera **recursiva**, pues contiene otros repositorios como submódulos.

2. Abrimos VSCode y abrimos la paleta de comandos.

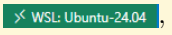
   Recuerde que se puede hacer de distintas formas:

   - Desde la pestaña "View → Command Palette":

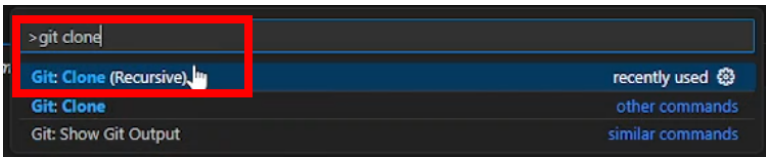
   - Presionando Alt+X
   - Presionando F1
   - Presionando Ctrl+Shift+P

ATENCIÓN

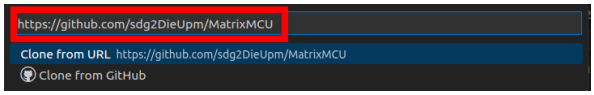
Si está utilizando **Windows**, asegúrese de que **NO está conectado a WSL**, pues queremos clonar en nuestro directorio *host*. Puede comprobarlo fijándose en la esquina inferior izquierda de de su ventana de VSCode. Si ve algo como `WSL:Ubuntu-24.04` 

, en la *"Command Palette"* busque y seleccione la opción *"Close Remote Connection"*.

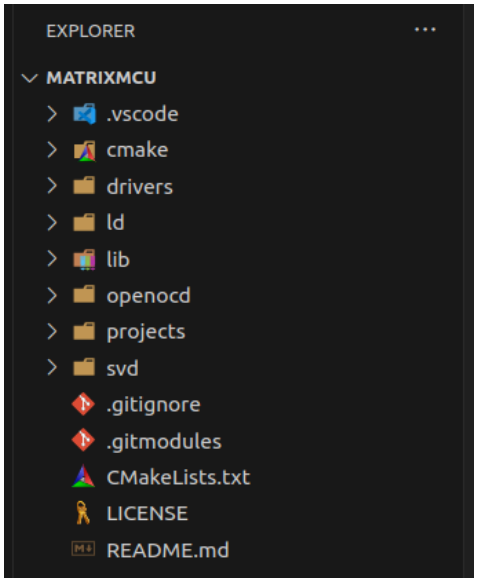
3. A continuación escribimos *"Git: Clone (Recursive)"* (**¡importante que sea recursivo!**).



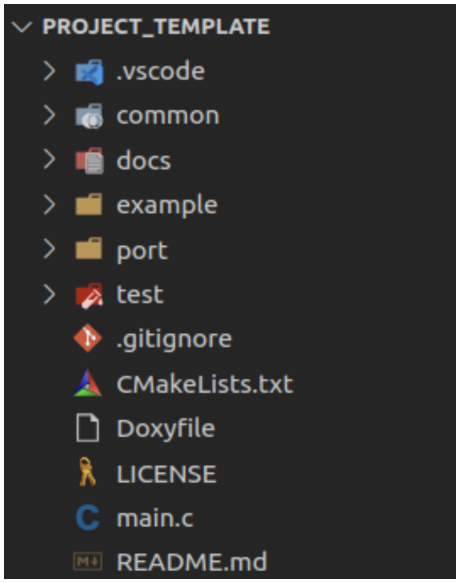
4. Inmediatamente nos pide la URL del repositorio. Escribimos o pegamos: `https://github.com/sdg2DieUpm/MatrixMCU` y pulsamos *enter*.



5. Nos pedirá la carpeta donde queremos clonar el repositorio. Seleccionamos la carpeta que hemos creado en el paso anterior. **¡Si está en Windows, seleccione la carpeta de Windows, no de WSL!**



(a) Workspace de MatrixMCU



(b) Carpeta de proyecto plantilla

6. Cuando acabe de clonar, nos preguntará si queremos abrir el repositorio. No importa si le decimos que sí, o no. A nosotros no nos interesa el *workspace* de **MatrixMCU** en sí, sino los proyectos que crearemos en él. No obstante, si lo abre, verá algo como lo de la Figura 2.4a.

Ya tenemos descargado el entorno de trabajo. Ahora vamos a descargar, desde VSCode, las extensiones necesarias para el correcto desarrollo del proyecto. Continue en la Sección 2.3.2.

2.3.2. Instalación de extensiones de VSCode

ATENCIÓN

Si está utilizando **Windows**, **conecte VSCode a WSL** en la distribución `Ubuntu-24.04` WSL: Ubuntu-24.04, ya que es donde deseamos instalar las extensiones. En la *"Command Palette"* busque y seleccione la opción ``WSL: Connect to WSL using Distro'' o ``WSL: Reopen folder in WSL'' y seleccione ``Ubuntu-24.04''.

Desde la barra lateral de VSCode accedemos a las Extensiones . En la cajetilla del buscador de la esquina superior izquierda buscamos e instalamos las siguientes:

1. La extensión "**Cortex-Debug**" que da soporte de depuración para arquitecturas ARM Cortex-M, se comunica con `OpenOCD`, permite ver registros y periféricos del micro, *etc.*

2. ${command:} La extensión "**Command Variable**", que permite sustitución de variables en los ficheros tasks.json y launch.json de VSCode.

3. La extensión "**Serial Monitor**", que permite abrir un terminal serie en VSCode. Es útil si vamos a comunicarnos con la placa por el puerto serie.

4. La extensión "**Todo Tree**", que es muy útil para ver las cosas que nos quedan por hacer (*to do*) en nuestro código cuando ponemos un comentario con la palabra clave *TODO*: `// TODO: comprobar XX de esta funcion`, por ejemplo.

5. La extensión **"C/C++"** que incluye el *intellisense* que nos permitirá autocompletar código, ver la documentación de las funciones, y mucho más.

