



IMAGINATION大學計劃

RVfpga 入門指南

致謝

iup Imagination university programme				Imagination			
AUTHORS		CONTRIBUTORS		ASSOCIATES			
Prof. Sarah Harris		Robert Owen		Prof. José Ignacio Gómez		Prof. Francisco Tirado	
Prof. Daniel Chaver		Olof Kindgren		Prof. Christian Tenllado		Prof. Román Hermida	
Zubair Kakakhel		Prof. Luis Piñuel		Prof. Daniel León		Prof. Ataur Patwary	
M. Hamza Liaqat		Ivan Kravets		Prof. Katzalin Olcoz		Cathal McCabe	
		Valerii Koval		Prof. Alberto del Barrio		Dan Hugo	
		Ted Marena		Prof. Fernando Castro		Braden Harwood	
		Prof. Roy Kravitz		Prof. Manuel Prieto		Prof. David Burnett	
						Gage Elerding	
						Prof. Brian Cruickshank	
						Deepen Parmar	
						Thong Doan	
						Oliver Rew	
						Niko Nikolay	
						Guanyang He	

ADVISER
Prof. David Patterson

Sponsors and Supporters



作者

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

顧問

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

貢獻者

- Robert Owen (<https://www.linkedin.com/in/robert-owen-4335931/>)
- Olof Kindgren (<https://www.linkedin.com/in/olofkindgren/>)
- Luis Piñuel教授 (<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/>)
- Roy Kravitz教授 (<https://www.linkedin.com/in/roy-kravitz-4725963/>)

聯合作者

- José Ignacio Gómez教授 (<https://www.linkedin.com/in/jos%C3%A9-ignacio-gomez-182b981/>)
- Christian Tenllado教授 (<https://www.linkedin.com/in/christian-tenllado-31578659/>)
- Daniel León教授 (<https://www.linkedin.com/in/danileon-ufv/>)
- Katzalin Olcoz教授 (<https://www.linkedin.com/in/katzalin-olcoz-herrero-5724b0200/>)
- Alberto del Barrio教授 (<https://www.linkedin.com/in/alberto-antonio-del-barrio-garc%C3%ADa-1a85586a/>)
- Fernando Castro教授 (<https://www.linkedin.com/in/fernando-castro-5993103a/>)
- Manuel Prieto教授 (<https://www.linkedin.com/in/manuel-prieto-matias-02470b8b/>)
- Francisco Tirado教授 (<https://www.linkedin.com/in/francisco-tirado-fern%C3%A1ndez-40a45570/>)
- Román Hermida教授 (<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/>)
- PAtaur Patwary教授 (<https://www.linkedin.com/in/ataurpatwary/>)

目錄

致謝	2
0. 前言	4
1. 簡介	5
2. 快速入門指南	9
3. RISC-V架構概述	18
4. RVFPGA系統概述	20
5. 安裝軟體工具	37
6. 執行RVfpgaNexys並對其進行程式設計	43
7. VERILATOR模擬	73
8. WHISPER模擬	79
9. 附錄	81

更新歷史：

- 版本1.0（2020年11月發佈）：
 - o RVfpga課程的原始版本。
- 版本1.1（2021年6月發佈）：
 - o 在實驗00中新增了實驗11-20的說明。
 - o 將SweRVolf版本更新為0.7.3，將Verilator版本更新為4.106。
 - o 新增了啟動ROM初始化程式。
 - o 在說明RVfpga系統的GSG中新增了圖1和表1
 - o 在實驗10中新增了UART練習。
 - o 修正了一些文字錯誤。
- 版本2.0（2021年11月發佈）：
 - o 新增了實驗11-20：文件、圖、軟體來源、練習和解答。
 - o 補充了新增實驗的幻燈片。
 - o 在GSG和實驗0-10中新增了一些細節，並修正了一些文字錯誤。

0. 前言

RVfpga電腦體系結構課程透過具體實驗幫助使用者瞭解商用RISC-V處理器、RISC-V SoC和RISC-V生態系統。本課程按照以下順序介紹RISC-V系統：從基礎數位設計和訊號到指令集架構和處理器，再到程式設計環境、啟動程式碼和編譯器。使用者可以透過RVfpga課程全面的瞭解RISC-V系統。使用者不僅可以瞭解RISC-V SoC和RISC-V生態系統的工作狀態，還能夠掌握如何使用並延伸RISC-V處理器和系統來支援未來的專案和研究。

David Patterson教授（因對RISC的貢獻而與John Hennessy共獲ACM A.M.杜林獎）表示：「RISC-V正在推動處理器設計以及軟體/硬體協同設計發生巨大變革。RISC-V是一種支援開放原始碼硬體實作的開放式架構。這種全新設計意味著軟體開發可與硬體開發同步進行，進而加快設計速度。RVfpga課程可加強對RISC-V處理器、RISC-V生態系統和RISC-V SoC的瞭解。本課程可幫助使用者深入瞭解日益普及的工業級處理器架構和系統，這將在他們的整個學術生涯和職業生涯中發揮巨大作用。」

1. 簡介

重要資訊：在開始之前，將從Imagination大學計劃下載的**RVfpga**資料夾複製到您的Ubuntu/Windows/macOS電腦。我們將此RVfpga資料夾所在目錄的絕對路徑稱為[RVfpgaPath]。

RVfpga包含兩個文件（**幻燈片**和**本入門指南**）和五個資料夾：

- (1) **範例**：使用本指南時將執行的範例程式
- (2) **src**：包含RVfpga系統的原始程式碼（Verilog和SystemVerilog）（參見下面的圖1）
- (3) **verilatorSIM**：包含用於在Verilator中執行RVfpgaSim模擬的指令碼
- (4) **driversLinux_NexysA7**：包含Nexys A7 FPGA電路板的Linux驅動程式
- (5) **實驗**：包含RVfpga實驗1-20期間將使用的程式、文件、圖表、說明、任務、練習和解答。實驗0是一份介紹性文件，其中包含RVfpga實驗概述，您應該在開始實驗之前閱讀該文件。

RISC-V FPGA（也稱為RVfpga）是一個包含使用者指導、工具和實驗的套件，可將商用RISC-V處理器應用於可現場程式化閘道陣列（Field Programmable Gate Array，FPGA）和模擬器，然後使用並擴展此處理器來學習電腦體系結構、數位設計、嵌入式系統和程式設計方面等相關知識。

「RVfpga入門指南」包含以下各節，簡述如下：

- **快速入門指南**（第2節）
- **背景和概述**
 - **RISC-V架構**（第3節）
 - **RVfpga系統**（第4部分）
- **在硬體中使用RVfpga系統**
 - **安裝軟體工具**（第5節）
 - **執行RVfpga系統並對其進行程式設計**（第6部分）
- **RVfpga系統模擬**
 - **使用Verilator**（一種HDL模擬器）（第7節）
 - **使用Whisper**（Western Digital的指令集模擬器）（第8節）
- **附錄**
 - **使用配套的RISC-V工具鏈和OpenOCD**（附錄A）
 - **在Windows中安裝並使用PlatformIO所需的驅動程式**（附錄B）
 - **在Windows中安裝Verilator和GTKWave**（附錄C）
 - **在macOS中安裝Verilator和GTKWave**（附錄D）
 - **使用Vivado將RVfpga系統載入到FPGA上**（附錄E）
 - **範例：在工業物聯網應用中使用RVfpga**（附錄F）

「快速入門指南」（第2節）介紹RVfpga所需安裝的最少軟體，並說明如何基於RVfpga系統下載和執行簡單的程式範例。如需更全面瞭解RVfpga，請跳過第2節，從第3節的完整指南開始。

第3部分簡要介紹RISC-V電腦架構。第4部分介紹RVfpga系統（第4.A – 4.C部分）和構成系統的Verilog檔案的組織結構（第4.D部分）。RVfpga系統是基於SweRVolf SoC

（<https://github.com/chipsalliance/Cores-SweRVolf>），同時後者又使用Western Digital（WD）的開放原始碼RISC-V SweRV EH1核心（<https://github.com/chipsalliance/Cores-SweRV>）。

圖1和表1說明RVfpga系統的層級結構（從SweRV EH1核心開始，一直到RVfpgaNexys和RVfpgaSim）。

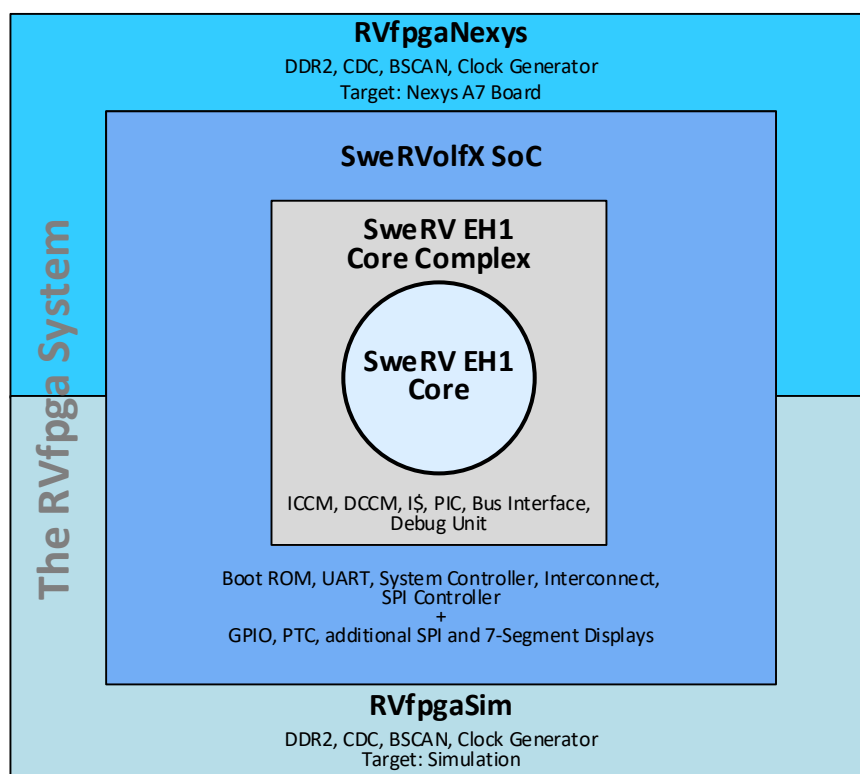


圖1. RVfpga系統層級

表1. RVfpga系統層級

名稱	說明
SweRV EH1核心	由Western Digital開發的開放原始碼商用RISC-V核心（ https://github.com/chipsalliance/Cores-SweRV ）。
SweRV EH1核心組合	一種新增了記憶體（ICCM、DCCM和指令快取）、可程式設定中斷控制器（Programmable Interrupt Controller，PIC）、匯流排介面和除錯單元的SweRV EH1核心（ https://github.com/chipsalliance/Cores-SweRV ）。
SweRVolfX（延伸SweRVolf）	我們在RVfpga課程中使用的晶片上系統。它是SweRVolf的延伸。 SweRVolf （ https://github.com/chipsalliance/Cores-SweRVolf ）：一種圍繞SweRV EH1核心組合建構的開放原始碼SoC。它新增了啟動ROM、UART介面、系統控制器、互連（AXI互連、Wishbone互連和AXI轉Wishbone橋接）以及SPI控制器。 SweRVolfX ：與SweRVolf相比新增了4個新周邊設備：GPIO、PTC、一個額外的SPI以及用於8位元7段顯示器的控制器。
RVfpgaNexys	以Nexys A7電路板及其周邊設備為目標的SweRVolfX SoC。它新增了DDR2介面、CDC（跨時脈變換）單元、BSCAN邏輯（用於JTAG介面）和時鐘產生器。 RVfpgaNexys與SweRVolf Nexys基本相同（ https://github.com/chipsalliance/Cores-SweRVolf ），只是後者基於SweRVolf。

RVfpgaSim	SweRVolfX SoC具有配套的測試平台和用於模擬的AXI記憶體。 RVfpgaSim與SweRVolf Sim基本相同 (https://github.com/chipsalliance/Cores-SweRVolf)，只是後者基於SweRVolf。
------------------	--

其餘部分說明如何將RVfpga系統用於硬體（RVfpgaNexys）和模擬（RVfpgaSim）。第5節說明如何安裝並使用RVfpga所需的軟體工具。第6部分說明如何使用PlatformIO將RVfpgaNexys下載到Nexys A7 FPGA電路板上（第6.A部分），以及基於RVfpgaNexys執行多個範例程式（第6.B-6.H部分）。第7部分和第8部分說明如何使用開放原始碼的HDL模擬器Verilator（第7部分）和Western Digital的RISC-V指令集模擬器（Instruction Set Simulator，ISS）Whisper（第8部分）模擬RVfpgaSim。

最後，附錄說明了如何在Linux中命令行介面下使用RVfpga（附錄A）、如何在Windows和macOS電腦上安裝所需的驅動程式和軟體（附錄B-D），以及如何使用Vivado將RVfpgaNexys下載到FPGA上（使用Vivado）（附錄E）。附錄F說明了如何在工業物聯網應用中使用RVfpga。

表2列出了RVfpga所需的軟體和硬體。本指南說明如何在Ubuntu 18.04作業系統（Operating System，OS）上安裝和使用這些工具及硬體。請對於其他作業系統（例如Windows或macOS）遵循類似（但是不完全相同）的步驟。若操作方式有所不同，我們會利用突出顯示功能為Windows和macOS插入具體說明。

附註：如果無法使用Nexys A7 FPGA電路板，仍然可以使用Western Digital的指令集模擬器（ISS）Whisper和開放原始碼HDL模擬器Verilator，透過模擬以完成實驗。在這種情況下，無需安裝Vivado（第5.A節）；只需安裝VSCode/PlatformIO（如第2.A節所述）和Verilator/GTKWave（如第5.C節所述）。

