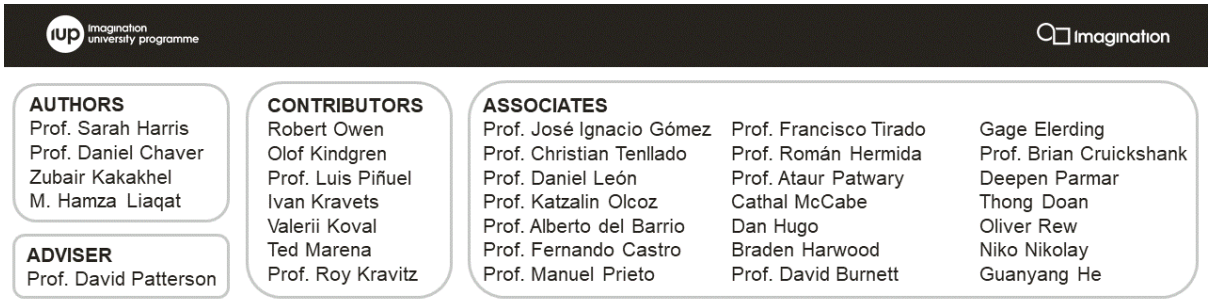
**PROGRAMA UNIVERSITARIO DE IMAGINATION**

# RVfpga
# Guía de inicio

# Agradecimientos




## Sponsors and Supporters


**AUTORES**

- Prof. Sarah Harris (https://www.linkedin.com/in/sarah-harris-12720697/)
- Prof. Daniel Chaver (https://www.linkedin.com/in/daniel-chaver-a5056a156/)
- Zubair Kakakhel (https://www.linkedin.com/in/zubairlk/)
- M. Hamza Liaqat (https://www.linkedin.com/in/muhammad-hamza-liaqat-ab73a0195/)

**ASESOR**

- Prof. David Patterson (https://www.linkedin.com/in/dave-patterson-408225/)

**COLABORADORES**

- Robert Owen (https://www.linkedin.com/in/robert-owen-4335931/)
- Olof Kindgren (https://www.linkedin.com/in/olofkindgren/)
- Prof. Luis Piñuel (https://www.linkedin.com/in/lpinuel/)
- Ivan Kravets (https://www.linkedin.com/in/ivankravets/)
- Valerii Koval (https://www.linkedin.com/in/valeros/)
- Ted Marena (https://www.linkedin.com/in/tedmarena/)
- Prof. Roy Kravitz (https://www.linkedin.com/in/roy-kravitz-4725963/)

**ADJUNTOS**

- Prof. José Ignacio Gómez (https://www.linkedin.com/in/jos%C3%A9-ignacio-gomez-182b981/)
- Prof. Christian Tenllado (https://www.linkedin.com/in/christian-tenllado-31578659/)
- Prof. Daniel León (www.linkedin.com/in/danileon-ufv)
- Prof. Katzalin Olcoz (https://www.linkedin.com/in/katzalin-olcoz-herrero-5724b0200/)
- Prof. Alberto del Barrio (https://www.linkedin.com/in/alberto-antonio-del-barrio-garc%C3%ADa-1a85586a/)
- Prof. Fernando Castro (https://www.linkedin.com/in/fernando-castro-5993103a/)
- Prof. Manuel Prieto (https://www.linkedin.com/in/manuel-prieto-matias-02470b8b/)
- Prof. Francisco Tirado (https://www.linkedin.com/in/francisco-tirado-fern%C3%A1ndez-40a45570/)
- Prof. Román Hermida (https://www.linkedin.com/in/roman-hermida-correa-a4175645/)
- Cathal McCabe (https://www.linkedin.com/in/cathalmccabe/)
- Dan Hugo (https://www.linkedin.com/in/danhugo/)
- Braden Harwood (https://www.linkedin.com/in/braden-harwood/ )
- David Burnett (https://www.linkedin.com/in/david-burnett-3b03778/)
- Gage Elerding (https://www.linkedin.com/in/gage-elerding-052b16106/)
- Brian Cruickshank (https://www.linkedin.com/in/bcruiksh/)
- Deepen Parmar (https://www.linkedin.com/in/deepen-parmar/)
- Thong Doan (https://www.linkedin.com/in/thong-doan/)
- Oliver Rew (https://www.linkedin.com/in/oliver-rew/)
- Niko Nikolay (https://www.linkedin.com/in/roy-kravitz-4725963/)
- Guanyang He (https://www.linkedin.com/in/guanyang-he-5775ba109/)
- Prof. Ataur Patwary (https://www.linkedin.com/in/ataurpatwary/)

# Índice



---

**Versiones de RVfpga:**

- Versión 1.0 (Publicada en noviembre de 2020):
  - Versión original de RVfpga.

- Versión 1.1 (Publicada en junio de 2021):
  - Añadida descripción de las prácticas 11-20 en el Lab 00.
  - SweRVolf actualizado a la versión 0.7.3.
  - Verilator actualizado a la versión 4.106.
  - Añadido un programa de inicialización en la Boot ROM.
  - Nueva figura y nueva tabla en la página 5 de la GSG para describir el Sistema RVfpga
  - Añadido un ejercicio de UART en el Lab 10.
  - Revisión de erratas.

---

**Autoría de la traducción:**

- Traducción al español llevada a cabo por Roberto Rodríguez Rodríguez (https://eie.ucr.ac.cr/profesores/roberto.rodriguez/).

- Revisión de la versión traducida realizada por Fernando Castro y Daniel Chaver.

---

# 1. INTRODUCCIÓN

RISC-V FPGA, también escrito RVfpga, es un material académico que incluye instrucciones, herramientas y prácticas de laboratorio para implementar un procesador comercial RISC-V en una FPGA (matriz de puertas lógicas programables en campo, por sus siglas en inglés) y en un simulador, y que a continuación permite usarlo y extenderlo para adquirir conocimientos sobre arquitectura de computadores, diseño digital, sistemas empotrados y programación.

Esta guía de inicio de RVfpga consta de las siguientes secciones, como se describe brevemente a continuación:

- **Guía de inicio rápido** (Sección 2)
- **Contexto y visión general**
  - **Arquitectura RISC-V** (Sección 3)
  - **El Sistema RVfpga** (Sección 4)
- **Uso del Sistema RVfpga en hardware**
  - **Instalación de las herramientas software** (Sección 5)
  - **Ejecución y Programación del Sistema RVfpga** (Sección 6)
- **Simulación del Sistema RVfpga**
  - **Uso de Verilator**, un simulador de HDL (Sección 7)
  - **Uso de Whisper**, simulador del repertorio de instrucciones de Western Digital (Sección 8)
- **Apéndices**
  - **Uso del conjunto de herramientas nativas de RISC-V y OpenOCD** (Apéndice A)
  - **Instalación de controladores en Windows para usar PlatformIO** (Apéndice B)
  - **Instalación de Verilator y GTKWave en Windows** (Apéndice C)
  - **Instalación de Verilator y GTKWave en macOS** (Apéndice D)
  - **Uso de Vivado para descargar el Sistema RVfpga en una FPGA** (Apéndice E)
  - **Ejemplo: Uso de RVfpga en una aplicación industrial de IoT** (Apéndice F)

La Guía de Inicio Rápido (Sección 2) describe la instalación mínima de software necesaria para utilizar RVfpga y luego muestra cómo descargar y ejecutar un programa simple de ejemplo en el Sistema RVfpga. Para una comprensión más profunda de RVfpga, omita la Sección 2 y vaya directamente a la guía completa que comienza en la Sección 3.

La Sección 3 proporciona una breve introducción a la arquitectura de computadores RISC-V. La Sección 4 describe el Sistema RVfpga (secciones 4.A – 4.C) y la organización de los archivos de Verilog que componen el sistema RVfpga (Sección 4.D). El Sistema RVfpga se basa en el SoC SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf) que a su vez utiliza el core de uso libre SweRV EH1 de Western Digital (WD's) (https://github.com/chipsalliance/Cores-SweRV). La Figura 1 y la Tabla 1, que se muestran a continuación, resumen la organización jerárquica del Sistema RVfpga, desde el core SweRV EH1 hasta RVfpgaNexys y RVfpgaSim.

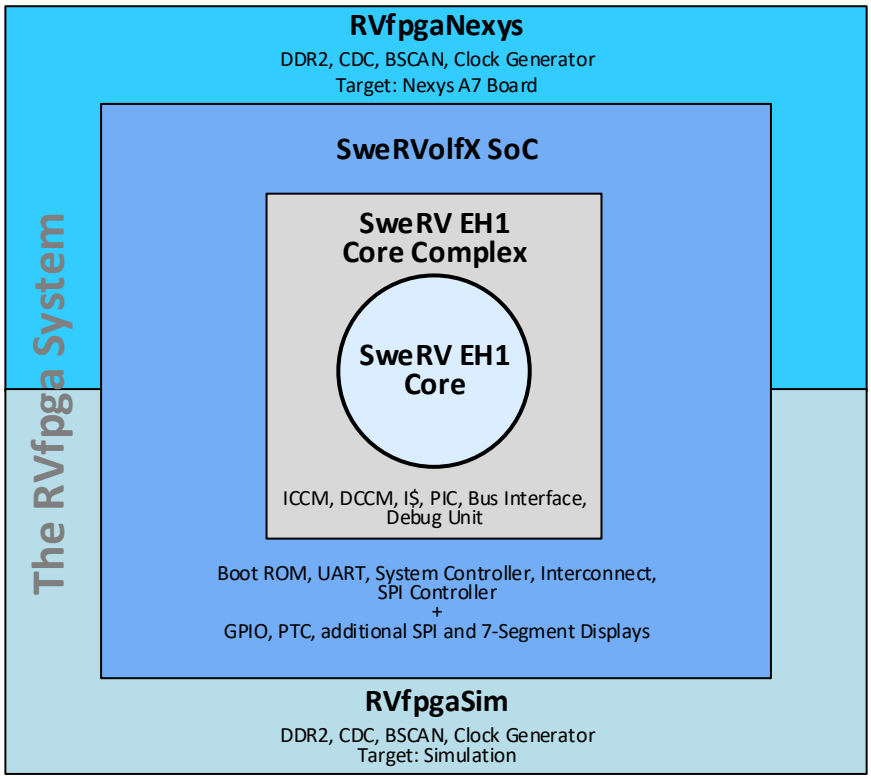



**Figura 1. El Sistema RVfpga**

**Tabla 1. Organización jerárquica del Sistema RVfpga**

| Nombre | Descripción |
|---|---|
| **SweRV EH1 Core** | Core de arquitectura RISC-V y de código abierto y uso libre desarrollado por Western Digital (https://github.com/chipsalliance/Cores-SweRV). |
| **SweRV EH1 Core Complex** | Core SweRV EH1 con varias memorias (ICCM, DCCM, and instruction cache), Controlador de Interrupciones Programable (PIC), interfaces de bus, y unidad de depuración (https://github.com/chipsalliance/Cores-SweRV). |
| **SweRVolfX** (Extended SweRVolf) | System on Chip utilizado en el curso RVfpga. Se trata de una extension de SweRVolf.<br>SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf): SoC de código abierto y uso libre basado en el SweRV EH1 Core Complex. Añade una Boot ROM, un interfaz UART, un Controlador de Sistema, módulos de interconexión (AXI Interconnect, Wishbone Interconnect, and AXI-to-Wishbone bridge), y un controlador SPI.<br>SweRVolfX: Añade 4 periféricos a SweRVolf: un GPIO, un PTC, otro SPI y un controlador para los 8 displays de 7 segmentos. |
| **RVfpgaNexys** | SweRVolfX adaptado para ejecutar en una placa Nexys A7. Añade un interfaz DDR2, una unidad CDC (clock domain crossing), lógica BSCAN (para el interfaz JTAG), y un generador de reloj.<br>RVfpgaNexys es igual que SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), salvo porque SweRVolf Nexys está basado en SweRVolf. |
| **RVfpgaSim** | SweRVolfX con un wrapper testbench y una memoria AXI para uso en simulador. |

| | |
|---|---|
| | RVfpgaSim es igual que SweRVolf sim, (https://github.com/chipsalliance/Cores-SweRVolf), salvo porque SweRVolf sim está basado en SweRVolf. |

El resto de las secciones muestran cómo usar el Sistema RVfpga tanto en hardware (RVfpgaNexys) como en simulación (RVfpgaSim). La sección 5 muestra cómo instalar las herramientas software necesarias para usar RVfpga. La sección 6 muestra cómo usar PlatformIO tanto para descargar RVfpgaNexys en la placa FPGA Nexys A7 (Sección 6.A) como para descargar y ejecutar varios programas de ejemplo en RVfpgaNexys (Secciones 6.B-6.H). Las secciones 7 y 8 muestran cómo simular RVfpgaSim usando Verilator (Sección 7), un simulador HDL de código abierto, y Whisper (Sección 8), el simulador del repertorio de instrucciones RISC-V de Western Digital (ISS).

Finalmente, los apéndices muestran cómo usar RVfpga en la línea de comandos de GNU/Linux (Apéndice A), cómo instalar los controladores y el software necesarios en máquinas Windows y macOS (Apéndices B-D), y cómo usar Vivado para descargar RVfpgaNexys en una FPGA (Apéndice E). Para finalizar, el Apéndice F muestra cómo usar RVfpga en una aplicación industrial IoT (Apéndice F).

Tabla 2 lista el software y el hardware necesarios para el uso de RVfpga. Esta guía muestra cómo instalar y utilizar estas herramientas y hardware en el sistema operativo (SO) Ubuntu 18.04. Otros sistemas operativos (como Windows o MacOS), siguen pasos similares (si no exactamente los mismos). Cuando las instrucciones difieren, se insertan instrucciones específicas para **Windows** y **macOS** usando este resaltado.

**Nota:** si no tiene acceso a la placa FPGA Nexys A7, las prácticas de laboratorio pueden ser completadas de forma simulada usando Whisper, el simulador del repertorio de instrucciones (ISS) de Western Digital, y Verilator, un simulador HDL de código abierto. En este caso, no es necesario instalar Vivado (Sección 5.A); sólo es necesario instalar VSCode/PlatformIO (como se explica en la Sección 2.A) y Verilator/GTKWave (como se explica en la Sección 5.C).

**Tabla 2. Software y hardware necesario para el uso de RVfpga**

| **Software** | | |
|---|---|---|
| **Nombre** | **Página web** | **Precio** |
| Vivado 2019.2 WebPACK | https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html | gratuito |
| VSCode | https://code.visualstudio.com/Download | gratuito |
| PlatformIO | https://platformio.org/<br>Instalado dentro del VSCode | gratuito |
| Verilator (un simulador de HDL) y GTKWave | https://github.com/verilator/verilator<br>http://gtkwave.sourceforge.net/ | gratuito |
| Whisper (Simulador del Repertorio de Instrucciones RISC-V de Western Digital) | https://github.com/chipsalliance/SweRV-ISS<br>Instalado dentro de PlatformIO | gratuito |
| RISC-V Toolchain y OpenOCD | https://github.com/riscv/riscv-gnu-toolchain<br>https://github.com/riscv/riscv-openocd<br>Instalado dentro de PlatformIO | gratuito |
| **Hardware** | | |
| **Nombre** | **Página web** | **Precio** |

| | | |
|---|---|---|
| Placa FPGA Nexys A7* | https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/ | $265 (precio académico: 199 dólares) |
| **Core y SoC RISC-V**** | | |
| **Nombre** | **Página web** | **Precio** |
| Western Digital SweRV EH1 Core | https://github.com/chipsalliance/Cores-SweRV | gratuito |
| SweRVolf | https://github.com/chipsalliance/Cores-SweRVolf | gratuito |

* Todos los pasos descritos en esta guía también funcionan en la placa FPGA Nexys4 DDR de Digilent.
** Proporcionado con la descarga de RVfpga de Imagination Technologies

**IMPORTANTE:** Antes de empezar, copie la carpeta **RVfpga** que descargó del Programa Universitario de Imagination a su máquina Ubuntu/Windows/macOS. Se hará referencia a la ruta absoluta del directorio donde se coloque la carpeta RVfpga como [RVfpgaPath]. RVfpga contiene cinco carpetas: (1) **examples**: programas de ejemplo que se ejecutarán durante el uso de esta guía, (2) **src**: contiene el código fuente (Verilog y SystemVerilog) del Sistema RVfpga, (3) **verilatorSIM**: contiene los *scripts* para ejecutar la simulación de RVfpgaSim en Verilator, (4) **driversLinux_NexysA7**: contiene los drivers de GNU/Linux para la placa FPGA Nexys A7, y (5) **Labs**: contiene los programas que se utilizarán durante las prácticas de laboratorio 1-10 de RVfpga.

## Conocimiento previo esperado:

Antes de completar este curso sobre RVfpga, que incluye esta Guía de iniciación a RVfpga y las prácticas de laboratorios de RVfpga, se espera que los usuarios posean al menos un conocimientos básicos de los siguientes temas:

- Diseño lógico digital
- Programación de alto nivel (preferiblemente C)
- Programación en lenguaje ensamblador
- Arquitectura del repertorio de instrucciones
- Microarquitectura del procesador
- Sistemas de memoria

Estos temas se abordan en el libro de texto *Digital Design and Computer Architecture: RISC-V Edition*, Harris & Harris, © Morgan Kaufmann, cuya publicación está prevista para el verano de 2021. Otros libros de texto, incluyendo *Computer Organization and Design RISC-V Edition*, Patterson & Hennessy, © Morgan Kaufmann 2017, cubren algunos de estos temas.

## 2. GUÍA DE INICIO RÁPIDO

Esta sección muestra cómo instalar el conjunto de herramientas mínimo necesario para usar Rvfpga, y a continuación explica cómo usar PlatformIO tanto para descargar RVfpgaNexys en la placa FPGA Nexys A7 como para ejecutar un programa en RVfpgaNexys. Tendrá que adquirir la placa FPGA (ver Tabla 2). Estos pasos también son válidos para la placa FPGA Nexys4-DDR, una versión anterior de la Nexys A7.

**A. Instalación mínima: VSCode, PlatformIO y los controladores de la placa Nexys A7**
**B. Descargar RVfpgaNexys en la FPGA y ejecutar programas en ella**

Las instrucciones mostradas a continuación son para un sistema Ubuntu 18.04. También son válidas para los sistemas operativos Windows 10 y macOS. Cuando las instrucciones difieren de las de Ubuntu, se insertan cuadros con instrucciones específicas para **Windows** y **macOS**. Si está usando Ubuntu, puede ignorar esos cuadros. Las rutas se escriben como rutas de GNU/Linux usando barras inclinadas hacia adelante (/), pero las rutas de Windows son típicamente las mismas pero con barras inclinadas hacia atrás (\).

### A. Instalación mínima: VSCode, PlatformIO y controladores de la placa Nexys A7

En esta etapa, se instalará el software y los controladores mínimos necesarios para usar RVfpga. Primero, se instalará el entorno de programación, y luego se instalarán los controladores para la placa FPGA Nexys A7.

**Instalación de VSCode y PlatformIO:** Se Utilizará PlatformIO, un entorno de desarrollo integrado (IDE por sus siglas en inglés) para descargar el sistema RVfpgaNexys en la placa Nexys A7 y también para descargar y ejecutar programas en RVfpgaNexys. PlatformIO está construido como una extensión del *Visual Studio Code* de Microsoft (VSCode). PlatformIO es multiplataforma e incluye un depurador integrado.

Debe seguir estos pasos para instalar tanto VSCode como PlatformIO:

1. Instalar VSCode:
   a. Descargar el archivo de instalación desde el siguiente enlace: https://code.visualstudio.com/Download

   b. Abrir una terminal, e instalar y ejecutar el VSCode:
```
cd ~/Downloads
sudo dpkg -i code*.deb
code
```

**Windows / macOS:** Los paquetes de VSCode también están disponibles para Windows (archivo .exe) y macOS (archivo .zip) en https://code.visualstudio.com/Download. Siga los pasos habituales para instalar y ejecutar una aplicación en estos sistemas operativos.

2. Instalar PlatformIO sobre VSCode:
   a. Instale las librerías de python3 escribiendo lo siguiente en una terminal:
```
sudo apt install -y python3-distutils python3-venv
```

**Windows / macOS:** este paso (2.a) no es necesario en Windows. En cuanto a macOS, si no está instalado, puede usar *homebrew* para instalar python3: `brew install python3`

b. Si aún no está abierto, inicie VSCode seleccionando el botón de inicio y escribiendo "VSCode" en el menú de búsqueda, luego seleccione VSCode, o escriba `code` en una terminal de Ubuntu.

c. En VSCode, haga clic en el icono de Extensiones situado en la barra lateral izquierda del programa (ver Figura 2).

**Figura 2. Icono de Extensiones de VSCode**

d. Escriba *PlatformIO* en el cuadro de búsqueda e instale el *IDE* PlatformIO haciendo clic en el botón de instalación que está a su lado (ver Figura 3).

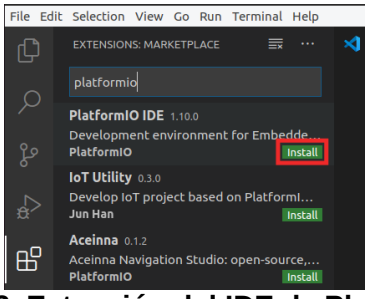



**Figura 3. Extensión del IDE de PlatformIO**

e. La ventana de SALIDA en la parte inferior le informará sobre el proceso de instalación. Una vez que haya terminado, haga clic en "Recargar ahora" ("*Reload Now*") en la ventana inferior derecha, y PlatformIO terminará de instalarse dentro de VSCode (ver Figura 4).

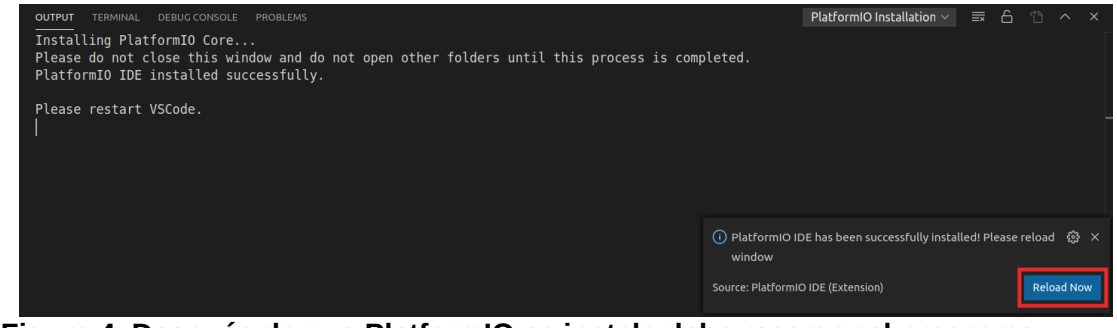



**Figura 4. Después de que PlatformIO se instale debe recargar el programa**

**Instalación de los controladores de la placa Nexys A7:** es necesario instalar manualmente los controladores de la placa Nexys A7.

- Abrir una terminal.
- Entrar en el directorio *[RVfpgaPath]/RVfpga/driversLinux_NexysA7*. (Para simplificar, se proporcionarán estos controladores dentro de la carpeta RVfpga. Cuando instale Vivado en la sección 5 de esta guía, también podrá encontrar estos controladores dentro del paquete descargado como se describe en esa sección).
- Ejecute el programa de instalación:

```
chmod 777 *
sudo ./install_drivers
```

- Desconecte la placa Nexys A7 de su computadora y reinicie ésta para que los cambios tengan efecto.

**Windows:** siga las instrucciones del Apéndice B para instalar los controladores de la placa Nexys A7.

**macOS:** no es necesario instalar ningún controlador adicional.

## B. Descargar RVfpgaNexys en la FPGA y ejecutar programas en ella

Ahora se descargará RVfpgaNexys, el sistema RISC-V creado para ser utilizado en una FPGA, enla placa FPGA Nexys A7. Aunque no se modificará en esta Guía de inicio, el código Verilog para el Sistema RVfpga está disponible en *[RVfpgaPath]/RVfpga/src*. Se describirá el código fuente del sistema RVfpga en la Sección 4 de esta GSG y con más detalle en las prácticas de laboratorio RVfpga 6-10. También se modificará el sistema RVfpga en algunos de los ejercicios de esas prácticas de laboratorio.

Para descargar RVfpgaNexys en la placa FPGA Nexys A7 se deben completar los siguientes pasos:

**Paso 1.** Conectar la placa FPGA Nexys A7 a la computadora y encenderla
**Paso 2.** Abrir PlatformIO y el programa en C
**Paso 3.** Descargar RVfpgaNexys a la placa Nexys A7
**Paso 4.** Descargar y ejecutar el programa en RVfpgaNexys

### Paso 1. Conectar la placa FPGA Nexys A7 a la computadora y encenderla

Figura 5 muestra las ubicaciones físicas de los LED y los interruptores (Switches) de la placa FPGA Nexys A7, así como el conector USB (*USB Connector*), el interruptor de encendido (*on switch*), los pulsadores (Pushbuttons) y los displays de 7 segmentos (7-Segment Displays). Conecte la placa Nexys A7 a su computadora con el cable USB suministrado y enciéndala.

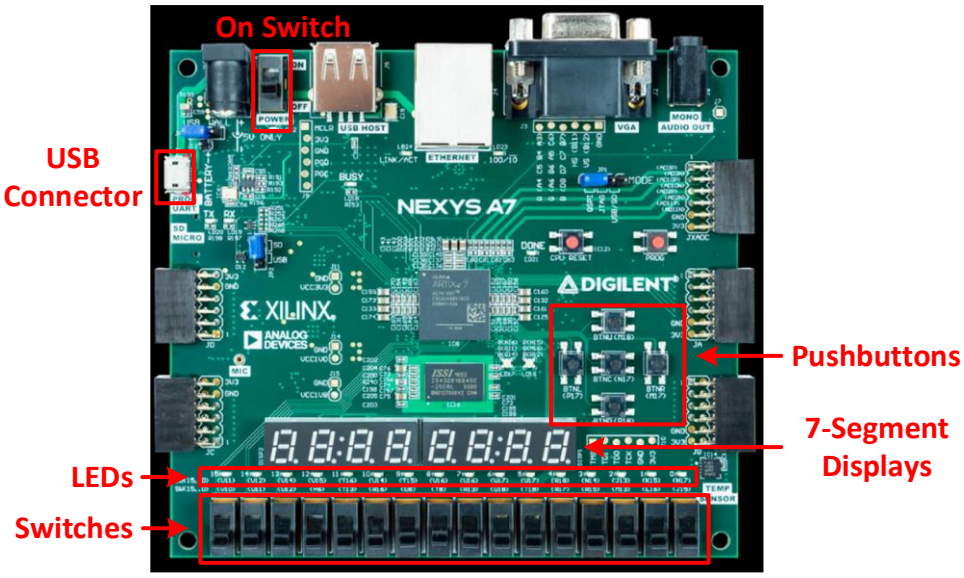



**Figura 5. Interfaces de E/S de la placa FPGA Nexys A7 de Digilent**
(figura tomada de https://reference.digilentinc.com/)

**Paso 2. Abrir PlatformIO y el programa en C**
Ahora abra VSCode escribiendo VSCode en el menú de inicio (ver Figura 6) o escribiendo `code` en una terminal.

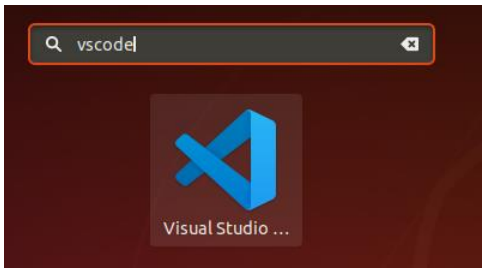



**Figura 6. Abrir VSCode**

Si la ventana de PlatformIO Home (PIO Home) no se abre automáticamente, haga clic en el icono de PlatformIO en el menú de la barra de la izquierda: . Luego expanda PIO Home y haga clic en Open. Ahora PIO Home se abrirá en la ventana de Bienvenida (ver Figura 7).

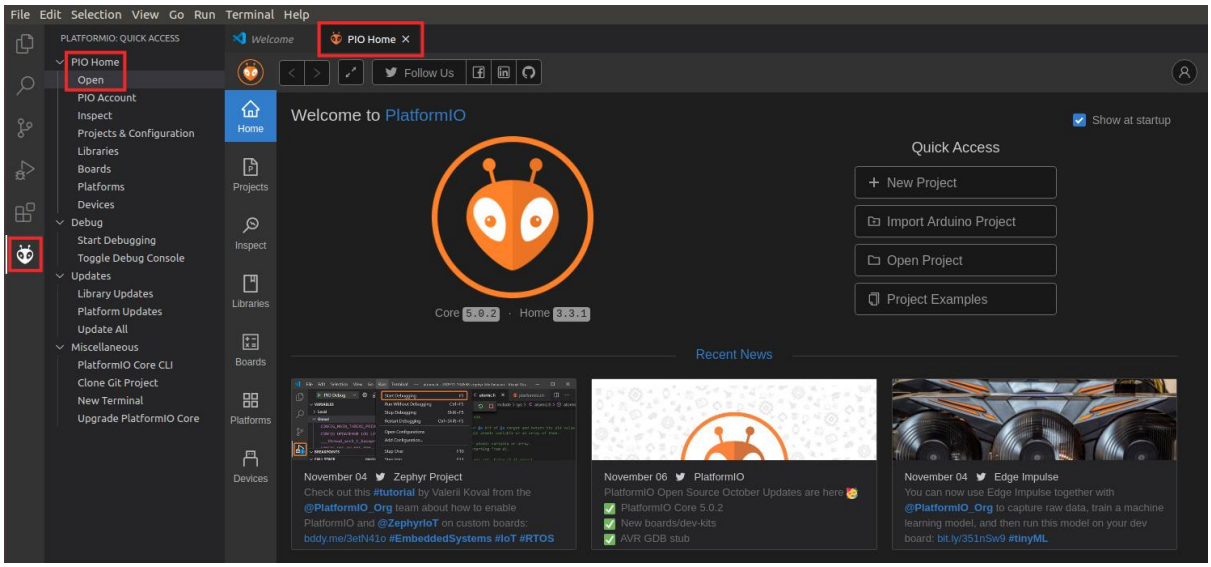



**Figura 7. Abrir PIO Home**

Ahora haga clic en File → Open Folder del menú superior de archivos y seleccione:
*[RVfpgaPath]\RVfpga\examples\LedsSwitches_C-Lang*

Seleccione la carpeta, pero no la abra (véase la Figura 8). PlatformIO abrirá el programa LedsSwitches_C-Lang, que lee los valores de los interruptores en la placa Nexys A7 y escribe sus valores en los LEDs de la misma.

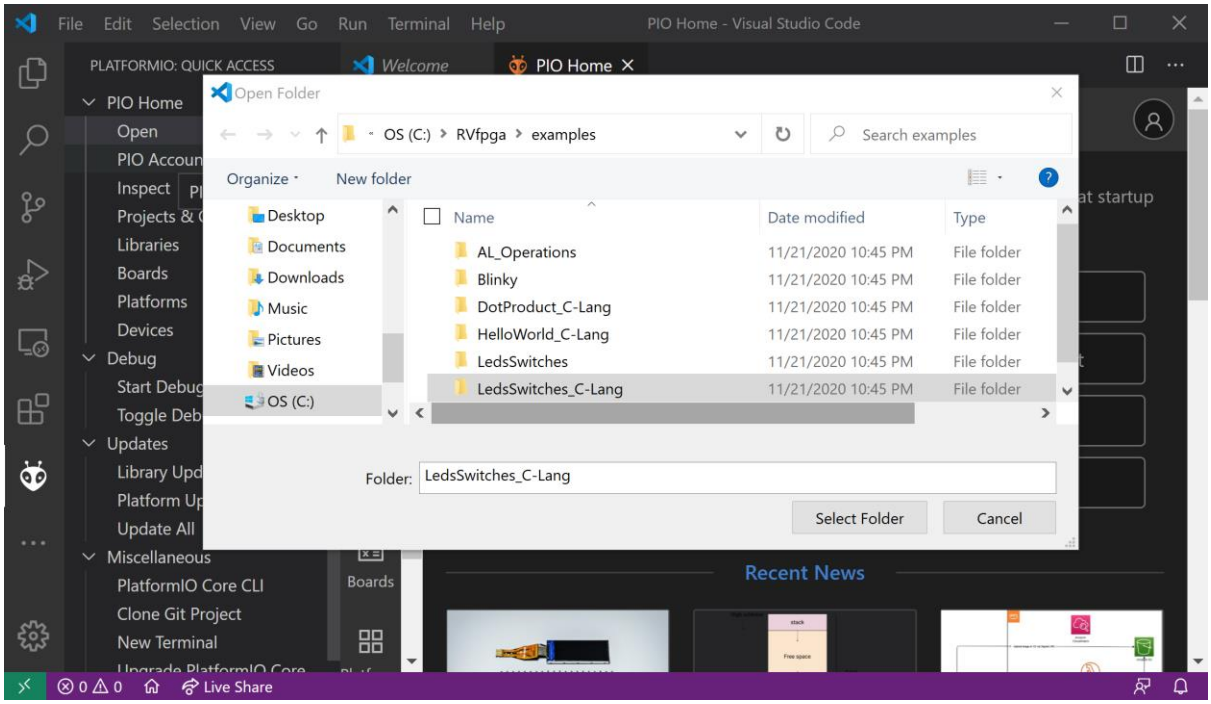



**Figura 8. Apertura del programa de ejemplo LedsSwitches_C-Lang**

Puede ver el programa LedsSwitches_C-Lang expandiendo la carpeta *src* y haciendo doble clic en LedsSwitches_C-Lang.c (Figura 9). Se discutirá este programa en detalle más adelante en esta Guía de inicio. Para esta Guía de Inicio Rápido, simplemente se descargará este programa en RVfpgaNexys que estará ejecutándose en la placa Nexys A7.

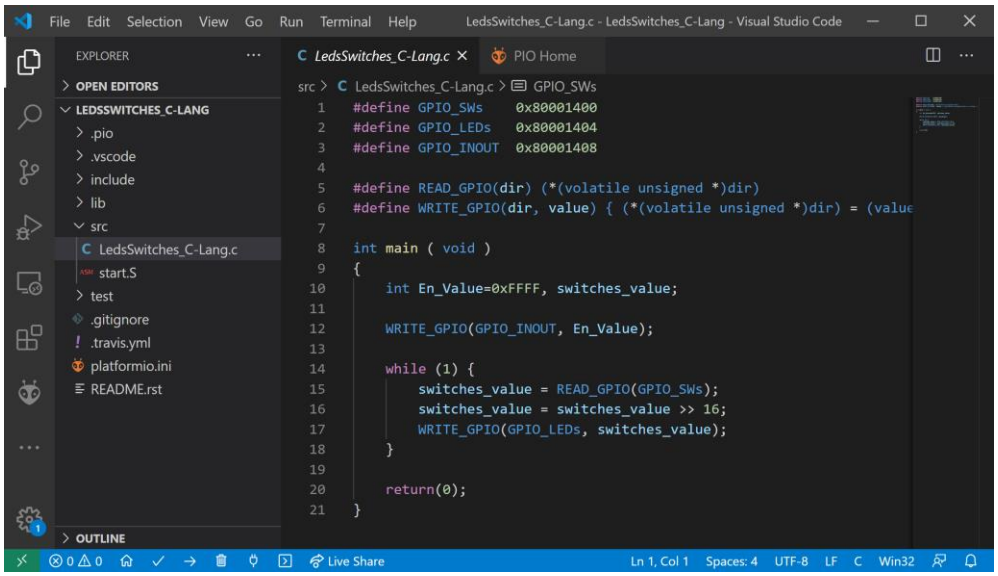



**Figura 9. Programa LedsSwitches_C-Lang.c**

Nótese que la primera vez que se abre un ejemplo de RVfpga en PlatformIO, la plataforma de Chips Alliance se instala automáticamente (se puede ver en PIO Home → Platforms, como se muestra en la Figura 10). Esta plataforma incluye varias herramientas que se usarán más adelante, como el conjunto de herramientas preconstruido de RISC-V, OpenOCD para RISC-V, un archivo *bitfile* con la implementación de RVfpgaNexys, RVfpgaSim, *scripts* de JavaScript y Python, y varios ejemplos. Si por alguna razón la plataforma Chips Alliance no se instalara automáticamente, podría instalarse manualmente, como se explicará en la Sección 6.A.

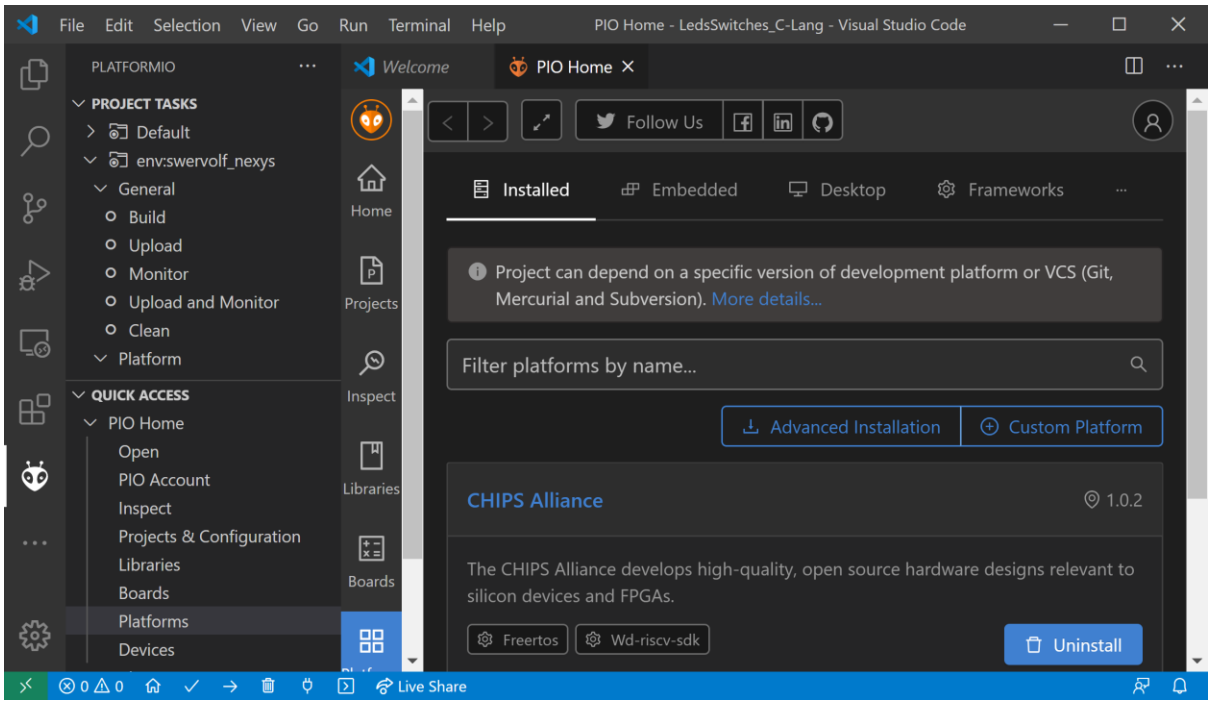



**Figura 10. La plataforma de Chips Alliance instalada en PlatformIO**

## Paso 3. Descargar RVfpga a la placa Nexys A7

Ya está todo listo para descargar RVfpgaNexys, el SoC RISC-V que incluye un procesador RISC-V con soporte para periféricos. Abra el archivo platformio.ini (archivo de inicialización de PlatformIO) haciendo doble clic en él en la ventana EXPLORER, como se muestra en la Figura 11. (Si la ventana del EXPLORER no está ya abierta, ábrala haciendo clic en en el menú de la barra izquierda). Ahora, agregue la ruta para la ubicación del *bitfile* que define RVfpgaNexys reemplazando la ruta board_build.bitstream_file por su propia ruta (ver Figura 11):

*board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpganexys.bit*

Guarde el archivo platformio.ini pulsando Ctrl-s.

Existen muchos comandos para el Archivo de Configuración del Proyecto (*platformio.ini*); se puede obtener más información sobre estas opciones en:
https://docs.platformio.org/en/latest/projectconf

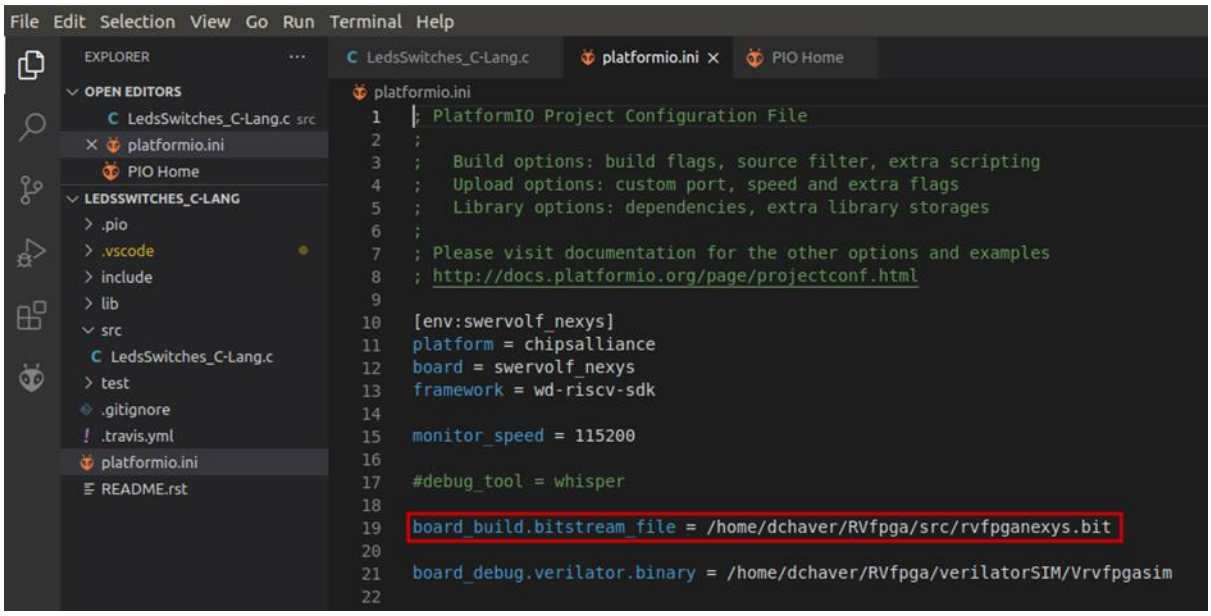



**Figura 11. Añadir la ruta al archivo bitfile de RVfpga**

Descargue RVfpgaNexys (como se define en el *bitfile* provisto) en la placa Nexys A7:

- Haga clic en el icono de PlatformIO en el menú de la barra izquierda (ver Figura 12).