Adicionalmente, **si está utilizando Windows**, instale también una extensión que le permitirá conectar WSL a la placa:

6. La extensión **"USBIP Connect"**, que permite hacer los procesos de "attach" y "detach" de la placa STM32 de manera sencilla desde la interfaz de VSCode, sin necesidad de abrir `Windows Powershell`.

   Recuerde nuevamente, que esta extensión debe ser instalada con el VSCode activo sobre WSL en la versión de `Ubuntu-24.04`.

   Para utilizarla active la *"Command Palette"* presionándo la tecla "F1" o utilizando el atajo de teclado "Ctrl + Shift + P" y busque "usbip". Deberían aparecer ambas opciones como se muestra en la Figura 2.4. Aunque también le aparecerá en la barra inferior de VSCode.

Figura 2.4: Opciones de Attach y Detach utilizando la extensión USBIP directamente desde VSCode.

Continuamos en la Sección 2.4 con la carga de un proyecto de ejemplo que servirá para verificar que tenemos el entorno de trabajo listo para el correcto desarrollo de la asignatura.

2.4. Compilación, depuración y carga de un proyecto

En esta sección vamos a probar que todo funciona correctamente y a crear nuestro primer proyecto, y vamos a compilarlo y cargarlo en la placa **Nucleo-STM32F446RE**. Vamos a probar un sencillo programa que hace parpadear el Light-Emitting Diode (Diodo Emisor de Luz) (LED) `LD2` (LED verde) de la placa **Nucleo-STM32F446RE** cada 1000 ms.

Toda la explicación de la estructura de ficheros de los proyectos para la *toolkit* **MatrixMCU** puede verla en detalle en el *"Capítulo: Desarrollando para Nucleo-STM32"* del libro de referencia [1].

ATENCIÓN

Lo que siempre tenemos que tener abierto en VSCode es un proyecto, no el *workspace* de MatrixMCU, ni tampoco la carpeta `projects`, sino alguno de los proyectos que tengamos dentro de dicha carpeta. Podemos estar seguros de que es un proyecto si vemos la carpeta `.vscode` dentro de la carpeta, así como el resto de directorios.

Para abrir un proyecto, seguimos los siguientes pasos:

1. Vamos a *"File → Open Folder"*, y nos movemos dentro de la carpeta *MatrixMCU*, y dentro de la carpeta `projects` seleccionamos nuestro primer proyecto, que será el proyecto plantilla `project_template`. Se abrirá y veremos algo como lo mostrado en la Figura 2.4b.

   Si es usuario de `Windows`, debe asegurarse de que abre el proyecto en WSL, ya que es en Ubuntu donde hemos instalado las dependencias del mismo. Fíjese en la esquina inferior izquierda y compruebe que muestra algo similar a "`WSL: Ubuntu-24.04`" 

. De no ser así, o mostrar una versión de Ubuntu diferente a `Ubuntu-24.04` debe hacer lo siguiente:

2. **Si ya tenía abierto el proyecto:** simplemente presione la tecla "F1" o el atajo del teclado "Ctrl + Shift + P" y busque la opción "WSL: Reopen folder in WSL"

3. **Si aún no tenía abierto el proyecto:** puede o bien abrirlo y, una vez abierto, seleccionar la opción indicada en el punto anterior o directamente pulsar la tecla "F1" o el atajo del teclado "Ctrl + Shift + P", buscar la opción "WSL: Connect to WSL" y posteriormente, buscar de manera normal con *"File → Open Folder"* el proyecto en su sistema de archivos.

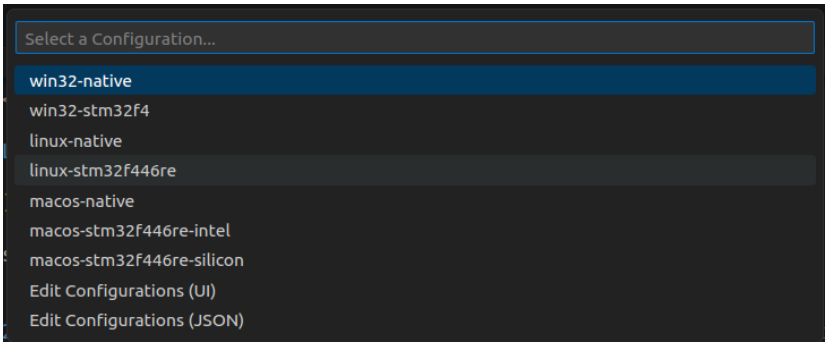
¡IMPORTANTE! Si en Windows la versión de Ubuntu que nos aparece en VSCode es diferente a `WSL: Ubuntu24.04`, será necesario que en su lugar seleccione "WSL: Reopen folder in WSL using distro" como "WSL: Connect to WSL using distro", respectivamente. Al hacerlo, debería aparecernos la opción de usar `Ubuntu24.04`. Pulse la tecla "Enter" sobre dicha opción.
Si en algún momento nos preguntara si confiamos en los autores de la carpeta, le decimos que **sí**.

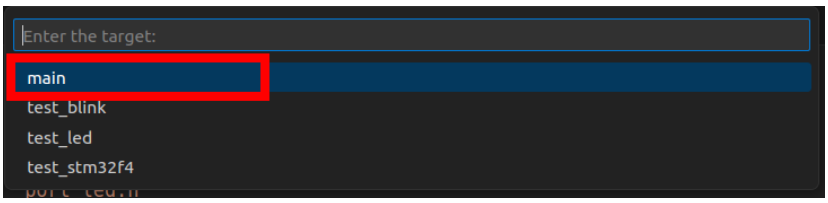
4. Abrimos el fichero `main.c` que contiene el código fuente de nuestro programa. En este caso, el programa es muy simple, solo parpadea un LED y se queda en un bucle infinito, pero es muy útil para probar que todo funciona correctamente, porque además está hecho con una máquina de estados.

   Esta carpeta no la edite, ni la borre. Es la plantilla que usaremos para crear nuevos proyectos. En los proyectos que copiemos sí podemos editarla y borrar los ficheros fuente (`.c` y `.h`) que no sean necesarios.