表2. RVfpga所需的軟體和硬體

軟體		
名稱	網站	費用
Vivado 2019.2 WebPACK	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html	免費
VSCode	https://code.visualstudio.com/Download	免費
PlatformIO	https://platformio.org/ 安裝在VSCode中	免費
Verilator（一種HDL模擬器） 和GTKWave	https://github.com/verilator/verilator http://gtkwave.sourceforge.net/	免費
Whisper（Western Digital的 RISC-V指令集模擬器）	https://github.com/chipsalliance/SweRV-ISS 安裝在PlatformIO中	免費
RISC-V工具鏈和OpenOCD	https://github.com/riscv/riscv-gnu-toolchain https://github.com/riscv/riscv-openocd 安裝在PlatformIO中	免費

硬體		
名稱	網站	費用
Nexys A7 FPGA電路板*	https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/	265美元 (學術優惠價： 199美元)
RISC-V核心和晶片上系統 (SoC) **		
名稱	網站	費用
Western Digital的SweRV EH1核心	https://github.com/chipsalliance/Cores-SweRV	免費
SweRVolf	https://github.com/chipsalliance/Cores-SweRVolf	免費

*本指南中描述的所有步驟也適用於Digilent的Nexys4 DDR FPGA電路板。

**從Imagination Technologies下載RVfpga時一併提供。

知識儲備要求：

在完成本RVfpga課程（包括本RVfpga入門指南和RVfpga實驗）之前，使用者至少應對以下內容有基本瞭解：

- 數位邏輯設計
- 高階程式設計（最好是C語言）
- 組合語言程式設計
- 指令集架構
- 處理器微架構
- 記憶體系統

這些內容已包含在以下教科書中：《數位設計和電腦結構》（RISC-V版），Harris & Harris，© Morgan Kaufmann（預計出版時間：2021年夏季）。其他教科書（包括《電腦組織結構和設計》（RISC-V版本），Patterson & Hennessy，© Morgan Kaufmann 2017）涵蓋其中一些內容。

2. 快速入門指南

本節依次說明了如何安裝使用RVfpga所需的必須工具，以及如何使用PlatformIO將RVfpgaNexys下載到Nexys A7 FPGA電路板上，然後在RVfpgaNexys上執行程式。同時需要購買FPGA電路板（請參閱表2）。這些步驟也適用於早期的電路板型號Nexys4-DDR FPGA電路板。

- A. 最低安裝要求：VSCode、PlatformIO和Nexys A7電路板驅動程式
- B. 將RVfpgaNexys下載到FPGA上並在RVfpgaNexys上執行程式

以下說明適用於Ubuntu 18.04系統。此外，也適用於Windows10和macOS作業系統 – 若操作與Ubuntu不同，我們會針對**Windows**和**macOS**插入包含具體說明的文字方塊。如果使用的是Ubuntu，則可以忽略這些文字方塊。路徑書寫形式如下：Linux路徑使用正斜線（/），Windows路徑通常相同，但帶有反斜線（\）。

A. 最低安裝要求：VSCode、PlatformIO和Nexys A7電路板驅動程式

在此步驟中，將安裝使用RVfpga所需的最少軟體和驅動程式。首先將安裝程式設計環境，然後安裝Nexys A7 FPGA電路板的驅動程式。

VSCode和PlatformIO安裝：將使用整合開發環境（Integrated Development Environment，IDE）PlatformIO將RVfpgaNexys下載到Nexys A7電路板上，然後在RVfpgaNexys上下載並執行程式。PlatformIO編譯為Microsoft的Visual Studio Code（VSCode）的延伸模組。PlatformIO支援跨平台操作，包含一個內建偵錯工具。

按照以下步驟安裝VSCode和PlatformIO：

1. 安裝VSCode：

- a. 從以下連結下載安裝檔案：<https://code.visualstudio.com/Download>

- b. 開啟終端機，然後安裝並執行VSCode：

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

Windows/macOS：VSCode套件也支援Windows（.exe檔案）和macOS（.zip檔案），可通過<https://code.visualstudio.com/Download>取得。請按照這些作業系統中用於安裝和執行應用程式的常規步驟操作。

2. 在VSCode軟體上安裝PlatformIO：

- a. 通過在終端機中輸入以下指令來安裝python3公程式：

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

Windows/macOS：Windows中不需要執行這一步（2.a）。對於macOS，可以使用**homebrew**來安裝python3：`brew install python3`

- b. 如果VSCode尚未開啟，請按照以下步驟將其啟動：選擇「Start」（開始）按鈕並在搜尋功能表中輸入「VSCode」，然後選擇「VSCode」，或者在Ubuntu終端機中輸入code。


- c. 在VSCode中，按一下VSCode左側欄中的「Extensions」（延伸模組）圖示（請參閱圖2）。



圖2. VSCode的「Extensions」（延伸模組）圖示

- d. 在搜尋方塊中輸入PlatformIO，然後按一下PlatformIO IDE旁邊的「Install」（安裝）按鈕進行安裝（請參閱圖3）。

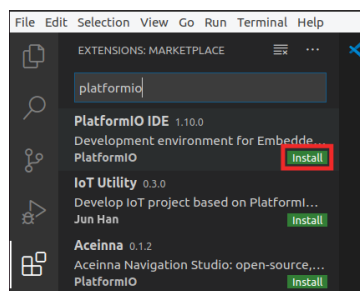


圖3. PlatformIO IDE延伸模組

- e. 底部的「OUTPUT」（輸出）視窗將通知使用者相關的安裝進度。安裝完畢後，按一下視窗右下角的「Reload Now」（立即重新載入），PlatformIO將在VSCode內部完成安裝（請參閱圖4）。

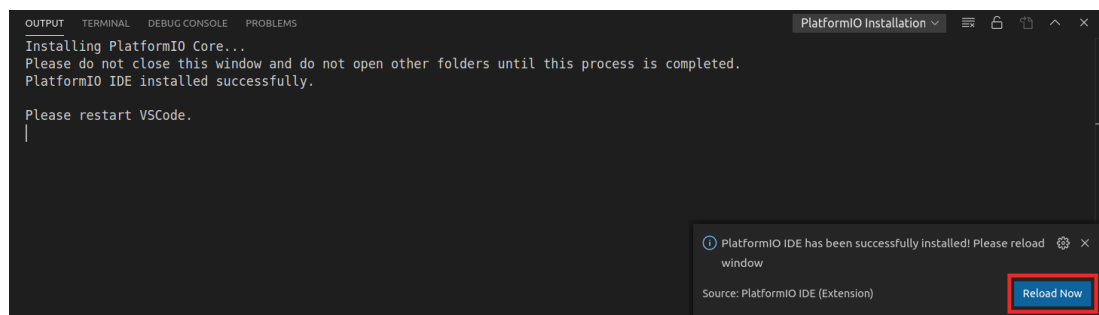


圖4. 安裝PlatformIO後立即重新載入

Nexys A7驅動程式安裝：需要手動安裝Nexys A7電路板的驅動程式。

- 開啟終端機。
- 移至目錄`[RVfpgaPath]/RVfpga/driversLinux_NexysA7`。（為簡便起見，我們在RVfpga資料夾內提供了這些驅動程式。此外，按照本指南第5節的說明安裝Vivado時，還可以在該節所述的下載套件內找到這些驅動程式。）

- 執行安裝指令碼：

```
chmod 777 *
sudo ./install_drivers
```
- 從電腦上拔掉Nexys A7電路板，然後重新啟動電腦，以使變更生效。

Windows：按照附錄B中的說明來安裝Nexys A7電路板的驅動程式。

macOS：不需要安裝任何其他驅動程式。

B. 將RVfpgaNexys下載到FPGA上並在RVfpgaNexys上執行程式

現在，將RVfpgaNexys（以FPGA為目標的RISC-V系統）下載到Nexys A7 FPGA電路板上。在本入門指南中，我們將不會對RVfpga系統進行修改，但[RVfpgaPath]/RVfpga/src中提供了RVfpga系統的Verilog。我們將在本指南的第4節介紹RVfpga系統的原始程式碼，並在RVfpga實驗6-20中加以詳細說明。此外，使用者還將透過這些實驗中的一些練習修改RVfpga系統。

透過完成以下步驟，在Nexys A7 FPGA電路板上執行RVfpgaNexys：

- 第1步. 將Nexys A7 FPGA電路板連接到電腦並開啟電路板
- 第2步. 開啟PlatformIO和C程式
- 第3步. 將RVfpgaNexys下載到Nexys A7電路板上
- 第4步. 在RVfpgaNexys上下載並執行程式

第1步. 將Nexys A7 FPGA電路板連接到電腦並開啟電路板

使用配套的USB電源線將Nexys A7電路板連接到電腦。圖5顯示了Nexys A7 FPGA電路板上的LED和開關，以及USB連接器、電源開關、按鈕和7段顯示器的物理位置。將USB電源線把電腦和Nexys A7電路板的USB連接器連接埠進行連接，然後開啟電路板開關。

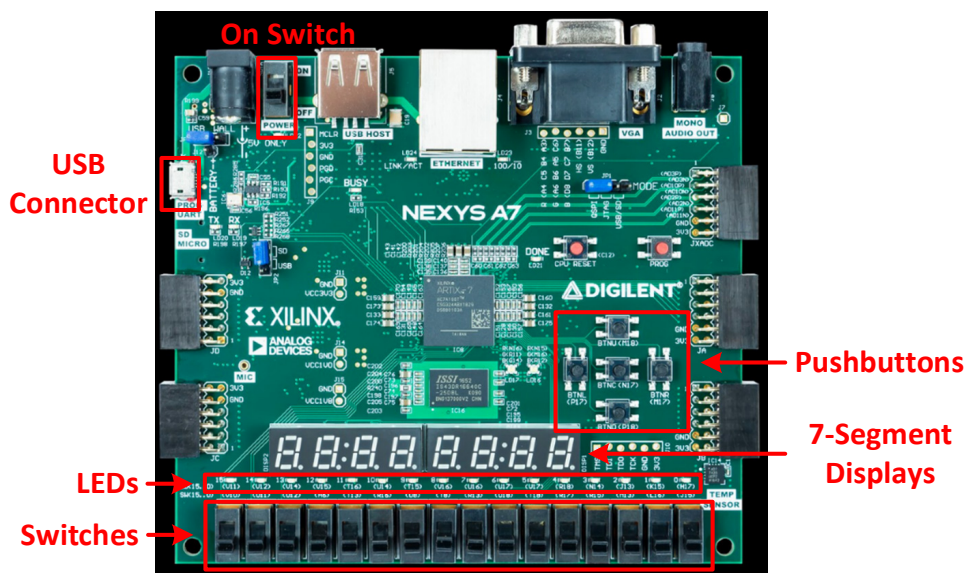


圖5. Digilent的Nexys A7 FPGA電路板的I/O介面

（電路板圖片來源：<https://reference.digilentinc.com/>）

第2步. 開啟PlatformIO和C程式

現在，通過以下方式開啟Visual Studio Code (VSCode)：在「Start」（開始）功能表中輸入VSCode（請參閱圖6），或者在終端機中輸入code。

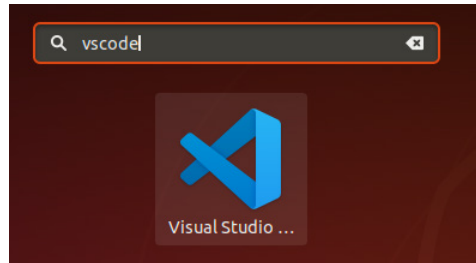



圖6. 開啟VSCode

如果PlatformIO首頁（「PIO Home」（PIO首頁））視窗沒有自動開啟，請按一下左側功能區功能表中的PlatformIO圖示：。然後展開「PIO Home」（PIO首頁），並按一下「Open」（開啟）。此時，「PIO Home」（PIO首頁）將開啟並顯示「Welcome」（歡迎）視窗（請參閱圖7）。

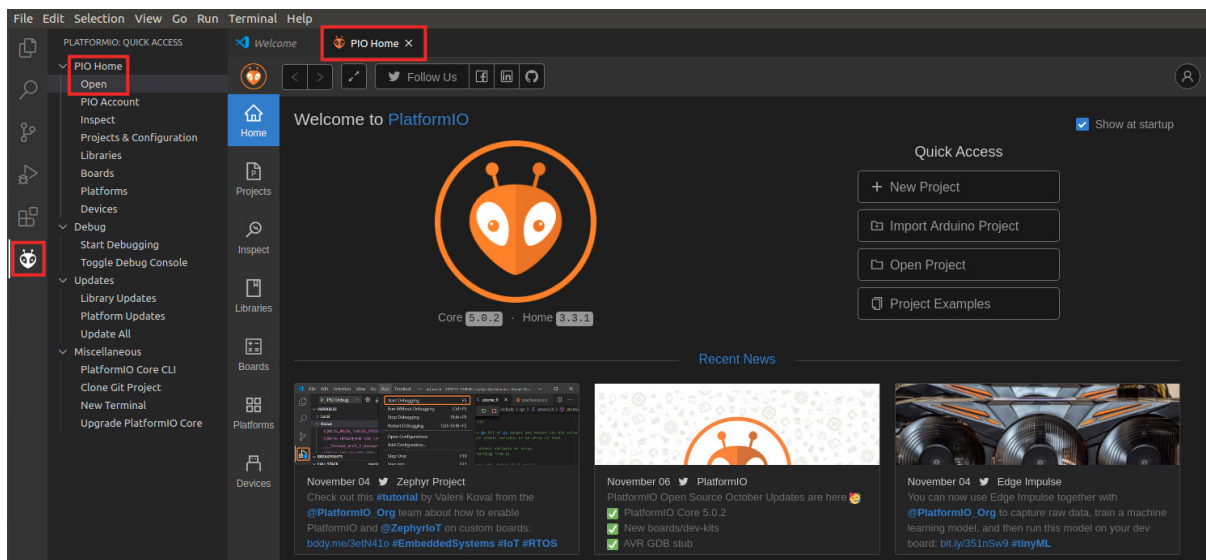


圖7. 開啟「PIO Home」（PIO首頁）

現在按一下頂端檔案功能表的「File」（檔案）→「Open Folder」（開啟資料夾），然後選擇：

`[RVfpgaPath]\RVfpga\examples\LedsSwitches_C-Lang`

選擇資料夾，但不要將其開啟（請參閱圖8）。PlatformIO現在將開啟程式LedsSwitches_C-Lang，該程式會讀取Nexys A7電路板上的開關值並將這些值寫入到電路板上的LED。