**Figura 12. Icono de PlatformIO**

- En el caso de que la ventana de Tareas del Proyecto (*Project Tasks*) esté vacía (Figura 13), debe refrescar ésta haciendo clic en  . Esto puede tomar varios minutos.

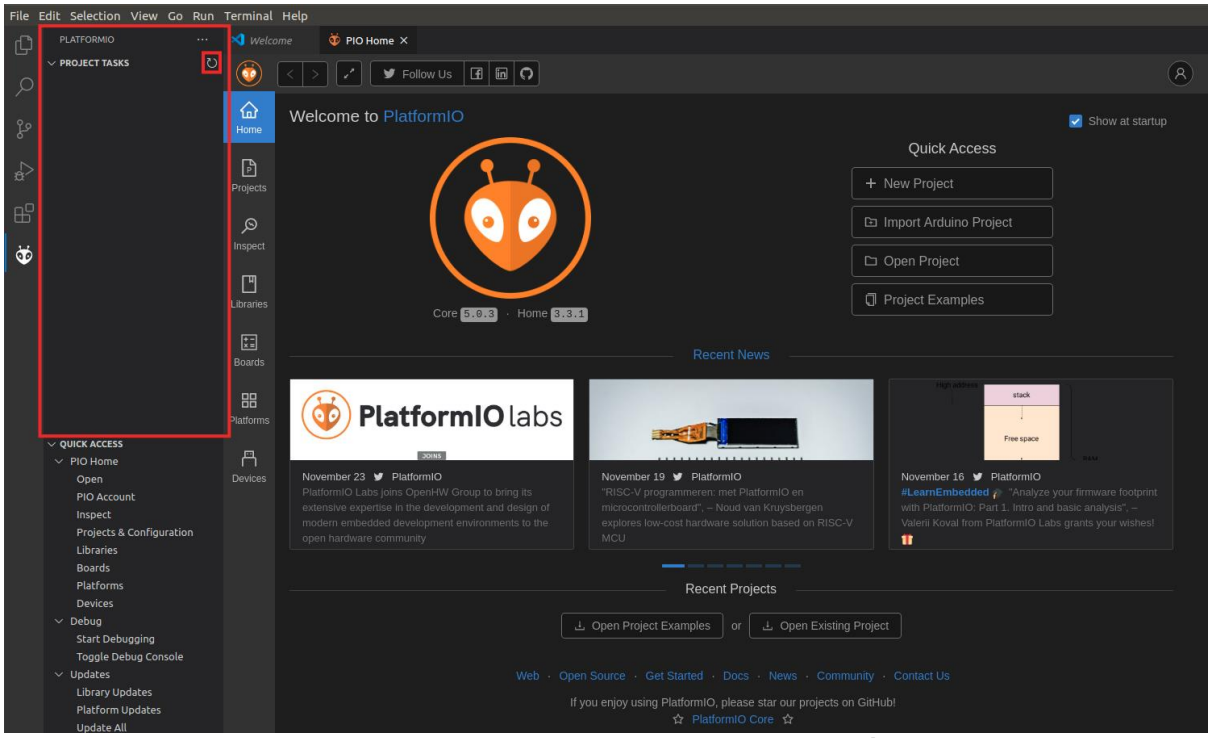



**Figura 13. Ventana de TAREAS DE PROYECTO vacía - Actualizar**

- A continuación, expanda *Project Tasks* → env:swervolf_nexys → Plataform y haga clic en Upload Bitstream, como se muestra en la Figura 14. **Después de uno o dos segundos, la FPGA será programada con RVfpgaNexys**.

  Por defecto, el procesador empieza a ejecutar a partir de la dirección 0x80000000, en la que la Boot ROM está mapeada en nuestro SoC (ver Tabla 6). La Boot ROM está inicializada con un programa (*boot_main.mem*) que enciende y apaga los LEDs y los Displays de 7 Segmentos 4 veces y luego apaga los LEDs, escribe un 0 en los

Displays de 7 Segmentos y se queda iterando en un bucle vacío. Este programa se encuentra disponible en el directorio: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*. Si deseas cambiarlo y recompilarlo, puedes hacerlo como se explica en el Apéndice A – Sección III (el archivo *boot_main.mem* es una copia del archivo *boot_main.vh*). En la Práctica 1 mostraremos cómo inicializar la Boot ROM con este programa al crear el *bitstream*.

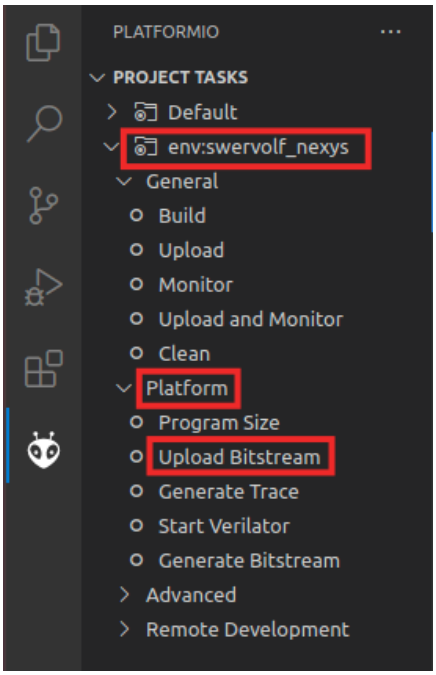
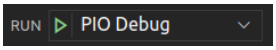



**Figura 14. Carga del archivo bitstream**

**Paso 4. Descargar y ejecutar el programa en RVfpgaNexys**

Ahora que RVfpgaNexys está descargado y funcionando en la placa Nexys A7, se procederá a descargar el programa en la memoria del RVfpgaNexys y se ejecutará y depurará el programa. Haga clic en el botón Ejecutar y Depurar "*Run and Debug*":  , que está disponible en la barra de la izquierda. Inicie el depurador haciendo clic en el botón de reproducción 

(la opción "PIO Debug" debe estar seleccionada). Este botón se encuentra cerca de la parte superior de la ventana (véase la Figura 15). El programa primero compilará y luego comenzará la depuración.

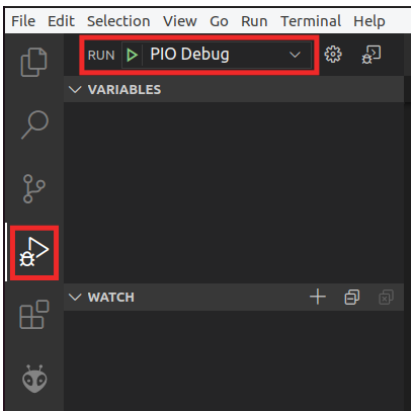



**Figura 15. Compilar y descargar el programa e iniciar el depurador**

Para controlar la sesión de depuración, puede utilizar la barra de herramientas de depuración que aparece cerca de la parte superior del editor (véase la Figura 16). Se describirán y probarán todas las opciones más adelante en esta Guía de inicio.

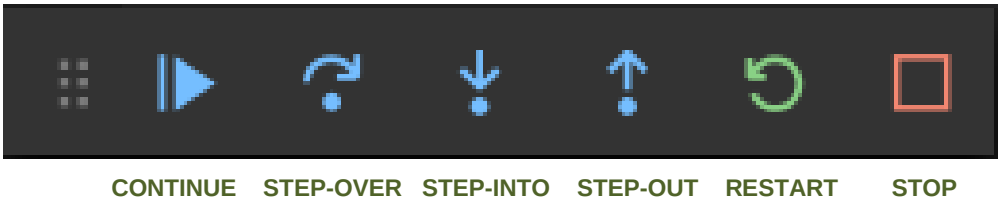



**Figura 16. Herramientas de depuración**

PlatformIO establece un punto de ruptura (*breakpoint*) temporal al comienzo de la función main. Por lo tanto, haga clic en el botón Continue  para ejecutar el programa. Ahora conmute los interruptores de la placa FPGA Nexys A7 y vea cómo se encienden los LEDs correspondientes.

## 3. LA ARQUITECTURA RISC-V

RISC-V es una arquitectura del repertorio de instrucciones (ISA por sus siglas en inglés) creada en 2011 en el laboratorio Par de la Universidad de California en Berkeley. El objetivo era que RISC-V se convirtiera en una "ISA Universal" para los procesadores utilizados en todo el espectro de aplicaciones, desde pequeños dispositivos IoT con grandes restricciones y pocos recursos hasta supercomputadores. Los diseñadores de RISC-V establecieron cinco principios para que la arquitectura alcanzara este objetivo:

- Debe ser compatible con una amplia gama de paquetes de software y lenguajes de programación.
- Su implementación debe ser factible en todas las opciones tecnológicas, desde las FPGAs hasta los ASIC (circuitos integrados de aplicación específica), así como en las tecnologías emergentes.
- Debe ser eficiente en los diversos escenariosmicroarquitectónicos, incluidos los que implementan control por microcódigo o bien por cableado, pipelines en orden o fuera de orden, diversos tipos de paralelismo, etc.
- Debe poder adaptarse a tareas específicas para lograr el máximo rendimiento requerido sin los inconvenientes impuestos por la propia ISA.
- Su repertorio de instrucciones básico debe ser estable y duradero, ofreciendo un marco común y sólido para los desarrolladores.

RISC-V es un estándar abierto, la especificación es de dominio público, y ha sido administrado desde 2015 por la **Fundación RISC-V**, ahora llamada **RISC-V Internacional**, una organización sin fines de lucro que promueve el desarrollo de hardware y software para la arquitectura RISC-V. En 2018, la Fundación RISC-V comenzó una colaboración continua con la Fundación Linux, y en marzo de 2020 la Fundación RISC-V se convirtió en la RISC-V Internacional con sede en Suiza. Esta transición disipó cualquier preocupación que la comunidad pudiera haber tenido sobre la continuidad del carácter abierto del estándar en el futuro. Desde 2020, la RISC-V Internacional cuenta con el apoyo de más de 200 actores clave de la investigación, el mundo académico y la industria, entre los que se encuentran Microchip, NXP, Samsung, Qualcomm, Micron, Google, Alibaba, Hitachi, Nvidia, Huawei, Western Digital, ETH Zurich, KU Leuven, UNLV y UCM.

RISC-V es una de las pocas y probablemente la única ISA de relevancia mundial, creada en los últimos 10 a 20 años, por ser un estándar abierto y modular, en lugar de incremental. Su modularidad la hace a la vez flexible y elegante. Los procesadores implementan la ISA base y sólo aquellas extensiones que utilizan. Este enfoque modular difiere de las ISA tradicionales, como x86 o ARM, que tienen arquitecturas incrementales, donde las ISA anteriores se expanden y cada nuevo procesador debe implementar todas las instrucciones, incluso las que están etiquetadas como "obsoletas", para garantizar la compatibilidad con los programas de software más antiguos. Por ejemplo, x86, que comenzó con 80 instrucciones, tiene ahora más de 1300, o 3600 si se consideran todos los diferentes códigos de operación disponibles en código máquina. Este gran número de instrucciones y el requisito de la compatibilidad hacia atrás dan como resultado procesadores grandes y con un gran consumo de energía que deben soportar instrucciones largas, porque la mayoría de los códigos de operación cortos, o instrucciones pequeñas, ya están en uso.

RISC-V tiene cuatro opciones de la ISA base: dos versiones de 32 bits (una versión de enteros y otra de empotrados, RV32I y RV32E) y las versiones de 64 y 128 bits (RV64I y RV128I), como se muestra en la Tabla 3. Los módulos de la ISA marcados como *Ratified* han sido ratificados actualmente. No se espera que los módulos marcados como *Frozen* cambien significativamente antes de ser puestos a prueba para su ratificación. Se espera que los módulos marcados como *Draft* sean modificados antes de su ratificación. La

capacidad de construir pequeños procesadores es un requisito particularmente importante para los dispositivos con restricciones de costo, área y energía. Se pueden añadir extensiones de instrucciones sobre estas ISA base para permitir tareas específicas, como por ejemplo operaciones en punto flotante, multiplicación y división, y operaciones vectoriales. Estas extensiones de hardware especializado también están incluidas en el estándar y son reconocidas por los compiladores, por lo que habilitar las opciones deseadas en un compilador permitirá la generación de un código binario específico. Cada una de estas extensiones se identifica con una letra que debe añadirse a la ISA base para representar las capacidades hardware de la implementación, como se muestra en la Tabla 4. Por ejemplo, RVM es la extensión que permite incluir la multiplicación/división, RVF es la extensión de punto flotante, y así sucesivamente.

**Tabla 3. ISAs base de RISC-V**
(tabla extraída de https://riscv.org/technical/specifications/)

| Base | Version | Status |
|---|---|---|
| RVWMO | 2.0 | **Ratified** |
| **RV32I** | **2.1** | **Ratified** |
| **RV64I** | **2.1** | **Ratified** |
| *RV32E* | *1.9* | *Draft* |
| *RV128I* | *1.7* | *Draft* |

**Tabla 4. Extensiones de la ISA estándar de RISC-V**
(tabla extraída de https://riscv.org/technical/specifications/)

| Extension | Version | Status |
|---|---|---|
| **Zifencei** | **2.0** | **Ratified** |
| **Zicsr** | **2.0** | **Ratified** |
| **M** | **2.0** | **Ratified** |
| *A* | *2.0* | Frozen |
| **F** | **2.2** | **Ratified** |
| **D** | **2.2** | **Ratified** |
| **Q** | **2.2** | **Ratified** |
| **C** | **2.0** | **Ratified** |
| *Ztso* | *0.1* | *Frozen* |
| *Counters* | *2.0* | *Draft* |
| *L* | *0.0* | *Draft* |
| *B* | *0.0* | *Draft* |
| *J* | *0.0* | *Draft* |
| *T* | *0.0* | *Draft* |
| *P* | *0.2* | *Draft* |
| *V* | *0.7* | *Draft* |
| *N* | *1.1* | *Draft* |
| *Zam* | *0.1* | *Draft* |

La letra G, que significa "general", se utiliza para indicar la inclusión de todas las extensiones MAFD. Obsérvese que una empresa o un particular pueden desarrollar extensiones patentadas utilizando códigos de operación que no han sido utilizados en los módulos estándar. Esto permite que las implementaciones de terceros se desarrollen con un menor tiempo de comercialización.

Por ejemplo, una implementación RISC-V de 64 bits que incluye las cuatro extensiones generales de la ISA más la **manipulación de bits** y **las interrupciones a nivel de usuario**, se denomina ISA RV64GBN. Todos estos módulos están cubiertos en la especificación de usuario o sin privilegios. La base de RISC-V también incluye un conjunto de requisitos e instrucciones para las operaciones con privilegios, necesarias para la ejecución de los sistemas operativos de propósito general.

# 4. VISIÓN GENERAL DEL SISTEMA RVFPGA

En esta sección se describe el Sistema RVfpga en su conjunto, desde el core hasta la interfaz de la placa FPGA. Figura 17la típica organización jerárquica de un sistema empotrado empezando por el core del procesador, luego el SoC construido alrededor del core y finalmente la interfaz del sistema y la placa. Primero se describirá el core del procesador (el **SweRV EH1 Core de Western Digital**), que ejecuta las instrucciones de RISC-V; luego, en la sección B, se describe el **SoC SweRVolfX**, que integra los componentes hardware del sistema (core, memoria y entrada/salida), y las modificaciones realizadas para utilizarlo dentro del Sistema RVfpga; en la Sección C se describe el SoC SweRVolfX implementado en la placa FPGA Nexys A7 (**RVfpgaNexys**) y también se describe el SoC SweRVolfX utilizado en la simulación (**RVfpgaSim**). Finalmente, en la Sección D se explica la estructura de archivos de todo el Sistema RVfpga.

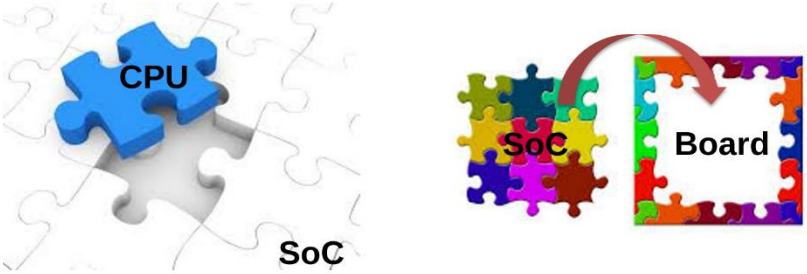



**Figura 17. Organización del sistema integrado**

## A. Core SweRV EH1 y Core Complex SweRV EH1

Western Digital ha desarrollado tres cores RISC-V en los últimos años: **SweRV EH1** (el core utilizado en el Sistema RVfpga), SweRV EH2 y SweRV EL2 (las futuras versiones de RVfpga podrían incluir estos cores). Cada core tiene una licencia Apache 2.0. El core SweRV EH1 es de 32 bits, superescalar de 2 vías y con un pipeline de 9 etapas. El Core SweRV EH2 se basa en el Core EH1 y lo amplía para añadir la capacidad de ejecución de dos hilos para la obtención de rendimiento adicional. El Core SweRV EL2 es un core más pequeño con un rendimiento moderado. En la página web https://www.westerndigital.com/company/innovations/risc-v se describen los tres cores disponibles, cuyas características principales se indican en la Tabla 5.

**Tabla 5. Principales características de los tres cores RISC-V de WD**
(tabla extraída de https://www.westerndigital.com/company/innovations/risc-v)

| Core Name | RISC-V Type | Pipeline Stages | Threads | Size @ TSMC | CoreMarks/Mhz |
|---|---|---|---|---|---|
| SweRV Core EH1 | RV32IMC | 9- dual issue | Single | .11mm @ 28nm | 4.9 |
| SweRV Core EH2 | RV32IMC | 9- dual issue | Dual | .067 @ 16nm | 6.3 |
| SweRV Core EL2 | RV32IMC | 4- single issue | Single | .023 @ 16nm | 3.6 |

De los tres cores, se decide utilizar el **core SweRV EH1** (suministrado con el paquete RVfpga y también disponible en https://github.com/chipsalliance/Cores-SweRV) por su alto rendimiento/MHz y su sencilla estructura de hilos. Además, Chips Alliance, un grupo comprometido con el suministro de hardware de código abierto, proporciona un SoC completo y verificado, denominado **SweRVolf** (suministrado con el paquete RVfpga y también disponible en https://github.com/chipsalliance/Cores-SweRVolf). El Sistema RVfpga

utiliza una extensión del SoC SweRVolf que, a su vez, utiliza la versión **1.8 del Core SweRV EH1** de Western Digital.

El **core SweRV EH1** es de 32 bits y solo admite el modo máquina (modo M). Este core soporta las siguientes extensiones de RISC_V: números enteros (I), instrucciones comprimidas (C) y multiplicación y división de números enteros (M). El Manual de Referencia del Programador (https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf) describe en detalle todos los aspectos del core, desde su estructura hasta la información de temporización y mapeo de memoria. El SweRV EH1 es un core superescalar de dos vías, con un pipeline dual de 9 etapas (ver Figura 18) que soporta cuatro unidades aritmético-lógicas (ALU por sus siglas en inglés), etiquetadas de EX1 a EX4 en dos pipelines, I0 e I1. Ambas vías del pipeline soportan las operaciones de las ALUs. Asimismo, una vía del pipeline soporta instrucciones de load/store y la otra tiene un multiplicador con una latencia de 3 ciclos. El procesador también tiene un divisor, con una latencia de 34 ciclos, que no está dentro del pipeline. Existen cuatro stalls en el pipeline: 'Fetch 1', 'Align', 'Decode', y 'Commit'. La etapa " Fetch 1" incluye un predictor de saltos Gshare. En la etapa "Align", las instrucciones se toman de tres buffers de búsqueda. En la etapa "Decode", se decodifican hasta dos instrucciones provenientes de cuatro buffers de instrucciones. En la etapa de "Commit", finalizan hasta dos instrucciones por ciclo. Por último, en la etapa de " Writeback", se actualizan los registros arquitectónicos.

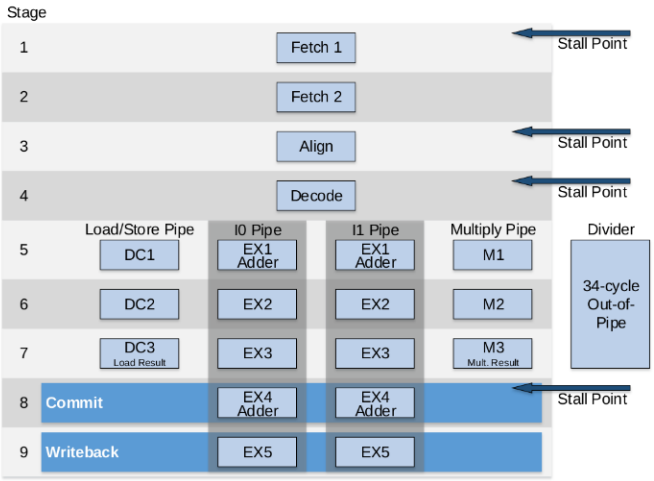



**Figura 18. Microarquitectura del core SweRV EH1**
(figura extraída de https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf)

Figura 19 muestra una comparación de diferentes cores y procesadores actuales. El rendimiento por MHz del core SweRV EH1 es impresionantemente alto con 4,9 CM/MHz (CoreMark por MHz): es el doble de rápido que el Cortex A8 de ARM y su rendimiento incluso supera el del Cortex A15 de ARM.

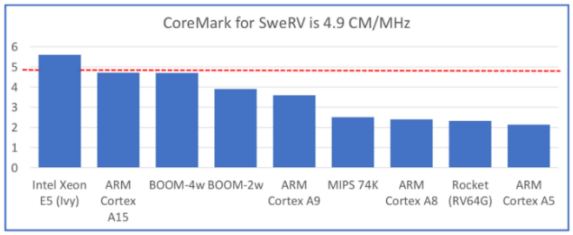



**Figura 19. Rendimiento por hilo y MHz en la ejecución del benchmark CoreMark**
(figura extraída de https://content.riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD_SweRV_Cores_Roadmap_v4SCR.pdf)

Western Digital también proporciona una extensión del core SweRV EH1 denominada **SweRV EH1 Core Complex** (véase Figura 20), que añade los siguientes elementos al core EH1 descrito anteriormente y coloreado en azul en la figura:

- Dos memorias dedicadas, una para instrucciones (ICCM) y la otra para datos (DCCM), que están conectadas al core. Estas memorias proporcionan un acceso de baja latencia y protección SECDED ECC (códigos de corrección de errores individuales y detección de errores dobles). Cada una de las memorias puede ser configurada con tamaños de 4, 8, 16, 32, 48, 64, 128, 256, o 512KB.
- Una caché de instrucciones opcional, asociativa de 4 vías y con protección de paridad o ECC.
- Un Controlador de Interrupciones Programable (PIC) opcional, que soporta hasta 255 interrupciones externas.
- Cuatro interfaces de bus de sistema para la búsqueda de instrucciones (bus maestro IFU), accesos de datos (bus maestro LSU), accesos de depuración (bus maestro de depuración) y accesos DMA externos (puerto esclavo DMA) a memorias acopladas (configurables como buses AXI4 o AHB-Lite de 64 bits).
- La Unidad de Depuración del Core, que cumple con la especificación de depuración de RISC-V.

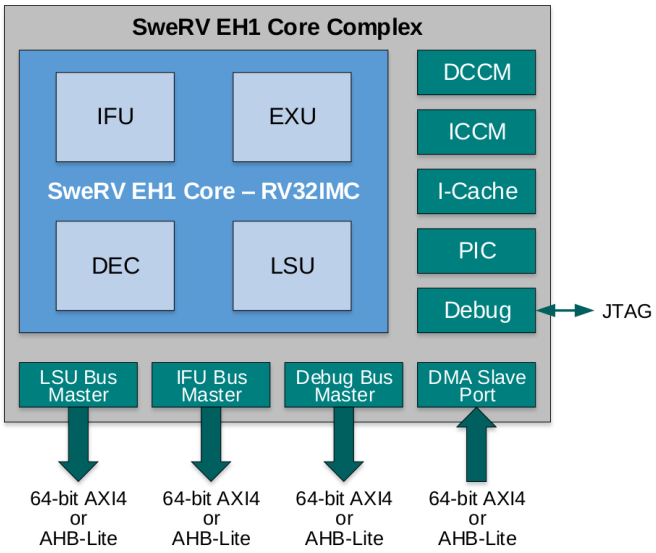



**Figura 20. Core Complex SweRV EH1**

(figura extraída de **https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf**)

## B. El SoC SweRVolfX

El SoC utilizado en el Sistema RVfpga, llamado SweRVolfX e ilustrado en la Figura 21, utiliza el **SoC SweRVolf versión 0.7.3** (https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3), que se basa en el SweRV EH1 Core Complex. Además del SweRV EH1 Core Complex (ver Figura 20), el SoC SweRVolf también incluye una ROM de arranque, una UART, un Controlador de Sistema y un controlador SPI (la Figura 21 muestra estos elementos en blanco). Dado que el core SweRV EH1 utiliza un bus AXI y que los periféricos utilizan un bus Wishbone, el SoC implementa también un bridge AXI-Wishbone.

En este curso se ampliará el SoC SweRVolf con algunas funciones más, como otro controlador SPI (SPI2), un GPIO (entrada/salida de propósito general), un PTC (PWM,

Temporizador y Contador), y los displays de 7 segmentos (la Figura 21rojo, excepto los displays de 7 segmentos, que se incluyen en el controlador del sistema). A este nuevo SoC lo llamamos SweRVolfX (la X proviene de eXtendido).

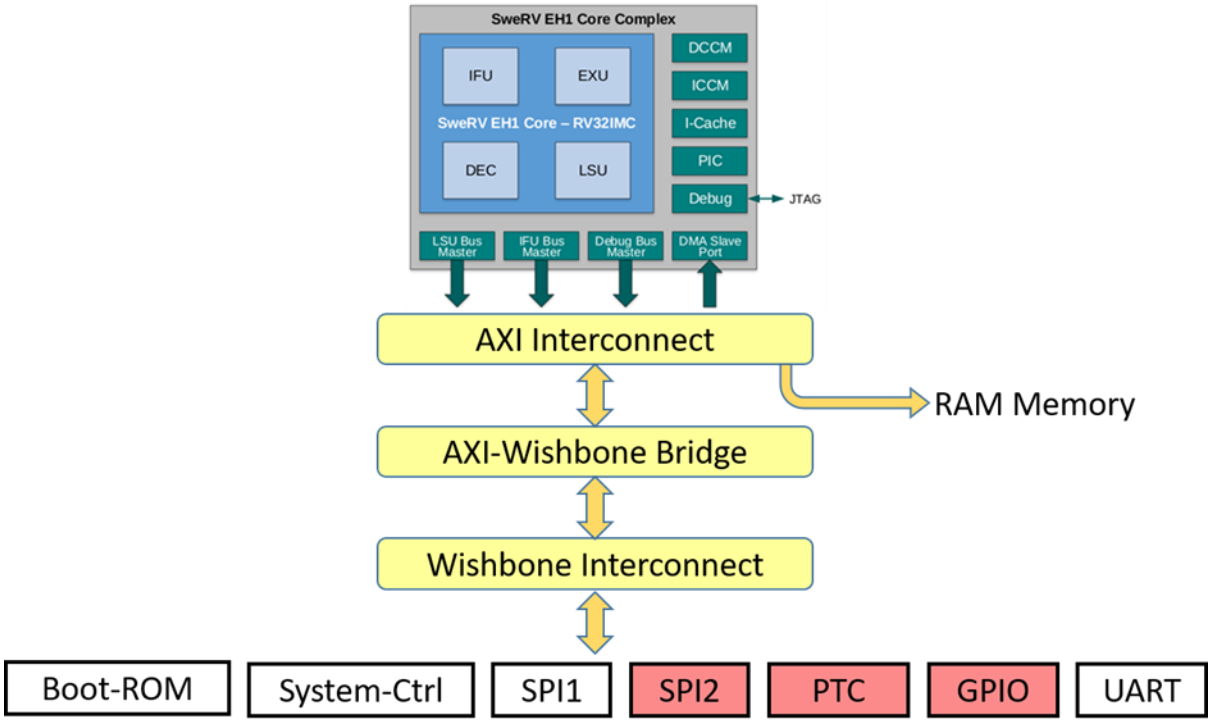



**Figura 21. SoC SweRVolfX (SweRVolf eXtendido con 4 nuevos periféricos).**

Tabla 6 muestra las direcciones en memoria de los periféricos conectados al core a través de la interfaz Wishbone.

**Tabla 6. Direcciones de memoria de los periféricos del SoC SweRVolfX**

| Sistema | Dirección |
|---|---|
| Boot ROM | 0x80000000 - 0x80000FFF |
| Controlador del sistema | 0x80001000 - 0x8000103F |
| SPI1 | 0x80001040 - 0x8000107F |
| SPI2 | 0x80001100 - 0x8000113F |
| PTC | 0x80001200 - 0x8000123F |
| GPIO | 0x80001400 - 0x8000143F |
| UART | 0x80002000 - 0x80002FFF |

### i. Entrada/Salida

El SoC SweRVolfX utiliza dos tipos de controladores hardware para comunicarse con los periféricos: controladores personalizados escritos en Verilog y controladores de código abierto de OpenCores [https://opencores.org/], una comunidad virtual para el desarrollo de cores IP (Propiedad Intelectual) con espíritu de una colaboración gratuita y de código abierto. El SoC SweRVolfX que se usa en este curso incluye las interfaces de E/S listadas a continuación, que se explicarán en detalle e incluso se extenderán en las prácticas de laboratorio 6-10 de RVfpga.

- **Controlador del sistema:** el controlador del sistema contiene funcionalidades comunes del sistema como el mantenimiento del registro con la información de la versión del SoC, el estado de inicialización de la RAM y el temporizador del RISC-V

(en https://github.com/chipsalliance/Cores-SweRVolf se puede encontrar el mapa completo de la memoria). En SweRVolfX hemos modificado este controlador del siguiente modo:

- Hemos incluido un nuevo controlador para comunicarse con los 8 displays de 7 segmentos de la Nexys A7, llamado **SevSegDisplays_Controller**, y hemos añadido dos nuevos registros en este controlador mapeados en las direcciones 0x80001038 y 0x8000103C.
- Hemos añadido dos registros de 1 bit para gestionar interrupciones del GPIO y del PTC, mapeados en la dirección 0x80001018.
- Hemos eliminado los registros del GPIO incluido en SweRVolf, mapeados entre las direcciones 0x80001010 y 0x80001017. Tal y como se describe más adelante, hemos incluido un GPIO más completo.

- **SPI:** dos controladores SPI de código abierto (obtenidos de https://opencores.org/projects/simple_spi y llamados SPI1 y SPI2) están implementados en SweRVolfX. Sus registros expuestos (SPI_SPCR, SPI_SPSR, SPI_SPDR, SPI_SPER, SPI_SPSS) están mapeados entre las direcciones 0x80001040 y 0x8000107F (para SPI1) y entre las direcciones 0x80001100 y 0x8000113F (para SPI2).

- **PTC:** Se utiliza el módulo temporizador de https://opencores.org/projects/ptc. Sus registros están mapeados en el rango de direcciones 0x80001200 a 0x800012FF.

- **GPIO:** Se utiliza el controlador GPIO de https://opencores.org/projects/gpio. Incluye 32 puertos de Entrada/Salida mapeados en el rango de direcciones 0x80001400 a 0x800014FF. Cada pin está conectado con un módulo de buffers de tres estados, de modo que puede ser configurado como una entrada o una salida.

- **UART:** en SweRVolfX está disponible un controlador UART de código abierto (obtenido de https://opencores.org/projects/uart16550). Sus registros expuestos están mapeados entre las direcciones 0x80002000 y 0x80002FFF.

### ii. Memoria

El SoC SweRVolfX incluye una memoria Boot ROM y el hardware necesario para permitir al usuario incluir memorias RAM y SPI Flash.

- **Boot ROM:** una Boot ROM contiene un gestor de arranque primario. Después de reiniciar el sistema, el SoC SweRVolfX empezará a buscar las instrucciones iniciales en el espacio de memoria que se encuentra entre las direcciones 0x80000000 a 0x80000FFF.

- **RAM:** el SoC SweRVolfX no incluye un controlador de memoria, pero reserva los primeros 128MiB de su mapa de memoria (0x00000000-0x07FFFFFF) y expone el bus AXI, de modo que el usuario puede acceder a la memoria RAM utilizando un controlador de memoria.

- **SPI Flash:** también se puede incluir una memoria SPI Flash utilizando el controlador SPI1 descrito en la sección anterior (rango de direcciones: 0x80001040-0x8000107F).

### iii. Interconexión

El core SweRV EH1 utiliza un bus AXI4 para conectar el core con la memoria. El bus también podría ser configurado como un bus AHB-Lite, pero no se usará esa opción en

este material. Todos los periféricos (dispositivos de Entrada/Salida) están conectados a un bus Wishbone, un bus de código abierto muy utilizado en las CPU y los periféricos de OpenCore. Por ello, el sistema incluye una interfaz de AXI a Wishbone (como se muestra en la Figura 21) para conectar el core a los periféricos.

En esta sección, se describe brevemente el funcionamiento de un bus AXI4 y un bus Wishbone. Si está interesado en ampliar sus conocimientos sobre las especificaciones de estos buses, puede utilizar las referencias que se proporcionan a continuación.

## El bus AXI4

El Core Complex SweRV EH1 utiliza la interconexión AXI4 para comunicarse con el mundo exterior (ver Figura 19). El *Advanced eXtensible Interface* (AXI) es un bus común usado por muchos procesadores y es parte de la especificación de interconexión on-chip Advanced Microcontroller Bus Architecture de ARM.

En las siguientes subsecciones, se explicarán brevemente algunos de los principales aspectos de la interconexión AXI4. Puede encontrar la especificación completa de AXI en el siguiente documento:
https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf

- **Características principales del bus AXI**

Las principales características de la tecnología del bus AXI son las siguientes:

- Es adecuado tanto para diseños de gran ancho de banda como de baja latencia
- Proporciona un funcionamiento de alta frecuencia sin utilizar conexiones complejas
- Cumple los requisitos de interfaz de una amplia gama de componentes
- Es adecuado para controladores de memoria con alta latencia de acceso inicial
- Proporciona flexibilidad en la implementación de arquitecturas de interconexión
- Es compatible con las interfaces AHB y APB existentes
- Proporciona fases separadas de dirección/control y datos
- Incluye soporte a las transferencias de datos no alineados (utilizando bytes de señalización)
- Permite transacciones basadas en ráfagas con sólo la dirección de inicio
- Proporciona canales de datos de lectura y escritura separados, que permiten un DMA de bajo costo
- Permite que la información de la dirección se envíe antes de la transferencia real de datos
- Proporciona soporte para el envío de múltiples direcciones pendientes y la finalización de transacciones fuera de orden
- Permite añadir fácilmente etapas de registro para proporcionar una correcta temporización

- **Arquitectura AXI**

El protocolo AXI define los siguientes canales de transacción independientes:

- Leer dirección
- Leer datos
- Escribir dirección
- Escribir datos
- Escribir respuesta

Figura 22 muestra cómo una transacción de lectura utiliza los canales de leer dirección y leer datos. Primero se envían la dirección y los bits de control desde el

dispositivo maestro, luego el dispositivo esclavo responde con los datos en el canal leer datos.

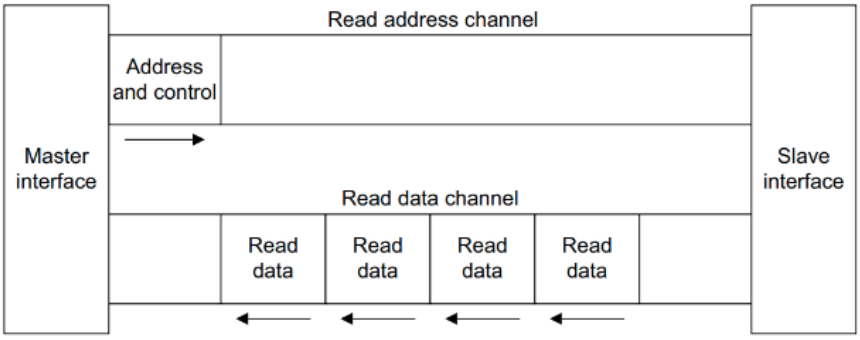



**Figura 22. Arquitectura de los canales de lectura**
(figura extraída de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

Figura 23 muestra cómo una transacción de escritura utiliza los canales escribir dirección, escribir datos y escribir respuesta. De manera similar a una lectura, el dispositivo maestro envía la dirección y los bits de control. Luego el dispositivo maestro envía los datos en el canal escribir datos y el dispositivo esclavo envía una respuesta.

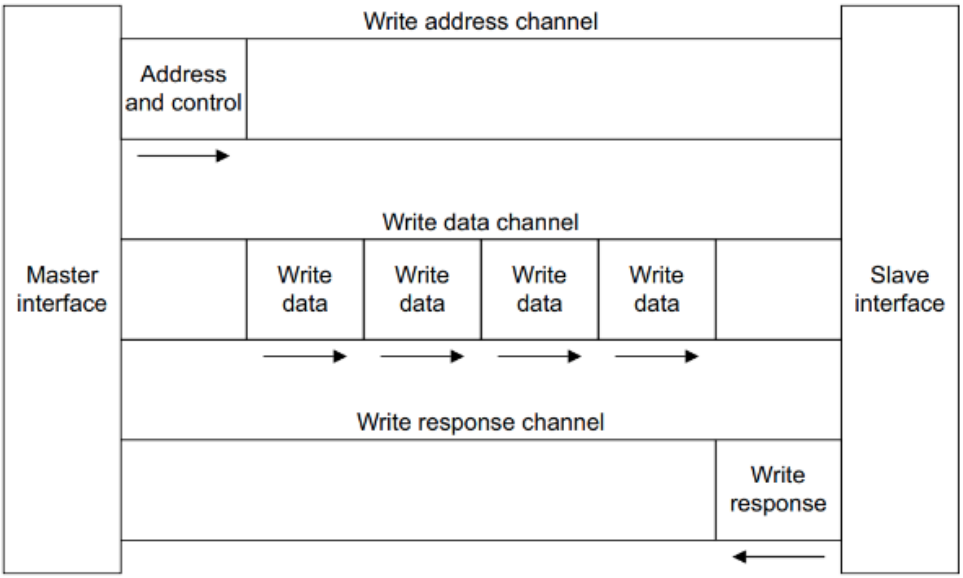



**Figura 23. La arquitectura de los canales de escritura**
(figura extraída de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

El canal de direcciones AXI lleva direcciones e información de control que describe la naturaleza de los datos a transferir. Los datos se transfieren entre el maestro y el esclavo usando:

- Buen un canal de leer datos para transferir datos del esclavo al maestro (Figura 22).
- O bien un canal de escribir datos para transferir datos del maestro al esclavo (Figura 23). En una transacción de escritura, el esclavo utiliza el canal de escribir respuesta para señalar la finalización de la transferencia al maestro (Figura 23).

- **Señales AXI**

Tabla 7. muestra las principales señales usadas en el bus AXI y una breve descripción de cada una de ellas. Las señales están organizadas en cinco grupos, que corresponden a los cinco canales descritos en la sección anterior:

- **Señales del canal de escribir dirección,** cuyos nombres comienzan con **AW**
- **Señales del canal de escribir datos,** cuyos nombres comienzan con **W**
- **Señales del canal de escribir respuesta,** cuyos nombres comienzan con **B**
- **Señales del canal de leer dirección,** cuyos nombres comienzan con **AR**
- **Señales del canal de leer datos,** cuyos nombres comienzan con **R**

**Tabla 7. Señales AXI**

(tabla extraída de https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

| Signal | Source: master/ slave | Input/ Output | Description |
|---|---|---|---|
| Aclk | Global | Input | Global clock signal. |
| AResetn | Global | Input | Global reset signal |
| AWID[3:0] | Master | Input | Write address ID. |
| AWADDR[31:0] | Master | Input | Write address. |
| AWLEN[3:0] | Master | Input | Write burst length. |
| AWSIZE[2:0] | Master | Input | Write burst size. |
| AWBURST[1:0] | Master | Input | Write burst type. |
| AWLOCK[1:0] | Master | Input | Write lock type. |
| AWCACHE[3:0] | Master | Input | Write cache type. |
| AWPROT[2:0] | Master | Input | Write protection type. |
| WDATA[31:0] | Master | Input | Write data. |
| ARID[3:0] | Master | Input | Read address ID. |
| ARADDR[31:0] | Master | Input | Read address. |
| ARLEN[3:0] | Master | Input | Read Burst length. |
| ARSIZE[2:0] | Master | Input | Read Burst size. |
| ARLOCK[1:0] | Master | Input | Read Lock type. |
| ARCACHE[3:0] | Master | Input | Read Cache type. |
| ARPROT[2:0] | Master | Input | Read Protection type. |
| RDATA[31:0] | Master | Input | Read data. |
| WLAST | Master | Input | Write last. |
| RLAST | Slave | Output | Read last. |
| AWVALID | Master | Output | Write address valid. |
| AWREADY | Slave | Output | Write address ready. |
| WVALID | Master | Output | Write valid. |
| RAVLID | Slave | Output | Read valid. |
| WREADY | Slave | Output | Write ready. |
| BID[3:0] | Slave | Output | Write Response ID. |
| RID[3:0] | Slave | Output | Read response ID. |
| BRESP[1:0] | Slave | Output | Write response. |
| RRESP[1:0] | Slave | Output | Read response. |
| BVALID | Slave | Output | Write response valid. |

## El Bus Wishbone

Los periféricos del SoC SweRVolfX utilizan la Arquitectura de Interconexión Wishbone SoC para cores IP Portátiles (https://opencores.org/howto/wishbone). El propósito principal de este bus es fomentar la reutilización del diseño reduciendo los problemas de integración del SoC. Anteriormente, los cores IP utilizaban esquemas de interconexión no estándar que dificultaban su integración. Estas interconexiones no estándar requerían la creación de una lógica personalizada para conectar cada uno de los cores

entre sí. Al adoptar un esquema de interconexión estándar como el bus Wishbone, los cores pueden ser integrados más rápida y fácilmente por el usuario final.