5. En la esquina inferior derecha de VSCode veremos algo como 

. Pinche sobre él para adaptarlo a su entorno. Esto ajusta las rutas de las librerías del compilador, y su nombre hace referencia a: (i) sistema operativo desde el que se trabaja, (ii) arquitectura de la plataforma destino, que puede ser un ordenador (compilación nativa) o un microcontrolador (compilación cruzada).





Al cambiarlo, si había líneas subrayadas en rojo, desaparecerán. Si no desaparecen, es que no ha elegido correctamente la configuración, o no se ha refrescado el fichero. Para refrescar el fichero, cierre el fichero y vuelva a abrirlo.

A continuación vamos a compilar y cargar el programa en la placa **Nucleo-STM32F446RE**.

6. Conecte la placa **Nucleo-STM32F446RE** al PC mediante el cable USB.

7. **Colocamos un punto de parada** en la llamada a `port_led_toggle()` haciendo clic junto al número de la línea.

8. **Abrimos la vista del depurador**, que aparece en la barra lateral izquierda con el icono .

9. Al abrirlo nos cambia la vista al entorno de depuración, y nos aparecen las ventanas de variables y los puntos de parada, entre otras cosas. Junto a la flecha verde nos aparece un desplegable con la configuración que tenemos definida (y otras si las hubiera). **Marcamos "Debug (stm32f446re)" y pulsamos** .

10. En ese instante nos preguntará qué *objetivo* queremos compilar. Seleccionamos `main`. El resto es para hacer pruebas de test.

    En este momento CMake habrá generado las reglas de compilación para nativo, y habrá creado los directorios necesarios para compilar como `build/stm32f446re/Debug` y `bin/stm32f446re/Debug`, donde se guardará el fichero binario ejecutable `.elf`. Instantáneamente, compilará el proyecto.

ATENCIÓN

Si está en Windows, en este punto, le habrá fallado. Esto se debe a que tenemos que decirle a Windows (la máquina *host*), que le comparta a WSL el acceso por USB a la placa **Nucleo-STM32F446RE** (**bind**). Además, le tendremos que decir a WSL que se conecte a dicha placa (**attach**). **Esto es un proceso que hará la primera vez o cada vez que cambie de placa**.
En máquinas virtuales como VMWare o VirtualBox esto es algo que también se hace, pero es algo más cómodo y se hace por interfaz gráfica. WSL no permite eso de momento.
Vaya al Apéndice C y siga los pasos. Cuando se inicie la depuración, vuelva aquí.

11. Al iniciarse la depuración, habrá aparecido la barra con las acciones del depurador: 

IMPORTANTE

Al depurar o al hacer *flash* hay veces que no "se carga" el código en la placa. Esto puede deberse a una inconsistencia en el estado del programador `STLINK`. Por ejemplo, a veces ocurre si una depuración previa no se termina correctamente y se desconecta la placa sin haberla detenido. Para solucionarlo:

1. Mantenemos pulsado el **botón de reset**.
2. Iniciamos la carga normalmente con *flash* o mediante la depuración.
3. Tenemos que soltar el botón cuando `OpenOCD` empiece a "subir" el código. Esto será cuando veamos una línea como: `Info : Unable to match requested speed 8000 kHz, using 4000 kHz`. Se para en esa línea un brevemente (alrededor de 1 segundo), por lo que tenemos que ser rápidos.

- A la izquierda, en el menú de depuración, aparecerán estas vistas de interés (Figura 2.5):
  - **Variables**: valores de las variables globales, locales, estáticas (si las hay) y registros.
  - **Watch**: visor de variables/ expresiones que se pretenden "vigilar".
  - **Call stack**: la pila de llamadas a función.
  - **Breakpoints**: la lista de puntos de parada.

- **XPeripherals**: registros de periféricos del microcontrolador y valores de los mismos (GPIOs, ADC, timers, etc.). Esta vista es posible gracias a los ficheros SVD que se han descargado con **MatrixMCU**.

- Fíjese en la Figura 2.5 que, en este punto de parada, el valor del 6° bit (**ODR5**) registro **GPIOA_ODR** vale `1`. Este registro contiene los valores de la GPIO cuando está configurada en modo salida. **En nuestra placa Nucleo-STM32F446RE, el `LD2` (LED verde) está conectado al puerto A, pin 5 (GPIOA 5)**.

IMPORTANTE

El microcontrolador está limitado a 7 puntos de parada por lo que si ponemos más, tendremos problemas al depurar. Si en el fichero `launch.json` tenemos la opción `stopAtEntry: true` —para que se pare en la primera línea del `main` al iniciar— entonces solo tendremos 6 puntos de parada. Podría ponerse a `false` para tener los 7.

12. Pulsamos continuar  , y se volverá a parar en el punto de parada que hemos colocado pasados 1000 ms (mitad del valor de espera del `#define BLINK_T_MS`).

    Compruebe que ahora el bit `ODR5` alterna entre `0` y `0` y se enciende y apaga el `LD2` (LED verde) de la placa.

13. A la derecha se nos habrá abierto una terminal con nombre `gdb-server`. Es una interfaz de depuración que permite, entre otras cosas, el uso de la función `printf` en depuración con la placa.

    Si pulsamos en dicha terminal podremos ver cómo se muestra también mensajes con el estado del LED y una marca de tiempo que es el `System tick` del microcontrolador (ver Figura 2.6).