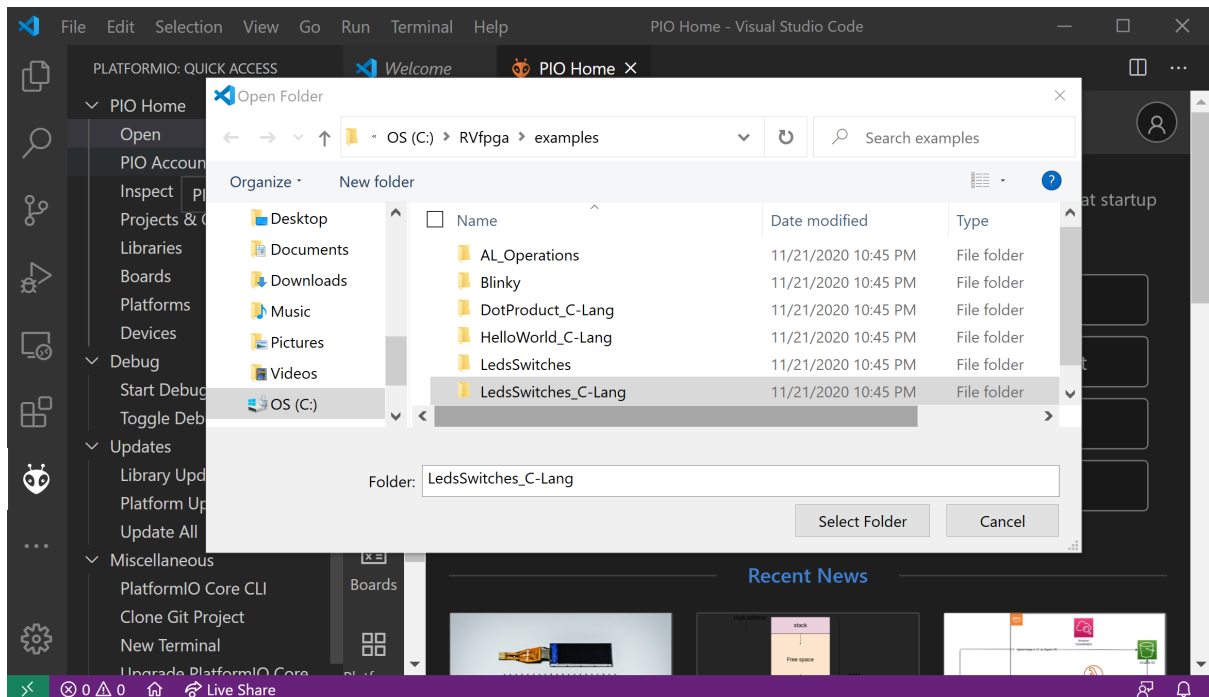


圖8. 開啟LedsSwitches_C-Lang範例

您可透過展開src資料夾，然後按兩下LedsSwitches_C-Lang.c來檢視LedsSwitches_C-Lang程式（圖9）。我們將在本入門指南的稍後章節詳細討論該程式。在本「快速入門指南」中，我們只會將此程式下載到將在Nexys A7電路板上執行的RVfpgaNexys。

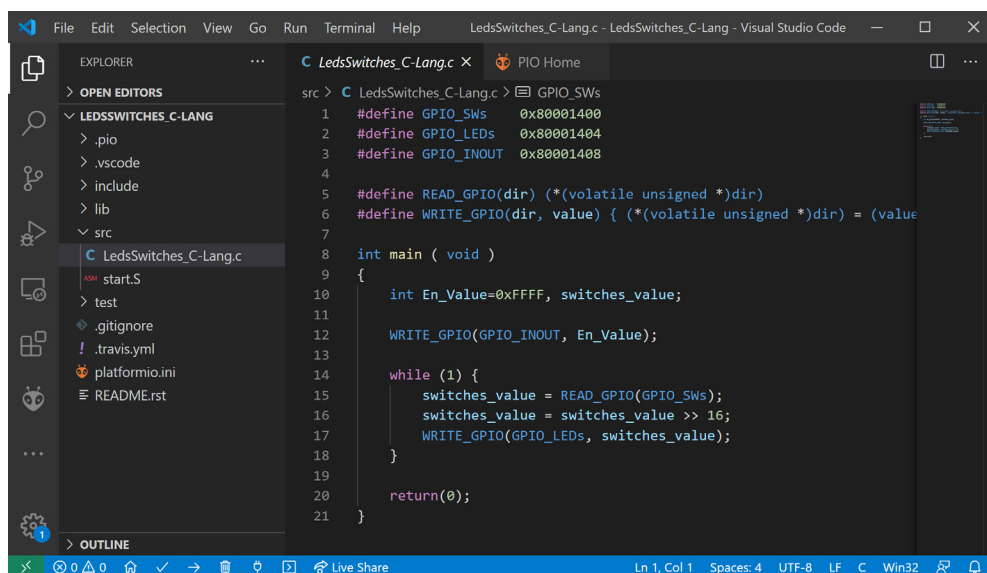


圖9. LedsSwitches_C-Lang.c程式

請注意，首次在PlatformIO中開啟RVfpga範例時會自動安裝Chips Alliance平台（可以在「PIO Home」（PIO首頁）→「Platforms」（平台）中檢視該平台，如圖10所示）。該平台包含以後將用到的幾個工具，例如預編譯的RISC-V工具鏈、用於RISC-V的OpenOCD、RVfpgaNexys bit檔和RVfpgaSim、JavaScript和Python指令碼以及一些範例。如果由於某種原因未自動安裝Chips Alliance平台，則可以手動完成安裝，具體過程請參閱第6.A節。

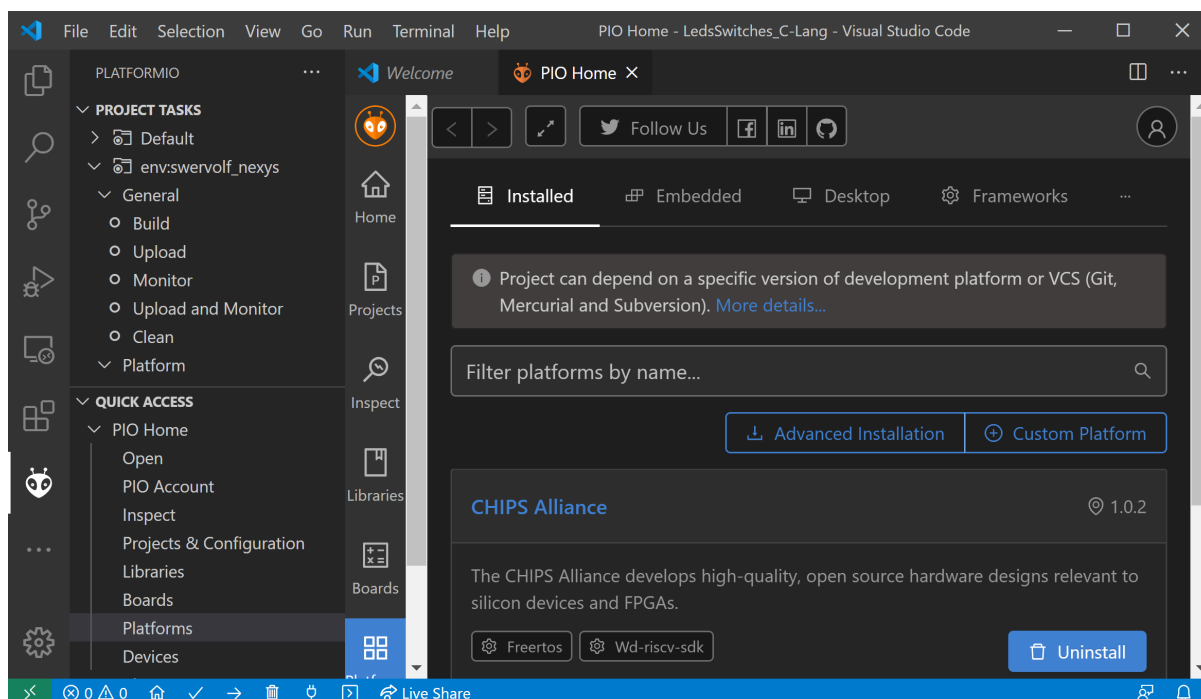



圖10. PlatformIO中安裝的Chips Alliance平台

第3步. 將RVfpgaNexys下載到Nexys A7電路板上

現在可以下載RVfpgaNexys，它是一個可以支援週邊設備的RISC-V處理器，屬於RISC-V SoC的一種。在「EXPLORER」（檔案總管）視窗中按兩下platformio.ini（PlatformIO初始化檔案）將其開啟，如圖11所示。（如果「EXPLORER」（檔案總管）視窗尚未開啟，請按一下左側功能區功能表中的將其開啟。）現在，通過將board_build.bitstream_file路徑替換為您自己的路徑，來新增用於定義RVfpgaNexys bit檔的位置路徑（請參閱圖11）：

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

按Ctrl-S儲存platformio.ini檔。

有多條命令可用於專案組態檔案（*platformio.ini*）；有關這些選項的更多資訊，請造訪：
<https://docs.platformio.org/en/latest/projectconf/>。

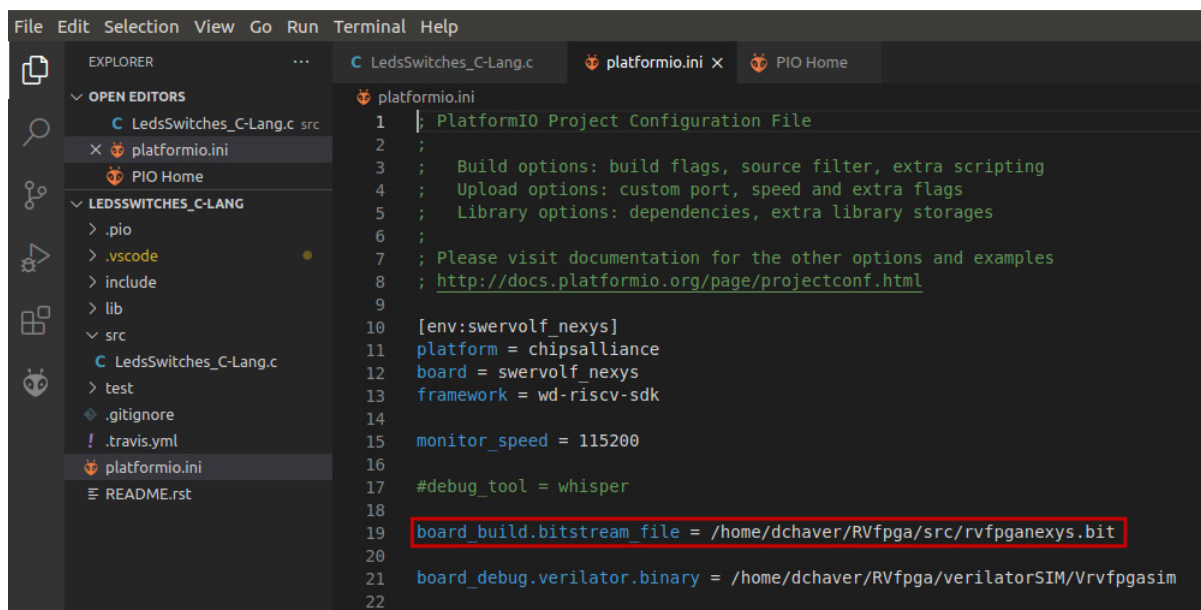


圖11. 在RVfpgaNexys bit檔案中新增路徑

將RVfpgaNexys（由該bit檔定義）下載到Nexys A7電路板：

- 按一下左側功能表功能區中的PlatformIO圖示（請參閱圖12）。

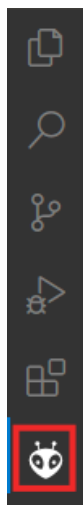



圖12. PlatformIO圖示

- 如果「Project Tasks」（專案任務）視窗為空（圖13），必須先按一下來重新整理「Project Tasks」（專案任務）。此程序可能需要幾分鐘。

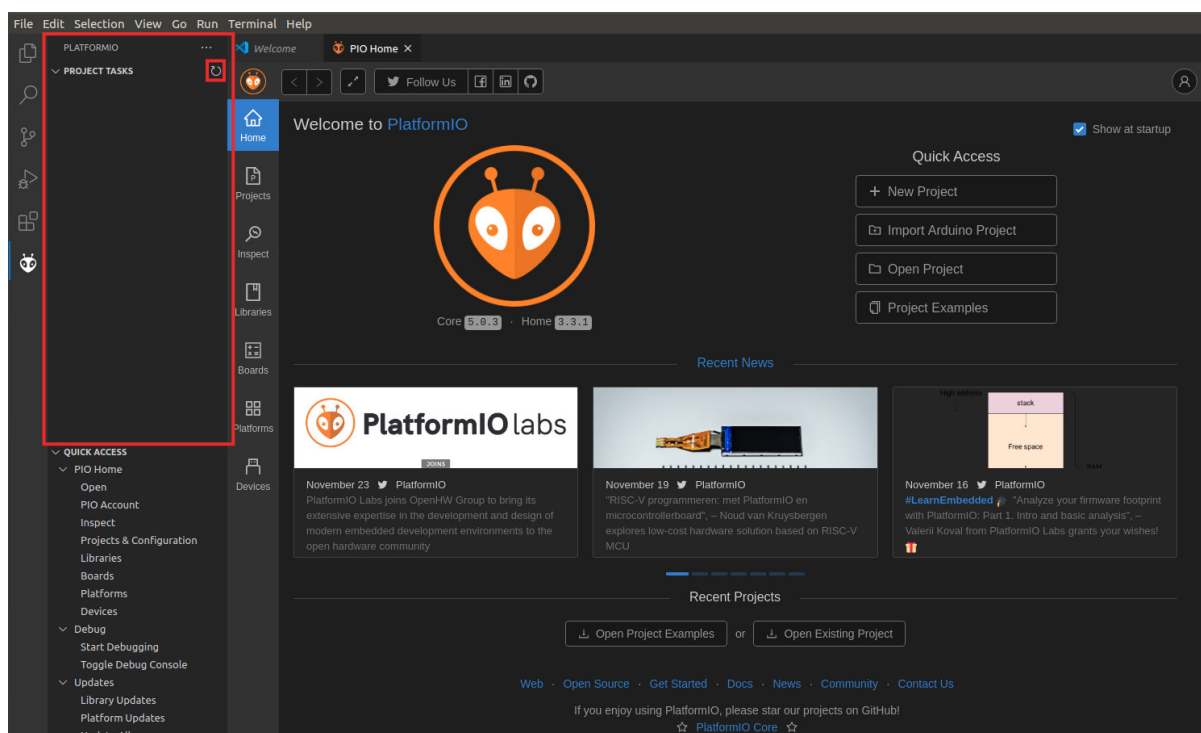


圖13.「PROJECT TASKS」（專案任務）視窗為空 – 重新整理

- 然後展開「Project Tasks」（專案任務）→ env:swervolf_nexys → 「Platform」（平台），再按一下「Upload Bitstream」（上傳位元串流），如圖14所示。一到兩秒後，將使用RVfpgaNexys SoC對FPGA進行程式設計。