- **Características principales de Wishbone**

Las principales características de la tecnología de bus Wishbone son las siguientes:

- Soporta las metodologías de diseño estructurado utilizadas por los equipos de grandes proyectos.
- Incluye un conjunto completo de protocolos populares de bus de transferencia de datos, incluyendo:
  i. Ciclos de LECTURA/ESCRITURA
  ii. Ciclos de transferencia de BLOQUE
  iii. Ciclos de LECTURA/MODIFICACIÓN/ESCRITURA
- Proporciona anchuras modulares de bus de datos y tamaños de operandos de hasta 64 bits.
- Soporta tanto el ordenamiento de datos BIG ENDIAN como LITTLE ENDIAN.
- Soporta varios métodos de interconexión de cores, incluyendo punto a punto, bus compartido, conmutador de barras cruzadas (*crossbar switch*), e interconexiones de matriz de conmutación (*switched fabric*).
- Incluye protocolos de establecimiento de comunicación que permiten que cada core IP modifique su velocidad de transferencia de datos.
- Soporta transferencias de datos con una única señal de reloj.
- Soporta terminación de ciclo normal, terminación por reintento y terminación debida a error.
- Incluye anchos de dirección modulares.
- Proporciona un esquema de decodificación parcial de direcciones para los esclavos. Esto facilita la decodificación de direcciones a alta velocidad, utiliza menos lógica redundante y soporta tamaños de direcciones variables y diferentes métodos de interconexión.
- Proporciona etiquetas definidas por el usuario. Éstas son útiles para aplicar la información a una dirección o bus de datos o a un ciclo de bus. Las etiquetas definidas por el usuario son especialmente útiles cuando se modifica un ciclo de bus para identificar información como:
  i. Transferencias de datos
  ii. Bits de paridad o de corrección de errores
  iii. Vectores de interrupción
  iv. Operaciones de control de la caché
- Incluye una arquitectura Maestro/Esclavo para diseño de sistemas flexibles.
- Tiene capacidades de multiprocesamiento (multi-MASTER). Esto permite una amplia variedad de configuraciones SoC
- Incluye una metodología de arbitraje que puede ser definida por el usuario final (árbitro de prioridad, árbitro de round-robin, etc.)

- **Arquitectura y señales del bus Wishbone**

Figura 24 la conexión estándar entre un maestro (en este caso, el Core SweRV EH1) y un esclavo (en este caso, periféricos como GPIO, SPI ...) a través de un bus Wishbone. El bus Wishbone es mucho más simple que el bus AXI4 y, como se muestra en la Tabla 8, utiliza menos señales.

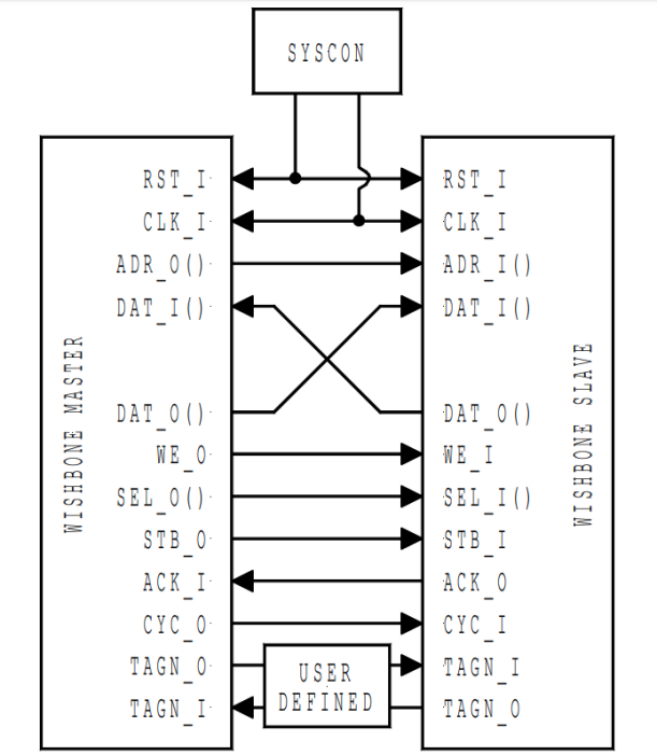



**Figura 24. Arquitectura del bus Wishbone**
(figura extraída de https://opencores.org/howto/wishbone)

**Tabla 8. Señales del bus Wishbone**
(tabla extraída de https://opencores.org/howto/wishbone)

| Signal name | description |
|---|---|
| CLK_O | It coordinates all activities for the internal logic within the WISHBONE interconnect. The INTERCON module connects the [CLK_O] output to the [CLK_I] input on MASTER and SLAVE |
| RST_O | It forces all WISHBONE interfaces to restart. All internal self-starting state machines are forced into an initial state. The INTERCON connects the [RST_O] output to the [RST_I] input on MASTER and SLAVE |

| Signal name | Description |
|---|---|
| CLK_I | All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals are stable before the rising edge of [CLK_I]. |
| DAT_I() | The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). |
| DAT_O() | The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). |
| RST_I() | The reset input [RST_I] forces the WISHBONE interface to restart |
| TGD_I() | Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I]. |
| TGD_O( ) | Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O] |

| Signal name | Description |
|---|---|
| ACK_I | The acknowledge input [ACK_I], when asserted, indicates the normal termination of a bus cycle |
| CYC_O | The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress |
| STALL_I | The pipeline stall input [STALL_I] indicates that current slave is not able to accept the transfer in the transaction queue |
| ERR_I | The error input [ERR_I] indicates an abnormal cycle termination |
| RTY_I | The retry input [RTY_I] indicates that the interface is not ready to accept or send data, and that the cycle should be retried |
| STB_O | The strobe output [STB_O] indicates a valid data transfer cycle |
| WE_O | The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle |

| Signal name | Description |
|---|---|
| ACK_O | The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle |
| CYC_I | The cycle input [CYC_I], when asserted, indicates that a valid bus cycle is in progress |
| STALL_O | The pipeline stall signal [STALL_O] indicates that the slave can not accept additional transactions in its queue |
| ERR_O | The error output [ERR_O] indicates an abnormal cycle termination |
| RTY_O | The retry output [RTY_O] indicates that the indicates that the interface is not ready to accept or send data, and that the cycle should be retried |
| STB_I | The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted |
| WE_I | The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle |

# C. SweRVolfX SoC en la placa FPGA Nexys A7 y en simulación

El SoC SweRVolfX (Figura 21) se puede ejecutar tanto (1) en la placa FPGA Nexys A7 (o Nexys4 DDR), cuya configuración se denomina **RVfpgaNexys,** o (2) en simulación, que se denomina **RVfpgaSim**.

## i. RVfpgaNexys

RVfpgaNexys es el SoC SweRVolfX (Figura 21) mapeado a la placa FPGA Nexys A7 y sus periféricos. (Recuerde que la placa FPGA Nexys 4 DDR también puede ser utilizada.) RVfpgaNexys es lo mismo que SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), excepto que este último está basado en SweRVolf. Los principales elementos utilizados por RVfpgaNexys se ilustran en la Figura 25:

- Hardware programado en la FPGA:
  - **SoC SweRVolfX** (Figura 21).
  - **Controlador Lite DRAM**
  - **Generador de reloj:** la placa Nexys A7 incluye un único oscilador de cristal de **100 MHz** que es usado por el **controlador Lite DRAM**. La frecuencia de este reloj se reduce a **50 MHz** para usarla en el **SoC SweRVolfX**.
  - **Módulo de Combinación de Dominios de Reloj:** conexión de 2 dominios de reloj: SoC SweRVolfX y Lite DRAM.
  - **Lógica BSCAN para el puerto JTAG**: puedes encontrar más información sobre este módulo en https://github.com/chipsalliance/Cores-SweRVolf/issues/29.

- Memoria/Periféricos de la placa FPGA Nexys A7 (o Nexys4 DDR) usados en RVfpgaNexys:
  - **Memoria DDR2** (a la que se accede a través del controlador Lite DRAM mencionado anteriormente)
  - **Conexión USB**
  - **Memoria Flash SPI**
  - **Acelerómetro SPI**
  - **16 LEDs** y **16 interruptores**
  - **Displays de 8 dígitos de 7 segmentos**

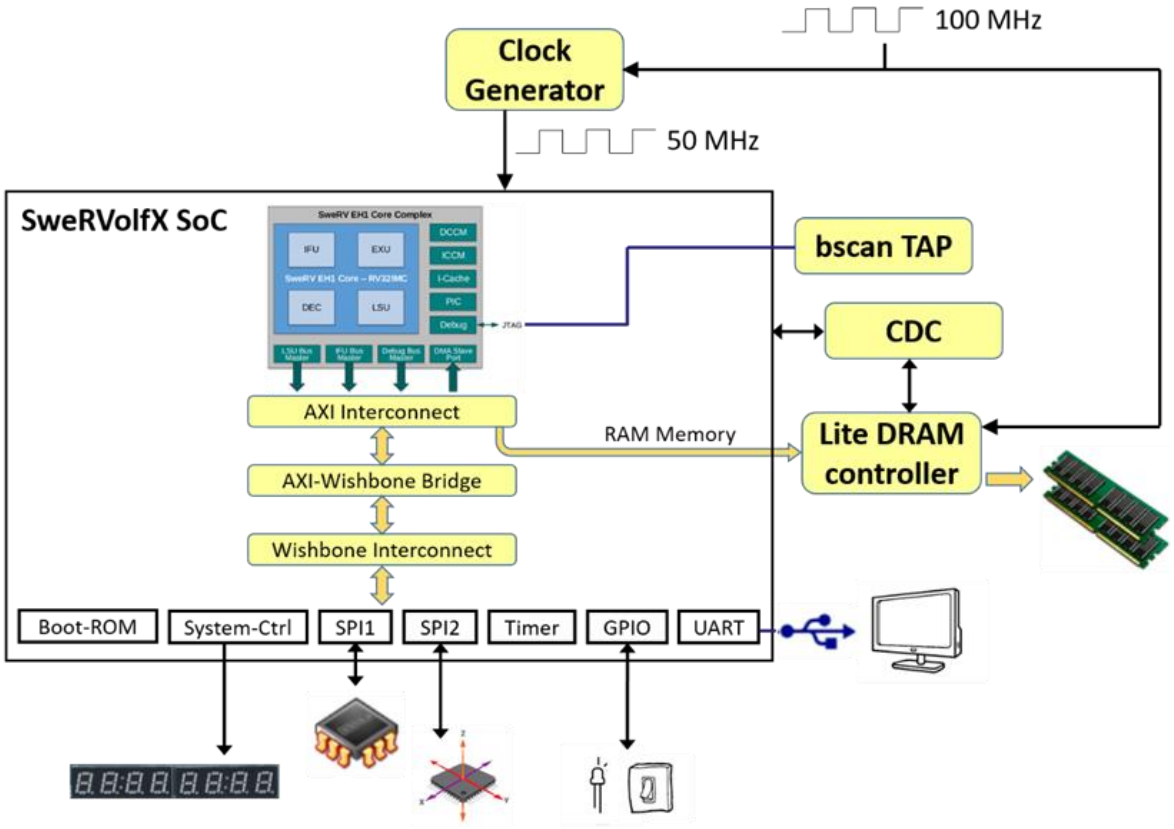



**Figura 25. RVfpgaNexys**

La placa Nexys A7 (Figura 26) es una placa de desarrollo recomendada para los planes de estudio de ingeniería eléctrica e informática. Cuesta 265 dólares (o 198,75 dólares con el descuento académico – para ello debe crear una cuenta en Digilent con una dirección de correo electrónico .edu). Digilent proporciona un extenso manual de referencia de la placa Nexys A7 en: https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf. Esta placa puede ser alimentada con un adaptador de 5V (no suministrado con la tarjeta) o con un PC a través del conector microUSB de la placa. Un microcontrolador Microchip PIC24 gestiona el proceso de carga en la FPGA, haciendo de esta placa una opción fácil de usar. La placa es programable usando la Suite de Diseño Vivado de Xilinx o OpenOCD. La configuración deseada puede ser descargada a la FPGA usando una de las siguientes cuatro fuentes: una tarjeta MicroSD formateada en FAT32, una memoria Flash USB formateada en FAT32, la memoria flash interna, o una interfaz JTAG.

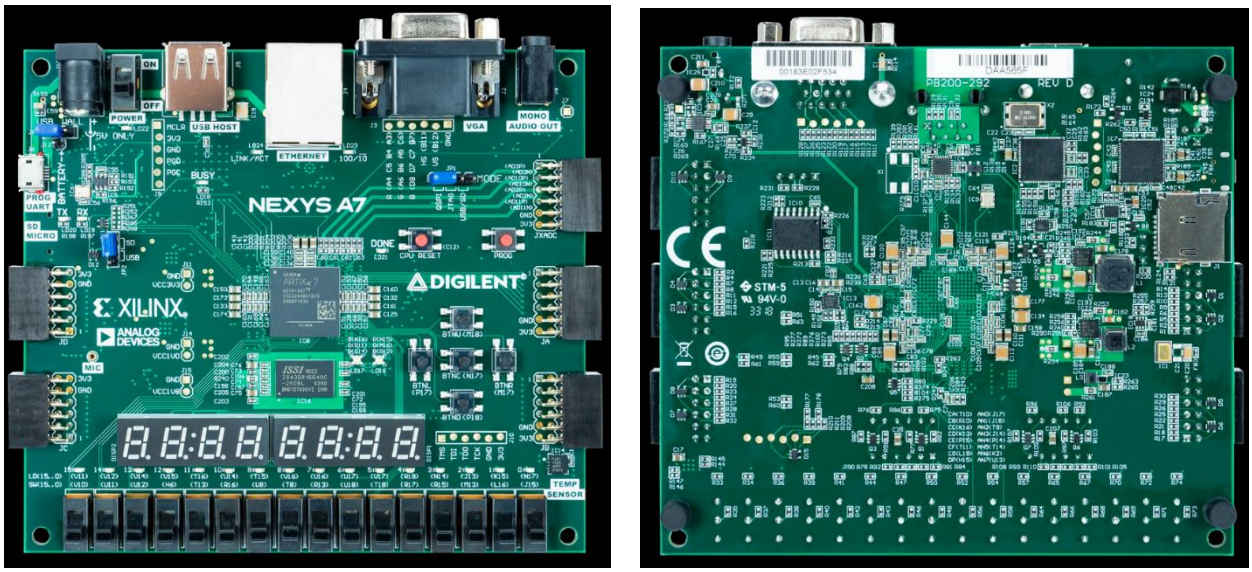



**Figura 26. Placa FPGA Nexys A7 de Digilent**
(figura extraída de https://reference.digilentinc.com/)

La placa FPGA Nexys A7-100T incluye las siguientes interfaces y dispositivos:

- 128 MiB DDR RAM
- Memoria Flash SPI de 128 Mibit
- Displays de 8 dígitos de 7 segmentos
- 16 Interruptores
- 16 LEDs
- Sensores y conectores, incluyendo un micrófono, un conector de audio, un puerto VGA 25, un puerto USB, LEDs RGB, un sensor de temperatura I2C, un acelerómetro SPI, entre otros.
- Xilinx Artix-7 FPGA, que tiene las siguientes características:
  - 15.850 estructuras lógicas organizadas en cuatro LUT de 6 entradas y 8 flip-flops.
  - 4.860 Kibits de RAM
  - 6 estructuras de gestión de reloj (CMT por sus siglas en inglés)
  - 170 pines de E/S
  - Frecuencia de reloj interno de 450 MHz

### ii. RVfpgaSim

El SoC SweRVolfX (Figura 21) también puede incluir un wrapper de Verilog para permitir la simulación. RVfpgaSim encapsula el SoC SweRVolf en un testbench para ser usado en simuladores de HDL. RVfpgaSim es lo mismo que SweRVolf sim (https://github.com/chipsalliance/Cores-SweRVolf), excepto que este último se basa en SweRVolf. Aunque existen muchos simuladores de HDL de código abierto, en RVfpga se usa Verilator (https://www.veripool.org/wiki/verilator). Este simulador HDL abierto y gratuito acepta Verilog sintetizable o SystemVerilog y afirma ser el simulador Verilog/SystemVerilog más rápido. Se utiliza ampliamente en la industria y en el mundo académico; proporciona un soporte inmediato de los vendedores de cores IP de ARM y RISC-V; y está dirigido por Chips Alliance y la Fundación Linux.

## D. Estructura de archivos

En las secciones anteriores se ha mostrado la organización de alto nivel del sistema que se utiliza en este material, desde el **SweRV EH1 Core Complex** (Figura 20), hasta el **SoC SweRVolfX** (Figura 21y, finalmente, las implementaciones **RVfpgaNexys** (Figura 25y **RVfpgaSim**.

En esta sección se describe la estructura de archivos de todo el sistema. Mientras lea estas explicaciones, abra los archivos y véalos en su computadora. Los archivos están disponibles en ***[RVfpgaPath]/RVfpga/src***.

### i. SweRV EH1 Core Complex

Figura 27Figura 20 muestra la estructura de archivos del **SweRV EH1 Core Complex** (Figura 20). El core está organizado en tres bloques principales: un wrapper SweRV (resaltado en gris) que incluye el core SweRV EH1 (resaltado en verde) y algunos otros elementos (como el Controlador de Interrupciones o la Unidad de Depuración), y las memorias de Datos/Instrucciones y la Caché de Instrucciones (resaltadas en rojo).

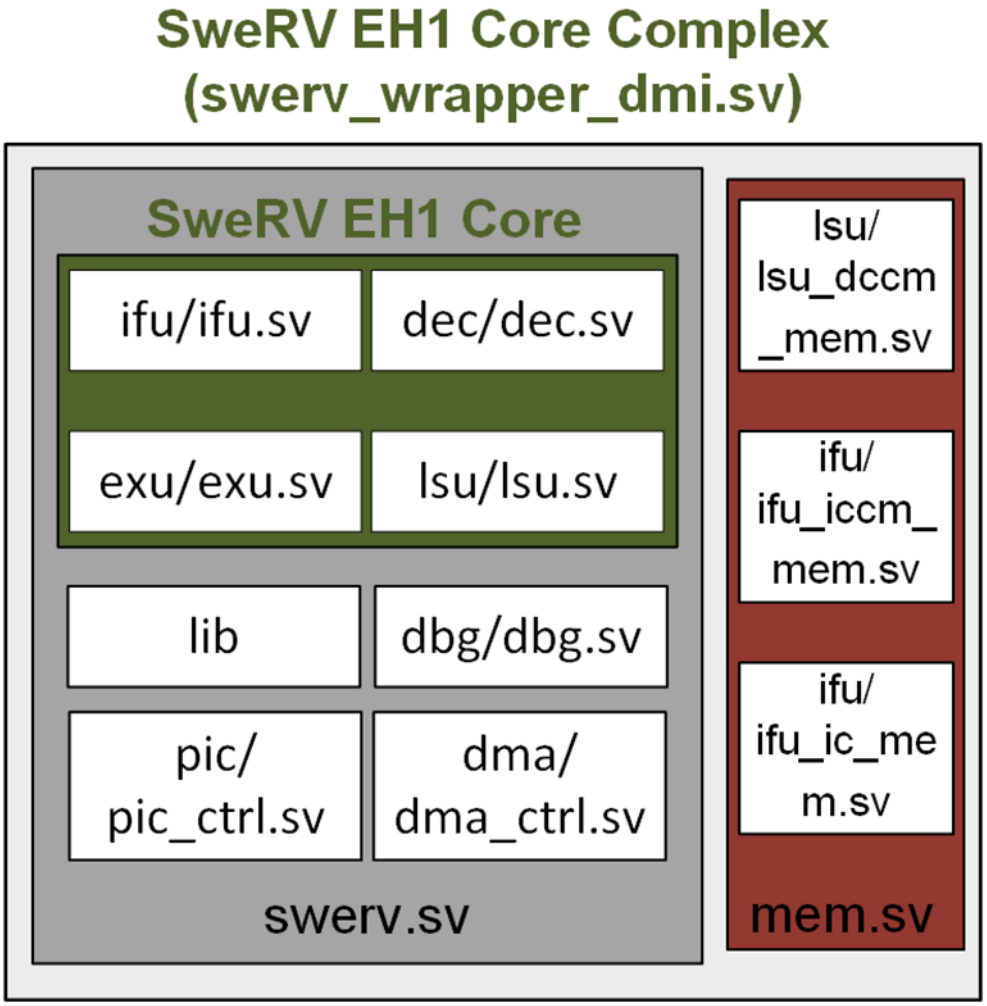



**Figura 27. SweRV EH1 Core Complex**

Los archivos de Verilog para el SweRV EH1 Core Complex están disponibles en esta carpeta:
***[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex***

Acceda a este directorio en su computadora para ver los archivos según se comentan en esta sección.

El archivo principal del SweRV EH1 Core Complex está en el archivo: *swerv_wrapper.sv*; el módulo superior se llama **swerv_wrapper**, y en él se instancian dos módulos que corresponden a los dos bloques resaltados en gris y rojo en la Figura 27:

- **mem** (implementado dentro de *mem.sv*): este módulo instancia los módulos para la implementación de la DCCM (**lsu_dccm_mem**, implementado en el archivo *lsu/lsu_dccm_mem.sv*), la ICCM (**ifu_iccm_mem**, implementado en el archivo *ifu/ifu_iccm_mem.sv*) y la Caché de Instrucciones (**ifu_ic_mem**, implementado en el archivo *ifu/ifu_ic_mem.sv*).

- **swerv** (implementado dentro de *swerv.sv*): este módulo instancia las unidades que componen el core.
  El core SweRV EH1 (resaltado en verde en la Figura 27) consta de las siguientes cuatro unidades:
  - Carpeta ***ifu*** (Instruction Fetch Unit): esta carpeta incluye los archivos de Verilog (el módulo principal está disponible en *ifu.sv*) para la Icache (caché de instrucciones), Fetch, Branch Predictor y Aligner.
  - Carpeta ***dec*** (Decode Unit): esta carpeta incluye los archivos de Verilog (el módulo principal está disponible en *dec.sv*) para la Decodificación de Instrucciones, el Scoreboard de Dependencias y el Banco de Registros.
  - Carpeta ***exu*** (Execution Unit): esta carpeta incluye los archivos de Verilog (el módulo principal está disponible en *exu.sv*) para las unidades aritméticas/lógicas disponibles en el core: dos ALU segmentadas, un Multiplicador segmentado y un Divisor fuera del pipeline.
  - Carpeta ***lsu*** (Load Store Unit): esta carpeta incluye los archivos de Verilog (el módulo principal está disponible en *lsu.sv*) para la unidad de Load/Store segmentada.

  Otras unidades incluidas en este módulo son:
  - Carpeta ***dbg*** (Debug Unit): esta carpeta incluye los archivos de Verilog (el módulo principal está disponible dentro de *dbg.sv*) de la unidad de depuración, que se encarga de poner el resto del core en modo inactivo, enviar los comandos/dirección, enviar los comandos/direcciones, enviar los datos de escritura y recibir los datos de lectura, y luego reanudar el core en modo normal.
  - Carpeta ***lib***: esta carpeta incluye los archivos de Verilog de los buses AXI y AHB-Lite.
  - Carpeta ***pic***: incluye el fichero *pic_ctlr.sv*, en el que se implementa el Controlador Programable de Interrupciones en el módulo **pic_ctrl**.
  - Carpeta ***dma***: incluye el fichero *dma_ctrl.sv*, que impelmenta el DMA en el módulo **dma_ctrl**.

### ii. SoC SweRVolfX

Figura 28 muestra la estructura de archivos para el **SoC SweRVolfX** (Figura 21). El SoC está organizado como los módulos que corresponden a los bloques mostrados en la Figura 21.

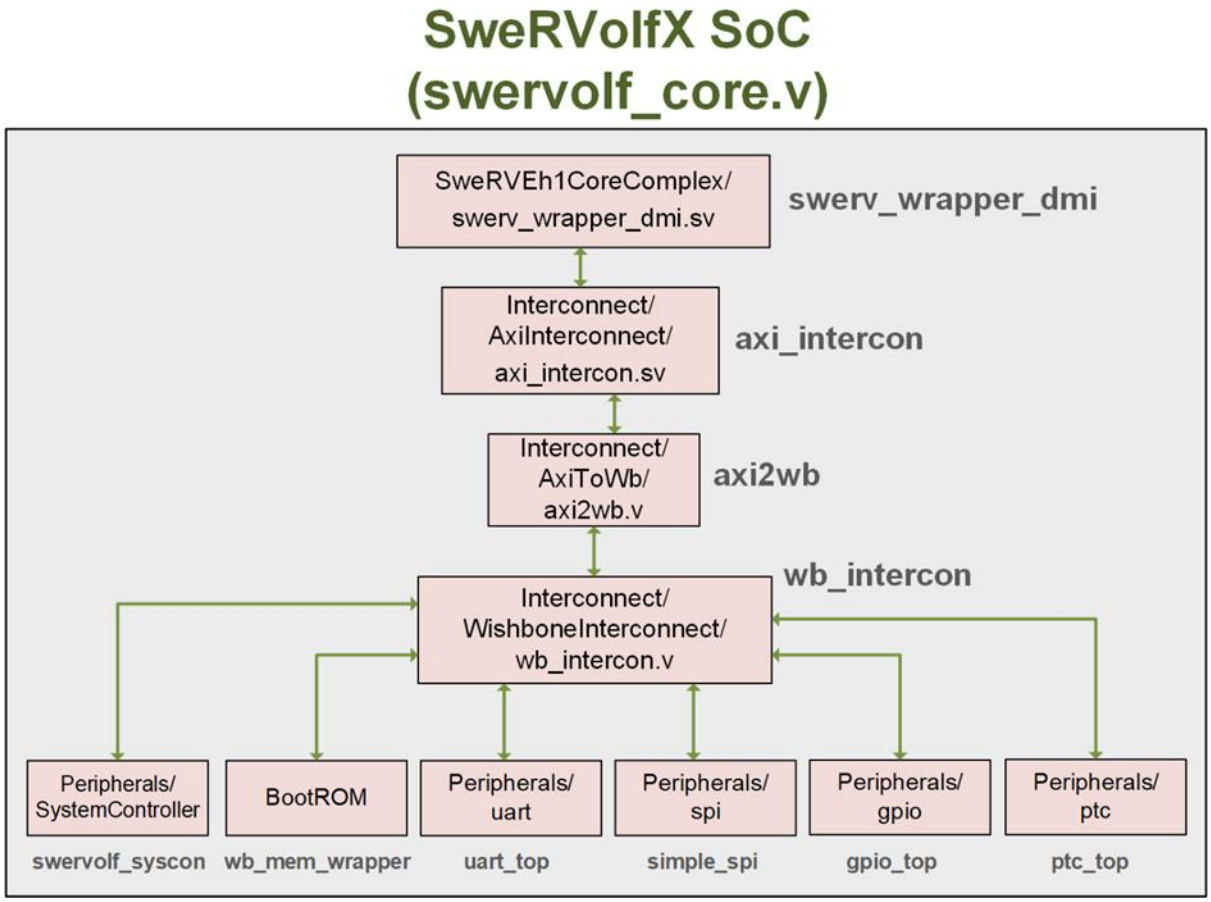



**Figura 28. SoC SweRVolfX**

Los archivos del SoC SweRVolfX se encuentran en:
***[RVfpgaPath]/RVfpga/src/SweRVolfSoC***

Encuentre ese directorio en su computadora para ver los archivos según los referimos en esta sección.

El módulo principal para el **SoC SweRVolfX** está disponible en: ***[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v***. Abra ese archivo y observe que incluye los módulos contenidos en el SoC SweRVolfX (Figura 21), específicamente:

- **axi_intercon** (disponible dentro de *Interconnect/AxiInterconnect/axi_intercon.v*): este módulo se agrega a través de otro archivo en la línea 100 (`` `include "axi_intercon.vh" ``). Este conecta el SweRV EH1 Core Complex con la interfaz AXI-to-Wishbone.
- **axi2wb** (disponible dentro de *Interconnect/AxiToWb/axi2wb.v*): este módulo, que está instanciado en la línea 153 de *swervolf_core.v*, es la interfaz AXI-to-Wishbone que permite la comunicación entre el core EH1 basado en AXI y los periféricos basados en Wishbone.
- **wb_intercon** (disponible dentro de *Interconnect/WishboneInterconnect/wb_intercon.v*): este módulo se agrega a través de otro archivo en la línea 145 (`` `include "wb_intercon.vh" ``). Conecta la interfaz AXI-to-Wishbone con los diferentes periféricos a través de un multiplexor que se analizará y modificará más adelante.

- **wb_mem_wrapper** (disponible dentro de *BootROM/wb_mem_wrapper.v*): el wrapper para la memoria de arranque descrita anteriormente está instanciada en la línea 197 de *swervolf_core.v*. Instancia el módulo **dpram64** (disponible dentro de *BootROM/dpram64.v*), que es un módulo básico de la RAM.
- **swervolf_syscon** (disponible dentro de *Peripherals/swervolf_0.7/rtl/swervolf_syscon.v*): este módulo, que está instanciado en la línea 215 de *swervolf_core.v*, define el Controlador del Sistema.
- **simple_spi**: Controlador SPI obtenido de OpenCores y disponible dentro de *Peripherals/spi/simple_spi_top.v*. Se instancia en las líneas 246 (SPI1) y 387 (SPI2) de *swervolf_core.v*.
- **uart_top**: Controlador UART obtenido de OpenCores y disponible dentro de *Peripherals/uart/uart_top.v*. Está instanciado en la línea 272 de *swervolf_core.v*.
- **gpio_top**: Controlador GPIO obtenido de OpenCores y disponible dentro de *Peripherals/gpio/gpio_top.v*. Está instanciado en la línea 338 de *swervolf_core.v*.
- **ptc_top**: Controlador PTC obtenido de OpenCores y disponible dentro de *Peripherals/ptc/ptc_top.v*. Está instanciado en la línea 361 de *swervolf_core.v*.
- **swerv_wrapper_dmi** (disponible dentro de *SweRVEh1CoreComplex/swerv_wrapper_dmi.v*): instancia (línea 407 de *swervolf_core.v*) del SweRV EH1 Core Complex de Western Digital, descrito en la sección anterior (Figura 27).

### iii. Wrappers para ejecución en placa y simulación

**SIMULACIÓN:**
**RVfpgaSim** contiene el **SoC SweRVolfX** encapsulado en un testbench que es utilizado por los simuladores de HDL. Está disponible en *[RVfpgaPath]/RVfpga/src/rvfpgasim.v*.

**EJECUCIÓN EN PLACA:**
**RVfpgaNexys** (disponible en: *[RVfpgaPath]/RVfpga/src/rvfpganexys.v*) contiene el **SoC SweRVolfX** en un entorno que permite cargarlo en la placa FPGA Nexys A7 y sus periféricos (véase Figura 25). Este módulo instancia, además de algunos otros módulos (como un módulo *generador de reloj*, **clk_gen_nexys**, un módulo de cruce de *dominio de reloj*, **axi_cdc_intf** o un módulo BSCAN para el puerto JTAG, **bscan_tap**), las dos principales estructuras del SoC:

- **swervolf_core**: instancia del **SoC SweRVolfX** descrito en la subsección anterior (Figura 28). Esto también requiere un archivo de restricciones llamado *rvfpganexys.xdc* (disponible en *[RVfpgaPath]/RVfpga/src/*), que define las conexiones entre el SoC y la placa.
- **litedram_top**: paquete para el Controlador DDR2 de LiteDRAM, que conecta el SoC SweRVolfX con la Memoria DDR2, y que se implementa en el archivo *[RVfpgaPath]/RVfpga/src/LiteDRAM/litedram_top.v.* Esto también requiere un archivo de restricciones llamado *litedram.xdc* (disponible dentro de *[RVfpgaPath]/RVfpga/src/LiteDRAM*), que define las conexiones entre el Controlador de la Memoria y la Memoria DDR2 incorporada.

Como resumen, la Figura 29 muestra la jerarquía para la implementación de RVfpgaNexys en la placa FPGA del Nexys A7.

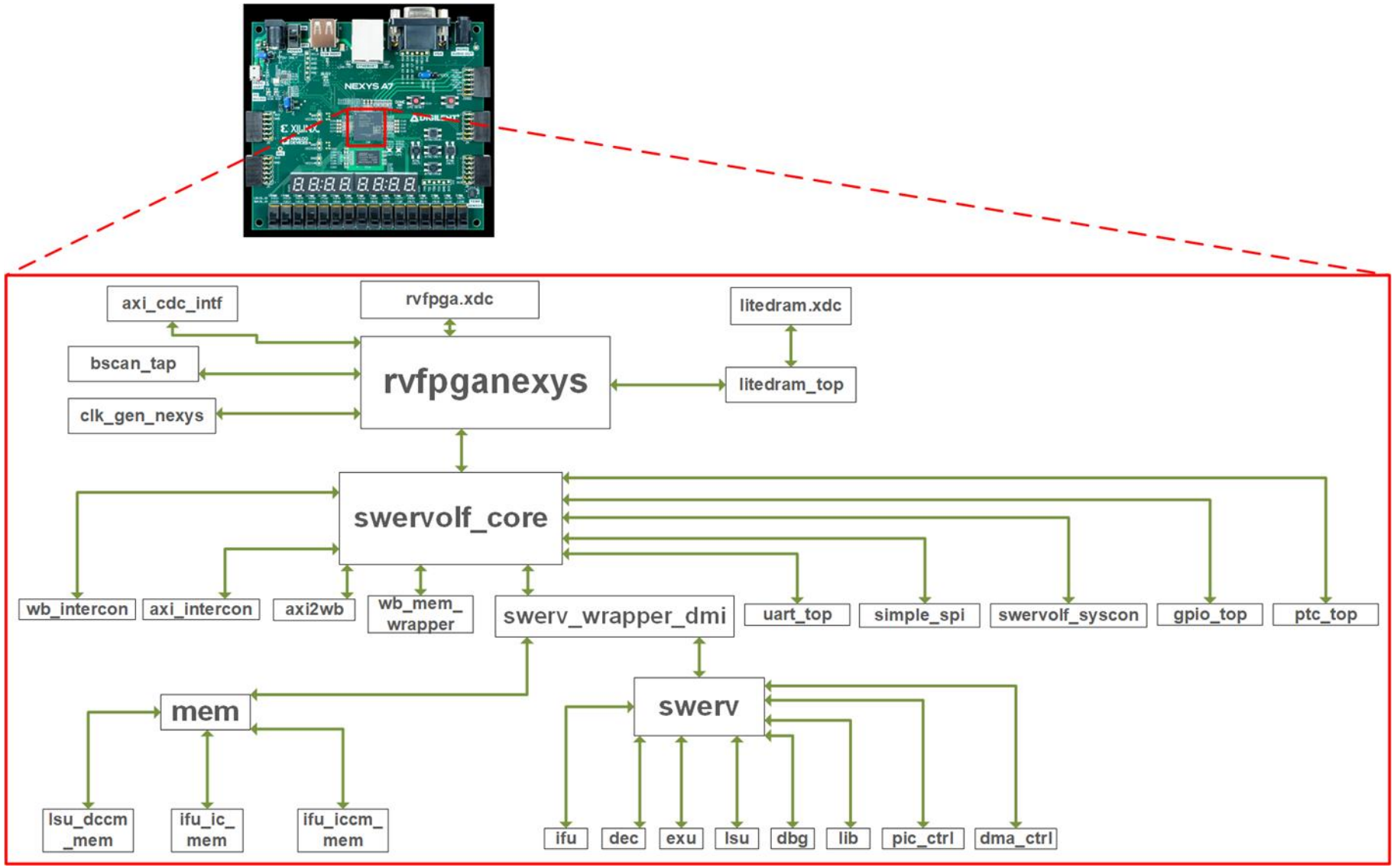



**Figura 29. Jerarquía de módulos para la implementación en la placa FPGA Nexys A7**

# 5. INSTALACIÓN DE HERRAMIENTAS DE SOFTWARE

**Las instrucciones a continuación son para un sistema operativo Ubuntu 18.04**, pero otras distribuciones de GNU/Linux, así como Windows o MacOS, siguen pasos similares (si no exactamente los mismos). En algunos casos, en este documento se insertan cuadros con instrucciones específicas para estos otros sistemas operativos. Si usa Ubuntu, puede ignorar esos cuadros.

Las instrucciones muestran cómo instalar las siguientes herramientas:

- **A. Vivado**: necesario para resintetizar el SoC. Esto es algo que se hará principalmente en las prácticas de laboratorio 6-10, donde se añadirán diferentes características al SoC básico.
- **B. VSCode (Visual Studio Code) y PlatformIO**: son las principales herramientas utilizadas en la GSG y en las prácticas. Se usan para programar la FPGA y para ejecutar/depurar programas en ella.
- **C. Verilator y GTKWave**: necesarios para simular el SoC y analizar las diferentes señales. De nuevo, estas herramientas se utilizarán principalmente en las prácticas de laboratorio 6-10.

Tenga en cuenta que, para la mayoría de las cosas que se harán en esta GSG y en los laboratorios, instalar VSCode y PlatformIO sería suficiente. Sin embargo, se recomienda instalar también las otras herramientas en este momento (Vivado, Verilator y GTKWave), de modo que no sean necesarias instalaciones poesteriores.

Este proceso puede prolongarse varias horas (o más, según la velocidad de descarga), pero la mayor parte del tiempo se emplea esperando mientras se descargan e instalan los programas.

## A. Instalar Vivado

Vivado es una herramienta de Xilinx para ver, modificar y sintetizar el código Verilog en una FPGA que soporte RISC-V. Se usará extensamente en las prácticas de laboratorio posteriores. Las instrucciones de instalación están disponibles en https://reference.digilentinc.com/vivado/installing-vivado/start y se resumen a continuación.

**Windows:** la página web a la que se hace referencia arriba (https://reference.digilentinc.com/vivado/installing-vivado/start) también incluye instrucciones detalladas para instalar Vivado en Windows. A continuación se insertan cuadros cuando se requieren instrucciones específicas para Windows.

**macOS:** Vivado no está soportado en macOS; por lo tanto, se necesita una máquina virtual Linux/Windows para ejecutar Vivado en este sistema operativo.

1. Acceda a https://reference.digilentinc.com/vivado/installing-vivado/start

2. Será dirigido a la página de descargas de Xilinx: https://www.xilinx.com/support/download.html

3. Se recomienda que instale el "Self Extracting Web Installer". En el momento de escribir este documento, se podía acceder al mismo a través del siguiente enlace en la página de descargas: Instalador unificado de Xilinx 2019.2: Instalador Web autoextraíble de Linux

**WINDOWS:** En el momento de escribir este documento, el "Self Extracting Web Installer" para Windows está en este enlace en la página de descargas: Instalador unificado de Xilinx 2019.2: Instalador web autoextraíble para Windows

4. Se le pedirá que acceda a su cuenta de Xilinx antes de que pueda descargar el instalador. Si aún no tiene una cuenta, tendrá que crear una.

5. Ejecute el archivo binario. Abra una terminal y hágala root (escriba "sudo su"). Luego arrastre el archivo binario (Xilinx_Unified_2019.2_1106_2127_Lin64.bin) a la terminal. Si le pide que haga el archivo ejecutable y lo ejecute, seleccione OK.

- **Solución de problemas:** Si la terminal dice permiso denegado, escriba lo siguiente en ésta (en el mismo directorio que el archivo binario está ubicado):
  > sudo `chmod +x ./Xilinx_Unified_2019.2_1106_2127_Lin64.bin`
  > sudo`./Xilinx_Unified_2019.2_1106_2127_Lin64.bin`

**WINDOWS:** En Windows puede simplemente ejecutar el archivo .exe que descargó en los pasos 3 y 4 haciendo doble clic en él.

6. El instalador de Vivado le guiará a través del proceso de instalación. Notas importantes:
- Seleccione **Vivado** (*no* Vitis) como el producto a instalar.
- Seleccione Vivado HL **Webpack** (*no* Vivado HL System Edition); la versión Webpack es gratuita.
- A menos que se indique otra cosa, se deben seleccionar los valores predeterminados.

**Sugerencia:** Si ha cambiado el directorio de instalación de Vivado, tendrá que modificar la ruta apropiadamente en los siguientes pasos.

**WINDOWS:** Los pasos 7 y 8, no son necesarios en Windows. Puede simplemente ignorar estos dos pasos e ir directamente al paso 9.

7. Después de que Vivado se haya instalado, tiene que configurar el entorno. Abra un terminal y escriba:

```
source /tools/Xilinx/Vivado2019.2/settings64.sh
```

Añada esa línea (`source /tools/Xilinx/Vivado2019.2/settings64.sh`) a su archivo `~/.bashrc` para que se ejecute cada vez que se abra una terminal.

8. Pruebe Vivado escribiendo lo siguiente en una terminal:

```
vivado
```

**Solución de problemas:**
- Si su sistema no puede encontrar ese ejecutable, tendrá que añadir lo siguiente a su ruta:

```
/tools/Xilinx/DocNav
/tools/Xilinx/Vivado/2019.2/bin
```

- Si aparece un error como "falló la inicialización específica de la aplicación..." ("application-specific initalization failed…"), escriba lo siguiente en una terminal:

```
sudo ln -s /lib/x86_64-linux-gnu/libtinfo.so.6 /lib/x86_64-
linux-gnu/libtinfo.so.5
```

9. Tendrá que **instalar manualmente los drivers del cable para la placa FPGA Nexys A7**. Escriba lo siguiente en una ventana de terminal:

```
cd
/tools/Xilinx/Vivado/2019.2/data/xicom/cable_drivers/lin64/ins
tall_script/install_drivers/

sudo ./install_drivers
```

**WINDOWS:** La instalación de Vivado en Windows instala automáticamente los drivers para la placa FPGA Nexys A7 que no son compatibles con PlatformIO. Por lo tanto, si utiliza Windows, **debe actualizar los drivers como se explica en el Apéndice B.** Debe **hacerlo aunque ya lo haya hecho en la sección Guía de Inicio Rápido porque los drivers fueron sobrescritos por la instalación de Vivado**.

10. También tendrá que instalar manualmente los archivos de la placa Digilent.
    - Descargue el archivo de las placas de vivado del repositorio de Github y extráigalo.
    - Abra la carpeta extraída del archivo y acceda al directorio *new/board_files*. Seleccione todas las carpetas dentro de este directorio y cópielas.
    - Abra la carpeta en la que se instaló Vivado (*/tools/Xilinx/Vivado* por defecto). En esta carpeta, navegue hasta el directorio *<version>/data/boards/board_files*, y luego pegue los archivos copiados en este directorio.
    - También puede usar la terminal, entrando en el directorio *new/board_files* y escribiendo:

```
sudo cp -r *
/tools/Xilinx/Vivado/2019.2/data/boards/board_files
```

**WINDOWS:** Copie y pegue las carpetas descargadas como se explica en el paso 10. En Windows, puede encontrar la carpeta *board_files* de Vivado en: *C:\Xilinx\Vivado\2019.2\data\boards\board_files*

## B. Instalar VSCode y PlatformIO

Ahora instalará VSCode y PlatformIO. **Si ya lo ha hecho en la Guía de Inicio Rápido – Sección 1 – no necesita repetir el proceso de nuevo y puede ir directamente a la Sección C**.