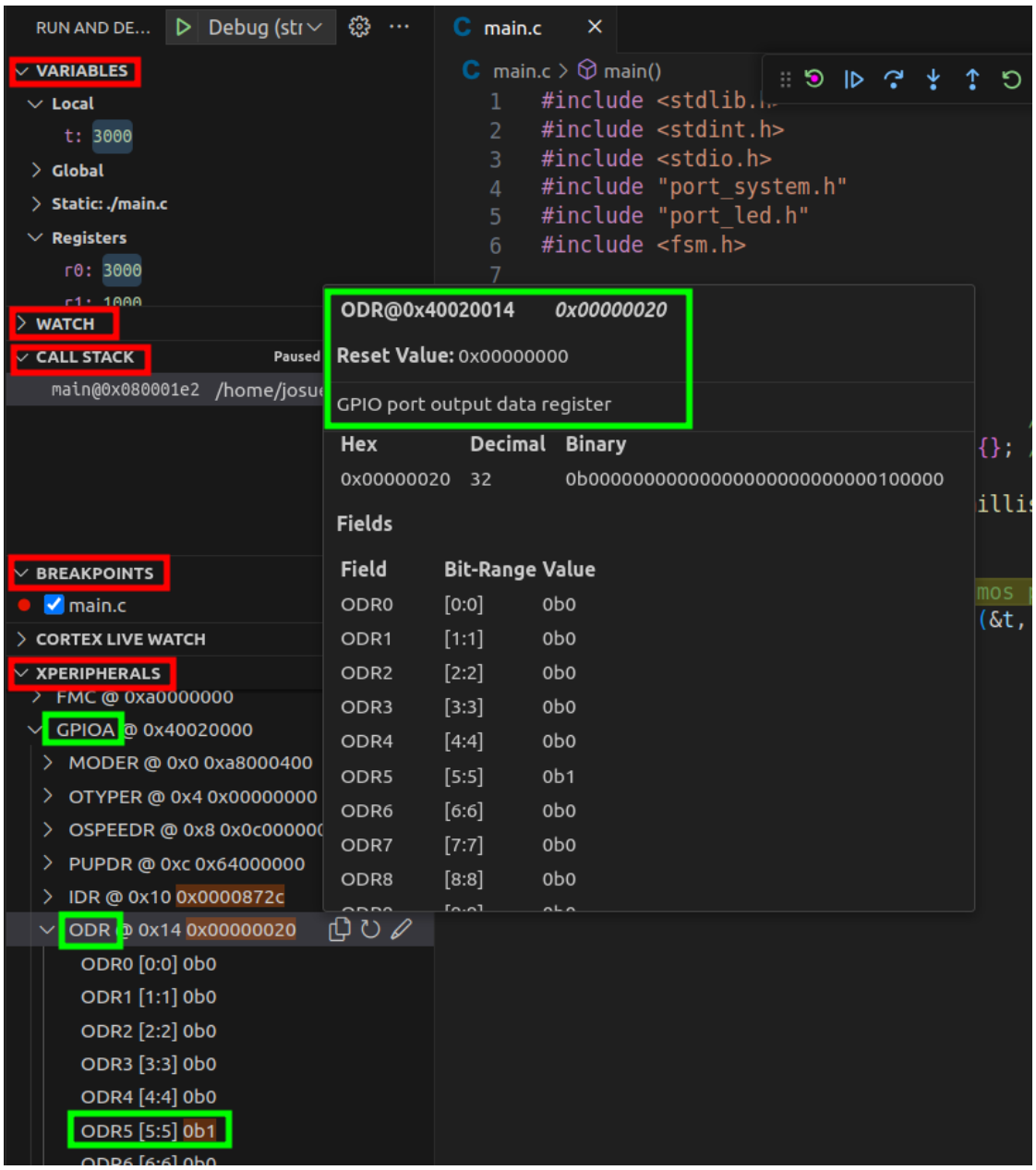


Figura 2.5: Vista del menú de depuración. Destacar el valor de ODR5 en el registro GPIOA_ODR.

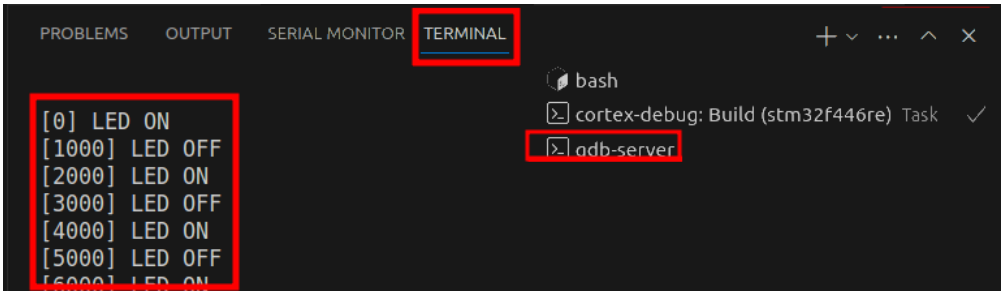


Figura 2.6: Ejemplo de salida de printf en la consola gdb-server indicando el estado del LED `LD2` cada $1000\ ms$.

Bibliografía

[1] J. Pagán Ortiz and R. Cárdenas Rodríguez, *Fundamentos teóricos de sistemas basados en microcontrolador STM32*. Universidad Politécnica de Madrid, 2023.

Apéndice A

Actualizar la versión de Ubuntu en WSL en Windows

En este apartado se detallarán los pasos a seguir para instalar la versión adecuada de Ubuntu en su WSL. No obstante, para mayor detalle sobre el procedimiento que vamos a seguir, puede referirse al enlace "Cambio de la distribución predeterminada de Linux instalada".
Siga estos pasos solo si su versión de Ubuntu instalada en WSL es diferente a la `Ubuntu-24.04`:

1. Abra una ventana de `Windows Powershell` haciendo clic derecho sobre ella y seleccione la opción de **"Ejecutar como administrador"**

2. Ejecute los siguientes comandos para ver, en primer lugar, una lista de las distribuciones disponibles. Verá algo como lo que se muestra en la Figura A.1. A continuación, proceda a instalar, con el segundo comando, la distribución de interés, en este caso `Ubuntu-24.04` :

```
wsl --list --online
wsl --install -d Ubuntu-24.04
```

   La instalación puede tardar un tiempo considerable así que no cancele el proceso y espere a que este termine.