預設情況下，處理器從位址0x80000000處開始擷取指令，其中啟動ROM位於我們的SoC中（參見表6）。啟動ROM使用一個程式（*boot_main.mem*）進行初始化，該程式使LED和7段顯示器閃爍四次，然後熄滅所有LED，將0寫入8個7段顯示器並保持空迴圈。該程式位於以下資料夾：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*。如果要進行更改和重新編譯，請按照附錄A – 第III部分中的說明進行操作（注意檔案*boot_main.mem*只是檔案*boot_main.vh*的副本）。在實驗1中，我們將介紹如何在建立位元串流時使用該程式初始化啟動ROM。

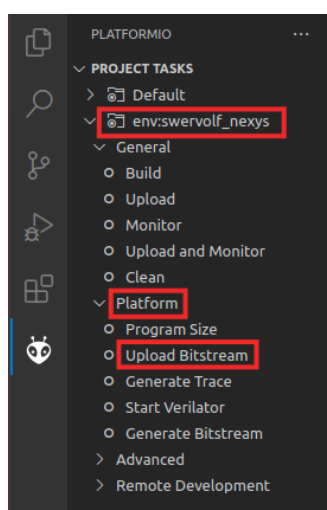




圖14. 上傳位元串流

第4步. 在RVfpgaNexys上下載並執行程式

RVfpgaNexys現在已下載到Nexys A7電路板上並執行，接下來會將程式下載到RVfpgaNexys的記憶體中，然後執行/偵錯該程式。按一下「Run and Debug」（執行和偵錯）按鈕：，該按鈕位於左側欄中。按一下播放按鈕 **PIO Debug** 啟動偵錯工具（確定已選擇「PIO Debug」（PIO偵錯）選項）。播放按鈕靠近視窗頂端位置（請參閱圖15）。程式將先編譯，然後開始偵錯。

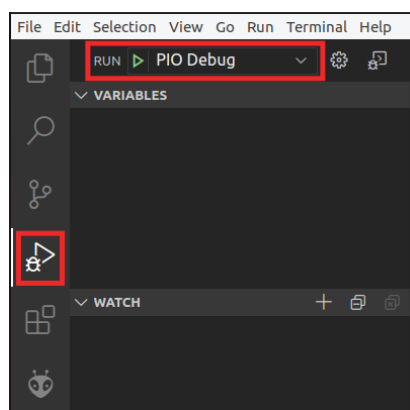



圖15. 編譯和下載程式，以及啟動偵錯工具

要控制偵錯工作階段，可使用編輯器頂端附近的偵錯工具列（請參閱圖16）。我們將在本入門指南的後續章節描述和測試所有選項。



圖16. 偵錯工具

PlatformIO在main函數的開頭設定一個臨時中斷點。因此，按一下「Continue」（繼續）按鈕可執行程式。現在開啟Nexys A7 FPGA電路板上的開關，並檢視相應的LED是否點亮。

3. RISC-V架構概述

RISC-V是一種指令集架構（Instruction Set Architecture，ISA），於2011年在加州大學伯克萊分校的Par實驗室中建立。RISC-V旨在成為各種應用範圍（涵蓋小型、受限、低資源物聯網裝置到超級電腦等各種應用）的處理器所適用的「通用ISA」。為實現這一目標，RISC-V架構師制定了五項架構原則：

- 架構必須與各種套件和程式設計語言相容。
- 架構必須能夠通過各種技術實作，包括FPGA到ASIC（專用積體電路）以及新興技術。
- 架構必須適用於各種微架構場景，包括實作微程式碼或硬聯線控制、順序或亂序管線、各種類型的並行性等。
- 必須能夠針對特定任務量身定制架構，以實現所需的最大效能，而不會因ISA本身而引起缺陷。
- 架構的基本指令集必須穩定且持久，從而為開發人員提供一個通用且穩固的架構。

RISC-V是一種開放標準，實際上，此規格隸屬於公共領域，自2015年起一直由**RISC-V Foundation**（現稱為**RISC-V International**，是一家促進RISC-V架構軟硬體開發的非營利組織）管理。RISC-V Foundation與Linux Foundation在2018年開展了持續的合作；2020年3月，RISC-V Foundation更名為RISC-V International，總部設於瑞士。這種轉變去除了社群對標準未來開放性的任何疑慮。截至2020年，RISC-V International得到了來自研究機構、學術界和工業界的200多家關鍵企業的支持，其中包括Microchip、NXP、Samsung、Qualcomm、Micron、Google、Alibaba、Hitachi、Nvidia、Huawei、Western Digital、ETH Zurich、KU Leuven、UNLV和UCM。

RISC-V是過去10-20年間創建的少數全球性相關ISA之一，而且可能是唯一的全球相關ISA，因為它是一種開放標準並採用模組化，而不是增量式的設計。這種模組化使其既靈活又簡潔。處理器實作基本ISA，並且僅實作所使用的延伸模組。這種模組化方法不同於採用增量式架構的傳統ISA（例如x86或ARM），它對之前的ISA進行了擴展，且每個新處理器必須實作所有指令（包括標記為「過時」的指令）以確保與舊版軟體程式相容。例如，x86從一開始具有80行指令到現在具有超過1300行指令，如果考慮機器程式碼中所有可用的不同作業碼，則具有3600條指令。由於大多數短作業碼或短指令已在使用中，因此為了處理大量指令和滿足回溯相容性等要求，必須提供能夠支援長指令的大型高能耗處理器。

RISC-V具有四個基本ISA選項：兩個32位元版本（整數和嵌入式版本，RV32I和RV32E）以及64位元和128位元版本（RV64I和RV128I），如表3所示。標記為「Ratified」（批准）的ISA模組現已經經過批准。標記為「Frozen」（凍結）的模組預計在提交批准之前不會發生重大變更。標記為「Draft」（草稿）的模組預計在批准之前會發生變化。對於成本、空間和資源有限的裝置而言，尤其需要具備建立小型處理器的能力。可以在這些基本ISA的上層新增指令延伸模組以執行特定任務，例如浮點數運算、乘法和除法，以及向量運算。這些專用的硬體延伸模組也包含在標準中，且編譯器已知其存在，因此能在編譯器中啟用所需的選項來產生目標二進位程式碼。其中每個延伸模組都用一個字母標識，該字母必須新增到核心ISA中以表示實作的硬體功能，如表4所示。例如，RVM是乘法/除法延伸模組，RVF是浮點數延伸模組等等。

表3. RISC-V基本ISA

(表格來自 <https://riscv.org/technical/specifications/>)

Base	Version	Status
RVWMO	2.0	Ratified
RV32I	2.1	Ratified
RV64I	2.1	Ratified
<i>RV32E</i>	<i>1.9</i>	<i>Draft</i>
<i>RV128I</i>	<i>1.7</i>	<i>Draft</i>

表4. RISC-V標準ISA延伸模組

(表格來自 <https://riscv.org/technical/specifications/>)

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

字母**G**表示「通用」，用於指示包含所有**MAFD**延伸模組。請注意，公司或個人可能會使用標準模組中沒有使用的作業碼來開發專有延伸模組。這樣有助於第三方加快實作，進而縮短其上市時間。

例如，包含全部通用ISA延伸模組以及位元操作和使用者級中斷的64位元RISC-V被稱為**RV64GBN ISA**。所有這些模組都符合非特權或使用者規格。此外，RISC-V基金會還規定了執行通用作業系統所需的特權操作的一組要求和指令。

4. RVFPGA系統概述

本節介紹整個RVfpga系統，包括從核心到FPGA電路板介面的各個部分。圖17說明了嵌入式系統的一般階層結構，從處理器核心開始，然後是圍繞核心編譯的SoC，最後是系統和電路板介面。

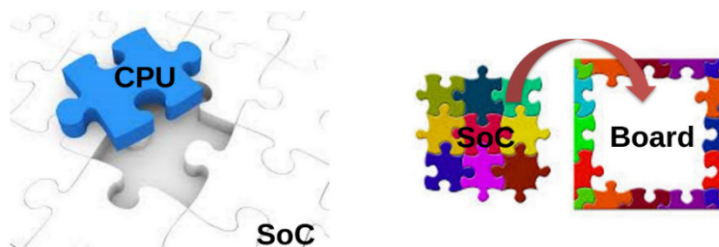


圖17. 嵌入式系統結構

圖1和表1說明系統的層級結構（從SweRV EH1核心開始，一直到RVfpgaNexys和RVfpgaSim）。在下面的小節中，我們首先介紹執行RISC-V指令的處理器核心（**Western Digital的SweRV EH1核心**）；接著在B節介紹**SweRVolfX SoC**，它整合了系統的硬體組件（核心、記憶體和輸入/輸出），以及為在RVfpga內使用此SoC而執行的延伸模組；在C節，介紹在Nexys A7 FPGA電路板（**RVfpgaNexys**）上實作的SweRVolfX SoC，以及在模擬（**RVfpgaSim**）中使用的SweRVolfX SoC。最後，在D節介紹整個RVfpga系統的檔案結構。

A. SweRV EH1核心和SweRV EH1核心組合

Western Digital在過去幾年中開發了三種RISC-V核心：**SweRV EH1**（RVfpga系統中使用的核心）、**SweRV EH2**和**SweRV EL2**（RVfpga的未來版本中可能包含這些核心）。每個核心都有一份Apache 2.0授權。**SweRV EH1**是一個32位核心，採用2路超標量和9階段管線設計。**SweRV Core EH2**是基於EH1核心基礎所建立，並對其進行擴展以支援雙執行緒功能，從而提升了效能。**SweRV Core EL2**是一種小型核心，可提供中等效能。RISC-V頁面（網址為<https://www.westerndigital.com/company/innovations/risc-v>）概述了這三種可用的核心，它們的主要特性如表5所示。

表5. 三種WD RISC-V核心的主要特性

（表格來自<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

在這三種核心中，**SweRV EH1核心**（隨RVfpga套件一起提供，也可從<https://github.com/chipsalliance/Cores-SweRV>取得）可提供高效能/MHz，並且採用簡單的執行緒結構，是首選核心。此外，致力於提供開放原始碼硬體的組織Chips Alliance提供了一個完整且經過驗證的SoC，稱為**SweRVolf**（隨RVfpga套件提供，也可從<https://github.com/chipsalliance/Cores-SweRVolf>取得）。RVfpga系統使用**SweRVolf SoC**的延伸，而後者又使用Western Digital的**SweRV EH1核心版本1.8**。

SweRV EH1核心是一種僅限機器模式（M模式）的32位元CPU核心，支援RISC-V的整數（I）、壓縮指令（C）以及整數乘法和除法（M）延伸模組。參考手冊

（https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf）詳細描述了核心從結構到時序資訊和記憶體映射的各個方面。

SweRV EH1是超標量核心，採用雙發射9階段管線設計（請參閱圖18），支援四個算術邏輯單元（Arithmetic Logic Unit, ALU），這些單元在兩條管線I0和I1中標記為EX1至EX4。兩路管線均支援ALU運算。其中一路管線支援載入/儲存，另一路具有3週期延遲乘法器。處理器還具有一個非管線式34週期的延遲除法器。管線中存在四個停頓點：「指令擷取1」、「對齊」、「解碼」和「提交」。「指令擷取1」階段包括Gshare分支預測器。在「對齊」階段，從三個緩衝區中進行指令擷取。在「解碼」階段，最多對四個指令緩衝區中的兩行指令進行解碼。在「提交」階段，每個週期最多提交兩行指令。最後在「回寫」階段，更新架構暫存器。

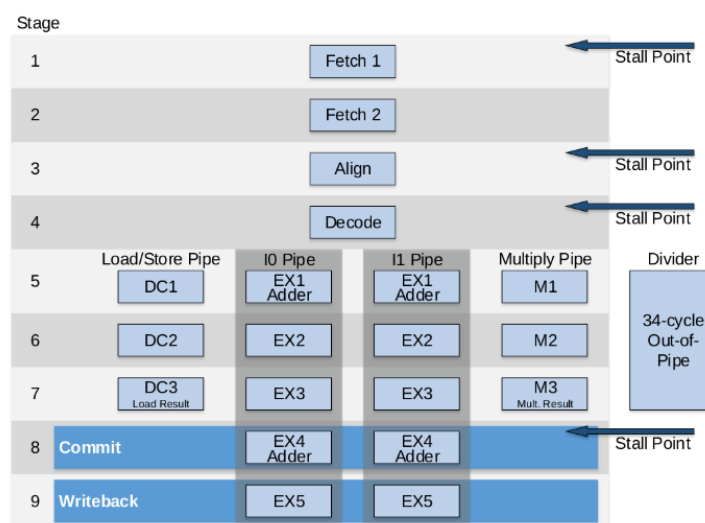


圖18. SweRV EH1核心微架構

（圖來自https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf）

圖19對目前的各種核心和處理器進行了比較。**SweRV EH1**核心的效能非常高，可達4.9 CM/MHz（CoreMark/MHz）：其執行速度為ARM Cortex A8的兩倍，效能甚至超過ARM Cortex A15。