PlatformIO es un entorno de desarrollo integrado (IDE) para sistemas empotrados que se implementa sobre Visual Studio (VS) Code de Microsoft. Permite programar el procesador RISC-V (que se encuentra en la FPGA) usando C o ensamblador. PlatformIO es multiplataforma e incluye un depurador incorporado.

Siga estos pasos para instalar tanto VSCode como PlatformIO:

**Línea de comandos de GNU/LINUX:** aunque el uso de VSCode+PlatformIO es el método recomendado, el Apéndice A proporciona instrucciones para cualquiera que esté interesado en instalar y usar la cadena de herramientas nativa de RISC-V y OpenOCD en Linux y usarlos en lugar de PlatformIO. Si va a usar PlatformIO, simplemente ignore el Apéndice A.

## 1. Instalar VSCode:

Siga estos pasos para instalar VSCode:

a. Descargue el archivo .deb del siguiente enlace: https://code.visualstudio.com/Download

b. Abra una terminal, e instale y ejecute VSCode escribiendo lo siguiente en la terminal:

```
cd ~/Downloads
sudo dpkg -i code*.deb
code
```

**Windows / macOS:** VSCode también está disponible para Windows (archivo .exe) y macOS (archivo .zip) en https://code.visualstudio.com/Download. Siga los pasos normalmente utilizados para instalar y ejecutar una aplicación en estos sistemas operativos.

## 2. Instalar PlatformIO sobre VSCode:

Siga estos pasos para instalar PlatformIO:

a. Instale las utilidades de python3 escribiendo lo siguiente en una terminal:

```
sudo apt install -y python3-distutils python3-venv
```

**Windows / macOS:** este paso (2.a) no es necesario en Windows. En cuanto a macOS, puede usar homebrew para instalar python3, si no está instalado: `brew install python3`

b. Si aún no está abierto, abra VSCode seleccionando el botón de inicio y escribiendo "VSCode" en el menú de búsqueda, luego seleccione VSCode, o escribiendo `code` en una terminal.

c. En VSCode, haga clic en el icono de Extensiones  situado en la barra lateral izquierda de VSCode (ver Figura 30).

**Figura 30. Icono de extensiones de VSCode**

d. Escriba *PlatformIO* en el cuadro de búsqueda e instale *PlatformIO IDE* haciendo clic

en el botón de instalación que está a su lado (ver Figura 31).

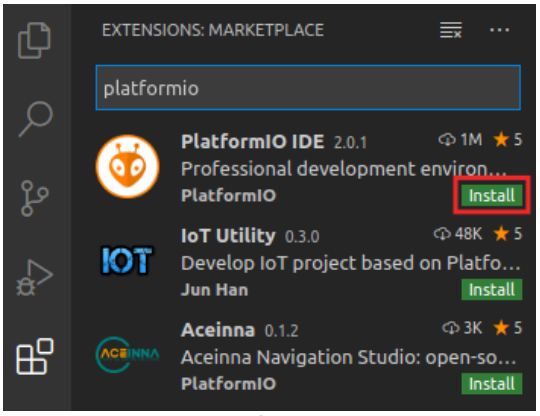



**Figura 31. Extensión de PlatformIO IDE**

e. La ventana de SALIDA en la parte inferior le informará sobre el proceso de instalación. Una vez finalizado, haga clic en "Recargar ahora" en la ventana inferior derecha, y PlatformIO se instalará dentro de VSCode (ver Figura 32).

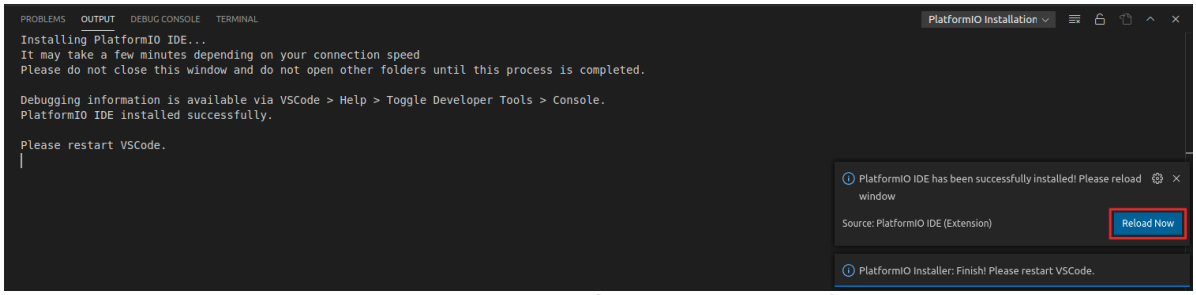



**Figura 32. Recarga ahora después de la instalación de PlatformIO**

## C. Instalar Verilator y GTKWave en Ubuntu 18.04

Las instrucciones de esta sección son válidas únicamente para sistemas GNU/Linux.

**Windows:** utilice el Apéndice C en lugar de las instrucciones que se proporcionan en esta sección.

**macOS:** utilice el Apéndice D en lugar de las instrucciones que se proporcionan en esta sección.

Siga los siguientes pasos para instalar Verilator (las instrucciones están disponibles en: https://www.veripool.org/projects/verilator/wiki/Installing pero también se resumen a continuación) y GTKWave en su sistema GNU/Linux Ubuntu 18.04. Este proceso implica una considerable cantidad de tiempo hasta su finalización.

- `sudo apt-get install git make autoconf g++ flex bison libfl2 libfl-dev`
- `sudo apt-get install -y gtkwave`
- `git clone https://git.veripool.org/git/verilator`
- `cd verilator`
- `git pull`

- `git checkout v4.106`
- `autoconf`
- `./configure`
- `make` (Alternativamente para hacerlo más rápido puede ejecutar `make -j$(nproc)`)
- `sudo make install`
- `export PATH=$PATH:/usr/local/bin` (cambiar la ruta en su sistema)

Para añadir */usr/local/bin* permanentemente a su ruta, añada la última línea a su archivo ~/. bashrc.

# 6. EJECUCIÓN Y PROGRAMACIÓN DE RVfpgaNexys

En esta sección, se muestra cómo ejecutar siete programas simples en RVfpgaNexys (ver Figura 25).

**GNU/LINUX / Windows / macOS:** Todas las instrucciones descritas en esta sección deben funcionar para los tres sistemas operativos, suponiendo que todas las herramientas y controladores necesarios se hayan instalado correctamente como se explica en la sección 5. En algunos casos, es posible que tenga que modificar algunos detalles menores, como el separador de carpetas, utilizado en GNU/Linux (/), por una barra invertida, utilizada en Windows (\).

Se hará una demostración de cómo usar RVfpgaNexys mostrando cómo ejecutar los siete programas de ejemplo listados en la Tabla 9. Los tres primeros programas están escritos en lenguaje ensamblador de RISC-V y los últimos cuatro programas están escritos en C. Las instrucciones para ejecutar cada uno de los programas en RVfpgaNexys se describen a continuación.

**Tabla 9. Programas de ejemplo de RVfpgaNexys**

| Nombre del programa | Descripción | Lenguaje de programación |
|---|---|---|
| **AL_Operations** | ejecuta operaciones aritméticas y lógicas | Ensamblador de RISC-V |
| **Blinky** | enciende intermitentemente un LED en la placa Nexys A7 | Ensamblador de RISC-V |
| **LedsSwitches** | lee los valores de los interruptores en la placa Nexys A7 y escribe ese valor en los LEDs | Ensamblador de RISC-V |
| **LedsSwitches_C-Lang** | lee los valores de los interruptores en la placa Nexys A7 y escribe ese valor en los LEDs | C |
| **HelloWorld_C-Lang** | imprime un mensaje corto en una terminal a través del puerto serie | C |
| **VectorSorting_C-Lang** | ordena de mayor a menor los valores de un vector | C |
| **DotProduct_C-Lang** | realiza el producto escalar de dos vectores | C |

Tenga en cuenta que, antes de poder ejecutar cualquiera de estos siete ejemplos, **debe programar la FPGA con RVfpgaNexys**, como se explica en la siguiente sección.

# A. Programar la FPGA con RVfpgaNexys

En esta sección, se explica el método recomendado para programar la FPGA con RVfpgaNexys, que utiliza PlatformIO. Siga los siguientes pasos para programar la FPGA con RVfpgaNexys:

(Si está interesado en usar Vivado para programar la FPGA, puede seguir las instrucciones del Apéndice E de esta guía en lugar de las que siguen a continuación. Sin embargo, el método allí descrito sólo es posible para sistemas GNU/Linux y Windows (*no* macOS) - y, en general, no se recomienda el método de usar Vivado para descargar RVfpgaNexys en la FPGA. En su lugar, se recomienda que siga las instrucciones siguientes e ignore el Apéndice E.)

a. Conecte la placa Nexys A7 a su computadora.
b. Encienda la placa Nexys A7 usando el interruptor de la parte superior izquierda.
c. Abra VSCode y PlatformIO si no lo ha hecho ya.
d. En la barra de menú superior, haga clic en *File → Open Folder* (ver Figura 33) y navegue al directorio *[RVfpgaPath]/RVfpga/examples/*

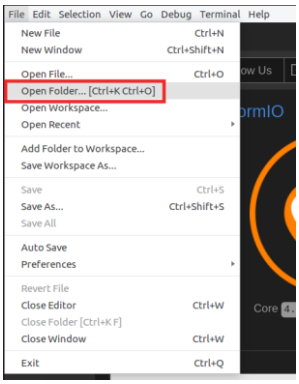



**Figura 33. Abrir carpeta**

e. Seleccione el proyecto de PlatformIO que va a utilizar. En esta sección, como ejemplo, se utilizará AL_Operations, el primer ejemplo mencionado en la Tabla 9, que se depurará en la siguiente sección, pero podría seguir los mismos pasos con cualquier otro ejemplo. Por lo tanto, seleccione el directorio *AL_Operations* (no lo abra, simplemente selecciónelo - vea la **Figura 34**) y haga clic en OK en la parte superior de la ventana. PlatformIO abrirá el ejemplo.

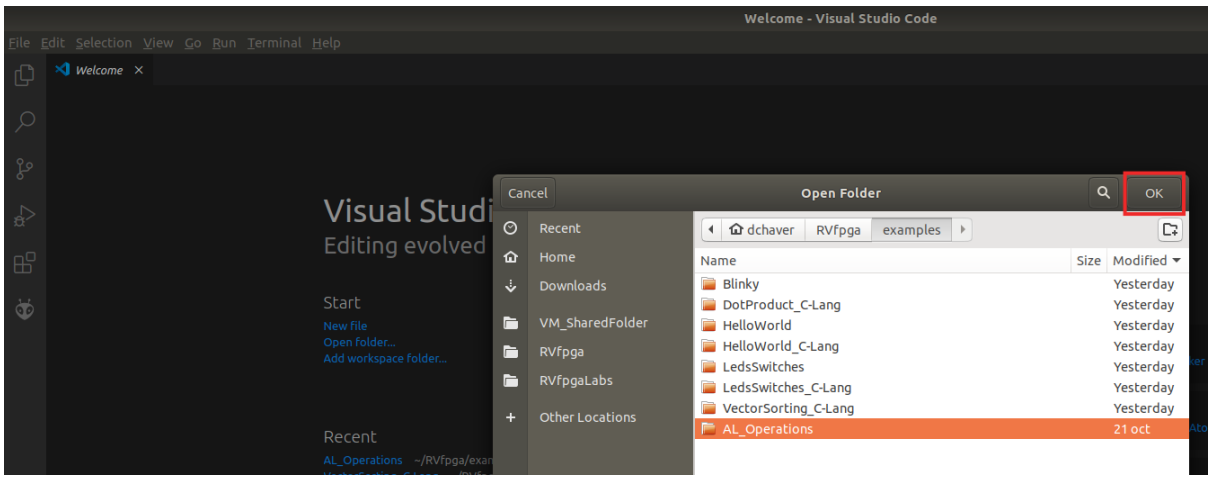



**Figura 34. Abra la carpeta AL_Operaciones**

f. Abra el archivo *platformio.ini*, haciendo clic en *platformio.ini* en la barra lateral izquierda (véase la Figura 35). Establezca la ruta del archivo bitstream de RVfpgaNexys en su sistema editando la línea que se muestra a continuación (y en la Figura 35). Tenga en cuenta que se proporciona un archivo bitstream presintetizado de RVfpgaNexys en la carpeta de RVfpga en *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*.

```
board_build.bitstream_file = [RVfpgaPath]/RVfpga/src/rvfpganexys.bit
```

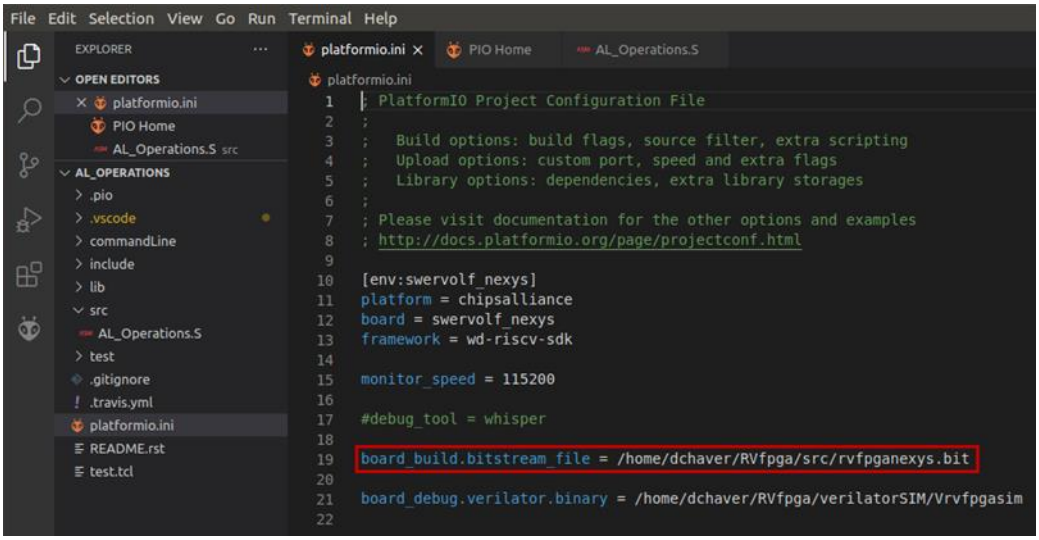



**Figura 35. Archivo de inicialización de Platformio: platformio.ini**

Hay muchos comandos diferentes que se pueden utilizar en el Archivo de Configuración del Proyecto (*platformio.ini*), y de los cuales se puede encontrar información en: https://docs.platformio.org/en/latest/projectconf/.

g. Haga clic en el icono de PlatformIO en el menú lateral izquierdo (ver Figura 36).

**Figura 36. Icono de PlatformIO**

En caso que la ventana de Tareas del Proyecto esté vacía (Figura 37), debe actualizar

primero las Tareas del Proyecto haciendo clic en  . Esto puede tomar varios minutos.

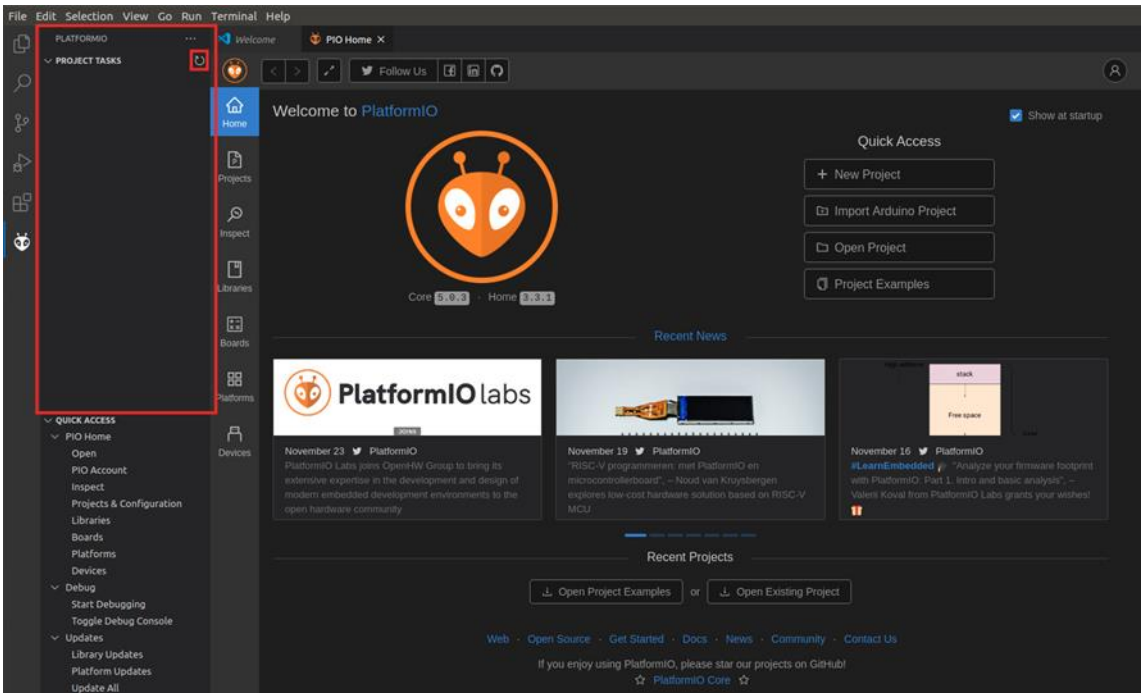



**Figura 37. Ventana de TAREAS DE PROYECTO vacía – Actualizar**

A continuación, expanda las tareas del proyecto → env:swervolf_nexys → Platform y haga clic en Upload Bitstream, como se muestra en la Figura 38. **Después de uno o dos segundos, la FPGA será programada con RVfpgaNexys.**

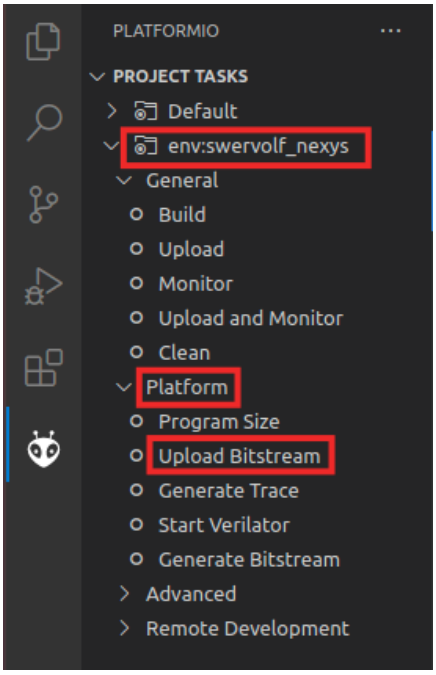



**Figura 38. Carga del Bitstream**

Por defecto, el procesador empieza a ejecutar a partir de la dirección 0x80000000, en la que la Boot ROM está mapeada en nuestro SoC (ver Tabla 6). La Boot ROM está inicializada con un programa (*boot_main.mem*) que enciende y apaga los LEDs y los

Displays de 7 Segmentos 4 veces y luego apaga los LEDs, escribe un 0 en los Displays de 7 Segmentos y se queda iterando en un bucle vacío. Este programa se encuentra disponible en el directorio: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*. Si deseas cambiarlo y recompilarlo, puedes hacerlo como se explica en el Apéndice A – Sección III (el archivo *boot_main.mem* es una copia del archivo *boot_main.vh*). En la Práctica 1 mostraremos cómo inicializar la Boot ROM con este programa al crear el *bitstream*.

h. Como alternativa al paso anterior (paso g), se puede descargar RVfpgaNexys desde una ventana de terminal de PlatformIO como se muestra en la Figura 39. Haga clic en el botón  (PlatformIO: Botón de nueva terminal) en la parte inferior de la ventana de PlatformIO para abrir una nueva terminal, y a continuación escriba (o copie) el siguiente comando en la terminal PlatformIO:

```
pio run -t program_fpga
```

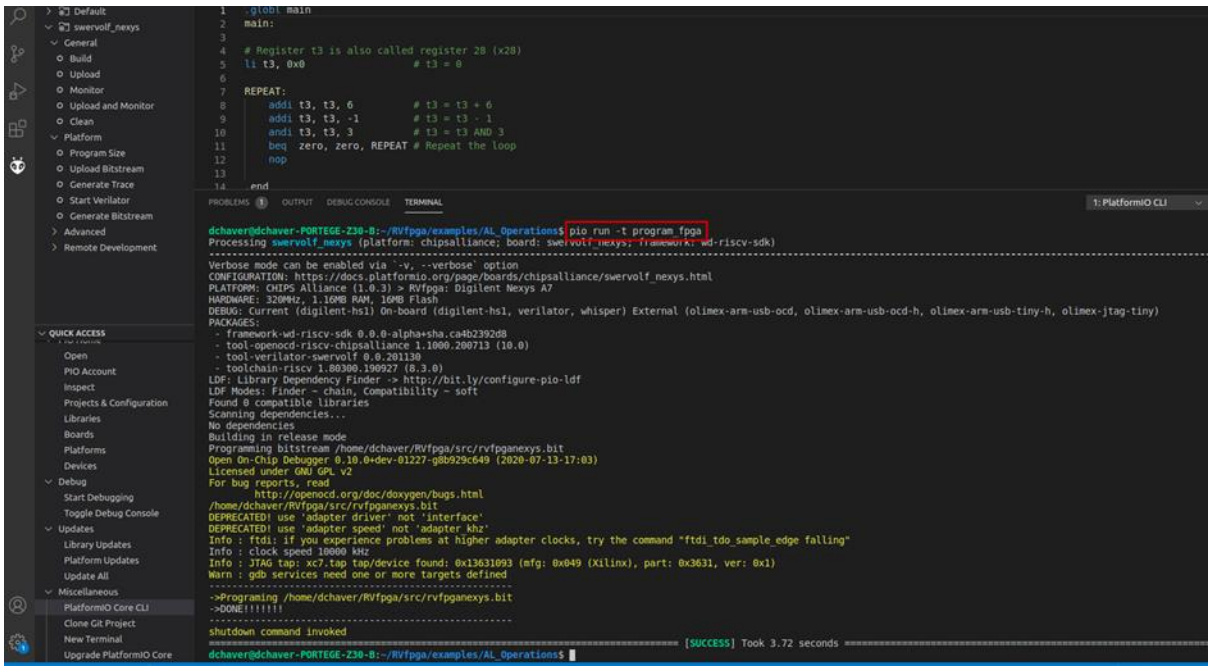



**Figura 39. Carga de RVfpgaNexys a la placa FPGA Nexys A7 usando PlatformIO**

Observe que la primera vez que se abre un ejemplo en PlatformIO, la plataforma de Chips Alliance se instala automáticamente (puede verla dentro del PIO Home, como se muestra en la Figura 40). Esta plataforma incluye varias herramientas que se usarán más adelante, como el conjunto de herramientas preconstruido de RISC-V, OpenOCD para RISC-V, un archivo bitfile de RVfpgaNexys y RVfpgaSim, scripts de JavaScript y Python, y varios ejemplos.

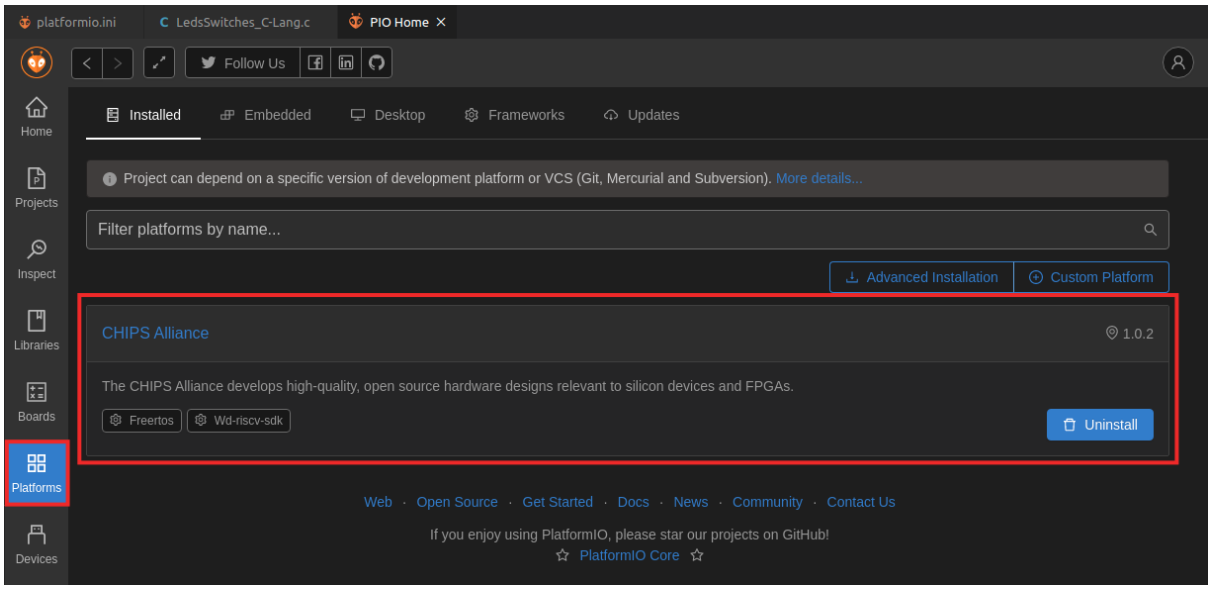
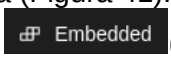
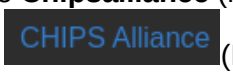



**Figura 40. La plataforma de Chips Alliance instalada en PlatformIO**

Si por alguna razón la plataforma Chips Alliance no se instaló automáticamente, puede instalarla manualmente siguiendo los siguientes pasos (normalmente, puede simplemente saltarse este procedimiento y continuar con la Sección B):

- Para ver el menú de acceso rápido, haga clic en el botón  localizado en la barra de la izquierda (Figura 41). Luego, en el PIO Home, haga clic en el botón 

y luego en la pestaña 

(Figura 41). Busque **Chipsalliance** (la plataforma que se usa en RVfpga) y ábrala haciendo clic en el botón 

(Figura 41).

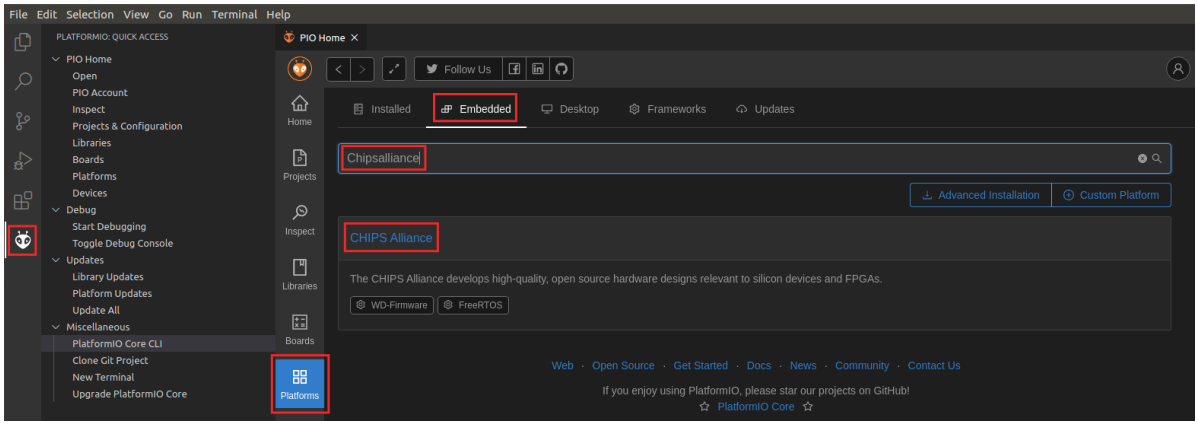



**Figura 41. Selección de la plataforma Chips Alliance**

- Después de hacer clic en el botón 

, verá los detalles de la plataforma Chips Alliance (Figura 42). Instálela haciendo clic en el botón 

(Figura 42).

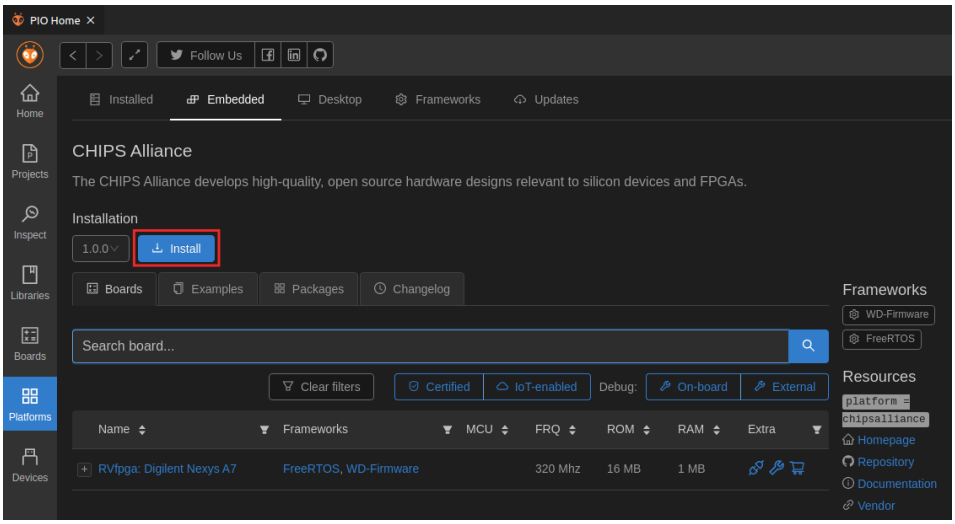



**Figura 42. Instalación de la plataforma Chips Alliance**

- Una vez completada la instalación, se muestra un resumen de las herramientas que se han instalado, (Figura 43). Haga clic en el botón  para cerrar esa ventana.

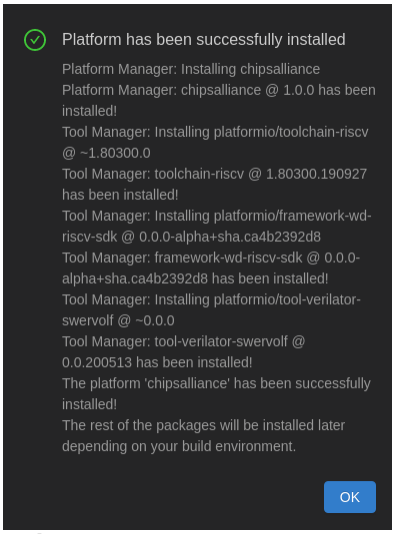




**Figura 43. Instalación correcta de la Plataforma Chips Alliance**

## B. Programa AL_Operations

El primer programa de ejemplo, AL_Operations.s (ver Figura 44), es un programa en lenguaje ensamblador que realiza tres instrucciones aritmético-lógicas (suma, resta y and lógica) sobre el mismo registro, `t3` (también llamado `x28`), dentro de un bucle infinito.

```
1  .globl main
2  main:
3
4  # Register t3 is also called register 28 (x28)
5  li t3, 0x0                    # t3 = 0
6
7  REPEAT:
8      addi t3, t3, 6            # t3 = t3 + 6
9      addi t3, t3, -1           # t3 = t3 - 1
10     andi t3, t3, 3            # t3 = t3 AND 3
11     beq  zero, zero, REPEAT   # Repeat the loop
```

**Figura 44. Programa AL__Operationss: AL_Operations.S**

Siga estos pasos para ejecutar y depurar este código en la placa FPGA Nexys A7 usando PlatformIO:

1. Programe la FPGA como se explica en la sección anterior. Es importante notar que ya tiene el proyecto *AL_Operations* abierto en PlatformIO.

2. Abra el programa en lenguaje ensamblador, AL_Operations.S, pulsando el icono del Explorador en la barra del menú de la izquierda  , expandiendo *src* bajo *AL_Operations* en la barra lateral izquierda y dando clic en AL_Operations.S (ver Figura 45).

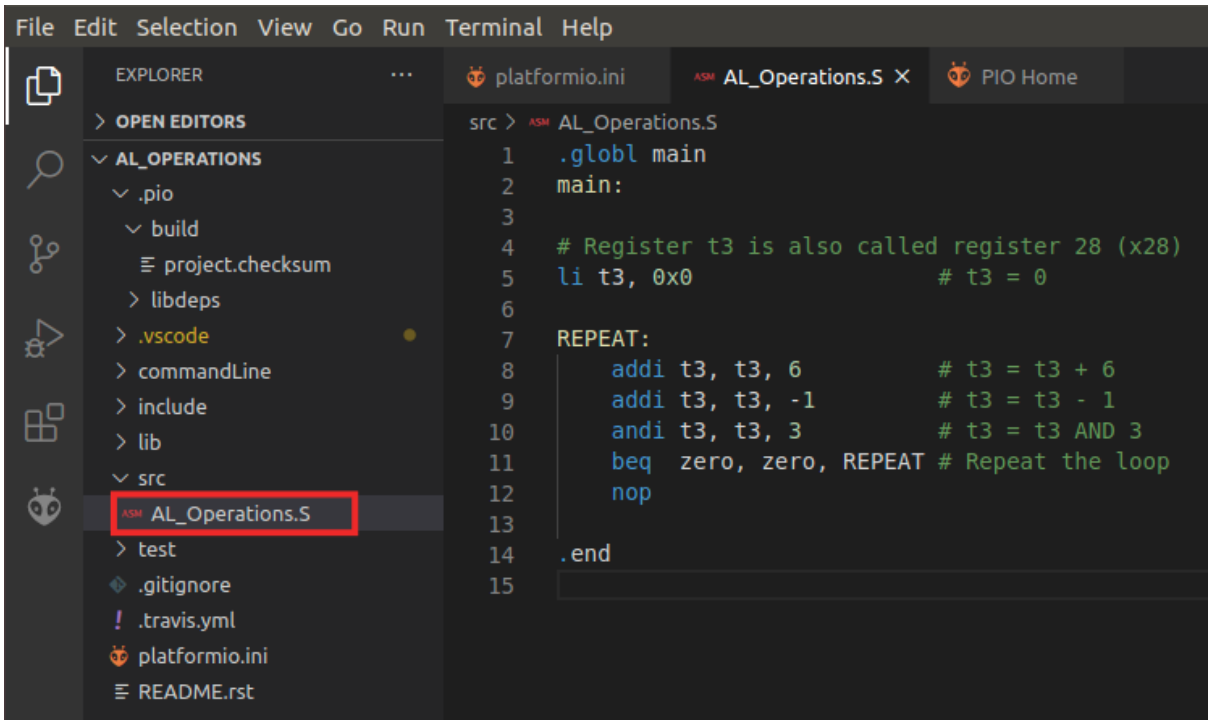
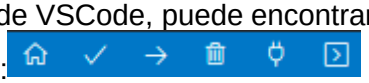
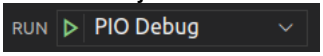



**Figura 45. Visualización del archivo de ensamblador AL_Operaciones.S**

3. VSCode y PlatformIO proporcionan diferentes formas de compilar, eliminar y depurar el programa. En la parte inferior de VSCode, puede encontrar algunos botones que proveen funcionalidades útiles:  . Por ejemplo,  se puede usar para construir el proyecto, o  se puede usar para eliminarlo. En la barra lateral izquierda (ver Figura 30), el botón "Ejecutar"  puede usarse para compilar el programa y luego abrir el depurador.

4. Haga clic en el botón "Ejecutar"  . Inicie el depurador haciendo clic en el botón de reproducción 

(compruebe que la opción "PIO Debug" esté seleccionada). Este botón se encuentra cerca de la parte superior de la ventana (véase la Figura 46). El programa primero compilará y luego comenzará la depuración. PlatformIO establece un punto de interrupción temporal al principio de la función

principal, por lo que la ejecución se detendrá allí.

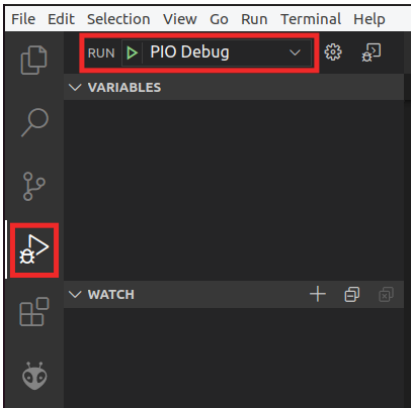



**Figura 46. Iniciar el depurador**

Para controlar la sesión de depuración, puede utilizar la barra de herramientas de depuración que aparece cerca de la parte superior del editor (véase la
5. **Figura 47**). A continuación se muestran las opciones:

- **Continue** Continúa ejecutando el programa hasta el siguiente punto de interrupción.
- **Breakpoints** Los puntos de interrupción se pueden añadir haciendo clic a la izquierda del número de línea en el editor.
- **Step Over** ejecuta la línea actual y luego se detiene.
- **Step Into** ejecuta la línea actual y si la línea actual incluye una llamada a función, saltará a esa función y se detendrá.
- **Step Out** ejecuta todo el código de la función en la que está y luego se detiene una vez que la función regresa.
- **Restart** reinicia la sesión de depuración desde el principio del programa.
- **Stop** detiene la sesión de depuración y vuelve al modo de edición normal.
- **Pause** detiene la ejecución. Cuando el programa se está ejecutando, el botón de Continue es reemplazado por el botón de Pause.

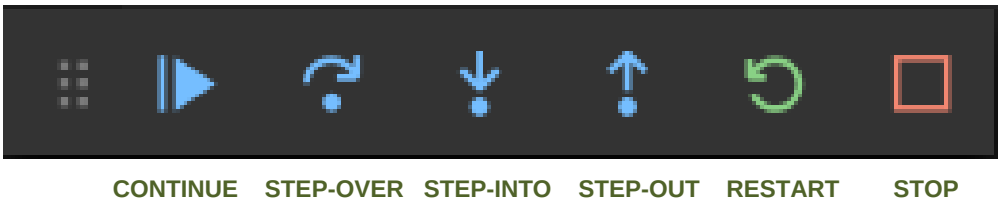



**Figura 47. Herramientas de depuración**

6. En la barra lateral izquierda, puede ver las opciones del Depurador. Las siguientes opciones están disponibles:
   - **Variables:** lista las variables locales, globales y estáticas presentes en su programa junto con sus valores.
   - **Call Stack:** muestra la función que se está ejecutando actualmente, la función que la llama (si la hay) y la ubicación de la instrucción actual en la memoria.
   - **Breakpoints:** muestra cualquier punto de interrupción establecido y resalta su número de línea. Los puntos de interrupción se pueden gestionar en esta sección. Los puntos de interrupción también pueden ser desactivados temporalmente sin

eliminarlos cambiando la casilla de verificación.

- **Peripherals**: muestra el estado de los registros de los periféricos del dispositivo mapeados en memoria (se cubrirán con más detalle en las Prácticas de Laboratorio de RVfpga).
- **Registers:** lista los valores actuales de cada uno de los registros del procesador.
- **Memory:** muestra el contenido de una dirección específica de la memoria.
- **Disassembly:** muestra el código ensamblado para una función específica – para código de alto nivel como C, esto permite ver el código ensamblado para depurar las instrucciones una por una.

7. Amplíe la opción de Registros en la barra lateral del depurador y continúe la ejecución paso a paso  . Observará que el registro `x28` (también llamado `t3`, como se muestra en la sección REGISTROS) almacena los resultados de las tres operaciones aritmético-lógicas: *suma*, resta y *and lógica*. Véase la Figura 48.

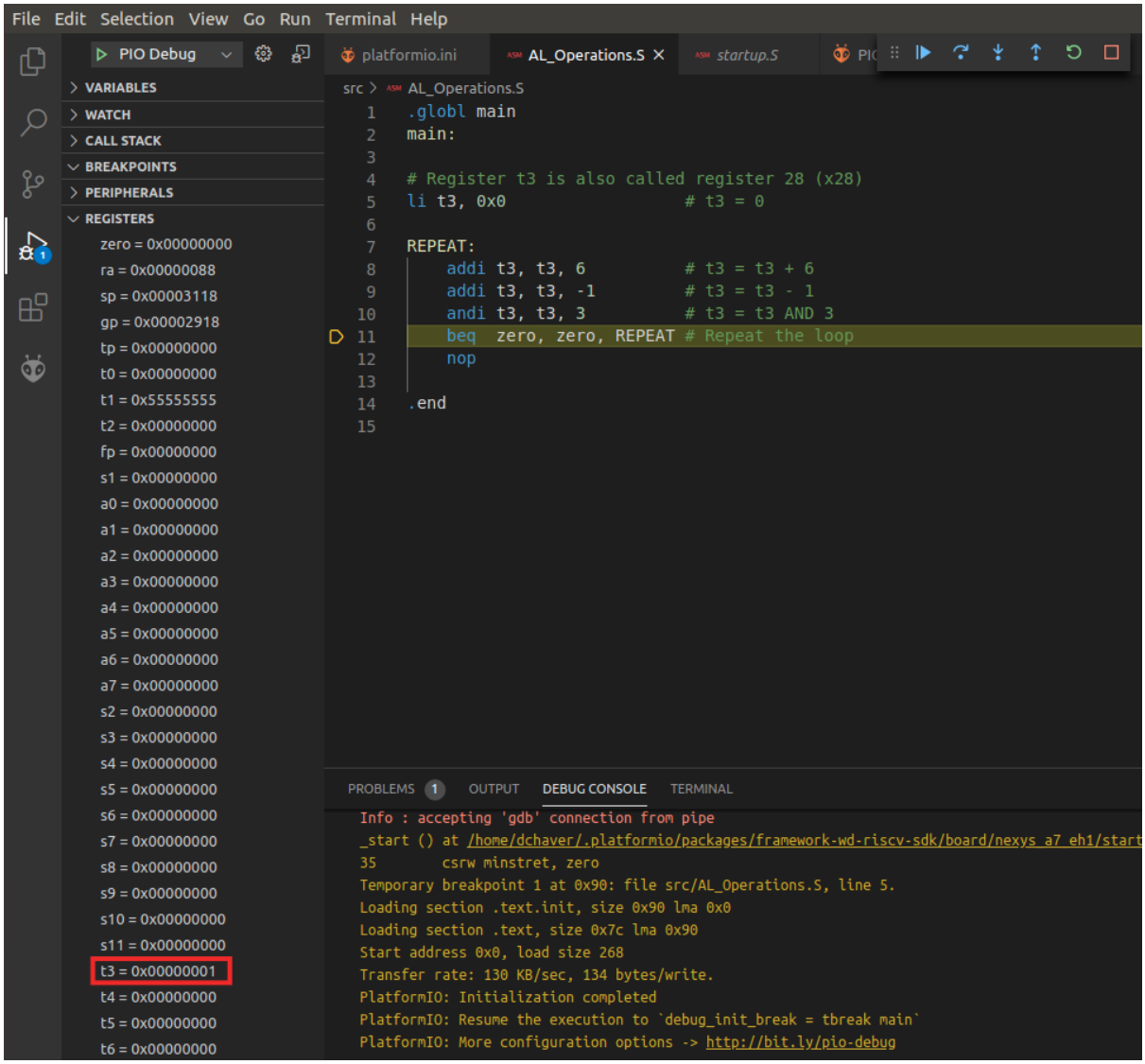



**Figura 48. Visualización del contenido de los registros**

8. Antes de llamar a la función *main*, se ejecuta un archivo de inicio, proporcionado por Western Digital en *~/.platformio/packages/framework-wd-riscv-*

*sdk/board/nexys_a7_eh1/startup.S*. Este archivo configura el core: inicialización de la caché de instrucciones, inicialización de registros (como *sp* o *gp*), etc. Cuando se inicia la depuración, este archivo se abre en la ventana principal (véase la Figura 48), y puede examinarlo allí.