Ahora vamos a comprobar, nuevamente, que la versión de Ubuntu que hemos instalado es la que es de nuestro interés para trabajar, así como mostrar cómo podemos acceder a ella. Para ello siga los pasos descritos a continuación:

1. Utilice el buscador de Windows y escriba "Ubuntu". Haga clic sobre dicha opción para abrir una terminal. Compruebe de que la versión de Ubuntu en la que se encuentra es la `Ubuntu-24.04`:

```
cat /etc/lsb-release
```

```
PS C:\WINDOWS\system32> wsl --list --online
A continuación, se muestra una lista de las distribuciones válidas que se pueden instalar.
Instalar con "wsl.exe --install <Distro>".

NAME                            FRIENDLY NAME
Ubuntu                          Ubuntu
Debian                          Debian GNU/Linux
kali-linux                      Kali Linux Rolling
Ubuntu-18.04                    Ubuntu 18.04 LTS
Ubuntu-20.04                    Ubuntu 20.04 LTS
Ubuntu-22.04                    Ubuntu 22.04 LTS
Ubuntu-24.04                    Ubuntu 24.04 LTS
OracleLinux_7_9                 Oracle Linux 7.9
OracleLinux_8_7                 Oracle Linux 8.7
OracleLinux_9_1                 Oracle Linux 9.1
openSUSE-Leap-15.6              openSUSE Leap 15.6
SUSE-Linux-Enterprise-15-SP5    SUSE Linux Enterprise 15 SP5
SUSE-Linux-Enterprise-15-SP6    SUSE Linux Enterprise 15 SP6
openSUSE-Tumbleweed             openSUSE Tumbleweed
```

Figura A.1: Salida por pantalla que muestra una lista de todas las distribuciones de Linux que soporta WSL para ser instaladas.

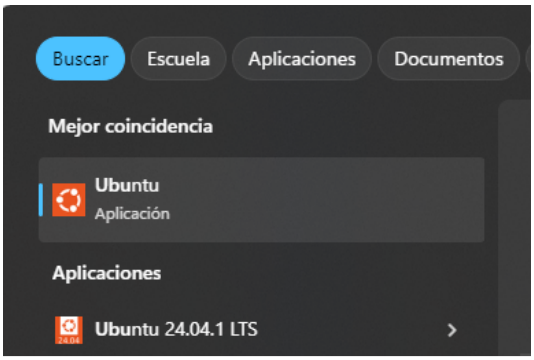


Figura A.2: Múltiples versiones de Ubuntu instaladas sobre un mismo WSL.

De ser así, ya ha acabado. De no serlo, continue leyendo.

En caso de persistir la versión de Ubuntu que ya tenía preinstalada:

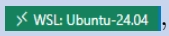
1. Vuelva nuevamente al buscador de Windows, pero en este caso busque específicamente por `Ubuntu-24.04` (con escribir "Ubuntu" en el buscador debería de ser suficiente).

   Por lo general, debería aparecer con un nombre similar a `‘‘Ubuntu 24.04.1 LTS’’` como se muestra en la Figura A.2 donde se ejemplifica el caso descrito en este punto.

2. Una vez identificado, haremos clic sobre `‘‘Ubuntu 24.04.1 LTS’’` y de nuevo comprobaremos la versión de Ubuntu.

   En este caso, debería de ser la misma que la mostrada en la Figura 2.2.

NOTA

Una vez esté trabajando sobre el proyecto de la asignatura en VS Code, tenga presente que trabajará sobre WSL. Si ha tenido que realizar los pasos descritos en esta sección asegúrese siempre que tras conectarse a WSL utilizando VSCode, la versión que le indica en la esquina inferior izquierda muestra algo similar a "`WSL: Ubuntu-24.04`" 

, ya que es donde va a realizar la instalación de todas las dependencias del proyecto.

Apéndice B

WSL con Visual Studio Code

Las herramientas de compilación y depuración no las tenemos instaladas en Windows, sino en el subsistema Ubuntu que hemos instalado con WSL. Es por ello que tenemos que conectar VSCode con WSL para que ejecute todo desde allí. Se puede hacer de hasta tres formas distintas como nos dice la web de VSCode (Developing in WSL). La más sencilla es:

1. Abre el proyecto en VSCode si no lo tenías ya abierto.

2. Pulsa F1 (o Ctrl+Shift+P) para que se abra la *Command Palette*, y selecciona el comando **WSL: Reabrir carpeta en WSL** (**WSL: Reopen Folder in WSL** en inglés). Se te preguntará qué distribución desea utilizar; selecciona **Ubuntu-24.04**. Se te abrirá tu carpeta de Windows en el subsistema Ubuntu:

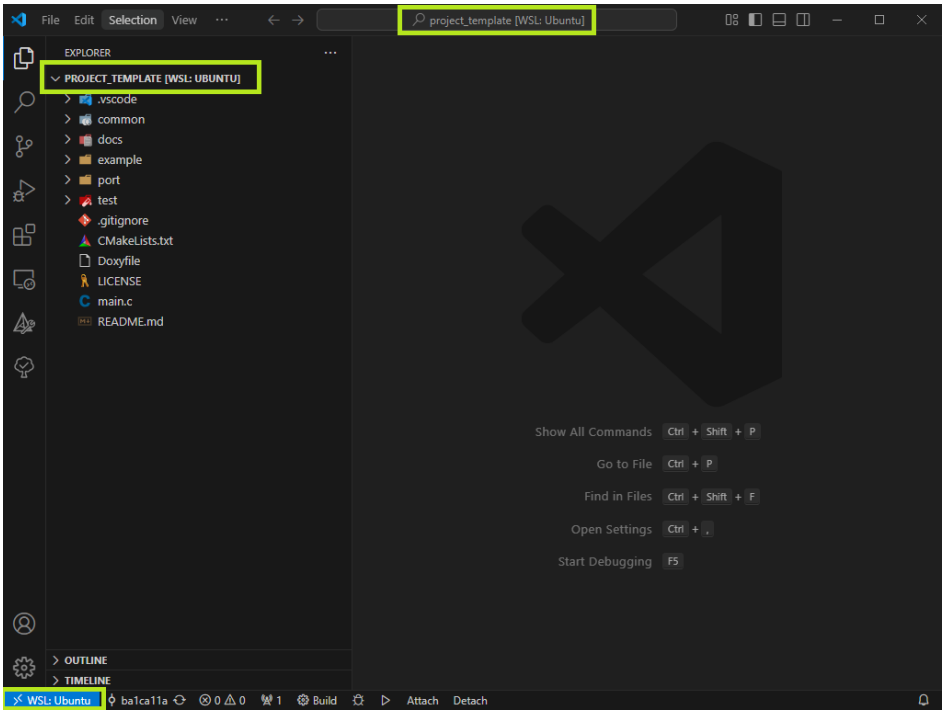


Figura B.1: Carpeta de proyecto abierta en WSL desde VSCode.