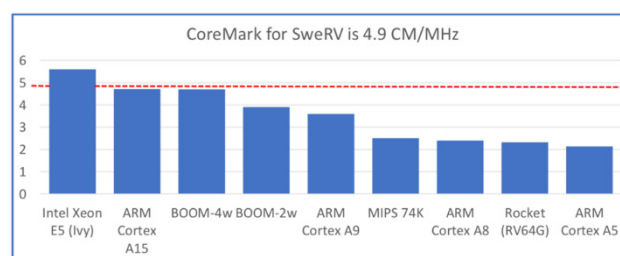


圖19. 單執行緒和頻率的基準比較

（圖來自https://content.riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD_SweRV_Cores_Roadmap_v4SCR.pdf）

Western Digital還提供了SweRV EH1核心的延伸模組，稱為**SweRV EH1核心組合**（請參閱圖20），此延伸模組在上述EH1核心的基礎上增加了以下元素，如圖中藍色部分所示：

- 兩個專屬記憶體，一個用於儲存指令（ICCM），另一個用於儲存資料（DCCM），它們與核心緊密耦合。這兩個記憶體提供低延遲存取和SECDED ECC（單一誤差校正和雙重誤差檢測糾錯碼）保護。每個記憶體可以配置為4、8、16、32、48、64、128、256或512KB。
- 可選的4向集關聯指令快取，具有同位檢查或ECC保護。
- 選用的可程式化插斷控制器（Programmable Interrupt Controller，PIC），最多支援255個外部中斷。
- 四個系統匯流排介面，用於指令擷取（IFU匯流排主控）、資料存取（LSU匯流排主控）、偵錯存取（偵錯匯流排主控）和外部DMA存取（DMA從屬連接埠）緊密耦合的記憶體（可配置為64位元AXI4或AHB-Lite匯流排）。
- 符合RISC-V偵錯規格的核心偵錯單元。

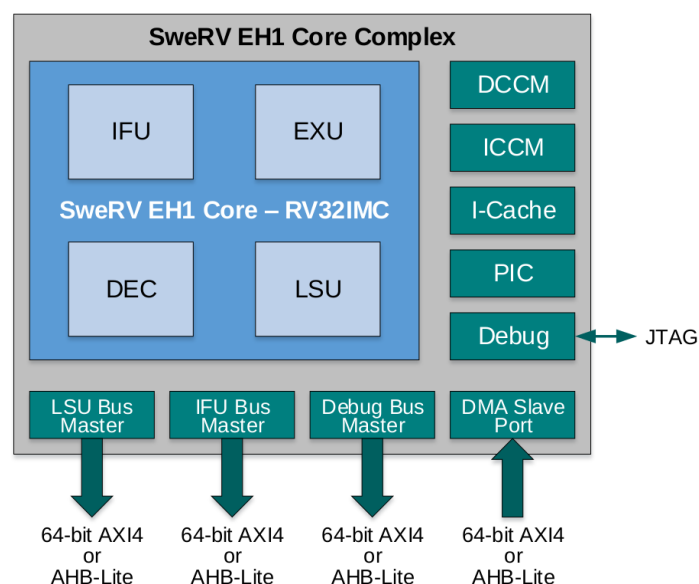


圖20. SweRV EH1核心組合

（圖來自https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf）

B. SweRVolfX SoC

此RVfpga套件中使用的晶片上系統（SoC）稱為SweRVolfX（如圖21所示），它基於SweRVolf 0.7.3版本（<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>），而後者基於SweRV EH1核心組合建構。除了SweRV EH1核心組合（參見圖20）外，SweRVolf SoC還包括一個啟動ROM、一個UART、一個系統控制器和一個SPI控制器（這些元件如圖21中的白色部分所示）。如果SweRV EH1核心使用AXI匯流排，而週邊設備使用Wishbone匯流排，則SoC也具有AXI-Wishbone橋接器。

在RVfpga中，SweRVolf SoC延伸了一些功能，例如一個額外SPI控制器（SPI2）、GPIO（通用輸入/輸出）控制器、PTC（PWM/計時器/計數器）模組以及與8位元7段顯示器介接的控制器。這些新周邊設備如圖21中紅色部分所示，但是7段顯示器控制器除外，它被包含在系統控制器當中。我們把這個系統稱為片上系統**SweRVolfX**（X代表延伸）。

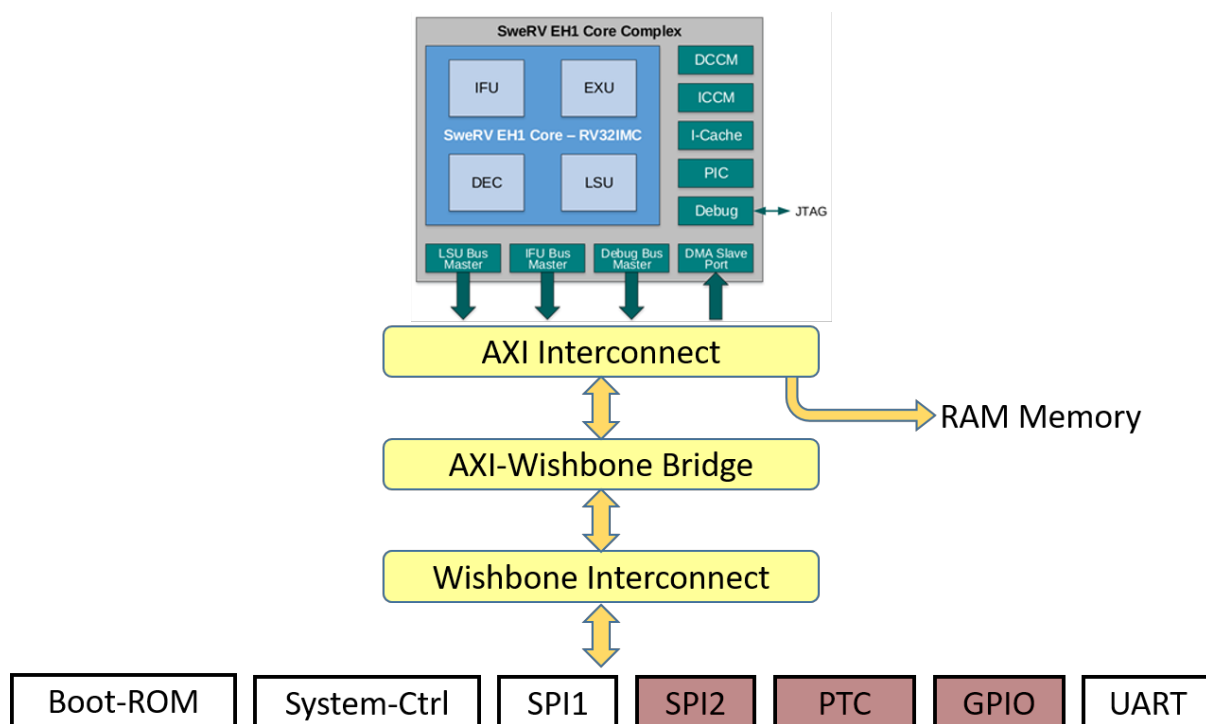


圖21. SweRVolfX（使用4個新周邊設備延伸SweRVolf）晶片上系統

表6列出了通過Wishbone互連連線到核心的週邊設備的記憶體映射位址。

表6. 擴展SweRVolfX SoC週邊設備的記憶體映射位址

系統	位址
開機ROM	0x80000000 - 0x80000FFF
系統控制器	0x80001000 - 0x8000103F
SPI1	0x80001040 - 0x8000107F
SPI2	0x80001100 - 0x8000113F
PTC	0x80001200 - 0x8000123F
GPIO	0x80001400 - 0x8000143F
UART	0x80002000 - 0x80002FFF

i. 輸入/輸出

SweRVolfX SoC使用兩種硬體控制器與週邊設備通訊：用Verilog編寫的自訂控制器和來自OpenCores [<https://opencores.org/>]的開放原始碼控制器，OpenCores是一個基於免費和開放原始碼合作的原則來開發gateway（閘道）IP（智慧財產權）核心的線上社群。我們在本課程中使用的SweRVolfX SoC包括下面列出的I/O介面，我們將在RVfpga的實驗6-20中使用、詳述這些介面，甚至對它們進行延伸。

- **系統控制器**：系統控制器包含常見的系統功能，例如在暫存器中儲存SoC版本資訊、RAM初始化狀態和RISC-V機器計時器（可以在<https://github.com/chipsalliance/Cores-SweRVolf>下找到完整的記憶體對映）。我們對此模組進行了如下修改：

- 新增了一個新的控制器（稱為**SevSegDisplays_Controller**）來與Nexys A7電路板上提供的8位元7段顯示器通訊，同時還為此控制器新增了兩個新的暫存器（對映到位址0x80001038和0x8000103C）。
 - 新增了兩個1位元暫存器（對映到位址0x80001018）來處理來自GPIO和PTC的中斷。
 - 刪除了SweRVolf提供的簡單GPIO暫存器（對映到位址0x80001010–0x80001017）。請注意，我們新增了一個更完整的GPIO控制器，如下所述。
- **SPI**：在SweRVolfX中實作了兩個開放原始碼SPI控制器（從https://opencores.org/projects/simple_spi取得，命名為SPI1和SPI2）。這些控制器的公開暫存器（SPI_SPCR、SPI_SPSR、SPI_SPDR、SPI_SPER、SPI_SPSS）將在位址0x80001040和0x8000107F（對於SPI1）之間映射，以及位址0x80001100與0x8000113F（對於SPI2）之間映射。
 - **PTC**：我們使用的計時器模組來自<https://opencores.org/projects/ptc>。其暫存器映射到位址範圍0x80001200至0x800012FF。
 - **GPIO**：我們使用來自<https://opencores.org/projects/gpio>的GPIO控制器。該控制器包括32個I/O連接埠，這些連接埠映射到位址範圍0x80001400至0x800014FF。每個針腳連接一個三態緩衝區，因此可以將其設定為輸入或輸出。
 - **UART**：SweRVolfX中提供了開放原始碼UART控制器（從<https://opencores.org/projects/uart16550>取得）。該控制器的公開暫存器映射到位址0x80002000和0x80002FFF之間。

ii. 記憶體

SweRVolfX SoC包括開機ROM記憶體，以及允許使用者加入RAM和SPI快閃記憶體的必要硬體。

- **開機ROM**：開機ROM包含第一階段的開機載入器。重設系統後，SweRVolfX SoC將開始從此區域擷取初始指令，該區域佔用的位址為0x80000000至0x80000FFF。
- **RAM**：SweRVolfX SoC不包含儲存控制器，但保留了其記憶體映射（0x00000000-0x07FFFFFF）的前128MiB，並且顯示AXI匯流排，讓使用者可以使用儲存控制器存取RAM記憶體。
- **SPI快閃記憶體**：也可以使用上一節介紹的SPI1控制器來加入SPI快閃記憶體（位址範圍：0x80001040-0x8000107F）。

iii. 互連

SweRV EH1核心使用AXI4匯流排來連接核心和記憶體。該匯流排也可以配置為AHB-Lite匯流排，但在本指南中我們不會使用該選項。所有週邊設備（I/O設備）都連接到Wishbone匯流排，這是一種在OpenCore CPU和週邊設備中廣泛使用的開放原始碼匯流排。系統包括AXI-Wishbone橋接器（如圖21所示），用於將核心連接到週邊設備。

在本節中，我們簡要介紹了AXI4匯流排和Wishbone匯流排的操作。如果您有興趣瞭解這些匯流排規格的詳細資訊，可以使用下面提供的參考資料。

AXI4匯流排

SweRV EH1核心組合使用AXI4互連與外界通訊（請參閱圖20）。高級可擴展介面（Advanced eXtensible Interface，AXI）是許多處理器的通用匯流排，並且是「ARM高級微處理器匯流排架構」晶片上互連規格的一部分。

在下面的小節中，我們將簡要說明AXI4互連的一些主要課題。有關完整的AXI規格，請參閱以下文件：

https://static.docs.arm.com/ih0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf

- **AXI匯流排的主要特性**

AXI匯流排技術的主要特性如下：

- 適用於高頻寬和低延遲設計
- 無需使用複雜的橋接器就能實現高頻率操作
- 可以滿足各種元件的介面要求
- 適用於初始存取時發生高延遲現象的儲存控制器
- 在互連架構實作方面具有靈活性
- 與現有AHB和APB介面回溯相容
- 分離的位址/控制和資料階段
- 支援非對齊資料的傳輸（使用位元組回應）
- 允許僅發出起始位址的突發型交易
- 提供支援低功耗DMA的單獨讀取和寫入資料通道
- 允許在實際傳輸資料前進行定址
- 支援多個未完成傳輸的位址進行操作和亂序傳輸
- 易於新增暫存器級來進行時序收斂

- **AXI架構**

AXI通訊協定定義了以下獨立的傳輸通道：

- 讀取位址通道
- 讀取資料通道
- 寫入位址通道
- 寫入資料通道
- 寫入回應通道

圖22顯示了資料讀取如何使用讀取位址和讀取資料通道。首先從主要裝置傳送位址和控制訊號，然後從屬裝置用讀取資料通道上的資料進行回應。

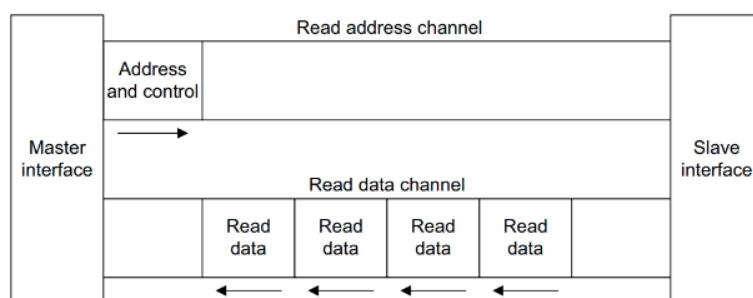


圖22. 讀取通道架構

(圖來自 https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

圖23顯示了資料寫入如何使用寫入位址、寫入資料和寫入回應通道。與資料讀取類似，主要裝置將傳送位址和控制訊號。然後，主要裝置會在寫入資料通道上傳送資料，從屬裝置隨後將傳送寫入回應。