**Windows:** La carpeta *.platformio* se encuentra dentro de su carpeta de usuario (C:\Usuarios\<USUARIO>). Tenga en cuenta que puede necesitar habilitar en el sistema la visualización de archivos/carpetas ocultos.

**MacOS:** Como en GNU/Linux, la carpeta *.platformio* se encuentra dentro de su carpeta home (*~/. platformio*).

9. También debemos destacar que en el mismo directorio (*~/.platformio/packages/framework-wd-riscv-sdk/board*) se proporciona el archivo *link.lds*, que constituye el script de enlazado que se utilizará en todos los proyectos. Este archivo determina la ubicación de las secciones de ensamblador (*texto*, *datos*, *bss*...) en la memoria.

10. Finalmente, se debe parar el depurador  (lo que hará que el programa de la Boot ROM se ejecute de nuevo) y volver a la ventana del Explorador haciendo clic en el botón  , que puede encontrar en la parte superior de la barra lateral izquierda. En la barra del menú superior, haga clic en *File → Close Folder*.

## C. El programa Blinky

El segundo programa de ejemplo, *blinky.S*, es un programa en lenguaje ensamblador que hace brillar intermitentemente el LED del lado derecho de la placa Nexys A7 (ver Figura 49). El programa invierte repetidamente y de modo periódico el valor mostrado en el LED que está ubicado más a la derecha.

```
1  #define GPIO_LEDs   0x80001404
2  #define GPIO_INOUT  0x80001408
3
4  #define DELAY 0x100000              /* Define the DELAY */
5
6  .globl main
7  main:
8
9         li x28, 0xFFFF
10        li a0, GPIO_INOUT
11        sw x28, 0(a0)               # Write the Enable Register
12
13        li   t1, DELAY              # Set timer value to control blink speed
14
15        li   t0, 0
16
17 bl1:
18        li   a0, GPIO_LEDs
19        sb   t0, 0(a0)              # Write to LEDs
20        xori t0, t0, 1              # invert LED
21        and  t2, zero, zero         # Reset timer
22
23 time1:                             # Delay loop
24        addi t2, t2, 1
25        bne  t1, t2, time1
26        j    bl1
```

**Figura 49. blinky.S**

Siga los siguientes pasos para ejecutar y depurar este código en RVfpgaNexys, el SoC RISC-V cargado en la placa FPGA:

1. Si ejecutó el primer ejemplo (*AL_Operations*), RVfpgaNexys ya está programado en la placa FPGA, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la sección A, usando el ejemplo de Blinky en lugar del ejemplo de AL_Operations.

2. En la barra superior, haga clic en *File* → *Open Folder*, y busque en el directorio *[RVfpgaPath]/RVfpga/examples/*

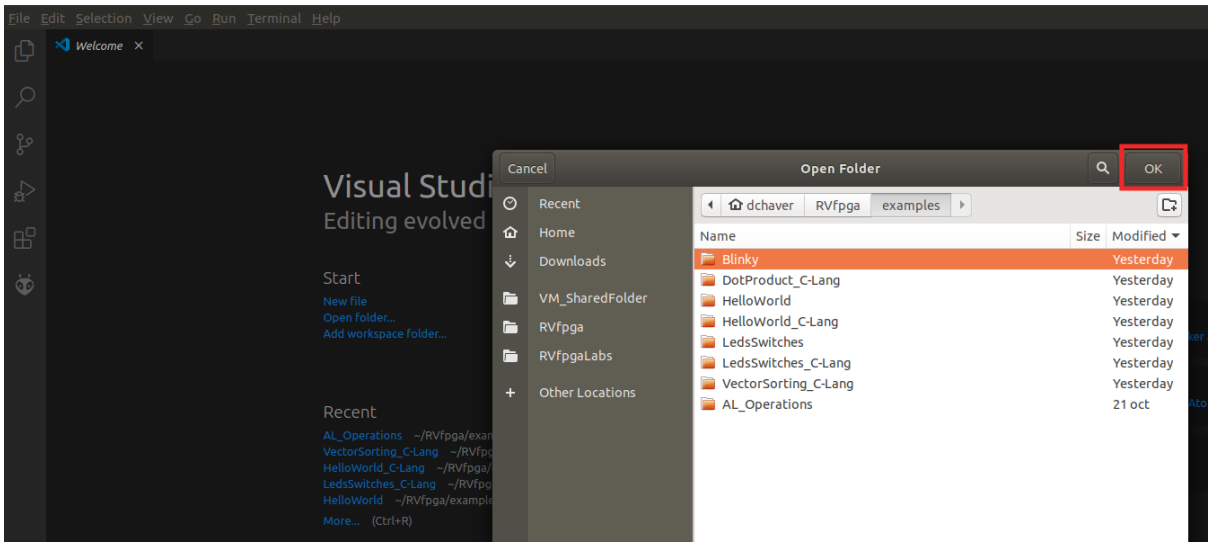



**Figura 50. Carpeta del programa Blinky**

3. Selecciona el directorio *Blinky* y haga clic en OK.

4. Abra el código en lenguaje ensamblador del ejemplo, archivo *blinky.S,* en el editor, haciendo clic en él (Figura 51).

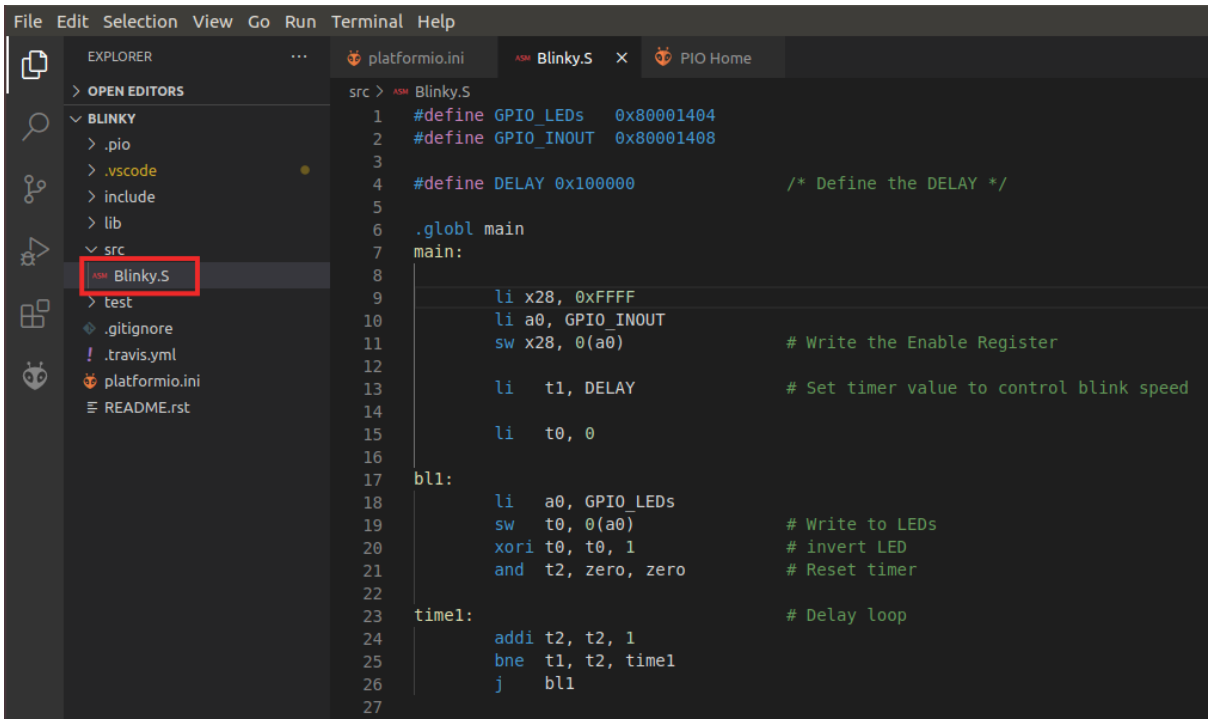
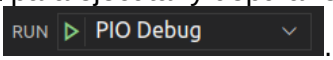



**Figura 51. blinky.S en PlatformIO**

5. Haga clic en el botón  para ejecutar y depurar el programa, luego inicie el depurador haciendo clic en el botón 

. PlatformIO establece un punto de interrupción temporal al principio de la función *main*. Por lo tanto, haga clic en el botón Continue  para ejecutar el programa.

6. En la placa verá que el LED de la derecha comienza a parpadear.

7. Detenga la ejecución haciendo clic en el botón de Pause  . La ejecución se detendrá en algún punto dentro del bucle infinito (probablemente dentro del bucle de retardo `time1`).

8. Establezca un punto de interrupción haciendo clic a la izquierda de la línea número 18. Aparecerá un punto rojo y el punto de interrupción se añadirá a la pestaña BREAKPOINTS (ver Figura 52).

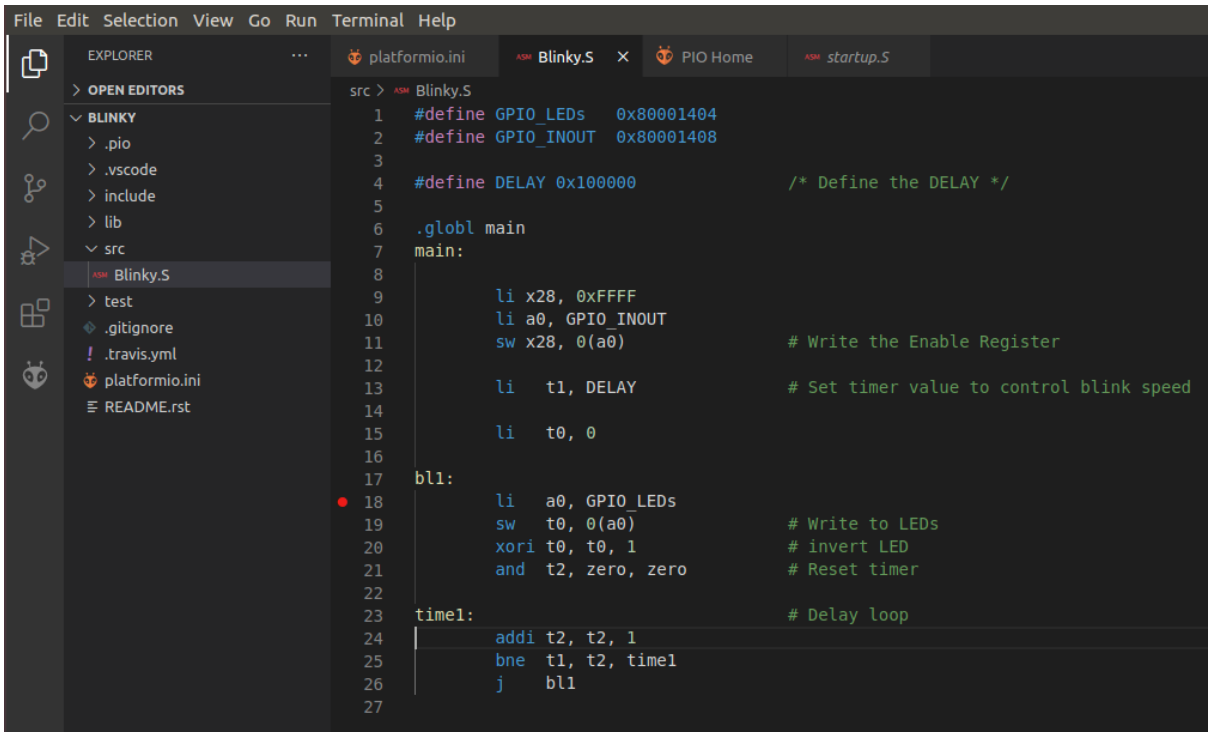



**Figura 52. Estableciendo un punto de ruptura en blinky.S**

9. Luego, continúe la ejecución haciendo clic en el botón Continue  . La ejecución continuará y se detendrá después de la instrucción store byte (`sb`), que escribe 1 (o 0) en el LED de la derecha.

10. Continúe la ejecución varias veces; verá que el valor que se establece en el LED de la derecha cambia cada vez.

11. Deje de depurar  y vuelva a la ventana del Explorador haciendo clic en  . Cierre el programa seleccionando *File → Close Folder.*

## D. El programa LedsSwitches

El tercer ejemplo en lenguaje ensamblador se comunica con los LED y los interruptores disponibles en la placa (ver Figura 53).

```
1  #define GPIO_SWs    0x80001400
2  #define GPIO_LEDs   0x80001404
3  #define GPIO_INOUT  0x80001408
4
5  .globl main
6  main:
7
8  li x28, 0xFFFF
9  li x29, GPIO_INOUT
10 sw x28, 0(x29)                  # Write the Enable Register
11
12 next:
13    li  a1, GPIO_SWs            # Read the Switches
14    lw  t0, 0(a1)
15
```

```
16    li  a0, GPIO_LEDs
17    srl t0, t0, 16
18    sw  t0, 0(a0)               # Write the LEDs
19
20    beq zero, zero, next
21 .end
```

**Figura 53. LedsSwitches.S**

Siga los siguientes pasos para ejecutar y depurar este código en la placa FPGA:

1. RVfpgaNexys ya está programado en la placa FPGA si ejecutó los ejemplos anteriores, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la Sección A, usando el ejemplo de LedsSwitches en lugar del ejemplo de AL_Operations.

2. En la barra superior, haga clic en *File → Open Folder*, y navegue hasta el directorio *[RVfpgaPath]/RVfpga/examples/.* Seleccione el directorio *LedsSwitches* y haga clic en OK.

3. El programa *LedsSwitches.S* tiene un bucle infinito donde se leen los interruptores y luego su estado se muestra en los LEDs.

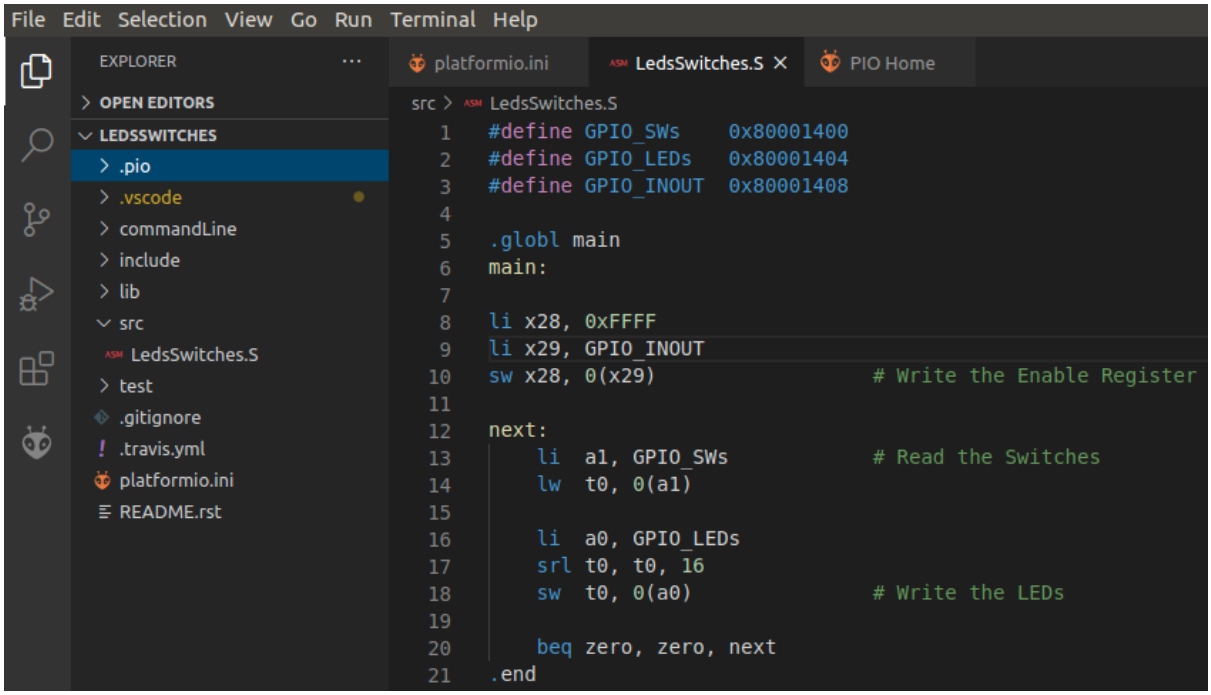



**Figura 54. LedsSwitches.S en PlatformIO**

4. Después de ejecutar el depurador como se explicó para los programas anteriores, el programa comienza su ejecución. PlatformIO establece un punto de interrupción temporal al principio de la función *main*. Por lo tanto, haga clic en el botón Continue para ejecutar el programa.

5. Conmute los interruptores en la parte inferior de la placa Nexys A7. Inmediatamente verá en la placa que los LEDs muestran el nuevo valor de los interruptores. Puede pausar la ejecución, ejecutar paso a paso e inspeccionar los registros como se ha explicado anteriormente. Cuando haya terminado, cierre el proyecto haciendo clic en *File → Close Folder.*

6. A veces, puede ser muy útil chequear los valores almacenados en la memoria. Para ello, PlatformIO proporciona un visualizador de la memoria.

    a. Pause la ejecución y ejecute paso a paso hasta el comienzo del *siguiente* bucle. Amplíe el visualizador de la memoria en la parte izquierda de la ventana (ver Figura 55) y haga clic en *Enter address...*

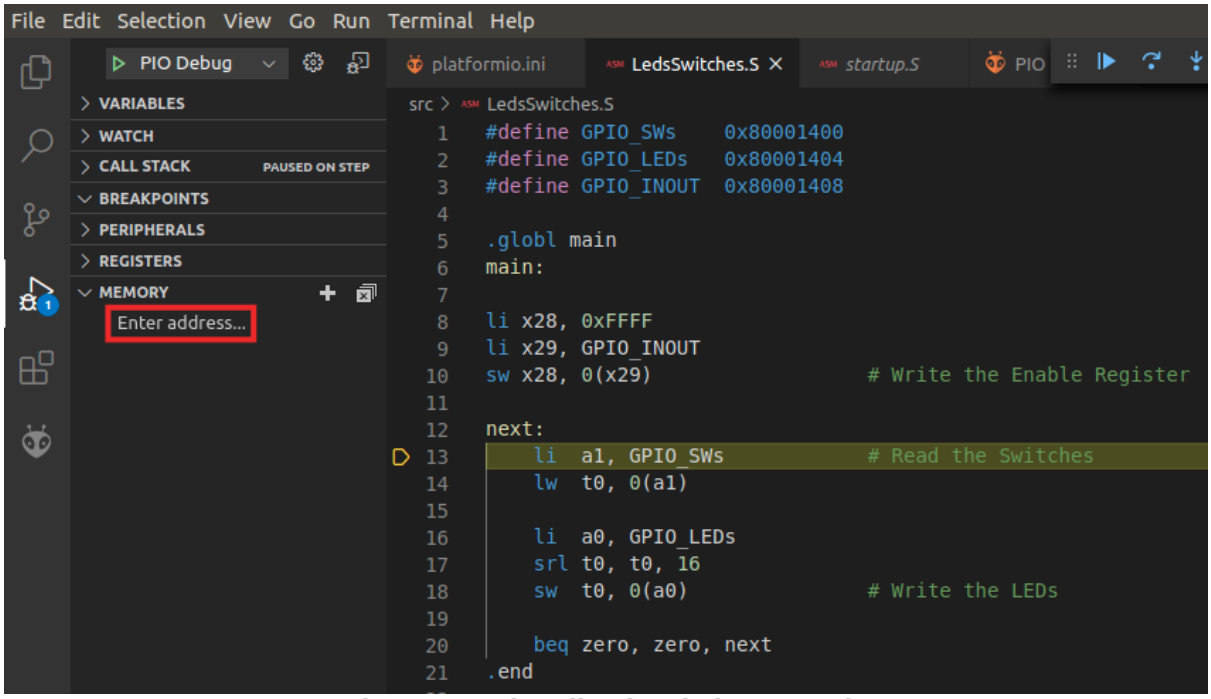



**Figura 55. Visualizador de la memoria**

    b. Se solicitará la dirección de memoria inicial (véase la Figura 56). Inserte la dirección inicial a la que los interruptores están mapeados, en este caso 0x80001400.

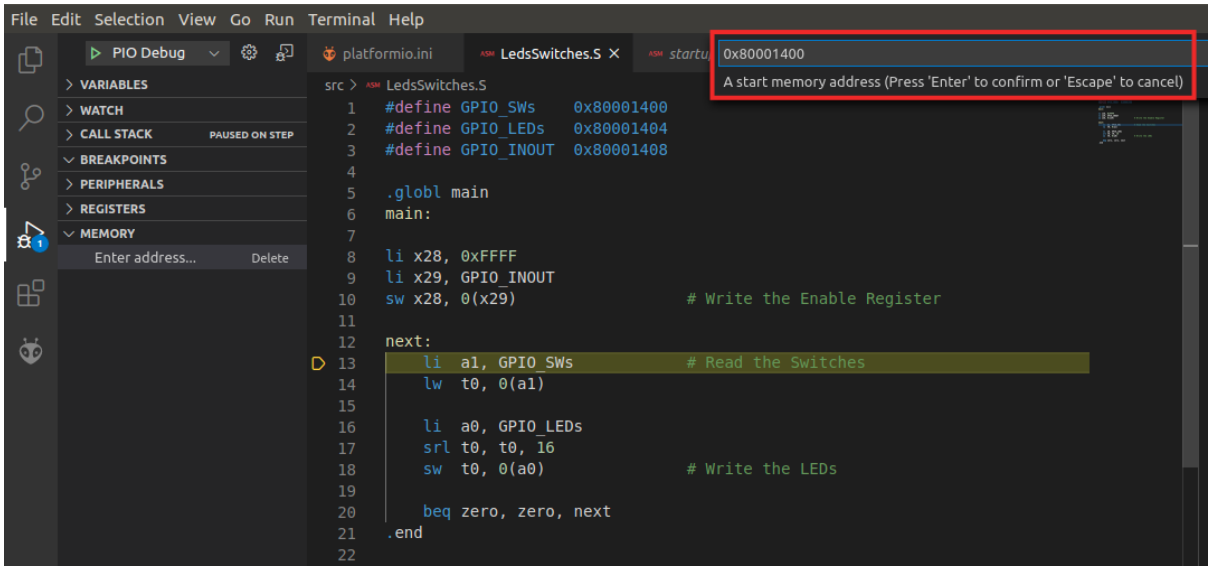



**Figura 56. Dirección de memoria inicial a mostrar**

c. A continuación se solicita el número de bytes que se desean inspeccionar (véase Figura 57), en este caso se inserta el valor de 0xc (se quiere inspeccionar tres registros de Entrada/Salida de 4 bytes, por lo que se necesitan 12 bytes).

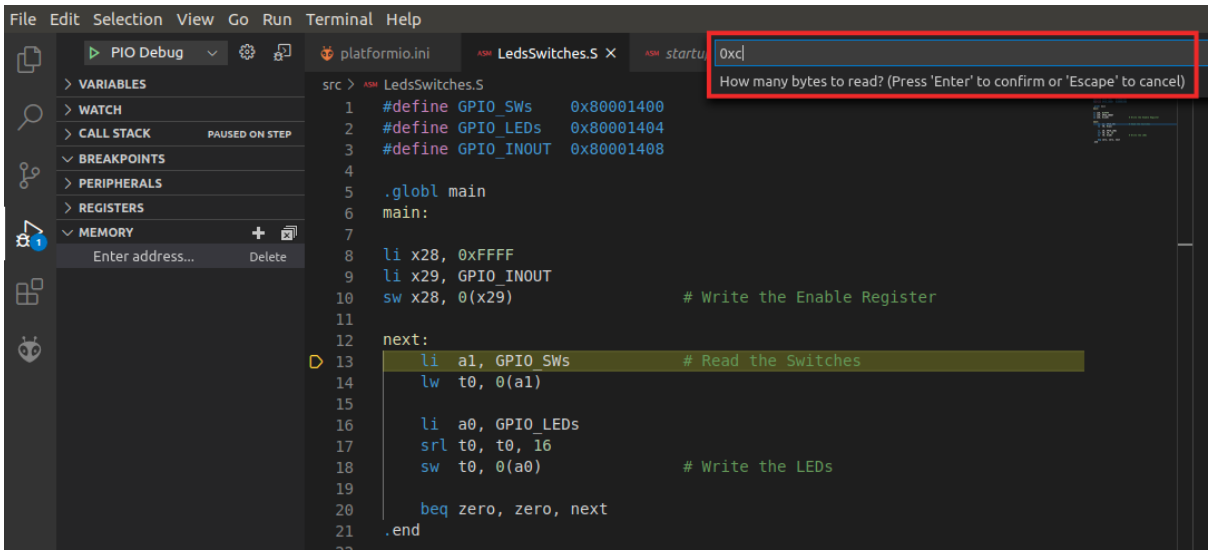



**Figura 57. Número de bytes a mostrar**

d. El visualizador de la memoria se abrirá a la derecha, mostrando los 12 bytes que se han solicitado (ver Figura 58). El valor que se tiene en los 16 interruptores es 0x123C (ver bytes en las direcciones 0x80001402 y 0x80001403). Teniendo en cuenta que la arquitectura RISC-V es little endian, el valor mostrado en la figura es coherente con eso. Los 16 LEDs (almacenados en las direcciones 0x80001404 y 0x80001405) muestran el mismo valor.

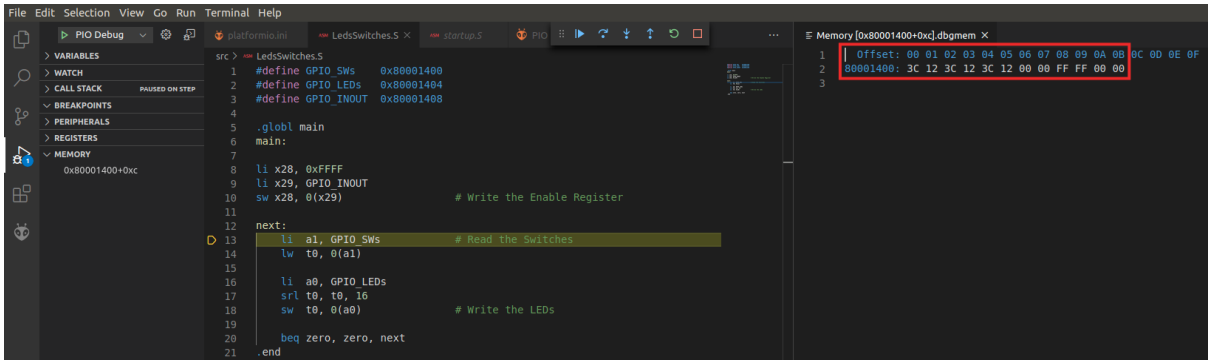



**Figura 58. Direcciones de memoria 0x80001400-0x8000140B**

e. Cambie los valores de los interruptores de la placa, por ejemplo a 0x5555, y ejecute una iteración más del bucle paso a paso. El valor de los interruptores en la memoria debe cambiar inmediatamente después de ejecutar la primera instrucción (Figura 59, arriba), y el valor de los LEDs debe cambiar en consecuencia después de ejecutar la instrucción `sw` (Figura 59, abajo).

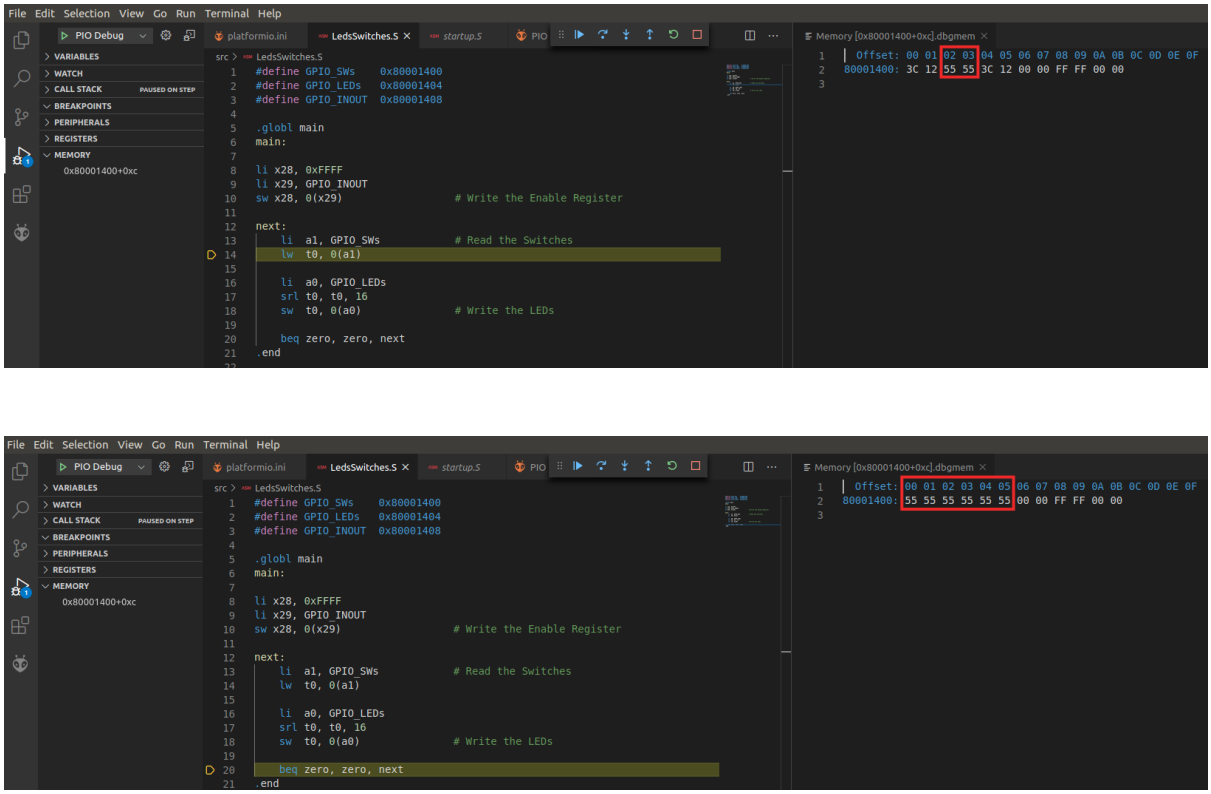






**Figura 59. Cambio en los valores de los interruptores y LEDs**

f. También puede visualizar otras posiciones de memoria, como las direcciones de RAM que almacenan las instrucciones máquina del programa. Abra otro rango de memoria empezando por 0x0 (dirección inicial asignada a la memoria RAM) y ocupando 0x100 bytes (Figura 60). Verá las instrucciones del programa LedsSwitches almacenadas en el rango de direcciones 0x90-0xC4, justo después del programa de inicio (Startup.S).

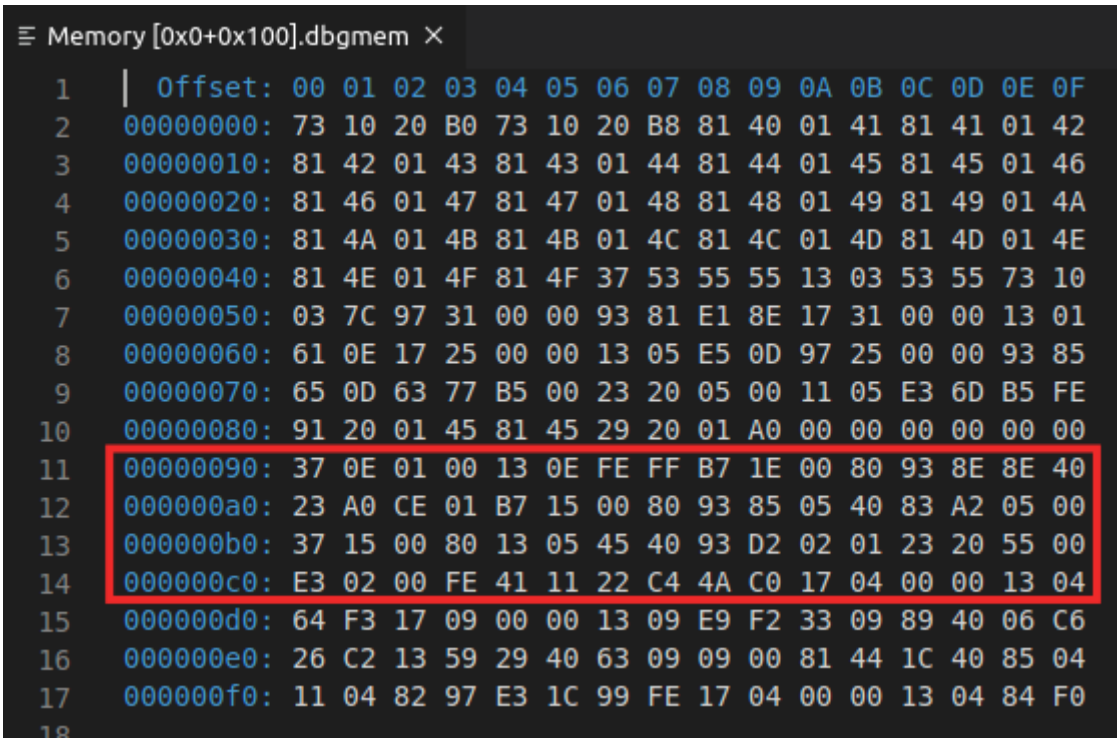



**Figura 60. Direcciones de memoria 0x0 a 0x100**

g. Puede ver el código máquina asociado a las instrucciones del programa abriendo el archivo que contiene el código desensamblado, disponible en *[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/firmware.dis* (ver Figura 61). Compare las dos figuras e intente identificar las instrucciones del programa.

```
Disassembly of section .text:

00000090 <main>:
  90: 00010e37          lui t3,0x10
  94: fffe0e13          addi  t3,t3,-1 # ffff <_sp+0xcebf>
  98: 80001eb7          lui t4,0x80001
  9c: 408e8e93          addi  t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
  a0: 01cea023          sw  t3,0(t4)

000000a4 <next>:
  a4: 800015b7          lui a1,0x80001
  a8: 40058593          addi  a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
  ac: 0005a283          lw  t0,0(a1)
  b0: 80001537          lui a0,0x80001
  b4: 40450513          addi  a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
  b8: 0102d293          srli  t0,t0,0x10
  bc: 00552023          sw  t0,0(a0)
  c0: fe0002e3          beqz  zero,a4 <next>
```

**Figura 61. Versión desensamblada del programa LedsSwitches**

## E. Programa LedsSwitches_C-Lang

El programa LedsSwitches_C-Lang.c (Figura 62) hace lo mismo que el programa LedsSwitches.s mostrado anteriormente (Figura 53) pero está escrito en C en lugar de en ensamblador.

```
1  #define GPIO_SWs    0x80001400
2  #define GPIO_LEDs   0x80001404
3  #define GPIO_INOUT  0x80001408
4
5  #define READ_GPIO(dir) (*(volatile unsigned *)dir)
6  #define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }
7
8  int main ( void )
9  {
10    int En_Value=0xFFFF, switches_value;
11
12    WRITE_GPIO(GPIO_INOUT, En_Value);
13
14    while (1) {
15        switches_value = READ_GPIO(GPIO_SWs);
16        switches_value = switches_value >> 16;
17        WRITE_GPIO(GPIO_LEDs, switches_value);
18    }
19
20    return(0);
21 }
```

**Figura 62. LedsSwitches_C-Lang.c**

Siga los siguientes pasos para ejecutar y depurar este programa en la placa FPGA:

1. RVfpgaNexys ya está programado en la placa FPGA si ejecutó los ejemplos anteriores, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la Sección A, usando el ejemplo LedsSwitches_C-Lang en lugar del ejemplo AL_Operations.

2. En la barra de menú superior, haga clic en *File → Open Folder*, y busque en el directorio *[RVfpgaPath]/RVfpga/examples/. Seleccione* el directorio *LedsSwitches_C-Lang* y haga clic en OK.

3. Antes de llamar al depurador, establezca un punto de interrupción (breakpoint) en la línea 15 del Código C.

4. A continuación, inicie el depurador. El programa comenzará a ejecutarse y se detendrá en el punto de interrupción (Figura 63).

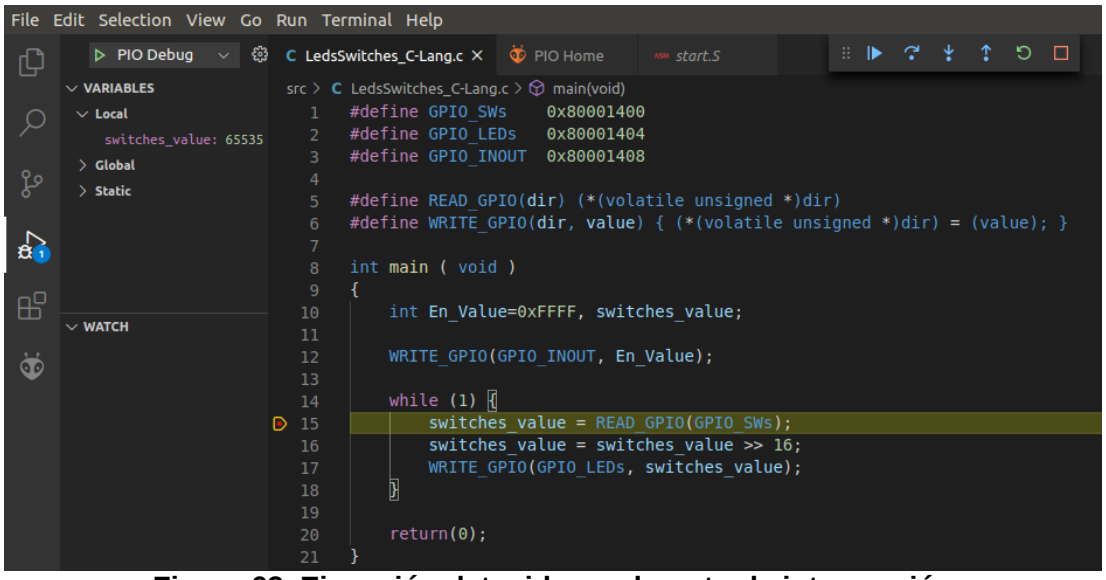



**Figura 63. Ejecución detenida en el punto de interrupción**

5. Haga que el programa continúe su ejecución  varias veces, pero cambie los interruptores entre cada clic. Los LEDs deberían mostrar el valor de los interruptores.

6. Se puede ver la ejecución del programa en C como se ha indicado previamente o se puede ver la ejecución del programa en lenguaje ensamblador generado por el compilador, haciendo clic en Switch to assembly como se muestra en la Figura 64.

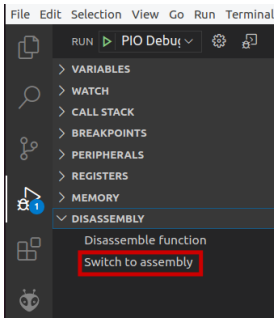



**Figura 64. Cambio a ensamblador**

7. El programa en ensamblador (Figura 65) primero lee el valor en los Interruptores con una instrucción de carga (`lw a5,1024(a4)`) y luego lo escribe en los LEDs con una instrucción de almacenamiento (`sw a5,1028(a4)`). Ejecútelo paso a paso, conmute los interruptores y verifique que los LEDs cambian para reflejar los nuevos valores de los interruptores.

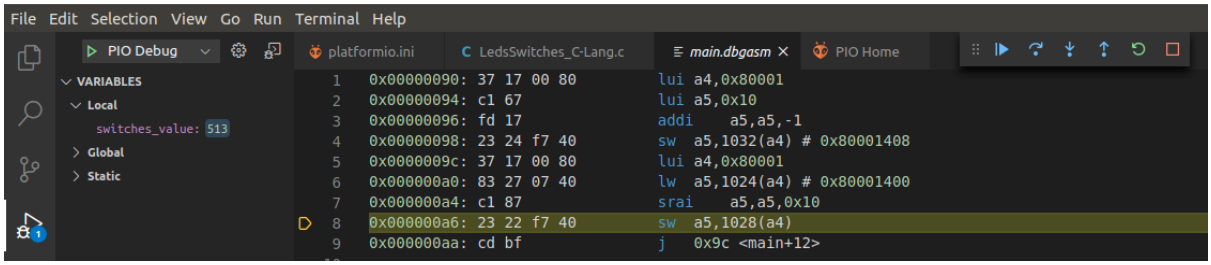



**Figura 65. Programa en lenguaje ensamblador**

# F. Programa HelloWorld_C-Lang

El segundo ejemplo en lenguaje C imprime un mensaje corto en la terminal a través del puerto serie. Para ver este mensaje, puede usar cualquier emulador de terminal como *gtkterm*, *minicom*, etc.; sin embargo, PlatformIO provee su propio monitor serie, por lo que aquí se muestra cómo usar este monitor.

Para configurar el monitor serie de PlatformIO se deben configurar algunos parámetros; específicamente, se debe establecer la velocidad de datos (en bits por segundo, o baudios) para la transmisión de datos en serie, lo cual se puede hacer utilizando el parámetro *monitor_speed* en el archivo *platformio.ini* (tenga en cuenta que este archivo forma parte de sus proyectos PlatformIO). Véase la Figura 66.