Apéndice C

Conexión de dispositivos USB con WSL en Windows

Vamos a proceder a describir los pasos necesarios para conectar el dispositivo USB con WSL y depurar correctamente nuestro código. Si se encuentra aquí muy probablemente se haya encontrado con el error que mostramos en la Figura C.1.

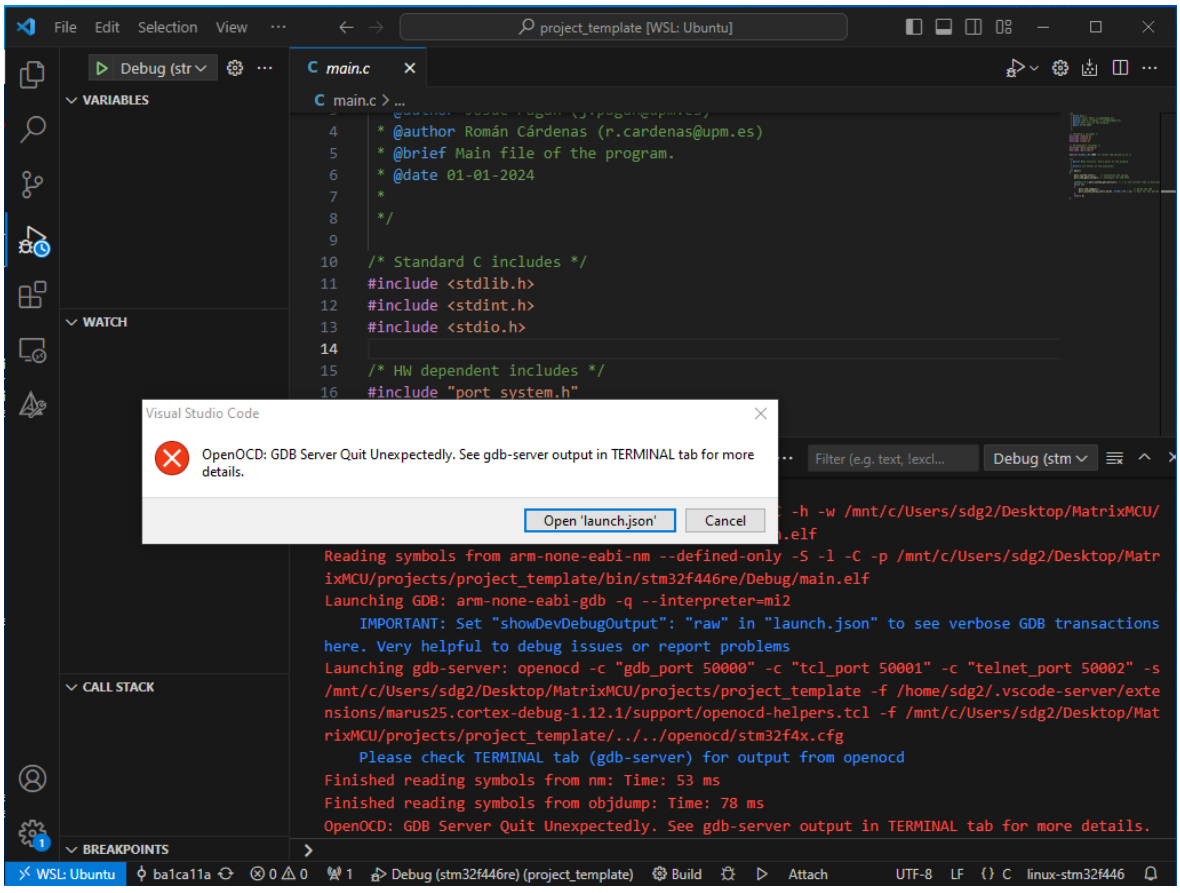


Figura C.1: Error relativo a OpenOCD al no haber haber hecho Attach del USB.

Para resolverlo, debe seguir los siguientes pasos:

1. Diríjase a la terminal de VSCode, y fíjese en que consiste en un error de conexión, seguramente sea similar a lo que se muestra en la Figura C.2. Este error también se puede producir como resultado de que OpenOCD se pase un tiempo esperando una respuesta de la placa y que esta no la esté generando adecuadamente. En cualquier caso, se refiere a un problema de conexión igualmente.