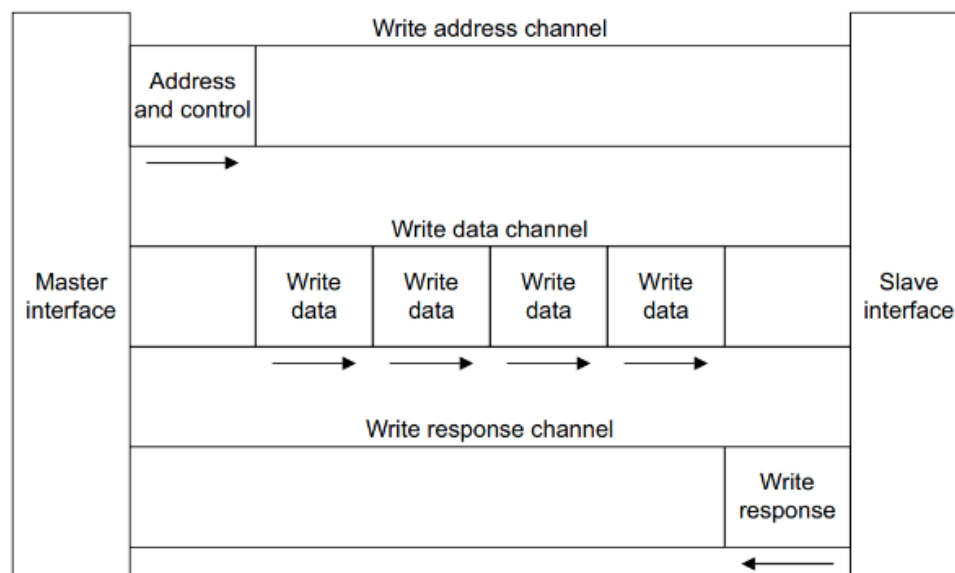


圖23. 寫入通道架構

(圖來自 https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

AXI位址通道帶有位址和控制資訊，這些資訊描述了要傳輸資料的性質。在主要與從屬裝置之間使用以下任一方式傳輸資料：

- 使用讀取資料通道，將資料自從屬裝置傳輸到主要裝置（圖22）。
- 使用寫入資料通道，將資料從主要裝置傳輸到從屬裝置（圖23）。在資料寫入過程中，從屬裝置會使用寫入回應通道，向主要裝置傳送傳輸完成訊號（圖23）。

• AXI訊號

表7.列出了AXI匯流排中使用的主要訊號以及每個訊號的簡要說明。這些訊號分為五個群組，分別對應於上節所說明的五個通道：

- 寫入位址通道訊號，名稱以**AW**開頭
- 寫入資料通道訊號，名稱以**W**開頭
- 寫入回應通道訊號，名稱以**B**開頭

- 讀取位址通道訊號，名稱以**AR**開頭
- 讀取資料通道訊號，名稱以**R**開頭

表7. AXI訊號

(表來自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.

Wishbone匯流排

SweRVofX週邊設備將Wishbone晶片上系統（SoC）互連架構用於可攜式的IP核心（<https://opencores.org/howto/wishbone>）。該匯流排主要用於通過解決晶片上系統整合問題來促進設計重複使用。IP核心以往採用非標準互連方案，因此難以整合。這些非標準互連要求建立自訂膠合邏輯，以將每個核心連接在一起。通過採用Wishbone匯流排等標準互連方案，最終使用者可以更快速、更輕鬆地整合核心。

• Wishbone主要特性

這種Wishbone匯流排技術的主要特性如下：

- 支援大型專案團隊使用的結構化設計方法。
- 包括一整套常用的資料傳輸匯流排通訊協定，具體如下：
 - i. 讀取/寫入週期
 - ii. 區塊傳輸週期
 - iii. 讀取/修改/寫入週期

- 提供高達64位元的模組化資料匯流排寬度和運算元大小。
- 支援大端和小端資料順序。
- 支援各種核心互連方法，包括點對點、共享匯流排、縱橫制交換器和交換式光纖互連。
- 包括握手通訊協定，該通訊協定允許每個IP核心限制其資料傳輸速度。
- 支援單時脈資料傳輸。
- 支援正常週期終止、重試終止和因錯誤終止。
- 包括模組化位址寬度。
- 為從屬裝置提供部分位址解碼方案。這有助於高速位址解碼、使用較少的冗餘邏輯，並支援可變位址大小的調節和互連。
- 提供使用者自訂標記。這些標記有助於將資訊應用於位址、資料匯流排或匯流排週期。在修改匯流排週期以標識下列資訊時，使用者定義的標記特別有用：
 - i. 資料傳輸
 - ii. 同位檢查位元或糾錯位元
 - iii. 中斷向量
 - iv. 快取控制操作
- 包括具有靈活系統設計的主/從架構。
- 具有多重處理（multi-MASTER）功能。這可實作各種SoC配置。
- 包括可由最終使用者定義的仲裁方法（優先級仲裁器、循環配置資源仲裁器等）。

• Wishbone架構和訊號

圖24說明了主要裝置（本例中為SweRV EH1核心）和從屬裝置（本例中為週邊設備，如GPIO、SPI等）之間通過Wishbone匯流排實作的標準連接。Wishbone匯流排比AXI4匯流排簡單得多，且如表8所示，Wishbone匯流排使用更少的訊號。

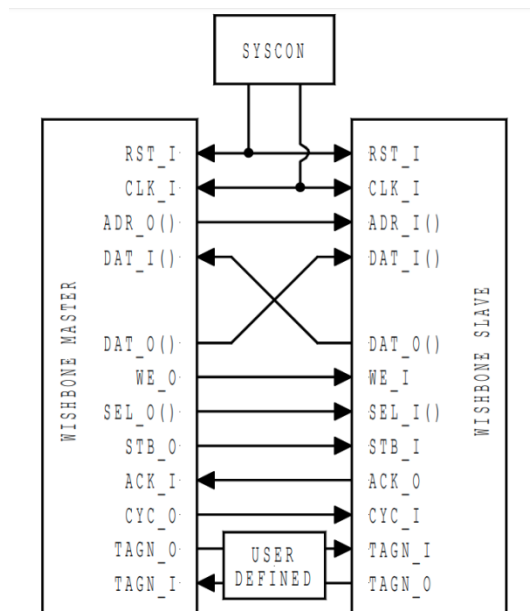


圖24. Wishbone架構

（圖來自<https://opencores.org/howto/wishbone>）

表8. Wishbone訊號

(表格來自<https://opencores.org/howto/wishbone>)

Signal name	description	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	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].
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	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_O(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	Signal name	Description
ACK_I	The acknowledge input [ACK_I], when asserted, indicates the normal termination of a bus cycle	ACK_O	The acknowledge output [ACK_O], when asserted, indicates the termination of a normal bus cycle
CYC_O	The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress	CYC_I	The cycle input [CYC_I], 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	STALL_O	The pipeline stall signal [STALL_O] indicates that the slave can not accept additional transactions in its queue
ERR_I	The error input [ERR_I] indicates an abnormal cycle termination	ERR_O	The error output [ERR_O] 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	RTY_O	The retry output [RTY_O] 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	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_O	The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle	WE_I	The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle

C. SweRVolfX SoC在Nexys A7 FPGA上進行模擬

SweRVolfX SoC (圖21) 可以(1)在Nexys A7 (或Nexys4 DDR) FPGA電路板上執行, 本例中其組態稱為RVfpgaNexys, 也可以(2)在模擬中執行, 本例中稱為RVfpgaSim。

i. RVfpgaNexys

RVfpgaNexys是以Digilent Nexys A7 FPGA電路板為目標的SweRVolfX SoC (圖25)。

RVfpgaNexys 與 SweRVolf Nexys 基本相同 (<https://github.com/chipsalliance/Cores-SweRVolf>), 只是後者基於SweRVolf。RVfpgaNexys使用的主要元素如圖25所示:

- 在FPGA中設計程式的硬體:
 - **SweRVolfX SoC** (如圖21所示)
 - **Lite DRAM控制器**
 - **時脈產生器**: Nexys A7電路板包括一個由**Lite DRAM控制器**使用的**100 MHz**晶振。此時脈的頻率按比例縮小至**50 MHz**, 以在**SweRVolfX SoC**中使用。
 - **跨時脈變換模組**: 連接2個時脈: **SweRVolfX SoC**和**Lite DRAM**。
 - **JTAG的BSCAN邏輯**: 有關此模組的更多資訊, 請造訪 <https://github.com/chipsalliance/Cores-SweRVolf/issues/29>。

- Nexys A7（或Nexys4 DDR）FPGA電路板上的RVfpgaNexys中使用的記憶體/週邊設備：
 - **DDR2記憶體**（通過上述Lite DRAM控制器存取）
 - **USB連接**
 - **SPI快閃記憶體**
 - **SPI加速計**
 - **16個LED和16個開關**
 - **8位7段顯示器**

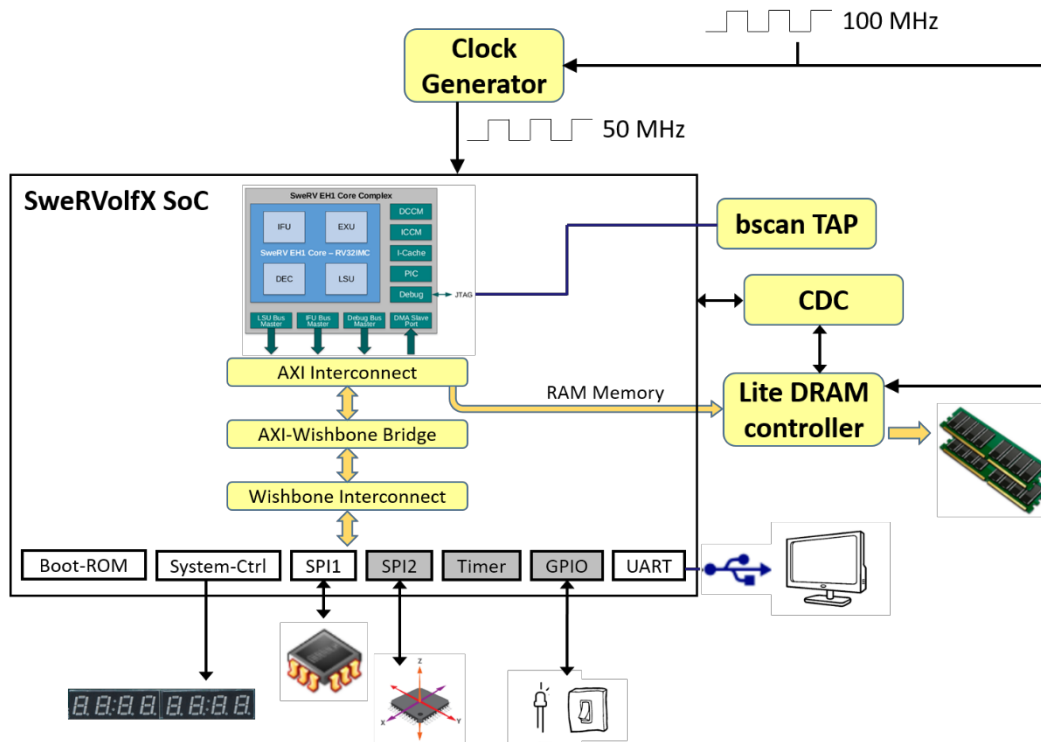


圖25. RVfpgaNexys

建議使用Nexys A7電路板（圖26）來開展電氣與電腦工程課程的培訓。此電路板的價格為265美元（學術優惠價為198.75美元 – 使用.edu電子郵件位址註冊一個Digilent帳戶）。Digilent在以下位置提供有關Nexys A7電路板的豐富參考手冊：
https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf。此電路板可以由5V壁式電源變壓器（未隨電路板提供）供電，也可以通過電路板上的microUSB連接器由PC供電。Microchip PIC24微處理器管理載入到FPGA的程序，因此該電路板非常簡單易用。此電路板可以使用Xilinx的Vivado Design Suite或OpenOCD進行程式設計。可以使用下列四種不同的來源之一將所需配置下載到FPGA：FAT32格式的MicroSD卡、FAT32格式的USB隨身碟、內部快閃記憶體或JTAG介面。

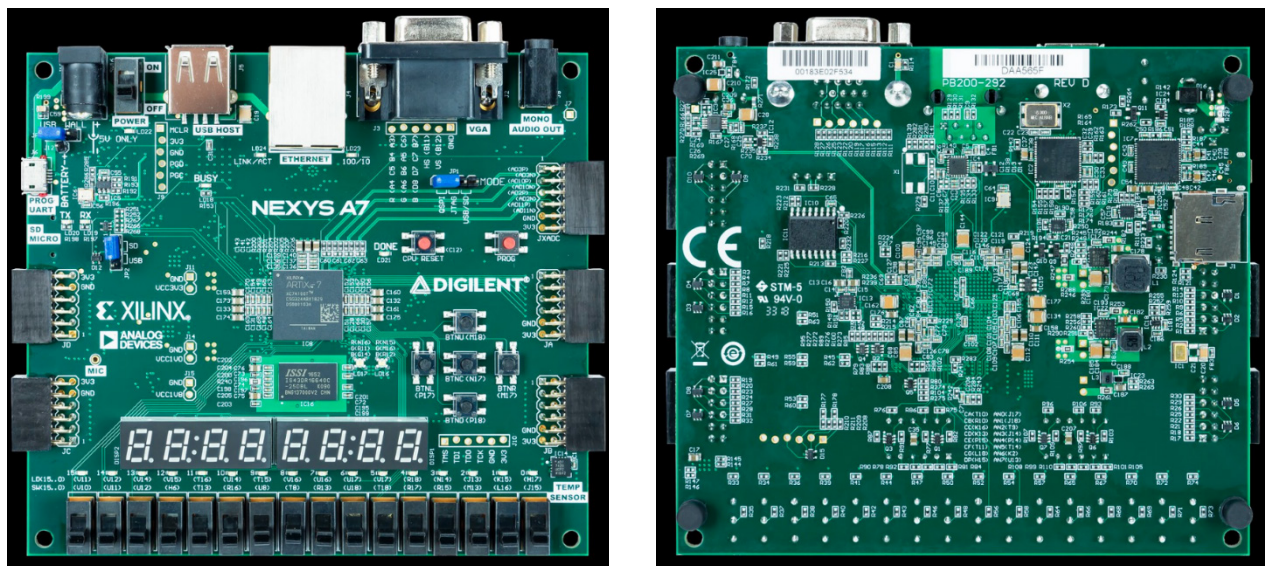


圖26. Digilent的Nexys A7 FPGA電路板

(圖來自<https://reference.digilentinc.com/>)