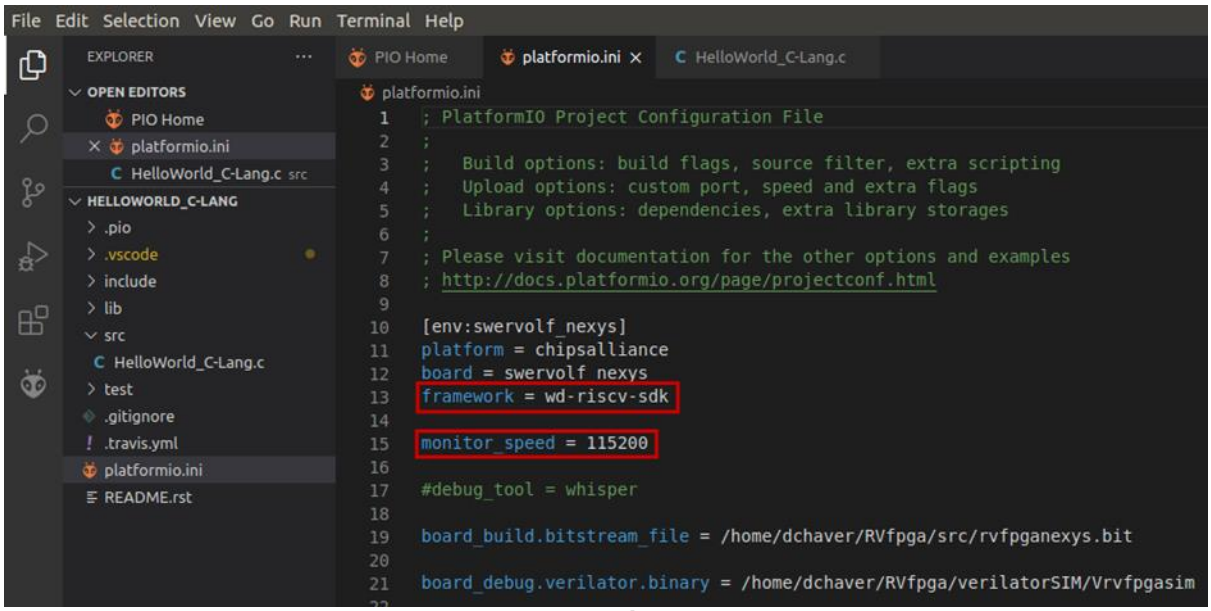



**Figura 66. Configuración del monitor serie**

Además, necesita añadir su usuario a los grupos `dialout, tty` y `uucp` escribiendo los siguientes comandos en una terminal:

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

Después de los tres comandos, reinicie la sesión para que los cambios en los grupos tengan efecto.

**Windows/macOS**: Los usuarios de Windows y MacOS no necesitan realizar el paso anterior.

Asimismo, este programa utiliza el Paquete de Soporte de Procesador (PSP por sus siglas en inglés) y el Paquete de Soporte de Tarjeta (BSP por sus siglas en inglés) proporcionados por WD dentro de su Paquete de Firmware (https://github.com/westerndigitalcorporation/riscv-fw-infrastructure). Estas librerías se incluyen en el proyecto utilizando un comando específico en platformio.ini (`framework = wd-riscv-sdk`), como se muestra en la Figura 66, e incluyendo los archivos apropiados al

principio del programa C, como se muestra en la Figura 67. Puede encontrar las librerías completas en su sistema en las siguientes rutas:

- PSP: `~/.platformio/packages/framework-wd-riscv-sdk/psp/`

- BSP: `~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/`

Estas librerías proporcionan muchas funciones y macros que permiten realizar diversas acciones como usar interrupciones, imprimir una cadena de caracteres, leer/escribir registros individuales, etc. En este ejemplo, se usa la función `printfNexys` para imprimir un mensaje en el monitor serie. En ejemplos posteriores y en las prácticas de laboratorio se mostrará cómo usar otras funciones y macros para diferentes propósitos.

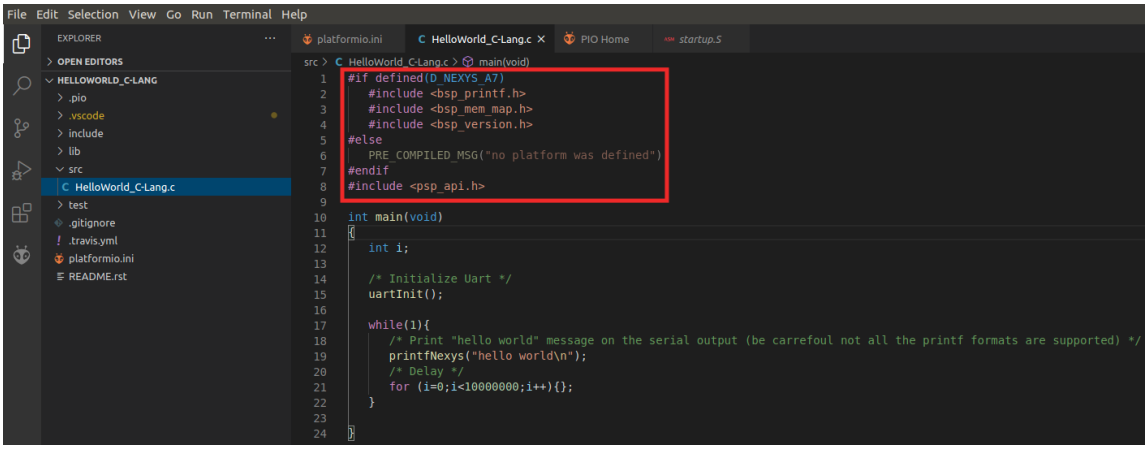



**Figura 67. Inclusión de archivos *.h* en *HelloWorld_C-Lang.c***

Siga los siguientes pasos para ejecutar y depurar este código en la placa FPGA:

1. RVfpgaNexys ya está programado en la placa FPGA si ejecutó los ejemplos anteriores, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la Sección A, usando el ejemplo de HelloWorld_C-Lang en lugar del ejemplo de AL_Operations.

2. Abra VSCode. PlatformIO debería abrirse automáticamente dentro del VSCode cuando se abre el VSCode. En la barra superior, haga clic en File → Open Folder, y navegue al directorio *[RVfpgaPath]/RVfpga/examples/*. Seleccione la carpeta *HelloWorld_C-Lang* y haga clic en OK.

3. El programa *HelloWorld_C-Lang. C* (Figura 68) inicializa la UART (función **uartInit**) y luego envía la cadena a través del puerto serie, utilizando la función **printfNexys** (puede encontrar la implementación de estas funciones en el archivo *~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/bsp_printf.c*). A continuación, se incluye un retardo antes de volver al principio del bucle.

```c
#if defined(D_NEXYS_A7)
   #include <bsp_printf.h>
   #include <bsp_mem_map.h>
   #include <bsp_version.h>
#else
   PRE_COMPILED_MSG("no platform was defined")
#endif
#include <psp_api.h>

int main(void)
{
   int i;

   /* Initialize Uart */
   uartInit();

   while(1){
      /* Print "hello world" message on the serial output (be carrefoul not all the printf formats are supported) */
      printfNexys("hello world\n");
      /* Delay */
      for (i=0;i<10000000;i++){};
   }

}
```

**Figura 68. Función main de HelloWorld_C-Lang**

4. Ejecute el depurador en PlatformIO. Cuando el programa empiece a funcionar, abra la terminal serie, haciendo clic en el botón con forma de enchufe disponible en la parte inferior de VS Code (Figura 69).

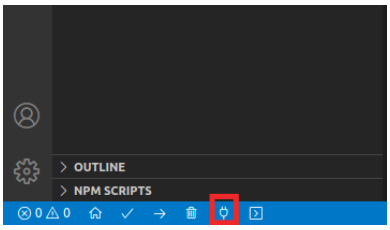



**Figura 69. Abrir la terminal serie**

5. La terminal serie imprime repetidamente el mensaje "HELLO WORLD !!!", como se muestra en la Figura 70.

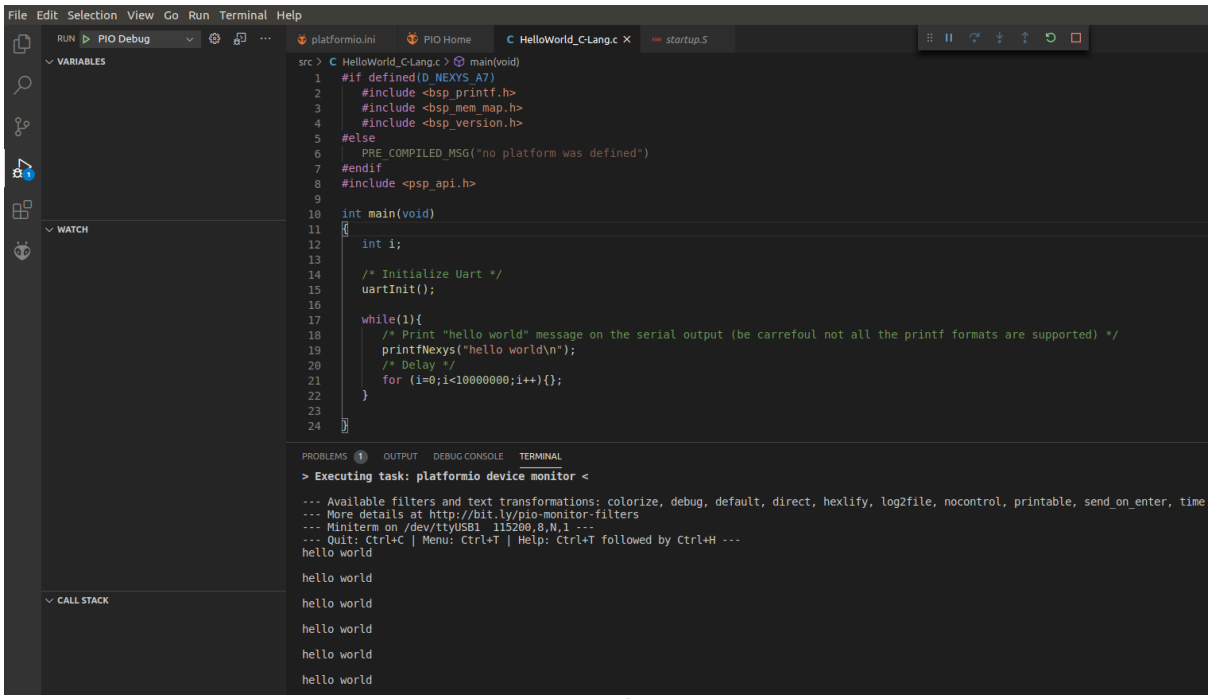



**Figura 70. Ejecución del programa**

# G. Programa VectorSorting_C-Lang

Finalmente, se muestra otro programa C que ordena los elementos de un vector, A, de mayor a menor y coloca los valores ordenados en un segundo vector, B. Los valores del vector A son reemplazados por ceros. Figura 71 muestra el programa.

```
1  #define N 8
2
3  int A[N]={7,3,25,4,75,2,1,1};
4  int B[N];
5
6  int main ( void )
7  {
8      int max, ind, i, j;
9
10     for(j=0; j<N; j++){
11         max=0;
12         for(i=0; i<N; i++){
13             if(A[i]>max){
14                 max=A[i];
15                 ind=i;
16             }
17         }
18         B[j]=A[ind];
19         A[ind]=0;
20     }
21
22     while(1);
23 }
```

**Figura 71. VectorSorting_C-Lang.c**

Siga los siguientes pasos para ejecutar y depurar este programa en la placa FPGA:

1. RVfpgaNexys ya está programado en la placa FPGA si ejecutó los ejemplos anteriores, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la Sección A, usando el ejemplo VectorSorting_C-Lang en lugar del ejemplo AL_Operations.

2. En la barra del menú superior, haga clic en *File → Open Folder*, y busque el directorio *[RVfpgaPath]/RVfpga/examples/.* Seleccione la carpeta *VectorSorting_C-Lang* y haga clic en OK.

3. Coloque un punto de interrupción en la línea 10 y comience a depurar. La ejecución se detendrá al principio del bucle `for` (Figura 72). Amplíe la sección VARIABLES en la barra lateral del depurador y analice los valores de los vectores A y B (resaltadas en rojo en la Figura 72).

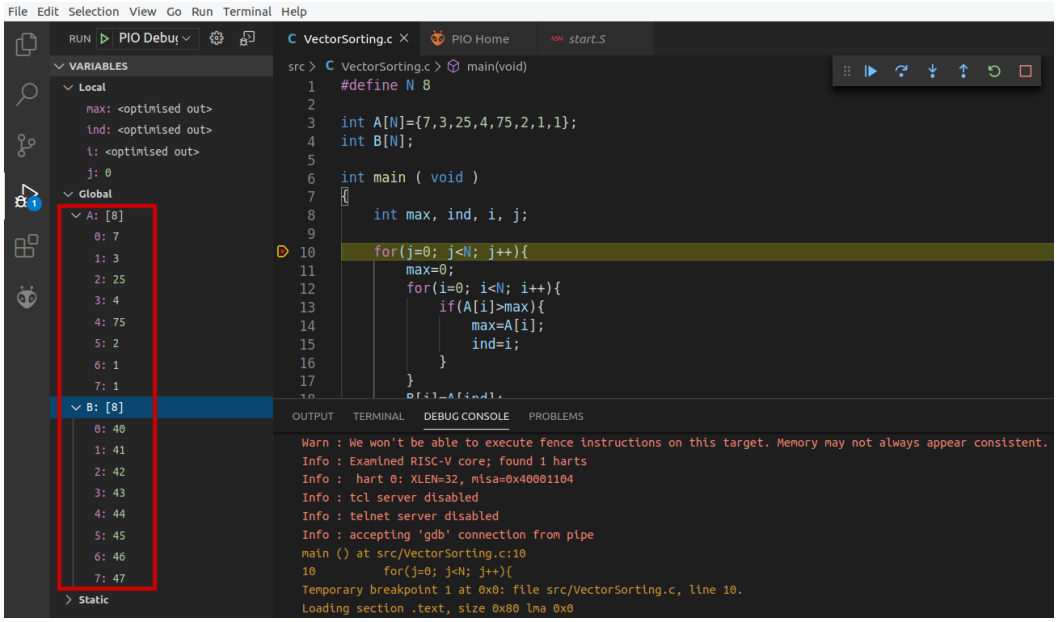



**Figura 72. Ejecución detenida al principio del programa**

4. Ahora coloque otro punto de interrupción en la línea 18 y continúe la ejecución haciendo clic en el botón  (ver Figura 73). Abra el visor de memoria (como se explicó para el programa LedsSwitches, Figura 55) y muestre 0x50 bytes a partir de la dirección 0x2148 (ver Figura 73), que es la dirección donde se almacena el vector A en la memoria para este programa. Se pueden ver los valores iniciales de los vectores A (en el rango 0x2148-0x2167) y B (en el rango 0x2178-0x2197).

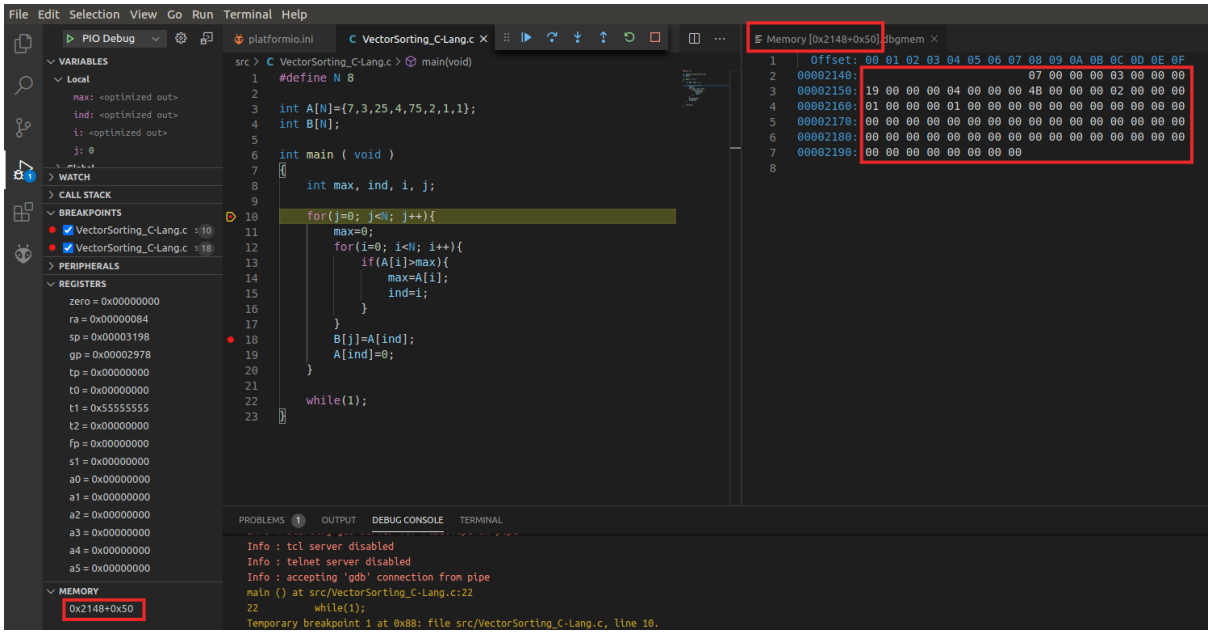



**Figura 73. Visor de memoria para los vectores A y B - Estado inicial.**

Tenga en cuenta que puede deducir fácilmente la dirección donde los vectores A y B están almacenados en la memoria cambiando a lenguaje ensamblador, como se explica en la Figura 64, y analizando cualquiera de las instrucciones que acceden a estos vectores (Figura 74). Como se ve en la figura, en la mayoría de los casos los comentarios

proporcionan esta información; sin embargo, también se puede ejecutar el programa hasta esas instrucciones y ver el valor que está almacenado en el registro.

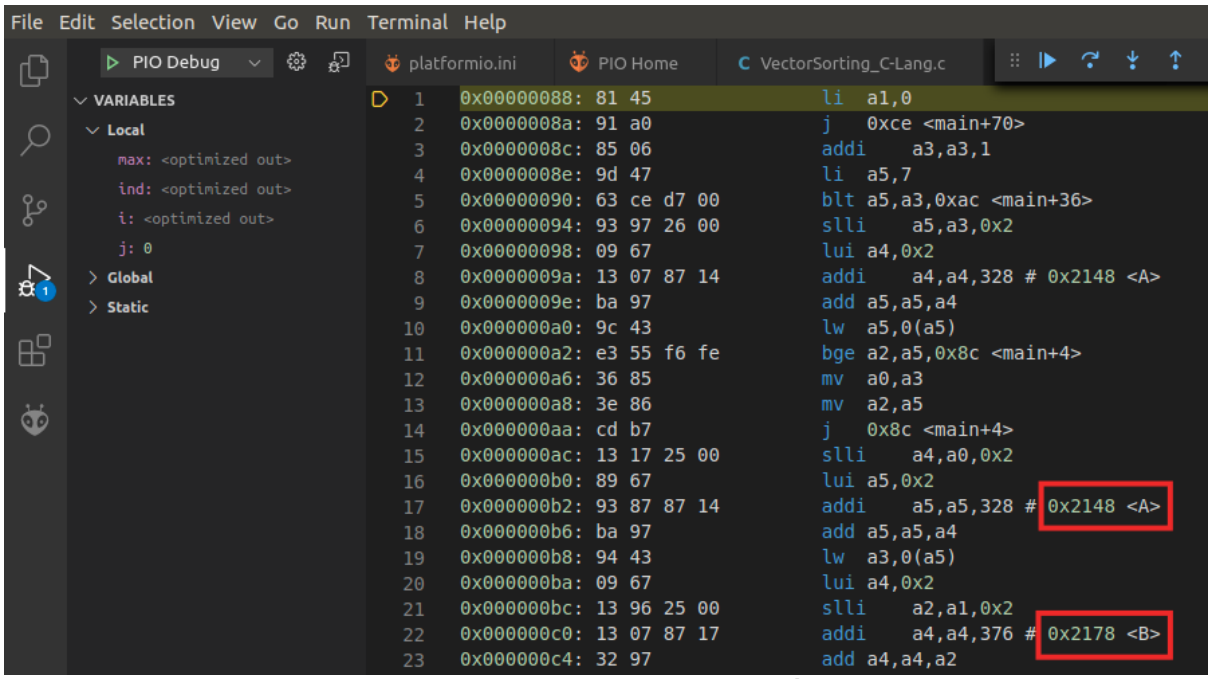



**Figura 74. Direcciones de memoria en las que están almacenados A y B..**

Haga clic dos veces en el botón Step Over (  ), y verá el primer elemento de B almacenado en la memoria y el valor correspondiente de A puesto a 0 (ver Figura 75).

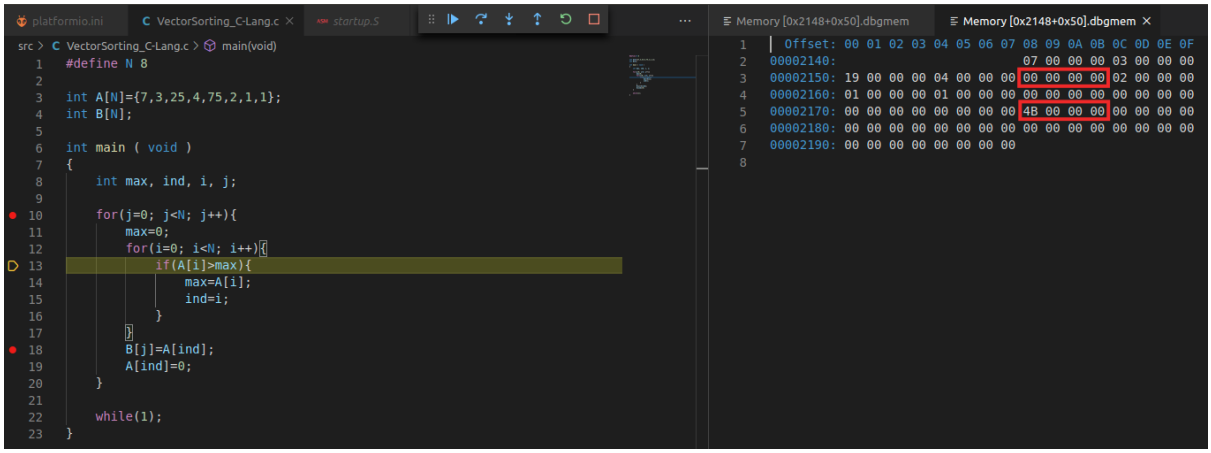



**Figura 75. Visor de memoria para los vectores A y B – Escritura del primer elemento de B y puesta a cero del elemento correspondiente en A.**

5. Elimine todos los puntos de interrupción, continúe la ejecución y deténgala después de varios segundos, con lo que el programa habrá terminado de ejecutarse. Analice, de nuevo, los valores almacenados en los vectores A y B. Como se muestra en la Figura 76, el vector B contiene los valores del vector A original ordenados de mayor a menor y el vector A contiene cero en todos los elementos (puede verlo tanto en la lista de variables de la izquierda como en el visor de memoria a la derecha).

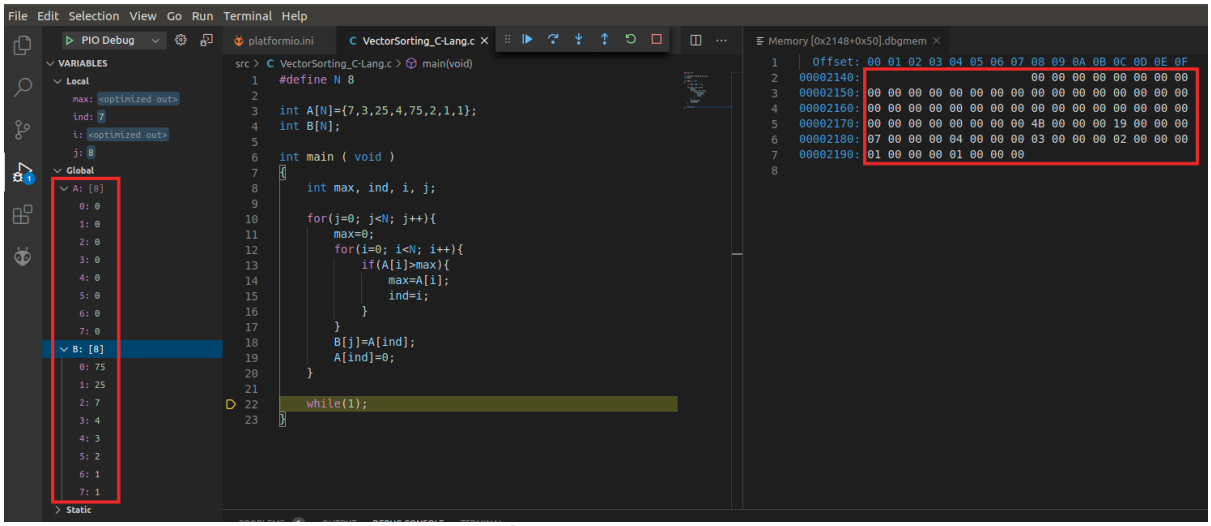



**Figura 76. Ejecución detenida al final del programa**

## H. Programa DotProduct_C-Lang

El último programa de ejemplo (Figura 77), DotProduct_C-Lang.c, calcula el producto escalar de dos vectores. El programa tiene dos funciones: *main* y *dotproduct*. La primera función invoca a la segunda con tres argumentos de entrada: el tamaño del vector y las direcciones iniciales de los dos vectores. Luego, la función *dotproduct* calcula el producto escalar de los dos vectores y devuelve el resultado.

```
1 #definir DIM 3
1  #define DIM 3
2
3  double dot;
4
5  double dotproduct(int n, double a[], double b[]){
6    volatile int i;
7    double sum=0;
8
9    for (i=0; i<n; i++) {
10         sum += a[i]*b[i];
11   }
12   return sum;
13 }
14
15 void main(void) {
16    double x[DIM] = {3.1, 4.3, 5.9};            // x is an array of size 3(DIM)
17    double y[DIM] = {1.4, 2.2, 3.7};            // same as x
18
19    dot = dotproduct(DIM, x, y);
20
21    return;
22 }
```

**Figura 77. DotProduct_C-Lang.c**

En este ejemplo se trabaja con números reales (nótese que el tipo de datos para las variables `x`, `y` y `dot`, es `double`). Sin embargo, el procesador SweRV EH1 no incluye soporte de punto flotante. Por lo tanto, el ejemplo utiliza la emulación de punto flotante a través de la librería de punto flotante por software proporcionada por *gcc* (https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html). Esta librería se utiliza

siempre que se incluye `-msoft-float` para desactivar la generación de instrucciones de punto flotante.

Siga los siguientes pasos para ejecutar y depurar este código en la placa FPGA:

1. RVfpgaNexys ya está programado en la placa FPGA si ejecutó los ejemplos anteriores, así que no debería necesitar programarlo de nuevo. Sin embargo, si necesita reprogramar RVfpgaNexys en la placa de nuevo, hágalo como se explica en la Sección A, usando el ejemplo DotProduct_C-Lang en lugar del ejemplo AL_Operations.

2. En la barra del menú superior, haga clic en *File → Open Folder*, y busque el directorio *[RVfpgaPath]/RVfpga/examples/. Seleccione* el directorio *DotProduct_C-Lang* y haga clic en OK.

3. Antes de llamar al depurador, establezca un punto de interrupción en la línea 10 y otro en la línea 19 (véase la Figura 78).

4. A continuación, empiece a depurar. El programa comenzará a ejecutarsedeteniéndose en el primer punto de interrupción (ver Figura 78).

5. En la barra lateral del Depurador, amplíe la sección de Variables (ver Figura 78). Los dos vectores contienen los valores iniciales asignados en *main*. La variable *dot se inicializa* a 0.

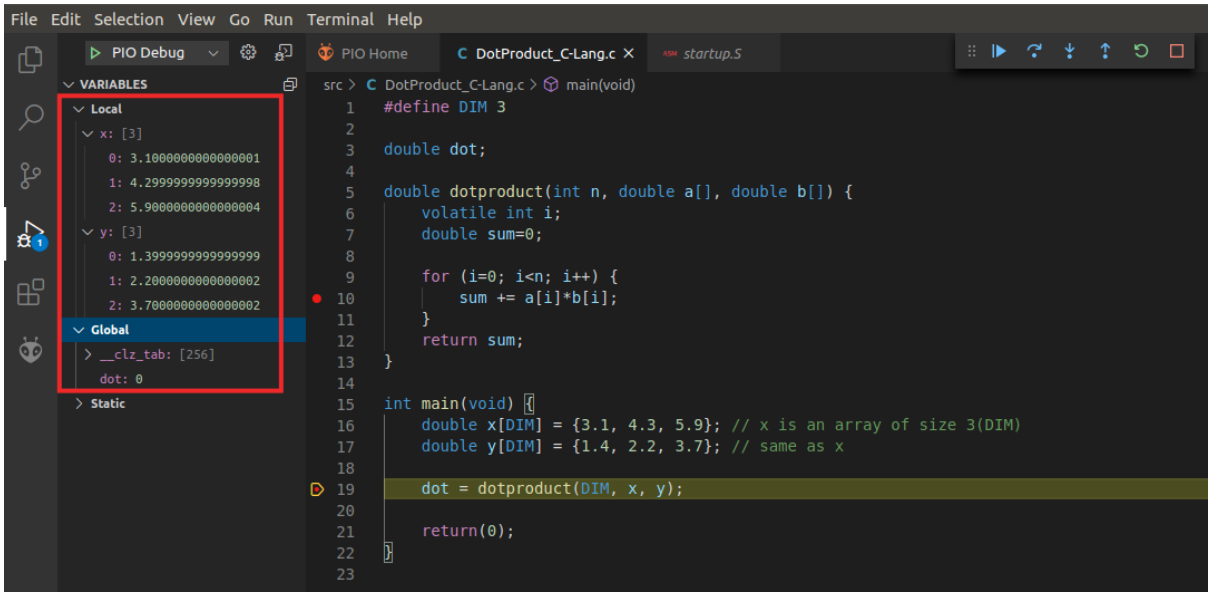
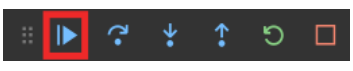



**Figura 78. Programa DotProduct_C-Lang: valores de las variables en el primer punto de interrupción**

6. Haga que el programa continúe su ejecución  . El programa se detiene en el segundo punto de interrupción (línea 10).

7. Cambie a lenguaje ensamblador (como se hizo en la Figura 64). Puede ver las rutinas de emulación de punto flotante y analizarlas en detalle entrando en ellas (ver Figura 79).

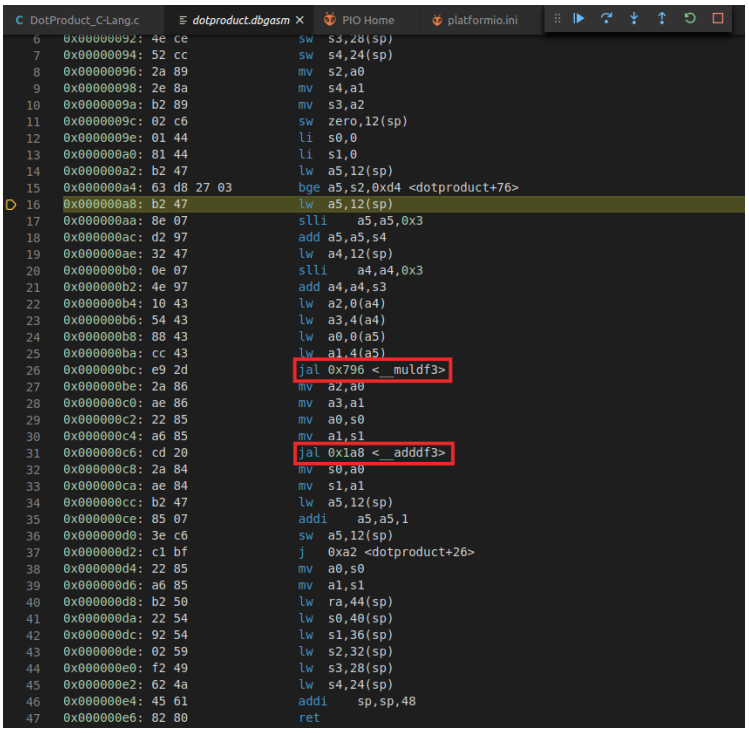



**Figura 79. Programa DotProduct_C-Lang: código en lenguaje ensamblador en el segundo punto de interrupción**

8. Vuelva a C y elimine los dos puntos de interrupción. Continúe la ejecución y póngala en pausa. Verá que el valor de la variable *dot* cambiará al producto escalar de los dos vectores (Figura 80).

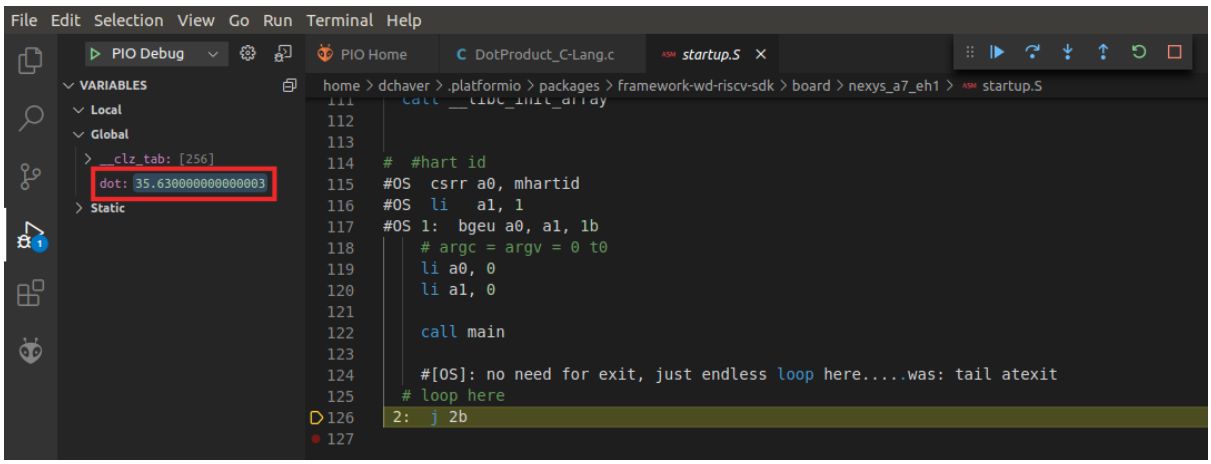



**Figura 80. Programa DotProduct_C-Lang: resultado del producto escalar**

9. Una vez que haya terminado de explorar este programa, cierre el proyecto haciendo clic en *File → Close Folder.*

# 7. SIMULACIÓN EN VERILATOR

En esta sección, ejecutará el primer programa utilizado en la sección anterior (*AL_Operations*) en RVfpgaSim usando Verilator. Verilator es un simulador de lenguaje de descripción de hardware (HDL) que simula el código Verilog que define el SoC (disponible en *[RVfpgaPath]/RVfpga/src*). Esta forma de ejecutar el SoC permite analizar las señales internas del sistema, lo que es especialmente útil para futuras prácticas de laboratorio y ejercicios en los que se añadirán operaciones internas o hardware nuevo al SoC.

Se mostrará cómo usar Verilator para ver las instrucciones y los valores de los registros ciclo a ciclo del programa *AL_Operations*, el primer programa en lenguaje ensamblador que se ejecutó y depuró en la Sección 6 (Figura 44). El usuario generará la traza de la simulación usando PlatformIO y luego agregará las señales de reloj, de las instrucciones para ambas vías del procesador superescalar y del registro `x28` (es decir, el registro `t3`) a las formas de onda de la simulación, y verán con GTKWave los cambios en las señales de las instrucciones y registros a medida que el programa se ejecuta.

## GENERAR EL ARCHIVO BINARIO DE LA SIMULACIÓN, Vrvfpgasim:

El directorio *[RVfpgaPath]/RVfpga/verilatorSIM* contiene el archivo *Makefile* y el *script* (*swervolf_0.7.vc*) para generar el archivo binario del simulador para RVfpgaSim. El *script* contiene información para que Verilator sepa, entre otras cosas, dónde encontrar los archivos fuente del SoC, que en este caso están disponibles en *[RVfpgaPath]/RVfpga/src*. A continuación se muestra cómo se puede generar el binario para RVfpgaSim, que más tarde se utilizará para crear la traza de simulación del programa *AL-Operations* que se ejecuta en RVfpgaSim.

1. En una terminal, genere el binario del simulador ejecutando los siguientes comandos:

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make
```

   El archivo ***Vrvfpgasim*** (el archivo binario de simulación RVfpgaSim), debe ser generado dentro del directorio *[RVfpgaPath]/RVfpga/verilatorSIM*.

**Windows:** si utiliza Windows, debe realizar estos mismos pasos dentro de la terminal Cygwin (consulte el Apéndice C para ver las instrucciones detalladas). Tenga en cuenta que la carpeta *C:\Windows* se puede encontrar dentro de Cygwin en: */cygdrive/c*. Todas las demás instrucciones de esta sección son las mismas que las descritas para GNU/Linux.

**MacOS:** Consulte el Apéndice D para las instrucciones detalladas.

## GENERAR LA TRAZA DE SIMULACIÓN DESDE PLATFORMIO, UTILIZANDO Vrvfpgasim:

Una vez generado el binario para el simulador (*Vrvfpgasim*), se utilizará dentro de PlatformIO para generar la traza de simulación (trace.*vcd*) del programa AL_Operations.

2. Abra VSCode y luego PlatformIO en su computadora.

3. En la barra superior, haga clic en *File→Open Folder...* (Figura 81), y navegue al

directorio *[RVfpgaPath]/RVfpga/examples/*

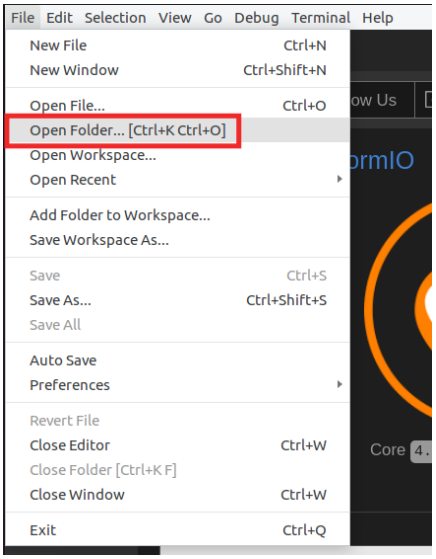



**Figura 81. Abra el ejemplo AL_Operations.S**

4. Seleccione el directorio *AL_Operations* (no lo abra, sólo selecciónelo) y haga clic en OK. El ejemplo se abrirá en PlatformIO.

5. Abra el archivo *platformio.ini*. Establezca la ruta hacia el archivo binario de simulación RVfpgaSim generado en el primer paso (*Vrvfpgasim*) editando la siguiente línea (ver Figura 82).

```
board_debug.verilator.binary =
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```

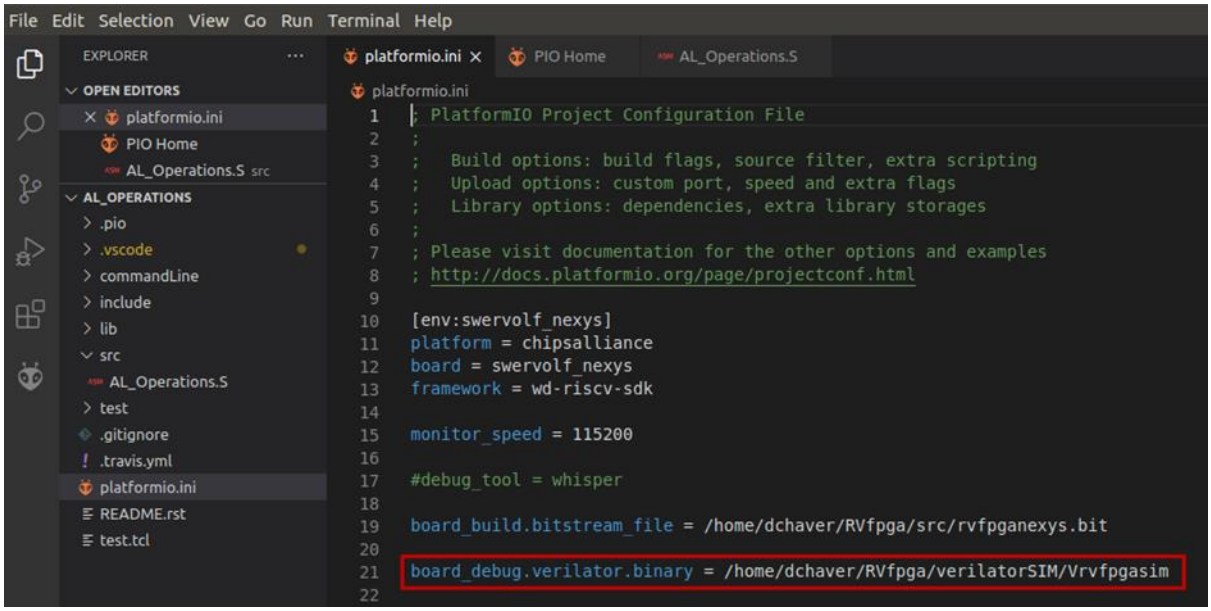



**Figura 82. Archivo de inicialización de PlatformIO: platformio.ini**

**Windows:** en Windows, el ejecutable de simulación RVfpgaSim se llama `Vrvfpgasim.exe`. Por lo tanto:

```
board_debug.verilator.binary = [RVfpgaPath]\RVfpga\verilatorSIM\Vrvfpgasim.exe
```

6. Ejecute la simulación haciendo clic en el icono de PlatformIO en la barra del menú de la izquierda  , luego expanda las Project Tasks → env:swervolf_nexys → Platform y haga clic en Generate Trace, como se muestra en la Figura 83.