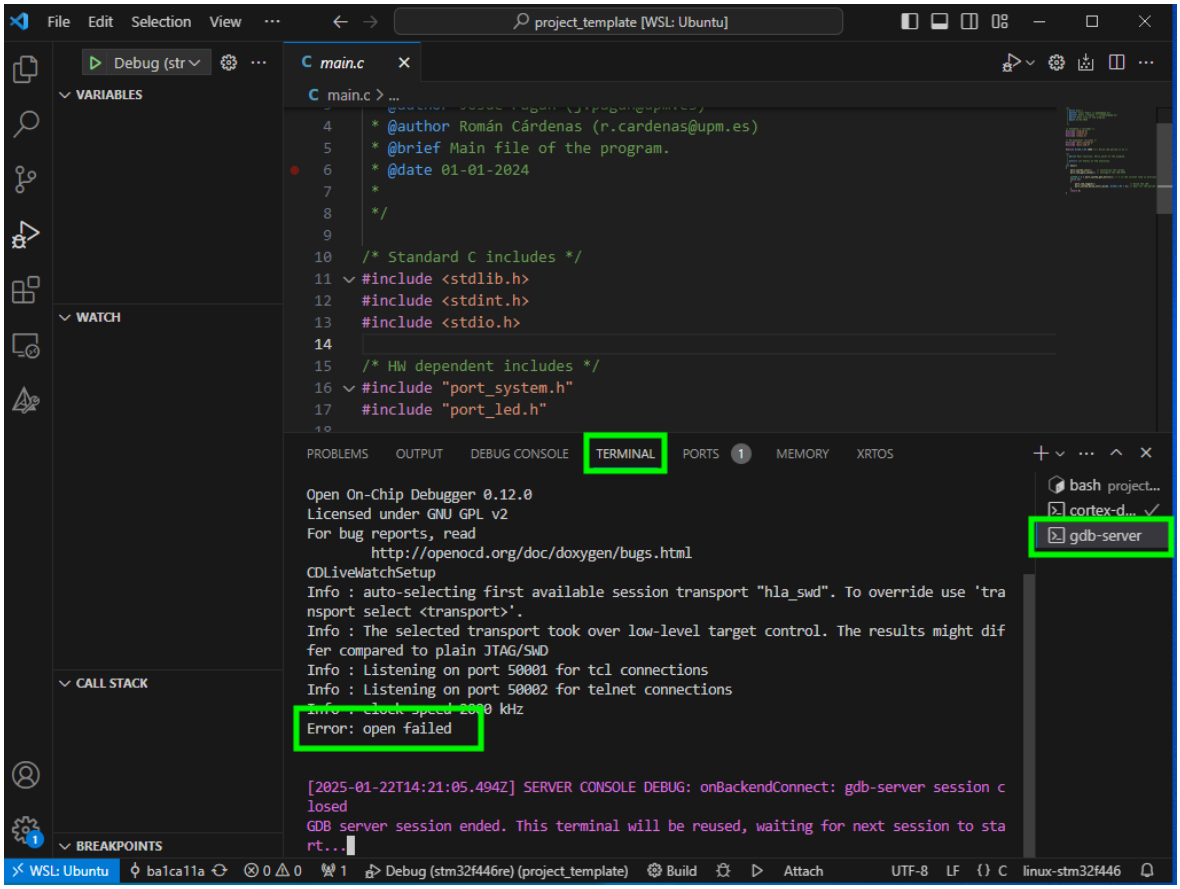


Figura C.2: Terminal de VSCode que muestra que el error es de conexión con la placa STM32.

2. Para solucionarlo, es necesario hacer el *Attach* de la placa con el PC. Para ello, utilizaremos la extensión de "**USBIP Connect**" en VSCode para intentar hacer el "Attach" y el "Detach" de la placa directamente desde el VSCode sin necesidad de utilizar `Windows Powershell`, como se muestra en la Figura C.3.

Si es la primera vez que lo hace, es probable que le genere otro mensaje de error y no pueda realizar la conexión. Esto se debe a que aún no se ha hecho el ``'bind'`` con la placa STM32. **Este procedimiento es necesario realizarlo cada vez que conectemos una placa distinta a nuestro PC**.

3. Para realizar el ``'bind'`` fíjese en que en la esquina inferior derecha debe haber aparecido un mensaje de error, como el que se muestra en la Figura C.4, cuando ha

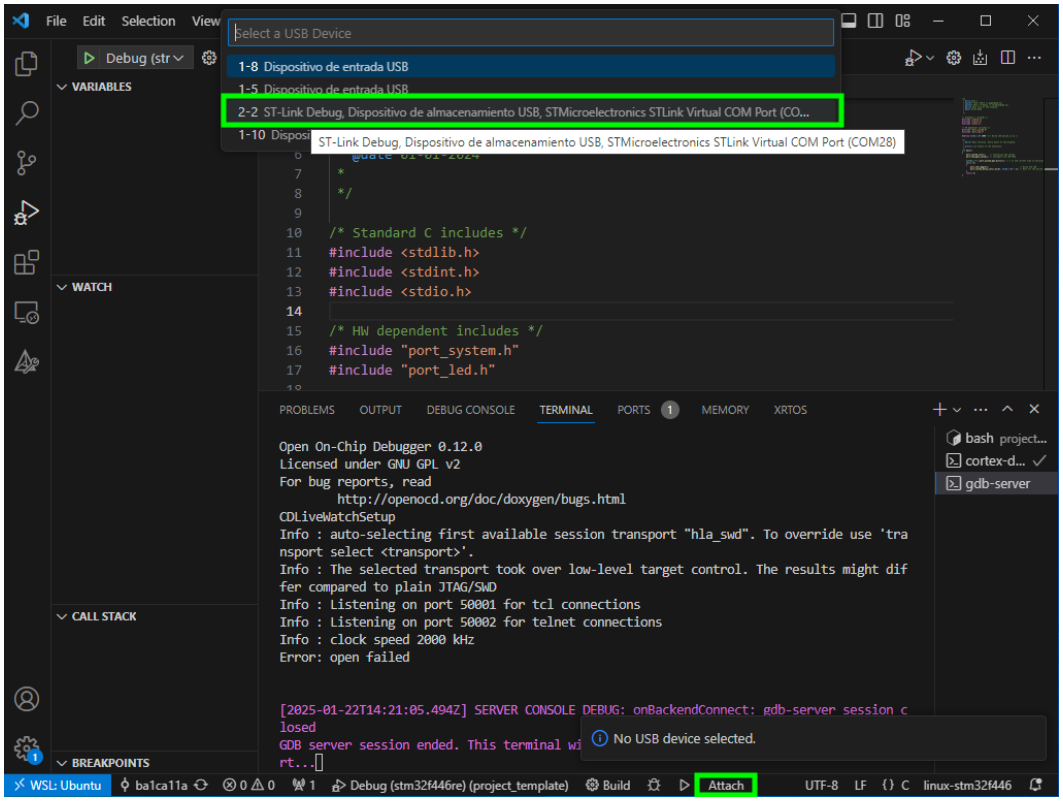


Figura C.3: Terminal de VSCode que muestra que el error es de conexión con la placa STM32.

intentado hacer el attach. En ese mensaje aparece el comando que será necesario ejecutar para llevar a cabo el ‘‘`bind`’’ de nuestra STM32 para que sea accesible por WSL.