Nexys A7-100T FPGA電路板包括以下介面和裝置：

- 128 MiB DDR RAM
- 128 Mibit SPI快閃記憶體
- 8位7段顯示器
- 16個開關
- 16個LED
- 感應器和連接器，包括麥克風、音訊插孔、VGA 25連接埠、USB主機連接埠、RGB-LED、I2C溫度感應器和SPI加速計等。
- Xilinx Artix-7 FPGA，具有以下功能：
 - 由四個6路輸入LUT和8個觸發器組成的15.850邏輯片。
 - 區塊RAM總計為4.860 Kb
 - 6個時脈管理圖塊（Clock Management Tile，CMT）
 - 170個I/O引腳
 - 450 MHz內部時脈頻率

ii. RVfpgaSim

SweRVolFX SoC（圖21）還可以包括Verilog包裝函式以啟用模擬。**RVfpgaSim**是包裝在HDL模擬器所用測試平台中的SweRVolFX SoC。RVfpgaSim與SweRVolFX Sim基本相同（<https://github.com/chipsalliance/Cores-SweRVolFX>），只是後者基於SweRVolFX。

雖然存在許多開放原始碼HDL模擬器，但我們使用Verilator

（<https://www.veripool.org/wiki/verilator>）。這是一款開放且免費的HDL模擬器，接受可綜合的Verilog或SystemVerilog，並且宣稱是最快的Verilog/SystemVerilog模擬器。這款模擬器廣泛應用於工業和學術領域；它提供現成的支援，只需提供ARM和RISC-V供應商IP；它以Chips Alliance和Linux Foundation為指導。

D. 檔案結構

在前面的部分中，我們已經介紹了在這些資料中使用的系統高階組織，從**SweRV EH1核心組合**（圖20）到**SweRVolfX SoC**（圖21），最後到**RVfpgaNexys**（圖25）和**RVfpgaSim**實作。

在本節中，我們介紹整個系統的檔案結構。閱讀這些說明時，請在PC上開啟並檢視檔案。這些檔案位於**[RVfpgaPath]/RVfpga/src**。

i. SweRV EH1核心組合

圖27顯示了**SweRV EH1核心組合**（圖20）的檔案結構。核心分為三個主要模組：**SweRV**包裝函式（以灰色突出顯示），其中又包括**SweRV EH1核心**（以綠色突出顯示）和一些其他元素（例如中斷控制器或偵錯單元），以及資料/指令記憶體和指令快取（以紅色突出顯示）。

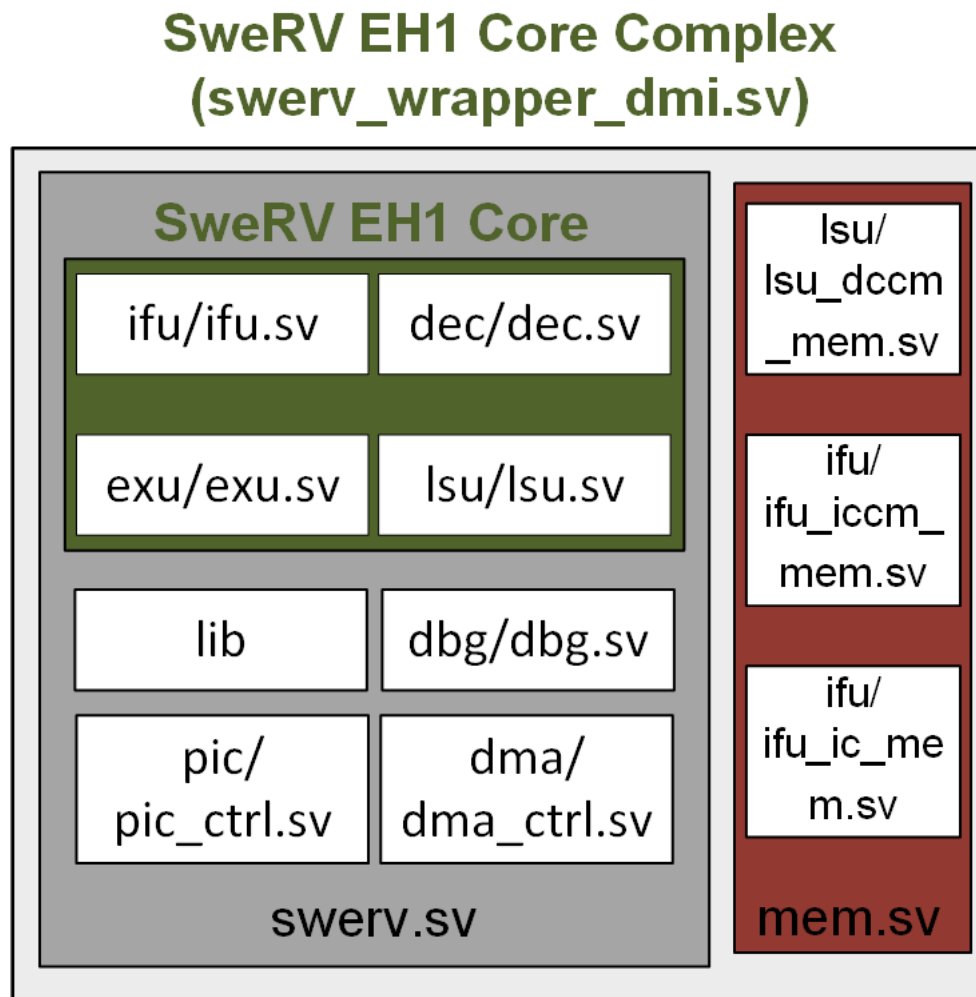


圖27. SweRV EH1核心組合

SweRV EH1核心組合的Verilog檔案位於以下資料夾中：

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex

在您的PC上找到該目錄，以檢視我們在本節中引用的檔案。

SweRV EH1核心組合的頂層檔案位於以下檔案中：**swerv_wrapper.sv**；頂層模組稱為**swerv_wrapper**，它可實例化與圖27中以灰色和紅色強調顯示的兩個區塊相對應的兩個模組：

- **mem**（在**mem.sv**內實作）：該模組實例化用於實作DCCM（**lsu_dccm_mem**，在**lsu/lsu_dccm_mem.sv**檔案中實作）、ICCM（**ifu_iccm_mem**，在**ifu/ifu_iccm_mem.sv**檔案中實作）和指令快取（**ifu_ic_mem**，在**ifu/ifu_ic_mem.sv**檔案中實作）的模組。
- **swerv**（在**swerv.sv**內部實作）：該模組實例化組成核心的單元。
SweRV EH1核心（在圖27中以綠色突出顯示）由以下四個單元組成：
 - 資料夾**ifu**（指令擷取單元）：該資料夾包含用於**lcache**（指令快取）、指令擷取、分支預測器和對齊器的Verilog檔案（**ifu.sv**內部提供的頂層模組）。
 - 資料夾**dec**（解碼單元）：該資料夾包含用於指令解碼、依賴性記分牌和暫存器檔案的Verilog檔案（**dec.sv**內部提供的頂層模組）。
 - 資料夾**exu**（執行單元）：該資料夾包含用於核心中算術/邏輯單元的Verilog檔案（**exu.sv**內部提供的頂層模組）：兩個管線式ALU、一個管線式乘法器和一個非管線式除法器。
 - 資料夾**lsu**（載入儲存單元）：該資料夾包含用於管線式載入/儲存單元的Verilog檔案（**lsu.sv**內部提供的頂層模組）。

該模組中還包含其他單元，具體如下：

- 資料夾**dbg**（除錯單元）：該資料夾包含除錯單元的Verilog檔案（**dbg.sv**內部提供的頂層模組），除錯單元負責將核心的其餘部分置於靜止模式，傳送命令/位址，傳送寫入資料和接收讀取資料，然後恢復核心以進入正常模式。
- 資料夾**lib**：該資料夾包含用於AXI和AHB-Lite匯流排的Verilog檔案。
- 資料夾**pic**：該資料夾包含檔案**pic_ctrl.sv**，用於在模組**pic_ctrl**中實作可程式設定中斷控制器。
- 資料夾**dma**：該資料夾包含檔案**dma_ctrl.sv**，用於在模組**dma_ctrl**中實作直接記憶體存取。

ii. SweRVofX SoC

圖28顯示圖21所示**SweRVofX SoC**的檔案結構。SoC被劃分為多個模組，這些模組對應於圖21中顯示的方塊。

SweRVolfX SoC (swervolf_core.v)

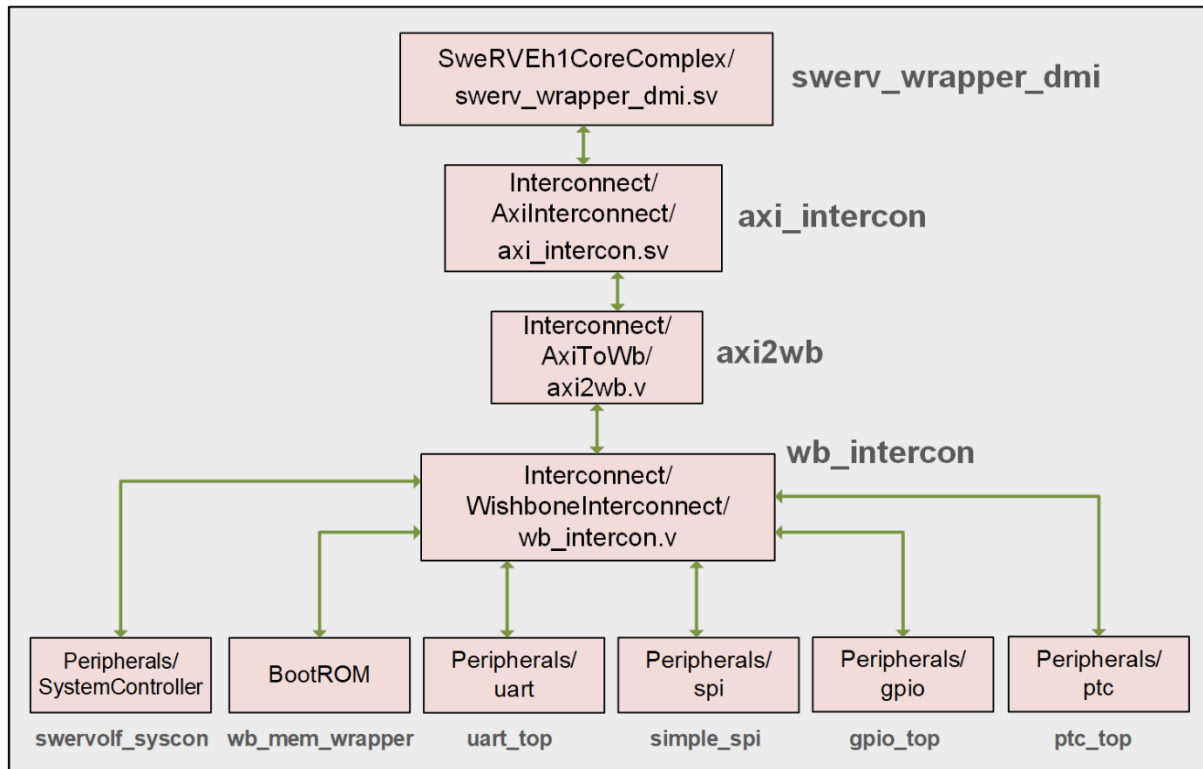


圖28. SweRVolfX SoC

SweRVolfX SoC的檔案位於：

[RVfpgaPath]/RVfpga/src/SweRVolfSoC

在您的PC上找到該目錄，以檢視我們在本節中引用的檔案。

SweRVolfX SoC的頂層模組位於：**[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v**。

開啟該檔案，注意其中包括SweRVolfX SoC（圖21）內包含的模組，具體如下：

- **axi_intercon**（在 **Interconnect/AxiInterconnect/axi_intercon.v** 內部提供）：該模組通過另一個檔案的第100行（``include "axi_intercon.vh"`）包含在內。它將SweRV EH1核心組合與AXI-Wishbone橋接器相連。
- **axi2wb**（在 **Interconnect/AxiToWb/axi2wb.v** 內部提供）：該模組在 **swervolf_core.v** 的第153行中實例化，它是AXI-Wishbone橋接器，允許在基於AXI的EH1核心與基於Wishbone的週邊設備之間進行通訊。
- **wb_intercon**（在 **Interconnect/WishboneInterconnect/wb_intercon.v** 內部提供）：該模組透過另一個檔案的第145行（``include "wb_intercon.vh"`）包括在內。它通過一個多工器將AXI-Wishbone橋接器與不同的週邊設備相連，我們稍後將對此多工器進行分析和修改。
- **wb_mem_wrapper**（在 **BootROM/wb_mem_wrapper.v** 內部提供）：上述啟動記憶體體的包裝函式在 **swervolf_core.v** 的第197行實例化。此外，它實例化了 **dpam64** 模組（在 **BootROM/dpam64.v** 內部提供），該模組是一個基本RAM模組。