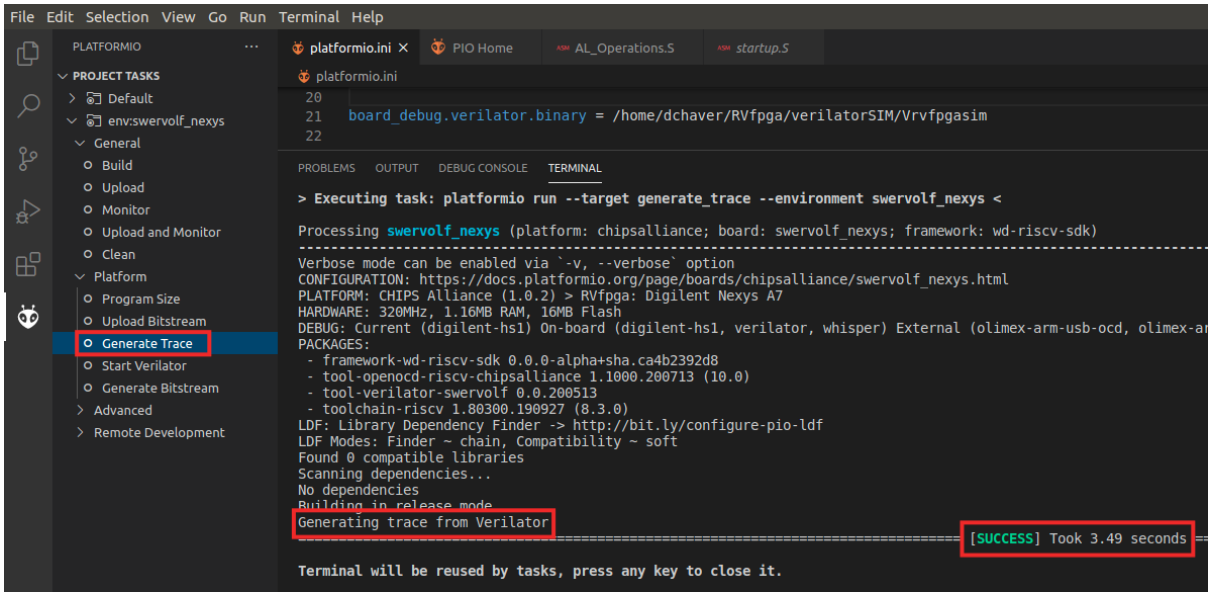



**Figura 83. Generando traza de Verilator**

Como alternativa, puede generar la traza desde la terminal de PlatformIO. Para ello, haga clic en el botón  (PlatformIO: New Terminal) en la parte inferior de la ventana de PlatformIO para abrir una nueva terminal, y luego escriba (o copie) el siguiente comando en el terminal de PlatformIO: `pio run --target generate_trace`

7. Unos segundos después del paso anterior, el archivo trace.*vcd* debería haberse generado dentro de *[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys*, y se puede abrir con *GTKWave*.

```
gtkwave [RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys/trace.vcd
```

**WINDOWS:** la carpeta *gtkwave64* que descargó, incluye una aplicación llamada *gtkwave.exe* dentro de la carpeta *bin*. Ejecute GTKWave haciendo doble clic en esa aplicación. En la parte superior de la aplicación, haga clic en **File - Open New Tab**, y abra el archivo trace.vcd generado en la carpeta *[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys*.

## ANALIZAR LA TRAZA DE LA SIMULACIÓN EN GTKWAVE:

8. Ahora añadirá las señales de reloj, de las instrucciones y de los registros. En el panel superior izquierdo de *GTKWave*, expanda la jerarquía del SoC para que pueda añadir señales al gráfico. Expanda la jerarquía a **TOP → rvfpgasim → swervolf → swerv_eh1**

→ **swerv,** y haga clic en el módulo **ifu** (se resaltará como se muestra en la Figura 84), seleccione la señal *clk* (que es el reloj usado para el core) y arrástrela al panel blanco de **señales** (Signals) o al panel negro de **ondas** (Waves) a la derecha.

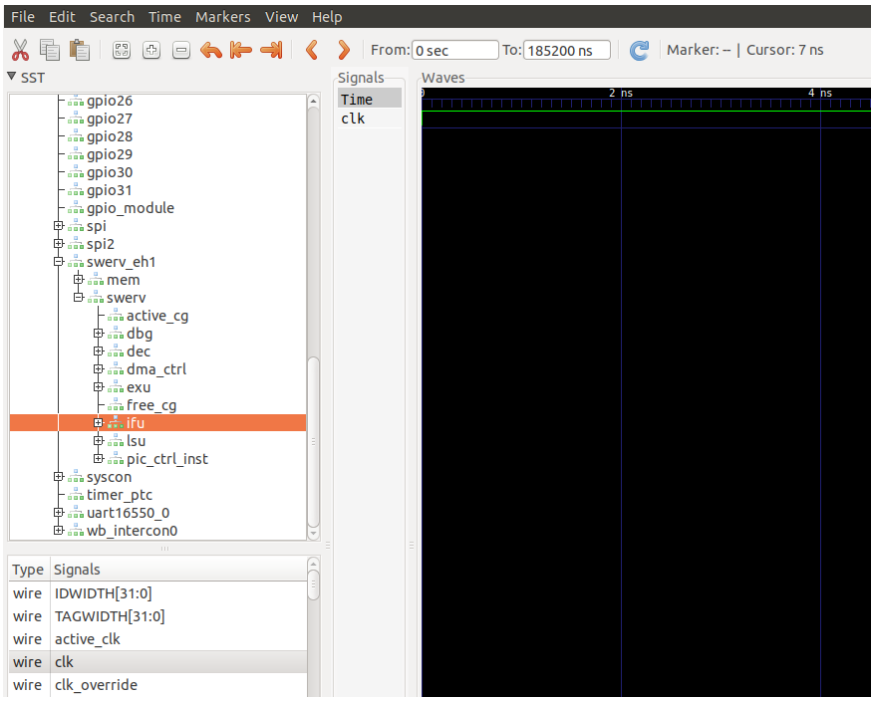



**Figura 84. Añada la señal *clk* al gráfico**

9. Haga un ajuste de zoom y luego haga clic en ampliar varias veces para que pueda ver el cambio de la señal del reloj (Figura 85).

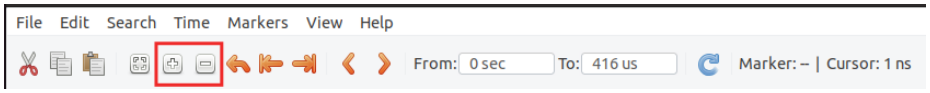



**Figura 85. Ampliar (+) Reducir (-)**

10. Ahora agregue las señales que muestran las instrucciones que se ejecutan en cada una de las dos vías del core superescalar RISC-V. En el mismo módulo (**ifu**) busque las señales *ifu_i0_instr[31:0]* e *ifu_i1_instr[31:0]* (Figura 86), y arrástrelas al panel negro de Ondas. El prefijo *ifu* indica la unidad de obtención (fetch) de la instrucción, *i0 indica la vía* superescalar 0 e *i1* indica la vía superescalar 1; *instr*[31:0] indica la instrucción de 32 bits.

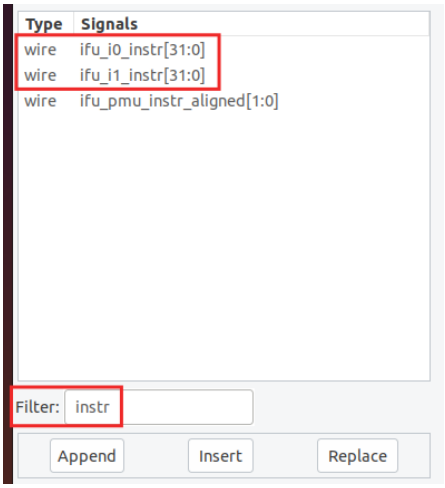



**Figura 86. Añada las señales *ifu_i0_instr[31:0]* e *ifu_i1_instr[31:0]* a la forma de onda**

11. Ahora agregue la señal que contiene el valor del registro `t3` (es decir, el registro número 28, `x28`). Expanda la jerarquía bajo **swerv** en **dec** → **arf** → **gpr_banks(0)** → **gpr(28)** y pulse sobre el módulo **gprff** (se resaltará como se muestra en la siguiente figura), seleccione la señal *dout[31:0]* (que muestra el contenido del registro `x28`, utilizado en el ejemplo de *AL_Operations.S)* y arrástrela al panel negro de ondas (Figura 87).

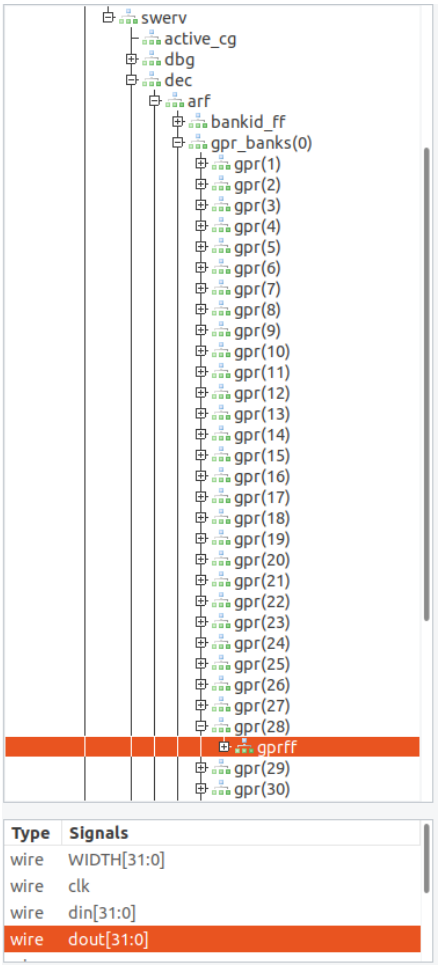



**Figura 87. Añada la señal *dout[31:0]* al gráfico**

12. Otra forma de mostrar señales en GTKWave es utilizando un fichero *.tcl*. En el directorio *[RVfpgaPath]/RVfpga/examples/AL_Operations* se proporciona el fichero *test.tcl*. Abre el fichero y analízalo. En cada línea se incluye la ruta y el nombre de cada señal que queremos mostrar en la gráfica.

```
gtkwave::addSignalsFromList rvfpgasim.clk
gtkwave::addSignalsFromList rvfpgasim.swervolf.swerv_eh1.swerv.ifu.ifu_i0_instr
gtkwave::addSignalsFromList rvfpgasim.swervolf.swerv_eh1.swerv.ifu.ifu_i1_instr
gtkwave::addSignalsFromList rvfpgasim.swervolf.swerv_eh1.swerv.dec.arf.gpr_banks(0).gpr(28).gprff.dout
```

Para utilizar el fichero *.tcl* en GTKWave tan solo tenemos que hacer clic en *File – Read Tcl Script File* y seleccionar el archivo *[RVfpgaPath]/RVfpga/examples/AL_Operations/test.tcl*.

13. Figura 88 muestra el programa AL_Operations.S y sus instrucciones equivalentes en lenguaje máquina.

```
# RISC-V assembly                 # comment (t3 = x28)        # machine code
    li   t3, 0x0                  # t3 = 0                    # 0x00000E13

REPEAT:
    addi t3, t3, 6                # t3 = t3 + 6               # 0x006E0E13
    addi t3, t3, -1               # t3 = t3 - 1               # 0xFFFE0E13
    andi t3, t3, 3                # t3 = t3 AND 3             # 0x003E7E13

    beq  zero, zero, REPEAT       # Repeat the loop           # 0xFE000CE3
    nop                           # nop                       # 0x00000013
```

**Figura 88. AL_Operations.S con su código máquina equivalente**

Ahora vea el cambio de las señales a medida que el programa se ejecuta. A medida que el programa se ejecuta en `t3` (registro `x28`) deben aparecer los valores mostrados en la Figura 89:

```
             li   t3, 0x0              # t3 = 0            # 0x00000E13
REPEAT:      addi t3, t3, 6            # t3 = 0 + 6 = 6    # 0x006E0E13
             addi t3, t3, -1           # t3 = 5            # 0xFFFE0E13
             andi t3, t3, 3            # t3 = 5 & 3 = 1    # 0x003E7E13
             beq  zero, zero, REPEAT   # Repeat the loop   # 0xFE000CE3
             nop                       # nop               # 0x00000013
REPEAT:      addi t3, t3, 6            # t3 = 1 + 6 = 7    # 0x006E0E13
             addi t3, t3, -1           # t3 = 7 - 1 = 6    # 0xFFFE0E13
             andi t3, t3, 3            # t3 = 6 & 3 = 2    # 0x003E7E13
             beq  zero, zero, REPEAT   # Repeat the loop   # 0xFE000CE3
             ...
             ...
```

**Figura 89. Flujo de instrucciones y valores del registro `t3` (`x28`) durante la ejecución de AL_Operations**

14. Amplíe el zoom alrededor de 10100 ns, donde analizará la ejecución de las tres instrucciones aritmético-lógicas de la primera y segunda iteración del bucle (Figura 90). Las dos primeras instrucciones (`li t3, 0x0` = `0x00000E13` y `addi t3, t3, 6` = `0x006E0E13`) se leen primero, una en cada vía del procesador superescalar RISC-V como se muestra en las señales *ifu_i0_instr*[31:0] e *ifu_i1_instr*[31:0]. Las dos instrucciones siguientes (`addi t3, t3, -1` = `0xFFFE0E13` y `andi t3, t3, 3` = `0x003E7E13`) se leen en el siguiente ciclo. Las dos últimas instrucciones (`beq zero,`

`zero, REPEAT` = `0xFE000CE3` y `nop` = `0x00000013)` se leen en el siguiente ciclo.

Debido al pipeline de 9 etapas del core SweRV y las dependencias entre instrucciones, los efectos de las instrucciones se ven ocho o más ciclos después que las instrucciones se leen. Ocho ciclos después que se leen las dos primeras instrucciones, en el registro `x28` (`t3`) se escribe un 0 (que ya lo había) debido a la primera instrucción: `li t3, 0x0` (`0x00000E13`). Un ciclo más tarde, el registro `x28` se actualiza a 0x6 debido a la siguiente instrucción: `addi t3, t3, 6` (`0x006E0E13`). Un ciclo más tarde, el registro `x28` se actualiza a 5 debido a la siguiente instrucción: `addi t3, t3, -1` (`0xFFFE0E13`). Finalmente, `x28` se actualiza a 1 debido a la siguiente instrucción: `andi t3, t3, 3` (`0x003E7E13`). Luego se leen las siguientes dos instrucciones: `beq zero, zero, REPEAT` (`0xFE000CE3`) y `nop` (`0x00000013`), se toma el salto y se repite el bucle, tal y como se predice en la Figura 89. Utilizando un razonamiento similar, se puede analizar la segunda iteración, que también se destaca en la Figura 90 y se pronostica en la Figura 89.

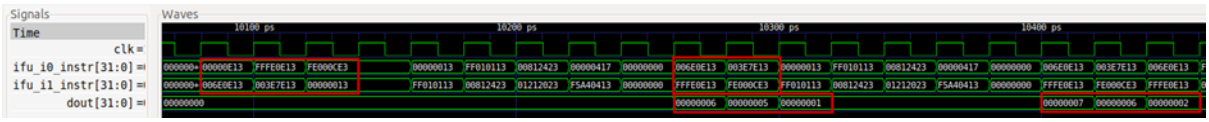



**Figura 90. Ejecución de las tres instrucciones aritmético-lógicas del ejemplo**

# 8. SIMULACIÓN EN WHISPER

Whisper (https://github.com/chipsalliance/SweRV-ISS) es un simulador de del repertorio de instrucciones RISC-V (ISS por sus siglas en inglés) desarrollado por Western Digital para la verificación del microcontrolador SweRV. Permite al usuario ejecutar código RISC-V sin necesidad de un hardware RISC-V subyacente. Usando Whisper, puede probar, ejecutar y depurar programas C o de lenguaje ensamblador usando PlatformIO sin requerir la placa FPGA Nexys A7.

**Windows:** Todas las instrucciones descritas en esta sección deberían funcionar para Windows (queremos agradecer a Jean-François Monestier, que fue el primero en portar Whisper a Windows: https://jean-francois.monestier.me/porting-western-digital-swerv-iss-to-windows/). Tenga en cuenta que una ventana emergente puede pedirle que permita a Whisper atravesar el firewall de Windows.

**macOS:** Todas las instrucciones descritas en esta sección también funcionan para MacOS.

Se puede ejecutar Whisper tanto usando la línea de comandos como usando un IDE (entorno de desarrollo integrado) como Eclipse o PlatformIO. En esta sección se muestra un ejemplo para indicar cómo simular un programa con Whisper en PlatformIO. Puede por tanto utilizar los mismos pasos descritos aquí para simular otros programas.

Se comenzará utilizando el ISS Whisper para simular *AL_Operations*, el primer programa en lenguaje ensamblador que se ejecutó y depuró en la Sección 6 (ver Figura 44). Siga los siguientes pasos para ejecutar y depurar este código en Whisper:

1. Abrir VSCode (y PlatformIO). En la barra de menú superior, haga clic en *File → Open Folder* y navegue a la carpeta *[RVfpgaPath]/RVfpga/examples/*, seleccione (pero no abra) la carpeta *AL_Operations* y luego haga clic en OK.

2. Haga clic en *File → Open File* y haga doble clic en *[RVfpgaPath]/RVfpga/examples/AL_Operations/platformio.ini*, y establezca **Whisper** como la herramienta de depuración al descomentar la línea 17 (ver Figura 91). Guarde el archivo (presione Ctrl+s).

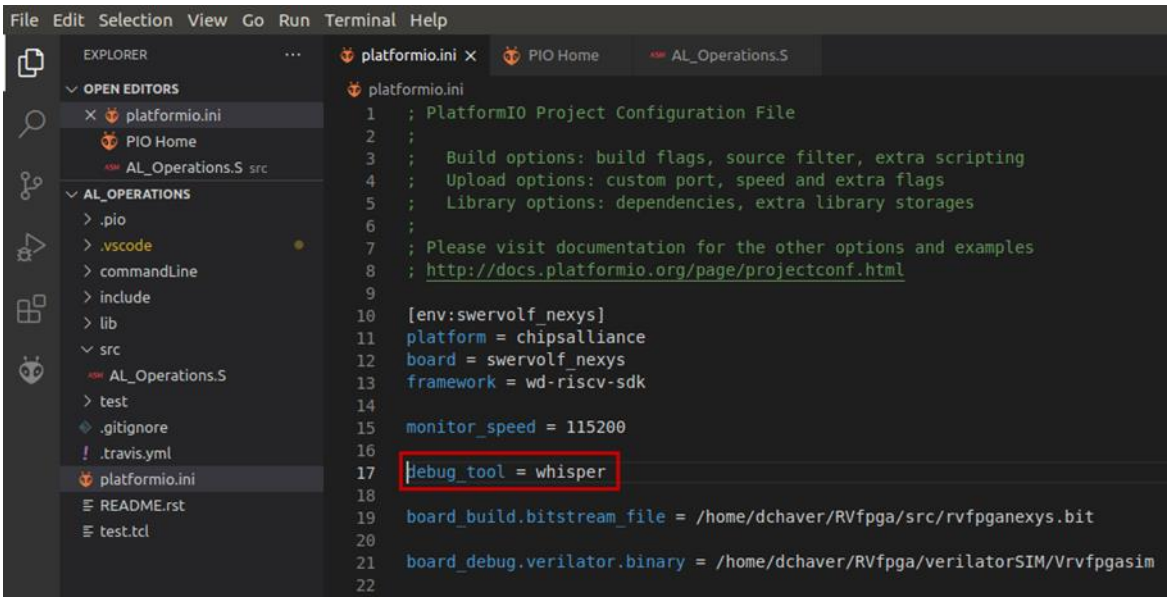
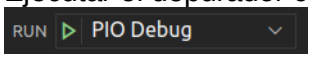



**Figura 91. Descomentar la línea 17.**

3. Ejecutar el depurador como de costumbre, haciendo clic en el botón  y luego en 


4. Ahora puede depurar el programa exactamente como se hizo en la sección 6.B, pero esta vez el programa se ejecuta simulado en Whisper en lugar de en la placa FPGA Nexys A7.

5. Si un programa utiliza la función *printfNexys* en Whisper, como el ejemplo HelloWorld_C-Lang (Sección 6.F), no debe abrir la terminal serie de PlatformIO, ya que en este caso los mensajes se muestran en la consola DEBUG (ver Figura 92).

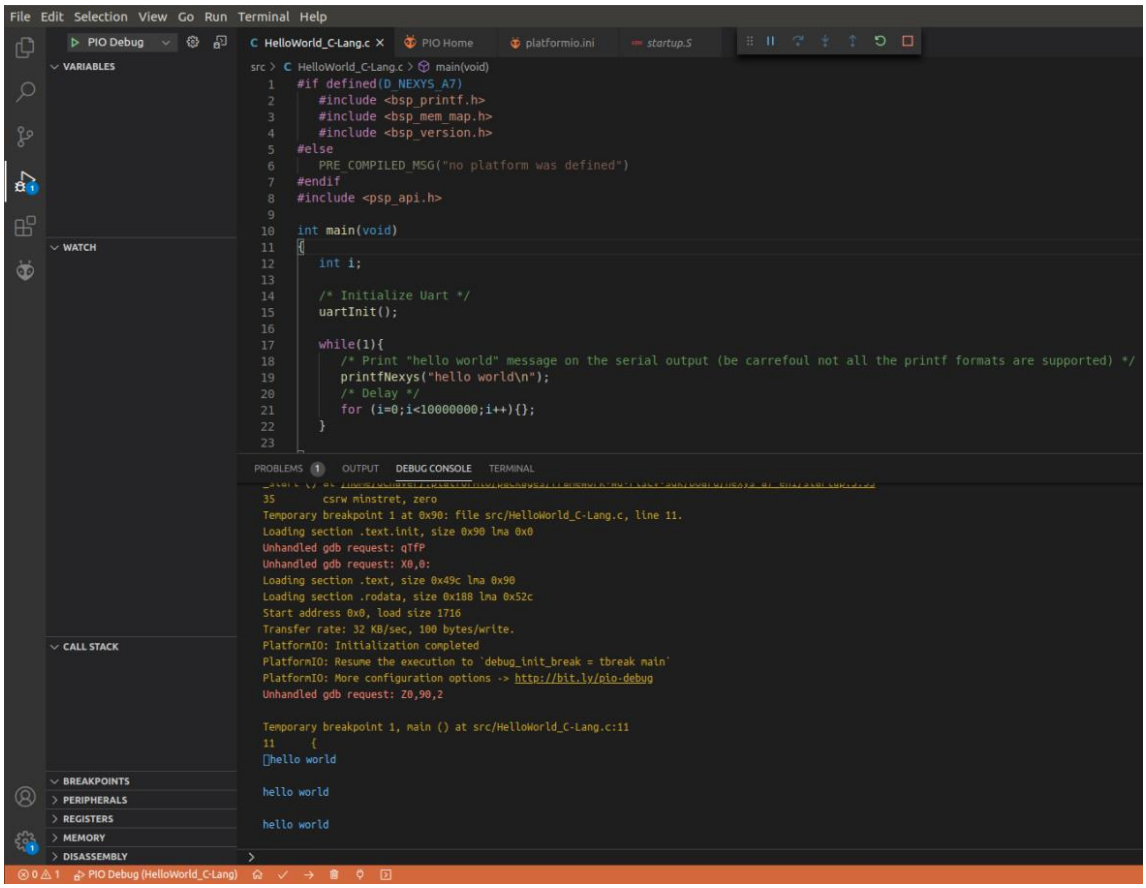



**Figura 92. Ejecución del ejemplo HelloWorld_C-Lang en Whisper**

# 9. APÉNDICES

Los siguientes apéndices muestran cómo usar las herramientas nativas de RISC-V y OpenOCD (en lugar de PlatformIO) en GNU/Linux, cómo instalar en Windows los drivers para descargar el archivo bitstream usando PlatformIO, cómo instalar Verilator y GTKWave en sistemas Windows y Mac OS, y cómo programar RVfpgaNexys usando Vivado. Tabla 10 lista todos los apéndices disponibles en esta Guía de inicio de RVfpga.

**Tabla 10. Lista de Apéndices**

| Apéndice | Descripción | Sistema operativo |
|---|---|---|
| A | Uso de las herramientas nativas de RISC-V y OpenOCD para RVfpga en Ubuntu 18.04 | GNU/Linux |
| B | Instalación de drivers en Windows para usar PlatformIO | **Windows** |
| C | Instalación de Verilator y GTKWave en Windows | **Windows** |
| D | Instalación de Verilator y GTKWave en macOS | **macOS** |
| E | Uso de Vivado para descargar RVfpgaNexys en una FPGA | **Windows** y GNU/Linux |
| F | Uso de RVfpga en una aplicación industrial de IoT | Todos |

El Apéndice A debería ser usado por aquellos que quieran compilar y ejecutar/depurar programas de forma nativa usando las herramientas nativas gcc/gdb y OpenOCD. Sin embargo, se **recomienda que el uso de PlatformIO** en su lugar, como se describe en esta Guía de inicio.

Los usuarios de **Windows** **deben seguir las instrucciones de los Apéndices B y C**. Las instrucciones del Apéndice B muestran cómo configurar los drivers para que los sistemas Windows puedan usar PlatformIO tanto para descargar programas como para descargar RVfpgaNexys en la placa FPGA Nexys A7. El Apéndice C muestra cómo instalar Verilator y GTKWave para que los usuarios de Windows puedan simular RVfpgaSim.

Los usuarios de **macOS** **deben seguir las instrucciones del Apéndice D para** simular RVfpgaSim usando Verilator y GTKWave.

**Se recomienda utilizar PlatformIO para descargar RVfpgaNexys** (como se define en el archivo bitfile, rvfpganexys.bit) en la placa FPGA Nexys A7. Este archivo bitfile (rvfpganexys.bit) puede ser generado por Vivado o PlatformIO. También es posible utilizar Vivado para descargar RVfpgaNexys en la placa FPGA Nexys A7, como se describe en el Apéndice E. Sin embargo, no se **recomienda utilizar** Vivado para descargar RVfpgaNexys en la placa, especialmente para los usuarios de **Windows**, ya que requeriría un intercambio continuo de drivers.

# Apéndice A: Uso de las herramientas nativas de RISC-V y OpenOCD para RVfpga en Ubuntu 18.04

Aunque se recomienda el uso de PlatformIO, en esta sección se muestra cómo instalar, ejecutar y utilizar las herramientas nativas de RISC-V y cómo utilizar OpenOCD para descargar RVfpgaNexys en la placa FPGA Nexys A7 y gdb para ejecutar y depurar programas en RVfpgaNexys. Las herramientas consisten en un compilador gnu, depurador, ensamblador, etc. Se muestra cómo instalar las herramientas de RISC-V y OpenOCD en un sistema operativo (SO) Ubuntu 18.04, pero este proceso también debería funcionar para otras distribuciones de GNU/Linux. Estas instrucciones suponen un sistema Ubuntu en sus versiones más recientes.

Los siguientes pasos no son necesarios si utiliza PlatformIO, como se ha descrito anteriormente en esta guía. El uso de PlatformIO, Vivado y Verilator o Whisper es el método recomendado para ejecutar, depurar y simular programas RISC-V, pero se proporcionan las siguientes instrucciones para cualquiera que esté interesado en utilizar las herramientas nativas de RISC-V y OpenOCD en lugar de PlatformIO y Vivado.

## I. Instalación nativa en un sistema operativo GNU/Linux Ubuntu

En esta sección se describe cómo instalar de forma nativa en su sistema Ubuntu 18.04 las herramientas nativas de RISC-V, OpenOCD y Whisper. Estas herramientas sólo sustituyen a PlatformIO; la instalación de Vivado y Verilator sigue siendo necesaria como se explica en la Sección 5 de esta Guía de Inicio

**Herramientas Nativas de RISC-V**

Aquí se muestra cómo instalar el conjunto completo de herramientas nativas de RISC-V, es decir, el compilador gnu, el depurador, etc. en su computadora. Las instrucciones de instalación son proporcionadas por RISC-V International en: https://github.com/riscv/riscv-gnu-toolchain. Estas instrucciones se resumen a continuación.

NOTA: La instalación del conjunto de herramientas nativas de RISC-V y OpenOCD podría llevar varias horas, la mayor parte del tiempo esperando mientras las herramientas se descargan, compilan e instalan.

En una terminal, escriba lo siguiente (el proceso puede llevar más de una hora, pero la mayor parte del tiempo se pasa esperando mientras se descargan e instalan los programas):

- `sudo apt-get install git autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev`
- `git clone --recursive https://github.com/riscv/riscv-gnu-toolchain`
- `cd riscv-gnu-toolchain/`
- `./configure --prefix=/opt/riscv --with-arch=rv32imc`
- `sudo make` (Si es posible use `sudo make -j$(nproc)` ya que disminuye significativamente el tiempo de compilación)
- `export PATH=$PATH:/opt/riscv/bin` (cambia la variable PATH en su sistema)

**OpenOCD**

OpenOCD es un depurador on-chip abierto, que permite a los usuarios programar y depurar dispositivos empotrados. Siga los siguientes pasos para instalar OpenOCD para RISC-V en su computadora:

- `sudo apt-get install libusb-1.*`
- `sudo apt-get install pkg-config`
- `git clone https://github.com/riscv/riscv-openocd.git`
- `cd riscv-openocd/`
- `./bootstrap`
- `./configure --prefix=/opt/riscv --program-prefix=riscv- --enable-ftdi --enable-jtag_vpi`
- `make`
- `sudo make install`

**Whisper**

Siga los siguientes pasos para instalar Whisper en su computadora (las instrucciones están disponibles en: https://github.com/chipsalliance/SweRV-ISS pero también se resumen a continuación):

- `apt-cache policy libboost-all-dev`
- `sudo apt-get install libboost-all-dev`
- `cd [RVfpgaPath]`
- `git clone https://github.com/chipsalliance/SweRV-ISS`
- `cd SweRV-ISS`
- `make BOOST_DIR=/usr/include/boost`
- `export PATH=$PATH:[RVfpgaPath]/SweRV-ISS/build-Linux` (sustituya `[RVfpgaPath]` según corresponda).

## II. Ejecución de un programa en RVfpgaNexys usando la placa FPGA Nexys A7 y utilizando OpenOCD

### Paso A. Descargar RVfpgaNexys (Figura 25) en la Nexys A7

1. Entre en el directorio del proyecto que contiene el archivo bitfile para RVfpgaNexys:
   ```
   cd [RVfpgaPath]/RVfpga/src
   ```

2. Descargue RVfpgaNexys en la placa FPGA usando OpenOCD:
   ```
   riscv-openocd -c "set BITFILE rvfpganexys.bit" -f
   OtherSources/ConfigFiles/swervolf_nexys_program.cfg
   ```

### Paso B. Ejecutar LedsSwitches, el programa que lee los Switches e imprime su estado en los LEDs

3. Entre en el directorio LedsSwitches/commandLine:
   ```
   cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine
   ```

   En ese directorio encontrará el Makefile para compilar las fuentes, el script de enlazado, un script de Python, y el programa *LedsSwitches.S.*

4. Construya el archivo .elf:
   ```
   make clean
   make LedsSwitches.elf
   ```

5. Conecte OpenOCD al SoC:
   ```
   riscv-openocd -f
   ../../../src/OtherSources/ConfigFiles/swervolf_nexys_debug.cfg
   ```

Una vez que OpenOCD comience a funcionar, verá varios mensajes incluyendo uno que dice:

```
Info : Listening on port 4444 for telnet connections
```

6. Abra una nueva terminal, y vaya al directorio de programas (`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`) y ejecute el siguiente comando:

```
telnet localhost 4444
```

Luego, dentro de la conexión telnet, teclee:

```
load_image LedsSwitches.elf
reg pc 0
resume
```

Estos tres comandos (1) cargan el programa LedsSwitches.elf en RVfpgaNexys, (2) ponen el contador de programa (PC) a 0 (la dirección de la primera instrucción del programa), y (3) reanudan la ejecución.

El programa comenzará a funcionar en RVfpgaNexys, el SoC SweRVolfX que ya se descargó en la placa FPGA Nexys A7 en el paso 2. El programa hace que los LEDs muestren el estado de los interruptores. Al conmutar los interruptores, los LEDs deben cambiar inmediatamente para reflejar el valor de los interruptores.

**Paso C. Depurar el programa AL_Operations_CommandLine que ejecuta operaciones aritmético-lógicas simples**

Ahora se mostrará cómo depurar otro programa (AL_Operations_CommandLine) usando OpenOCD y gdb.

7. Mantenga la conexión OpenOCD abierta (ver paso 5).

8. En la otra terminal donde se ejecuta telnet (desde el paso 6), salga de la conexión telnet tecleando:

```
exit
```

9. Cambie al directorio del proyecto que contiene AL_Operaciones/commandLine:

```
cd .. /../AL_Operations/commandLine
```

En ese directorio encontrará el Makefile para compilar las fuentes, el script de enlazado, un script de Python, y el programa *AL_Operations.S.*

10. Construya el archivo .elf:

```
make clean
make AL_Operations.elf
```

11. A continuación, en esta terminal, inicie gdb escribiendo lo siguiente:

```
riscv32-unknown-elf-gdb AL_Operations.elf
```

12. Dentro de la consola gdb, escriba:

```
target remote localhost:3333
load
```

Esto hará que se establezca la conexión con OpenOCD y cargará el programa *AL_Operations.elf* en la memoria.

13. Ahora debería ser capaz de depurar el programa. Escriba la siguiente secuencia y analice los resultados:

    i. `disas 0,20`

    Esto muestra el código en lenguaje ensamblador de la dirección 0 a la 20 (sin incluir la dirección 20).

```
(gdb) disas 0,20
Dump of assembler code from 0x0 to 0x14:
=> 0x00000000 <_start+0>:       li      t3,0
   0x00000004 <REPEAT+0>:       addi    t3,t3,6
   0x00000008 <REPEAT+4>:       addi    t3,t3,-1
   0x0000000c <REPEAT+8>:       andi    t3,t3,3
   0x00000010 <REPEAT+12>:      beqz    zero,0x4 <REPEAT>
End of assembler dump.
```

**Figura 93. Visualización del programa en lenguaje ensamblador**

    ii. `i r t3`

    Esto muestra el contenido del registro `t3`. Alternativamente, puede escribir la versión más larga: `info reg t3`.

```
(gdb) i r t3
t3             0x0      0
```

**Figura 94. Impresión del valor contenido en el registro `t3`**

    iii. `i r pc`

    Esto muestra el contenido del contador de programa (`pc`).

```
(gdb) i r pc
pc             0x0      0x0 <_start>
```

**Figura 95. Impresión del valor contenido en el registro PC, que apunta a la primera instrucción**

    iv.
```
stepi
i r t3
stepi
i r t3
stepi
i r t3
stepi
i r t3
```

    `stepi` hace que el programa ejecute una instrucción. `i r t3` muestra entonces el contenido del registro `t3`.

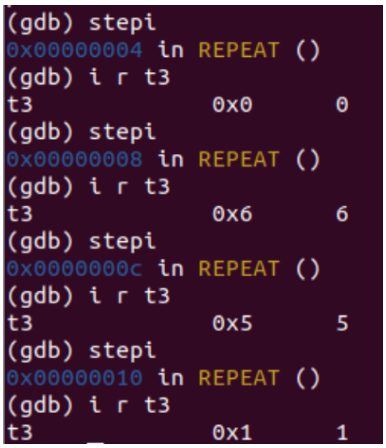



**Figura 96. Ejecutar varias instrucciones una por una y ver el registro `t3`**

Una vez que haya terminado de depurar y explorar el programa y los registros usando gdb, salga de gdb tecleando **quit** en la terminal gdb y salga de OpenOCD tecleando ^C en el terminal OpenOCD.

## III. Simulación de un programa en RVfpgaSim usando Verilator

1. Abra una terminal en Ubuntu

2. En una ventana de terminal, genere el archivo binario del simulador ejecutando los siguientes comandos:

```
cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean
make
```

   El archivo *Vrvfpgasim* (el binario de simulación RVfpgaSim), debe ser generado dentro del directorio *[RVfpgaPath]/RVfpga/verilatorSIM*.

3. Entre en la carpeta que contiene el programa de ejemplo:

```
cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine
```

4. Cree el programa hexadecimal para la simulación.

```
make clean
make AL_Operations.elf
make AL_Operations.bin
make AL_Operations.vh
```

5. Ejecute el simulador.

```
.. /.../../verilatorSIM/Vrvfpgasim
+ram_init_file=AL_Operations.vh +vcd=1
```

   Después de unos segundos, detenga la simulación introduciendo ^C en la terminal. Se debe generar el archivo trace*.vcd*, el cual puede abrir con `GTKWave`.

```
gtkwave trace.vcd
```

6. Siga las instrucciones de los pasos 8 a 12 de la Sección 7 para añadir señales al gráfico y analizarlas.

## IV. Simulación de un programa en Whisper

1. Abra una terminal en Ubuntu

2. Entre en la carpeta que contiene el programa de ejemplo:
   ```
   cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine
   ```

3. Cree el programa desensamblado.
   ```
   make AL_Operations.dis
   ```

4. Abra *AL_Operations.dis* en un editor. Esto es lo que debería ver:

   ```
   <_start>:
      0: 00000e13                li    t3,0
   <REPEAT>:
      4: 006e0e13                addi  t3,t3,6
      8: fffe0e13                addi  t3,t3,-1
      c: 003e7e13                andi  t3,t3,3
     10: fe000ae3                beqz  zero,4 <REPEAT>
     14: 00000013                nop
   ```

5. Ejecute el simulador en modo interactivo.
   ```
   whisper --interactive AL_Operations.elf
   ```

6. Depure el programa.

   ```
   whisper> step
     #1 0 00000000 00000e13 r 1c        00000000  addi     x28, x0, 0x0

   whisper> peek r x28
     0x00000000

   whisper> step
     #2 0 00000004 006e0e13 r 1c        00000006  addi     x28, x28, 0x6

   whisper> peek r x28
     0x00000006

   whisper> step
     #3 0 00000008 fffe0e13 r 1c        00000005  addi     x28, x28, -0x1

   whisper> peek r x28
     0x00000005

   whisper> step
     #4 0 0000000c 003e7e13 r 1c        00000001  andi     x28, x28, 0x3

   whisper> peek r x28
     0x00000001
   ```

Una vez que haya terminado de depurar y explorar el programa y los registros usando whisper, salga escribiendo **quit** en la terminal.

# Apéndice B: Instalación de los drivers en Windows para usar PlatformIO

Para descargar el ejecutable de Zadig, navegue hasta el siguiente sitio web (véase la Figura 97):

https://zadig.akeo.ie/

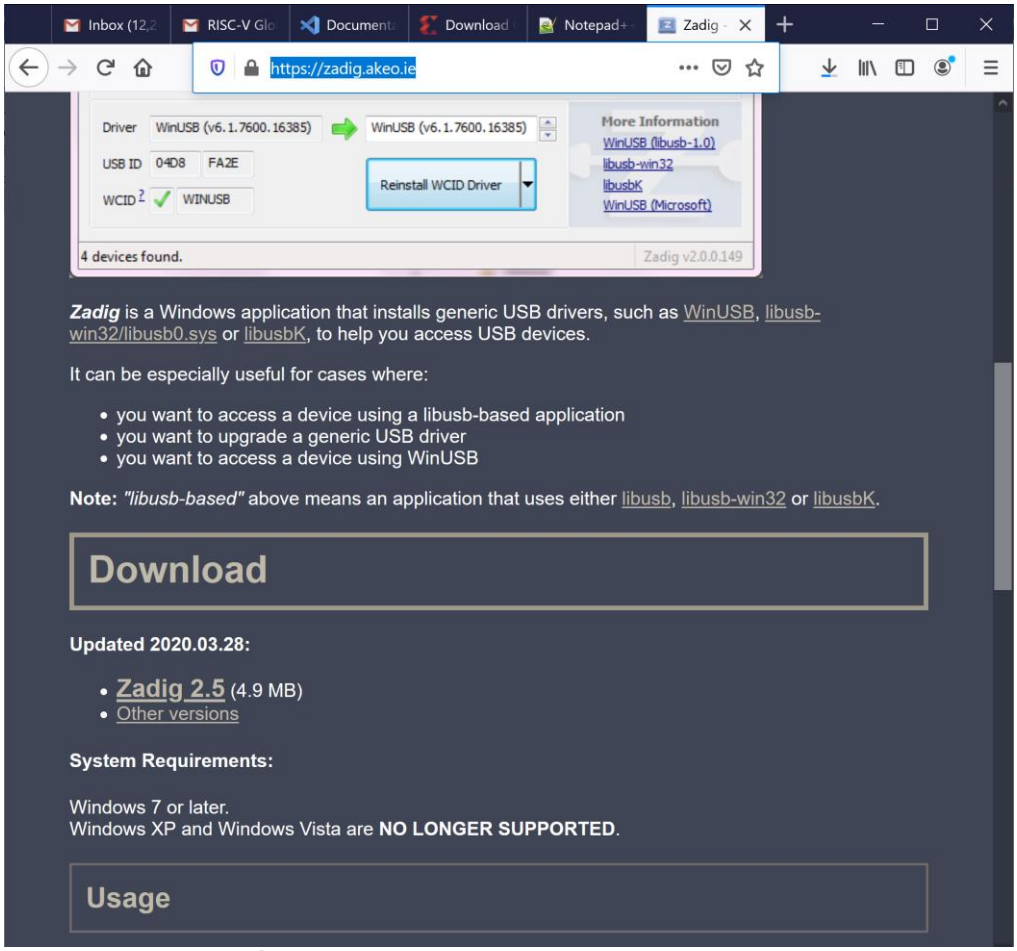



**Figura 97. Instalación del driver de la placa Nexys A7 usado por PlatformIO**

Haga clic en Zadig 2.5 y guarde el ejecutable. Luego ejecútelo (zadig-2.5.exe), éste se encuentra donde lo descargó. También puede escribir zadig en el menú de inicio para encontrarlo. Probablemente se le preguntará si quiere permitir que Zadig haga cambios en su computadora y si le permitirá buscar actualizaciones. Haga clic en Sí en ambos casos.

Conecte la placa Nexys A7 a su computadora y enciéndela. En Zadig, haga clic en Options → List All Devices (ver Figura 98).