4. Ahora, abrimos el buscador de Windows y buscamos `Windows Powershell`. Hacemos clic sobre él y ejecutamos el comando que nos indicó el mensaje de error de VSCode al que hacíamos referencia en el punto anterior. Un ejemplo de este procedimiento está ilustrado en la Figura C.4. Para comprobar si, efectivamente, el proceso se ha llevado a cabo correctamente, podemos ejecutar a continuación el siguiente comando. De estar hecho, correctamente este debería aprecer como ‘‘`Shared`’’.

```
usbipd list
```

5. Volvemos a VSCode e intentamos hacer de nuevo el “attach” de igual manera a la descrita anteriormente, en el segundo paso. En este caso, deberíamos de ser capaces de realizar el “attach” de nuestro dispositivo sin mayor problema, como se muestra en la Figura C.6, la cual refleja el mensaje que nos debería aparecer en la esquina inferior derecha.

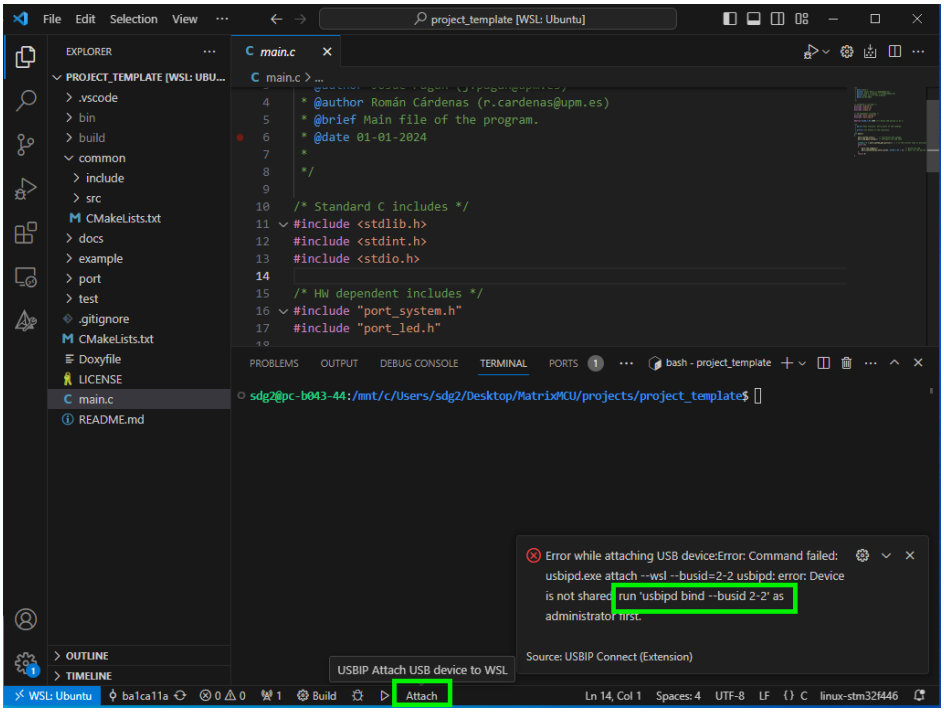


Figura C.4: Terminal de VSCode que muestra el error al hacer el "attach" por primera vez con la placa STM32 donde se ve el comando que necesitaremos para arreglar el problema.

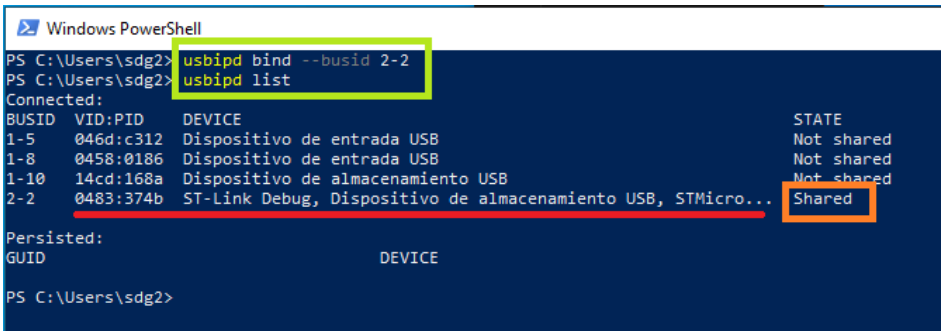


Figura C.5: Ejemplo de cómo realizar el "bind" de nuestra placa STM32 con nuestro PC.

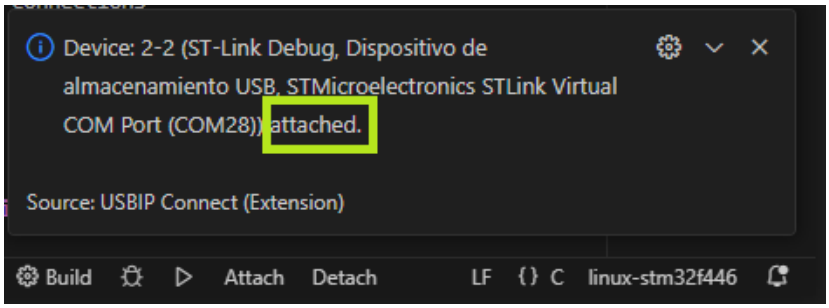


Figura C.6: Ejemplo de conexión realizada correctamente utilizando el "attach" de la placa desde VSCode.

6. Para asegurarnos del éxito del procedimiento llevado a cabo, intentaremos hacer de nuevo un "Clean and Debug" marancdo la opción **"Clean and Debug**

(stm32f446re)” y, en esta ocasión, debería de funcionar correctamente.