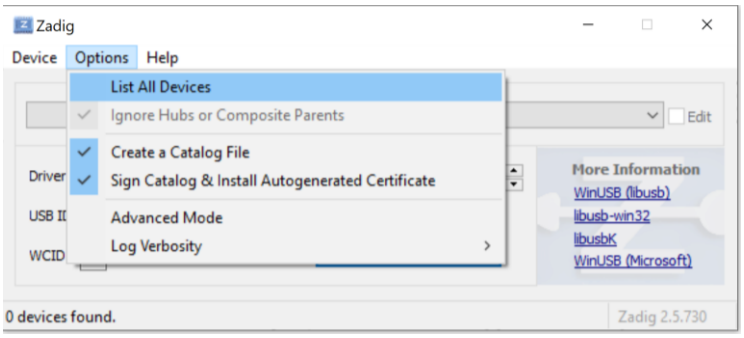



**Figura 98. Listado de todos los dispositivos en Zadig**

Si hace clic en el menú desplegable, verá en la lista Digilent USB Device (Interface 0) y Digilent USB Device (Interface 1). Instalará nuevos drivers sólo para Digilent USB Device (Interface 0) (véase la Figura 99).

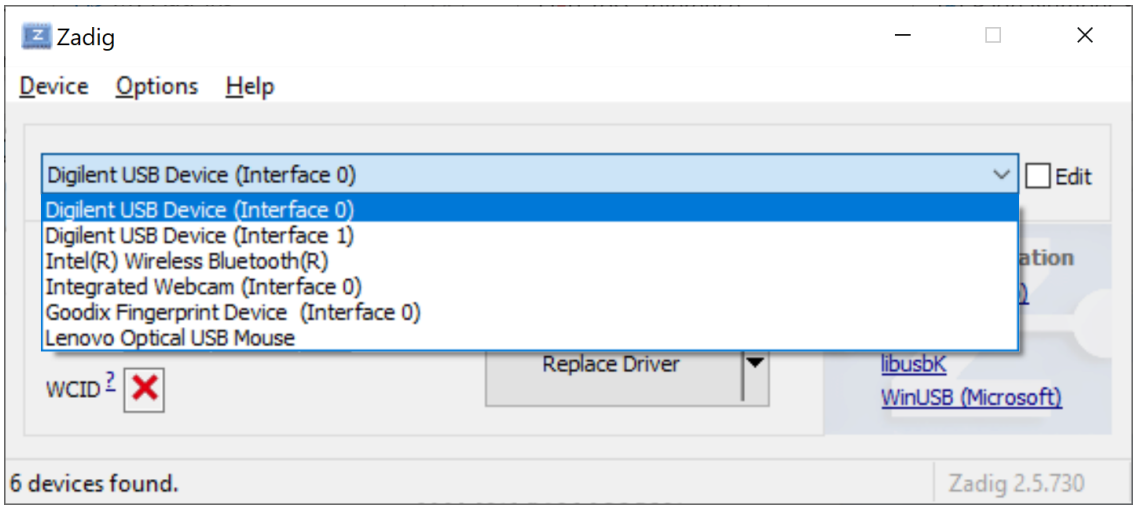



**Figura 99. Instalación del driver WinUSB para Digilent USB Device (Interface 0)**

Ahora sustituirá el driver FTDI por el driver WinUSB, como se muestra en la Figura 100. Haga clic en Replace Driver (o Install Driver) para Digilent USB Device (Interface 0). Está instalando el driver para la placa Nexys A7 o, si instaló previamente Vivado, está reemplazando el driver FTDI usado por Vivado por el driver WinUSB usado por PlatformIO.

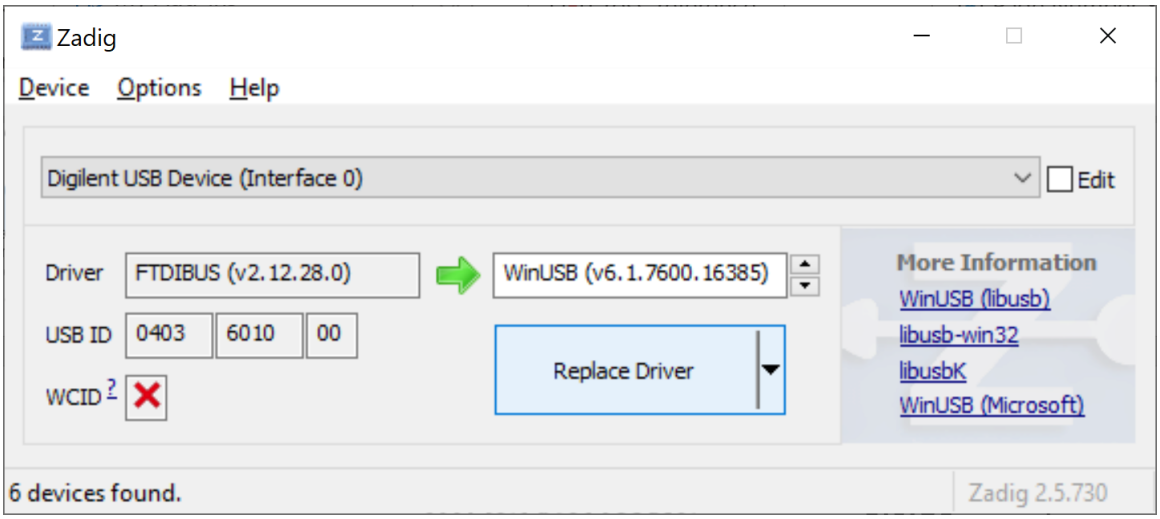



**Figura 100. Sustitución del driver para la placa Nexys A7**

Después de algún tiempo, normalmente varios minutos, Zadig indicará que el driver se instaló correctamente. Haga clic en Cerrar y luego cierre la ventana de Zadig.

La próxima vez que use PlatformIO no necesita reinstalar el driver. Sin embargo, tenga en cuenta que **este driver no es compatible con Vivado en Windows**. Por lo tanto, ya no podrá utilizar Vivado para descargar archivos bitfile a la placa FPGA. Si desea utilizar Vivado para descargar archivos bitfile (no recomendado), deberá volver al driver original instalado con Vivado, tal como se describe en el Apéndice E.

# Apéndice C: Instalación de Verilator y GTKWave en Windows

En esta sección, se explica cómo instalar Verilator y GTKWave en Windows 10. En Windows, se debe usar Cygwin para instalar Verilator, así que primero se explica cómo instalar este entorno de programación/ejecución.

## Instalación de Cygwin:

Como se describe en su página web (https://www.cygwin.com), Cygwin consiste en herramientas GNU y de código abierto que proporcionan una funcionalidad en Windows similar a la de una distribución GNU/Linux. Siga los siguientes pasos para instalar Cygwin en Windows 10.

1. Abra en un navegador la página web de instalación (https://cygwin.com/install.html) y descargue el archivo de instalación, llamado *setup-x86_64.exe* (Figura 101).


**Figura 101. Página web de la instalación de Cygwin**

2. Ejecute el archivo de configuración en su máquina haciendo doble clic en él (Figura 102). Haga clic en **Siguiente** varias veces, manteniendo las opciones predeterminadas. El instalador le pedirá que elija **un sitio de descarga** (Figura 103), puede elegir cualquiera de ellos.

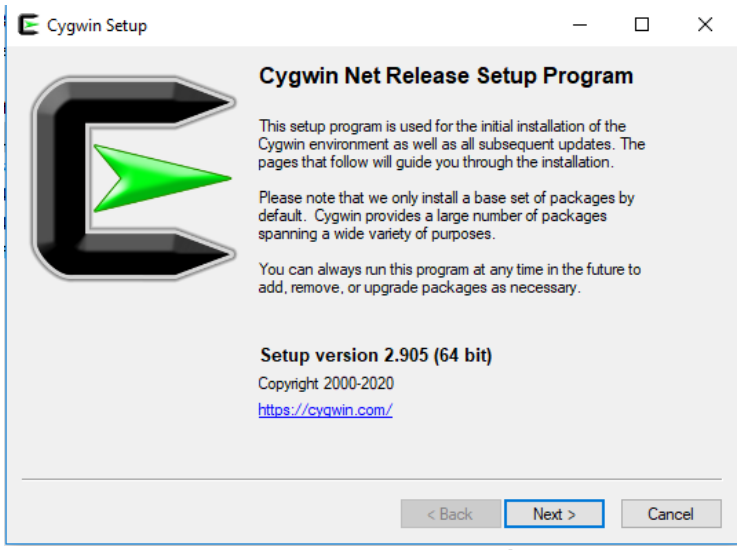



**Figura 102. Ventana de instalación de Cygwin**

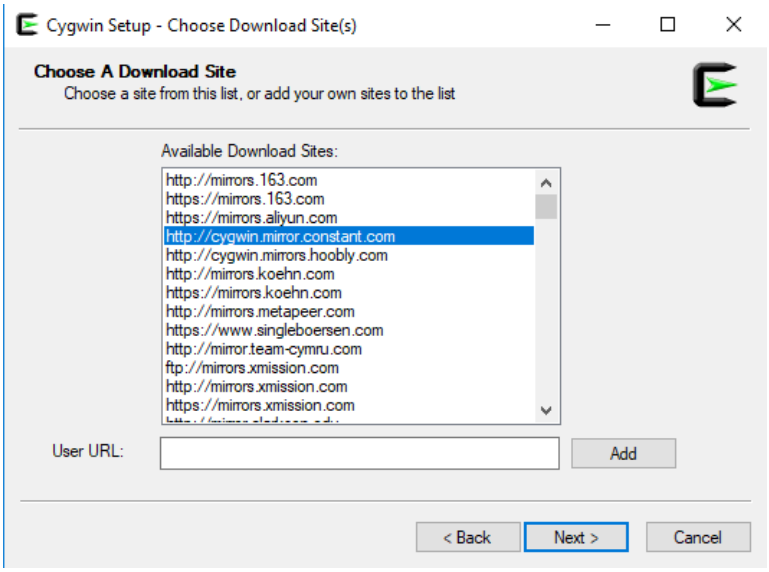



**Figura 103. Elija el sitio de descarga**

3. Después de varios pasos, llegará a la ventana de selección de paquetes (Figura 104). En las categorías de paquetes seleccione Full, como se muestra en la Figura 104.

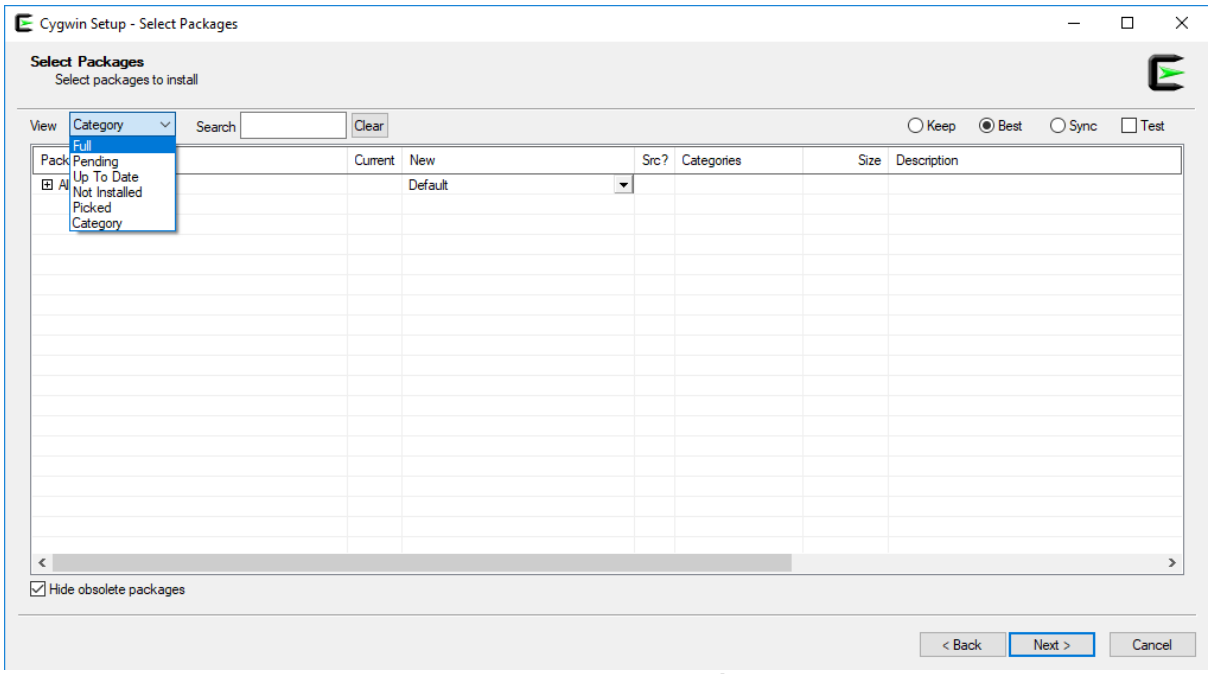



**Figura 104. Ventana de selección de Paquetes**

4. Aparecerá la lista completa de paquetes que puede instalar (Figura 105). En el cuadro de **búsqueda**, seleccione los paquetes específicos que desea instalar.

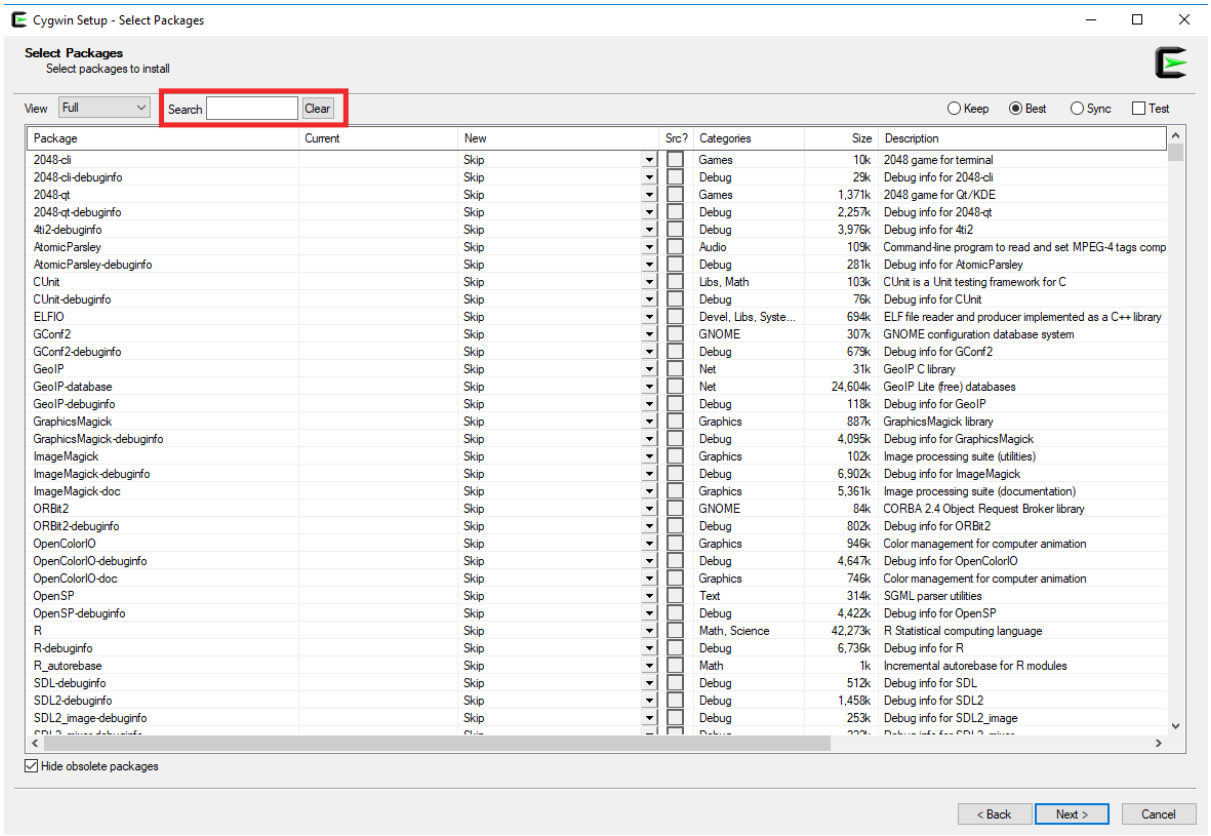



**Figura 105. Ventana de selección de paquetes - Vista completa**

Para poder compilar Verilator y generar un nuevo binario del simulador, es necesario instalar los siguientes paquetes:

- `git`
- `make`
- `autoconf`
- `gcc-core`
- `gcc-g++`
- `flex`
- `bison`
- `perl`
- `libargp-devel`

Incluya al menos estos paquetes en su instalación de Cygwin. Selecciónelos uno a uno siguiendo los pasos que se indican a continuación (sólo se muestran los pasos detallados para el primer paquete de la lista, `git`; el proceso es el mismo para los otros paquetes):

- Busque el paquete `git` en el cuadro de **búsqueda** (Figura 106).

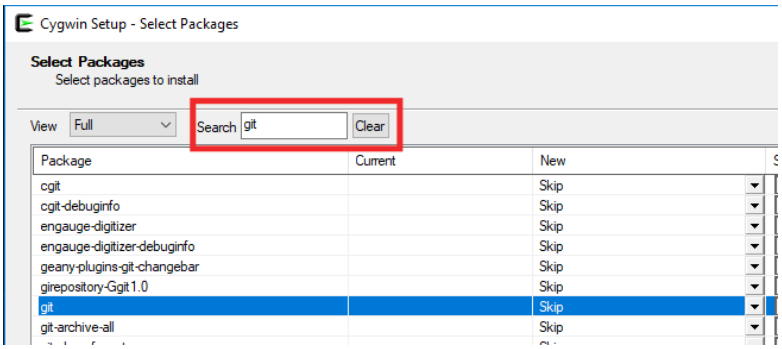



**Figura 106. Búsqueda del paquete `git`**

- Seleccione la versión más actualizada en el menú desplegable y marque la casilla (Figura 107).


**Figura 107. Seleccione la versión más actualizada y marque la casilla**

- Haga lo mismo con los demás paquetes de la lista anterior.

5. Una vez que haya seleccionado los nueve paquetes, haga clic en **Siguiente** en las ventanas subsiguientes para incluir estos paquetes en su instalación de Cygwin (el proceso de instalación, ver Figura 108, puede tardar varios minutos) y finalice la instalación haciendo clic en Finalizar (Figura 109).

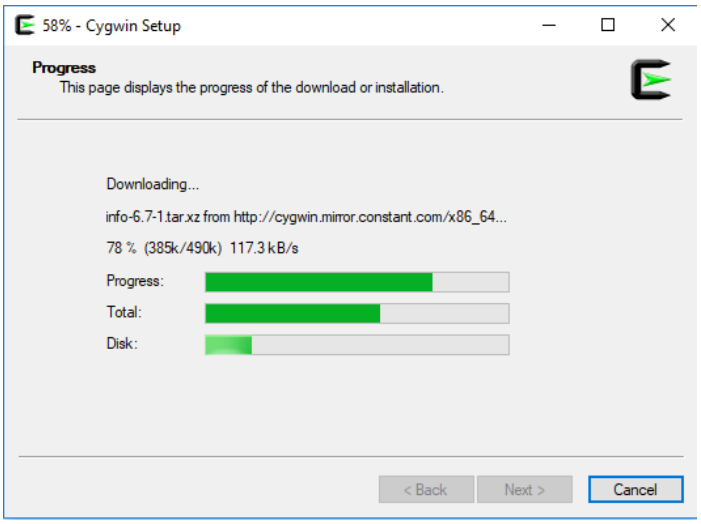



**Figura 108. Configuración de Cygwin**

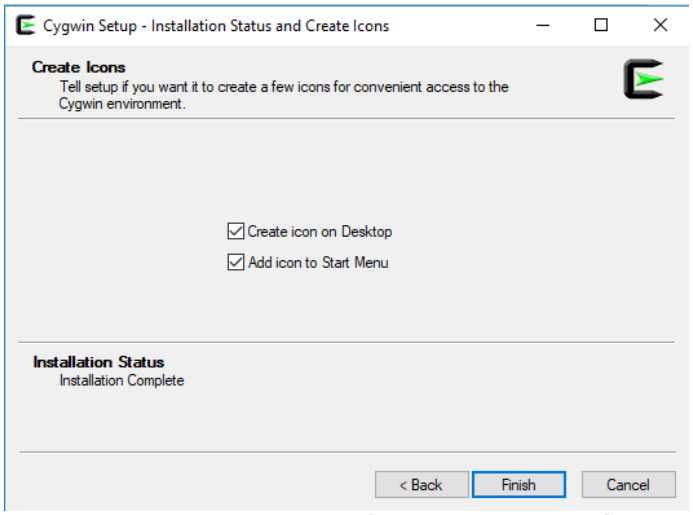



**Figura 109. Finalización de la instalación**

6. Si necesita añadir un paquete a su instalación Cygwin, repita los pasos 2-5 para ese paquete.

## Instalación de Verilator:

Siga los siguientes pasos para instalar Verilator en Windows 10.

1. Abra la terminal de Cygwin (Figura 110), disponible en el Escritorio de Windows o en el menú de Inicio.


**Figura 110. Terminal de Cygwin**

2. Compile e instale Verilator siguiendo estos pasos. Esto puede llevar algún tiempo (incluso horas), dependiendo de la velocidad de su computadora:

   - `git clone https://git.veripool.org/git/verilator`
   - `cd verilator`
   - `git pull`
   - `git checkout v4.106`
   - `autoconf`
   - `./configure`
   - `make`
   - `make install`

## Instalación de GTKWave:

Se puede descargar GTKWave como un paquete precompilado en https://sourceforge.net/projects/gtkwave/files/. Busque el paquete de Windows más reciente (en el momento en que se escribió este documento, se llamaba ***gtkwave-3.3.100-bin-win64***), y descárguelo y descomprímalo. Puede encontrar un archivo ejecutable llamado *gtkwave* dentro de la carpeta *bin*, que puede ejecutar y usar en su máquina de Windows.

# Apéndice D: Instalación de Verilator y GTKWave en macOS

En esta sección, se explica cómo instalar Verilator y GTKWave en macOS. Las instrucciones se probaron con macOS Catalina 10.15.6 pero se espera que funcionen en otras versiones del sistema operativo. Para la instalación se utiliza el gestor de paquetes Homebrew (https://brew.sh/). Se pueden encontrar instrucciones similares para MacPorts, el otro gestor de paquetes ampliamente utilizado en macOS (https://www.macports.org/).

**Instalación de gcc:**

Para construir un nuevo simulador utilizando Verilator, es necesario instalar un conjunto de herramientas de compilación en el sistema. Existen muchas maneras de instalar un conjunto válido de herramientas de compilación. A continuación citamos dos de ellas:

1. Instale las herramientas de línea de comandos de XCode. Tenga en cuenta que esto instalará LLVM, pero un comando *gcc* estará disponible de todos modos después de la instalación. Para ello, escriba el siguiente comando en una ventana de Terminal:

- `xcode-select –install`

2. Instale *gcc* usando Homebrew. Utilice el siguiente comando:

- `brew install gcc@9`

**Instalación de verilator:**

Instalar Verilator con Homebrew es tan simple como escribir el siguiente comando en una Terminal abierta:

- `brew install verilator`

**Instalación de gtkwave:**

Una vez más, se usará Homebrew para instalar *gtkwave*. Pero esta vez se necesita usar *cask* porque es una aplicación GUI de macOS. Escriba los siguientes comandos en una Terminal abierta:

- `brew tap homebrew/cask`
- `brew cask install xquartz`
- `brew cask install gtkwave`

Después de la instalación, debería aparecer un icono para *gtkwave.app* en la carpeta de aplicaciones . Para poder usarlo desde la línea de comandos, es posible que tenga que instalar el módulo Switch de Perl:

- `cpan install Switch`

# Apéndice E: Uso de Vivado para descargar RVfpgaNexys en una FPGA

Siga los siguientes pasos para programar la FPGA con el SoC RVfpgaNexys usando Vivado:

**WINDOWS:** Antes de seguir los siguientes pasos, en Windows es necesario volver a los drivers utilizados por Vivado como se explica al final de este apéndice (Apéndice E).

a. Conecte la placa Nexys A7 a su computadora.
b. Encienda la placa Nexys A7 usando el interruptor en la parte superior izquierda.
c. Abra Vivado 2019.2.
d. Abra el *Administrador de Hardware* disponible en Vivado y resaltado en la Figura 111.

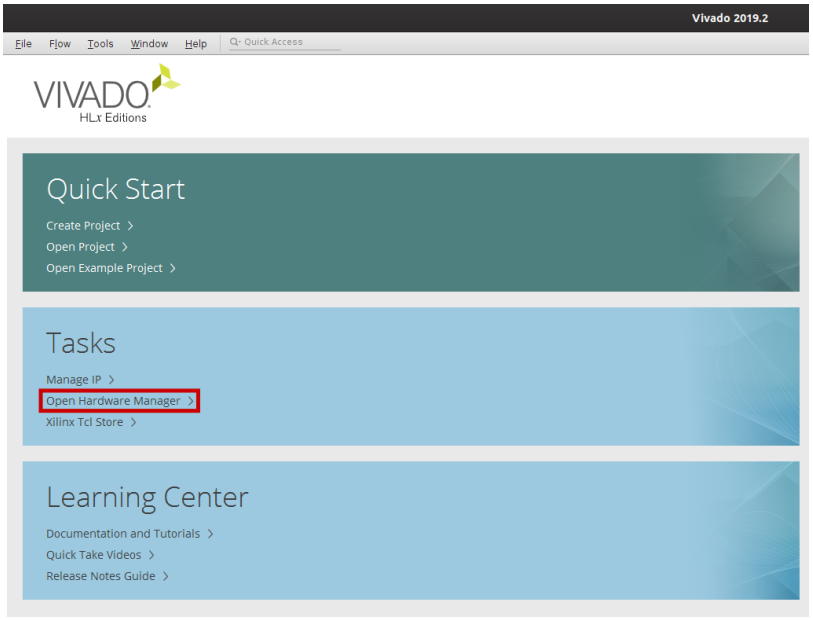



**Figura 111. Abrir el Administrador de Hardware**

e. Al abrir el administrador de hardware se informa que no hay ningún hardware objetivo abierto. Abra el objetivo haciendo clic en *Open target – Auto connect* (Figura 112).

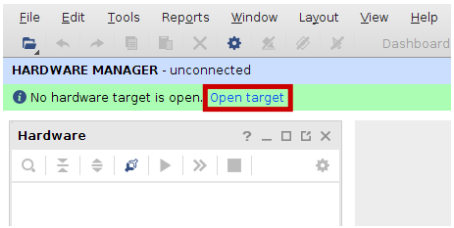



**Figura 112. Apertura de objetivo**

f. Seleccione *Program device* como se muestra en la
g. **Figura 113**. Ahora cargará RVfpgaNexys en la FPGA. En la nueva ventana, seleccione el *archivo Bitstream* de *[RVfpgaPath]/RVfpga/src/rvfpganexys.bit*. Haga clic en *Program*.

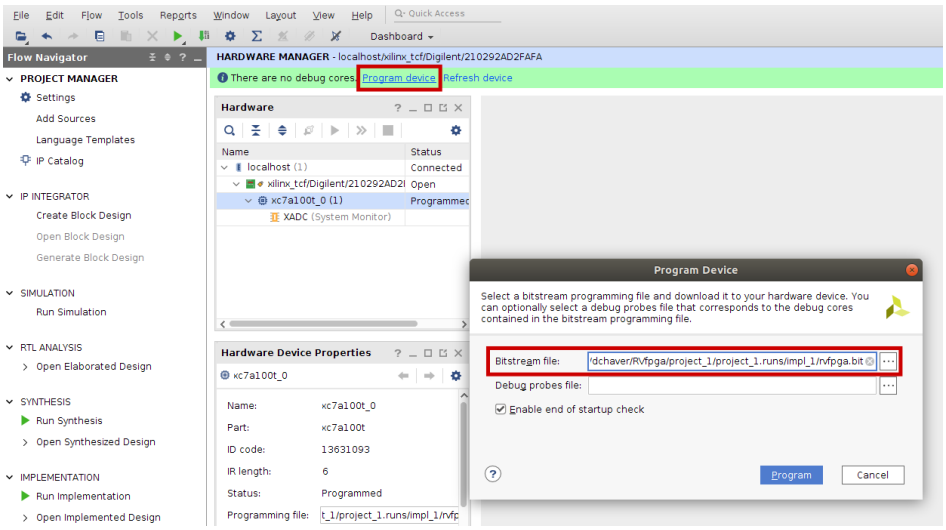



**Figura 113. Programación del dispositivo**

h. Después de unos segundos, la FPGA se programará con RVfpgaNexys, el **SoC SweRVolfX orientado a una FPGA** (ver Figura 25).
i. Finalmente, cierre **el Administrador de hardware** haciendo clic en el botón X de la parte superior derecha del panel del Administrador de hardware en Vivado (Figura 114), para que Vivado libere la placa.

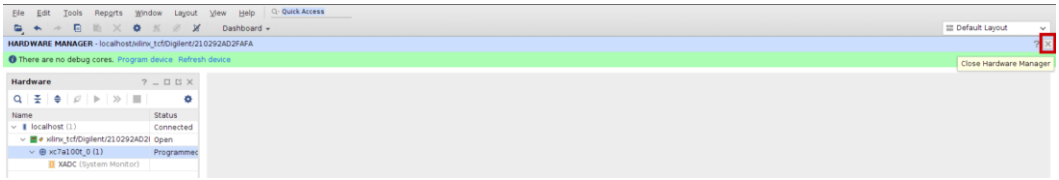



**Figura 114. Cierre del Administrador de Hardware**

## Cómo volver a los drivers usados por Vivado en Windows

Desafortunadamente, en Windows, los drivers de la placa FPGA Nexys A7 son diferentes para Vivado y PlatformIO. Se recomienda encarecidamente **utilizar PlatformIO para programar la FPGA, como se explica en la sección 5.A de esta Guía de inicio**. Sin embargo, si desea utilizar Vivado para descargar archivos bitfile, debe revertir los drivers que instaló en el Apéndice B y volver a los drivers de Vivado (FTDI) para la placa FPGA Nexys A7. Para ello, abra el Device Manager haciendo clic en el menú Inicio, escribiendo Device Manager en el cuadro de búsqueda y haciendo clic en Device Manager (ver Figura 115).

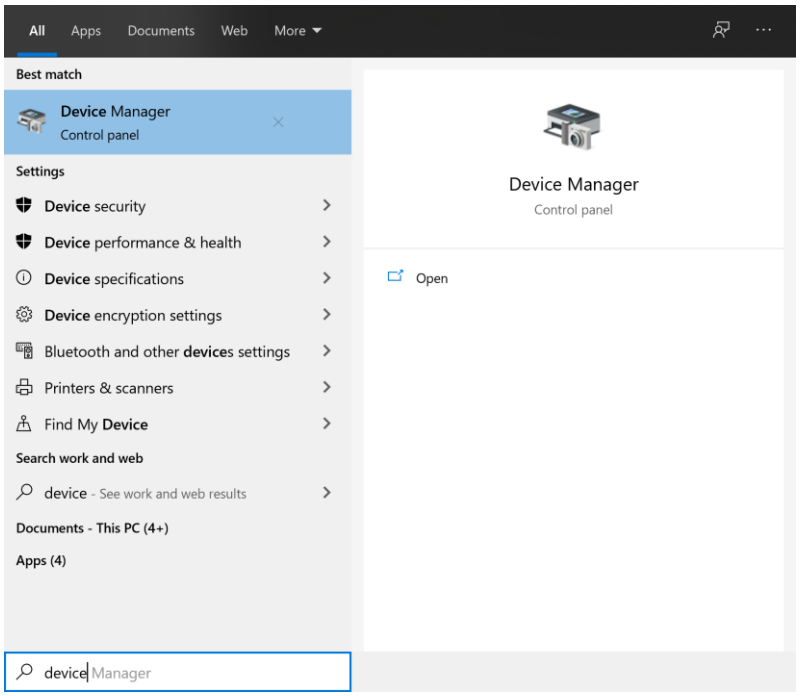



**Figura 115. Abrir el Device Manager (Administrador de Dispositivos)**

A continuación, amplíe el campo de Universal Serial Bus devices, haga clic con el botón derecho en Digilent USB Device y seleccione Properties (véase Figura 116).

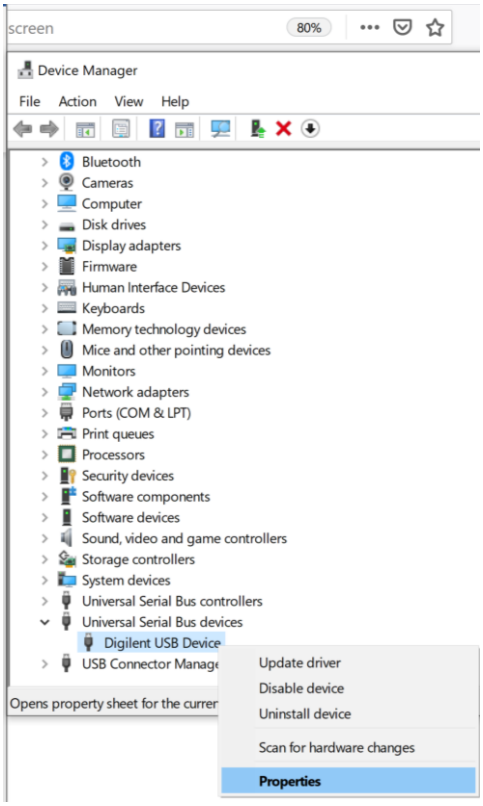



**Figura 116. Propiedades del driver para la placa FPGA Nexys A7 de Digilent**

En la ventana de Propiedades, haga clic en la pestaña Driver y seleccione Roll Back Driver (ver Figura 117).

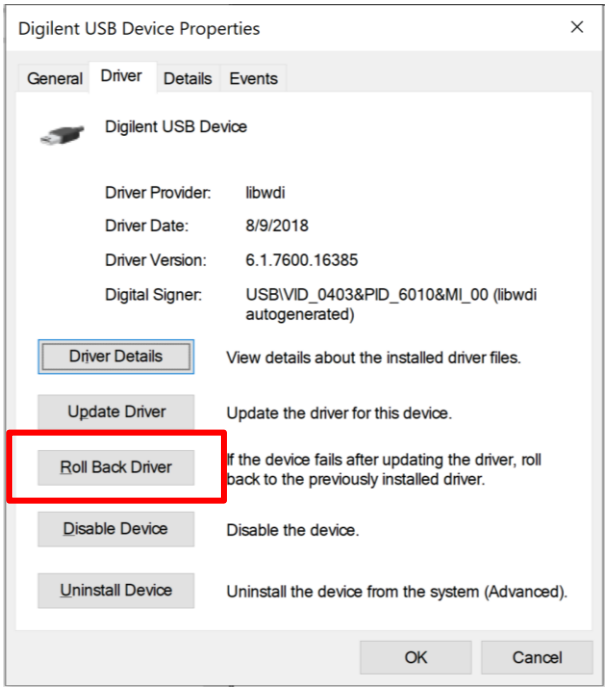



**Figura 117. Regreso al driver original**

Aparecerá una ventana preguntando por qué está revirtiendo el driver. Seleccione una razón y haga clic en Sí (ver Figura 118).

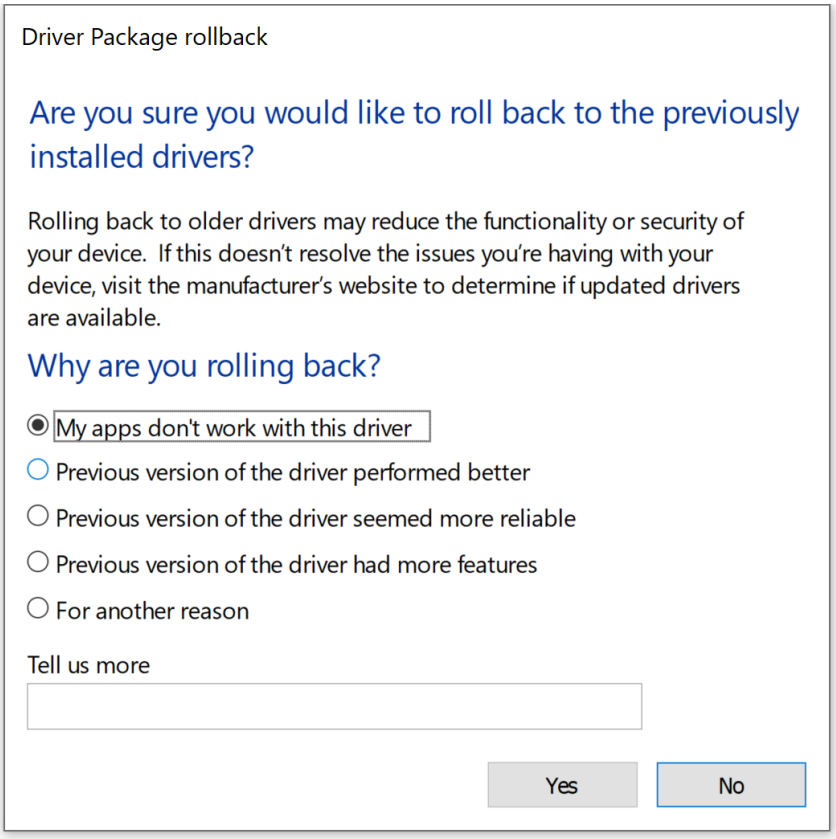



**Figura 118. Confirmación de la reversión**

Después de que se vuelva al driver anterior, el Proveedor del Driver debe figurar como FTDI (véase Figura 119).

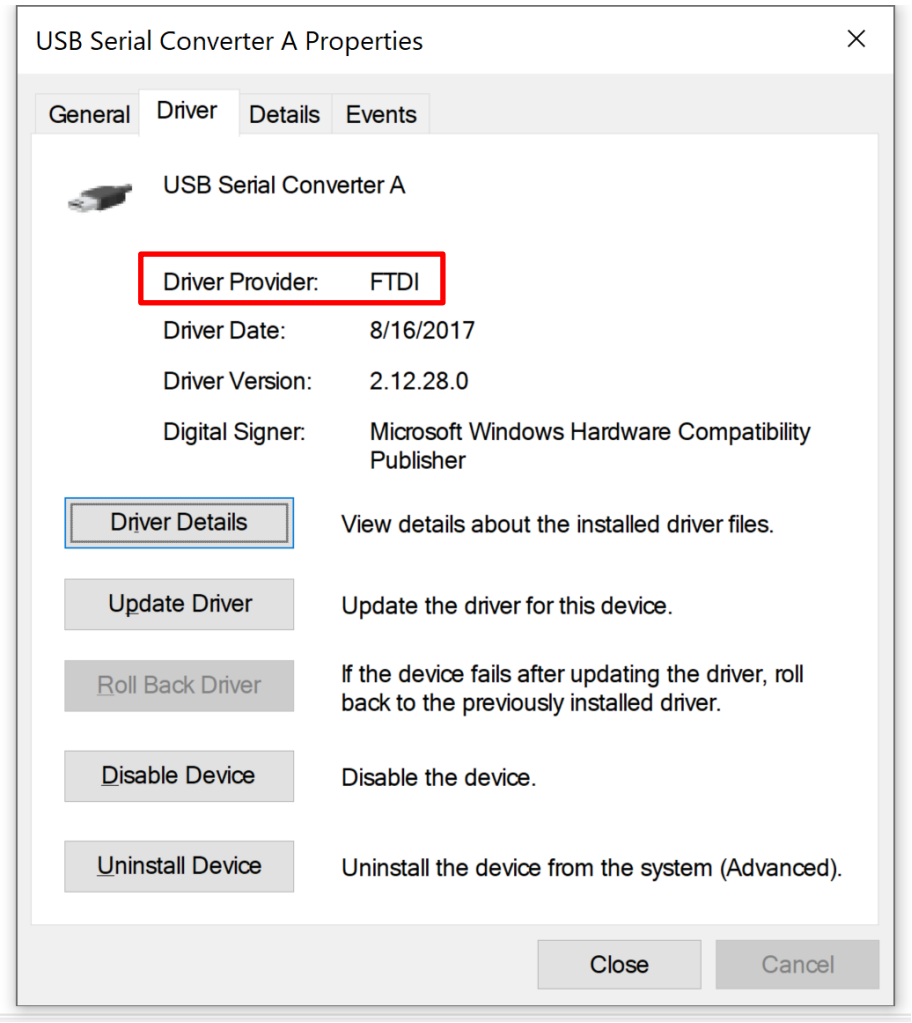



**Figura 119. FTDI se muestra como proveedor del driver**

Ahora puede cargar archivos bitfile en la placa FPGA usando Vivado. Sin embargo, todavía tendrá que usar Zadig para reemplazar el driver de la placa Nexys A7, de modo que PlatformIO pueda descargar el programa en RVfpgaNexys. Por lo tanto, se recomienda que use PlatformIO para descargar los archivos bitfile también (en lugar de usar Vivado), esto le evitará tener que intercambiar continuamente los drivers.

# Apéndice F: Uso de RVfpga en una aplicación de IoT industrial

En julio de 2020, Daniel León González, estudiante de maestría de la Universidad Complutense de Madrid, terminó su trabajo fin de máster titulado “Implementación FPGA de un SoC basado en RISC-V para IoT industrial”. Este trabajo muestra el uso de RVfpga en una aplicación real de IoT industrial. A continuación se presenta el resumen del proyecto, estando el trabajo completo disponible en: https://eprints.ucm.es/62106/1/DANIEL_LEON_GONZALEZ_DL_-_FPGA_Implementation_of_an_ad-hoc_RISC-V_SoC_for_Industrial_IoT__Graded__4286351_962908330.pdf.

### Implementación FPGA de un SoC basado en RISC-V para IoT industrial

**Resumen:** Los dispositivos de nodo para IoT necesitan, generalmente, ser eficientes energéticamente y tener un coste contenido, pero no precisan de una gran potencia de cómputo en un gran número de escenarios. Esto cambia sustancialmente en un entorno de IoT Industrial, donde los requerimientos sensoriales y de tiempo de respuesta precisan de una potencia de cálculo mayor. Un nodo desarrollado a medida, sobre un procesador eficiente y un sistema operativo de altas capacidades, puede balancear estos requerimientos ofreciendo una solución óptima. Este trabajo aborda la implementación hardware, sobre FPGA Artix-7, de un prototipo de nodo IIoT basado en la arquitectura de procesador RISC-V. El proyecto presenta la creación de un System-on-Chip a medida y el desarrollo de los drivers necesarios sobre el sistema operativo Zephyr para soportar una aplicación de prueba de concepto, que se despliega en una red de estrella con un rúter de borde. Mensajes de extremo a extremo pueden ser enviados y recibidos entre el nodo y la plataforma ThingSpeak en la nube. El documento incluye un análisis de las implementaciones existentes de procesadores RISC-V, una descripción de los elementos necesarios y una guía detallada de configuración de entorno y pasos para construir el proyecto completo.