- **swervolf_syscon**（在*Peripherals/SystemController/swervolf_syscon.v*內部提供）：該模組在*swervolf_core.v*的第215行實例化，用於定義系統控制器。
- **simple_spi**：從OpenCores獲得的SPI控制器，在*Peripherals/spi/simple_spi_top.v*內部提供。它在*swervolf_core.v*的第246（SPI）和387（spi2）行實例化。
- **uart_top**：從OpenCores獲得的UART控制器，在*Peripherals/uart/uart_top.v*內部提供。它在*swervolf_core.v*的第272行實例化。
- **gpio_top**：從OpenCores獲得的GPIO控制器，在*Peripherals/gpio/gpio_top.v*內部提供。它在*swervolf_core.v*的第338行實例化。
- **ptc_top**：從OpenCores取得的PTC控制器，在*Peripherals/ptc/ptc_top.v*內部提供。它在*swervolf_core.v*的第361行實例化。
- **swerv_wrapper_dmi**（在*SweRVEh1CoreComplex/swerv_wrapper_dmi.v*內部提供）：Western Digital的SweRV EH1核心組合的實例化（*swervolf_core.v*的第407行），如前面的部分（圖27）所述。

iii. 用於電路板上執行和模擬的包裝函式

模擬：

RVfpgaSim是一個模擬目標，用於將**SweRVolfX SoC**（圖21）包裝在HDL模擬器使用的測試平台中。它位於*[RVfpgaPath]/RVfpga/src/rvfpgasim.v*中。

電路板上執行：

RVfpgaNexys（位於*[RVfpgaPath]/RVfpga/src/rvfpganexys.v*中）將**SweRVolfX SoC**（圖21）包裝在以Nexys A7 FPGA電路板及其周邊設備（參見圖25）作為目標的包裝函式中。該模組除了實例化一些其他模組（例如時脈產生器模組**clk_gen_nexys**、跨鐘域模組**axi_cdc_intf**或JTAG連接埠的BSCAN模組**bscan_tap**）外，還實例化兩個主要SoC結構：

- **swervolf_core**：上一小節（圖28）所述**SweRVolfX SoC**的實例化。此外，還需要一個稱為*rvfpganexys.xdc*的約束檔案（位於*[RVfpgaPath]/RVfpga/src/*中），該檔案定義SoC與電路板之間的連接。
- **litedram_top**：LiteDRAM DDR2控制器的包裝函式，用於連接SweRVolfX SoC與DDR2記憶體，此包裝函式在檔案*[RVfpgaPath]/RVfpga/src/LiteDRAM/litedram_top.v*中實作。此外，還需要一個稱為*litedram.xdc*的約束檔案（位於*[RVfpgaPath]/RVfpga/src/LiteDRAM/*中），該檔案定義記憶體控制器與板上DDR2記憶體之間的連接。

總而言之，圖29顯示了Nexys A7 FPGA電路板上整個RVfpgaNexys實作的階層。

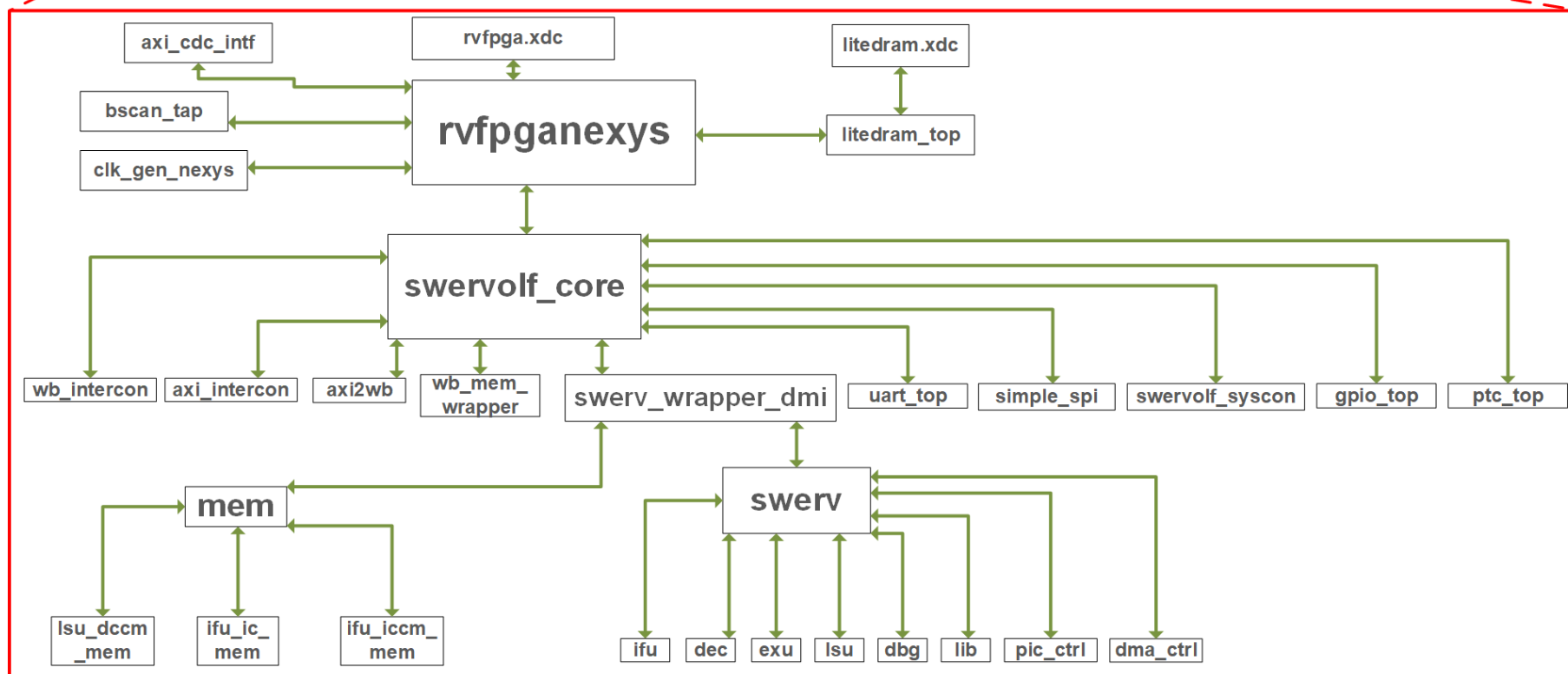
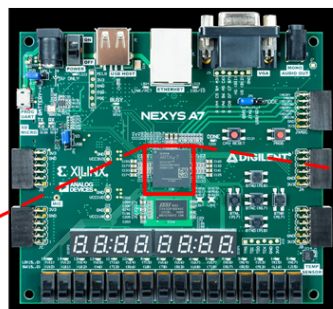


圖29. Nexys A7 FPGA電路板實作的模組階層

5. 安裝軟體工具

以下說明適用於**Ubuntu 18.04 OS**，但其他Linux作業系統以及Windows或macOS遵循類似（但是不完全相同）的步驟。在某些情況下，我們會插入文字方塊來針對這些不同的作業系統提供具體說明。如果使用的是**Ubuntu**，則可以忽略這些文字方塊。

這些說明顯示了如何安裝以下工具：

- A. Vivado**：重新綜合晶片上系統所需的工具。在實驗6-20中，將主要完成此操作，並且不同的特性將包含在基線SoC中。
- B. VSCode（Visual Studio Code）和PlatformIO**：這些是GSG和實驗中使用的主要工具。這些工具用於對FPGA進行程式設計，以及在FPGA上執行/偵錯程式。
- C. Verilator和GTKWave**：模擬SoC和分析不同訊號時所需的工具。同樣，將主要在實驗6-20中使用這些工具。

請注意，對於將在此GSG和實驗中進行的大多數操作，安裝VSCode和PlatformIO已經足夠。但現在，我們建議使用者同時安裝其他工具（Vivado、Verilator和GTKWave），這樣以後便無需再安裝。

此程序可能需要幾個小時（或更長時間，具體取決於下載速度），但大部分時間都花在了等待程式下載和安裝上。

A. 安裝Vivado

Vivado是一種Xilinx工具，用於檢視、修改和綜合RISC-V FPGA的Verilog程式碼。您將在後續的實驗中廣泛使用此工具。可從<https://reference.digilentinc.com/vivado/installing-vivado/start>取得安裝說明，總結如下。

Windows：上面引用的網頁（<https://reference.digilentinc.com/vivado/installing-vivado/start>）還包括在Windows中安裝Vivado的詳細說明。在下文中，我們將在Windows需要具體說明時插入文字方塊。

macOS：macOS不支援Vivado；因此，需要Linux/Windows虛擬機器才能在此作業系統中執行Vivado。

1. 導覽至<https://reference.digilentinc.com/vivado/installing-vivado/start>
2. 您將被引導至Xilinx下載頁面：<https://www.xilinx.com/support/download.html>
3. 建議安裝「Self Extracting Web Installer」。在撰寫本文件時，可通過下載頁面的以下連結取得此程式：[Xilinx Unified Installer 2019.2: Linux Self Extracting Web Installer](#)

WINDOWS：在撰寫本文件時，可通過下載頁面的以下連結取得適用於Windows的「Self Extracting Web Installer」：[Xilinx Unified Installer 2019.2: Windows Self Extracting Web Installer](#)

4. 在下載安裝程式之前，系統將要求您登入到Xilinx帳戶。如果您還沒有帳戶，則需要建立一個帳戶。

5. 執行二進位檔案。開啟一個終端機，將其設定為根（輸入「**sudo su**」）。然後將二進位檔案（**Xilinx_Unified_2019.2_1106_2127_Lin64.bin**）拖曳至終端機。如果系統提示將檔案設定為可執行檔並執行該檔案，請選擇「**OK**」（確定）。

- **疑難排解：**如果終端機顯示權限被拒絕，則在終端機中輸入以下內容（與二進位檔案位於同一目錄中）：

```
> sudo chmod +x ./Xilinx_Unified_2019.2_1106_2127_Lin64.bin
> sudo ./Xilinx_Unified_2019.2_1106_2127_Lin64.bin
```

WINDOWS：在Windows中，只需按兩下執行步驟3和4所下載的.exe檔案。

6. Vivado安裝程式將引導您完成安裝程序。重要注意事項：

- 選擇**Vivado**（*非Vitis*）作為要安裝的產品。
- 選擇**Vivado HL Webpack**（*非Vivado HL系統版本*）；Webpack是免費的。
- 其他選項應為預設值。

提示：如果變更了Vivado的安裝目錄，將需要在以下步驟中適當地修改路徑。

WINDOWS：在Windows中不需要步驟7和8。只需忽略這兩個步驟，直接移至步驟9即可。

7. Vivado安裝完畢後，需要設定環境。開啟一個終端機並輸入以下內容：

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

將上述內容（`source /tools/Xilinx/Vivado2019.2/settings64.sh`）新增到`~/.bashrc`檔案，以便該檔案在每次啟動終端機時執行。

8. 通過在終端機中輸入以下內容來測試Vivado：

```
vivado
```

疑難排解：

- 如果系統找不到該可執行檔，則需要在路徑中新增以下內容：

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

- 如果收到一則錯誤訊息，例如「**application-specific initialization failed...**」（應用程式特定的初始化失敗...），則在終端機輸入以下內容：

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

9. 將需要**手動安裝Nexys A7 FPGA電路板的驅動程式**。在終端機視窗中輸入以下內容：

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

sudo ./install_drivers
```

WINDOWS：在Windows中安裝Vivado時會自動安裝Nexys A7電路板的驅動程式，這些驅動程式與PlatformIO不相容。因此，如果使用的是Windows，必須按照附錄B中的說明更新驅動程式。務必執行此操作，即使已經在「快速入門指南」節中完成了此操作時也如此，因為驅動程式已被Vivado安裝覆蓋。

10. 此外，還需要手動安裝Digilent Board Files。

- 從Github存放庫下載Vivado電路板的**封存檔**並解壓縮檔案。
- 開啟從封存檔中解壓縮的資料夾，導覽至其`new/board_files`目錄。選擇並複製此目錄中的所有資料夾。
- 開啟安裝Vivado的資料夾（預設為`/tools/Xilinx/Vivado`）。在此資料夾下，導覽至`<version>/data/boards/board_files`目錄，然後將電路板檔案貼到該目錄中。
- 此外，也可以使用終端機，方式是移至`new/board_files`目錄並輸入以下內容：

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

WINDOWS：按照步驟10的說明複製/貼上下載的資料夾。在Windows中，可以在下列位置找到Vivado的`board_files`資料夾：`C:\Xilinx\Vivado\2019.2\data\boards\board_files`

B. 安裝VSCode和PlatformIO

現在將安裝VSCode和PlatformIO。如果已經在第1節「快速入門指南」中完成此操作，則無需再重複此程序，可以直接移至C節。

PlatformIO是一種面向嵌入式系統的整合開發環境（IDE），它在Microsoft的Visual Studio（VS）Code基礎上進行編譯。借助此開發環境，使用者可以使用C語言或組合語言對RISC-V處理器（位於FPGA上）進行程式設計。PlatformIO支援跨平台操作，包含一個內建偵錯工具。

按照以下步驟安裝VSCode和PlatformIO：

LINUX命令列：儘管建議使用VSCode+PlatformIO，但附錄A為有興趣在Linux中安裝和使用原生RISC-V工具鏈和OpenOCD來代替PlatformIO的使用者提供了說明。如果要使用PlatformIO，可直接忽略附錄A。

1. 安裝VSCode：

按照以下步驟安裝VSCode：

- a. 從以下連結下載.deb檔案：<https://code.visualstudio.com/Download>
- b. 開啟終端機，然後在此終端機中輸入以下內容來安裝並執行VSCode：


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

Windows/macOS：VSCode套件也支援Windows（.exe檔案）和macOS（.zip檔案），可通過<https://code.visualstudio.com/Download>取得。請遵循在這些作業系統中安裝和執行應用程式的通用步驟。

2. 在VSCode軟體上安裝PlatformIO：

按照以下步驟安裝PlatformIO：

- a. 通過在終端機中輸入以下指令來安裝python3公用程式：

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

Windows/macOS：Windows中不需要執行這一步（2.a）。對於macOS，可以使用homebrew來安裝python3：`brew install python3`


- b. 如果VSCode尚未開啟，請通過以下方式將其啟動：選擇「Start」（開始）按鈕並在搜尋功能表中輸入「VSCode」，然後選擇VSCode，或在終端機中輸入code。
- c. 在VSCode中，按一下VSCode左側欄中的「Extensions」（延伸模組）圖示（請參閱圖30）。



圖30. VSCode的「Extensions」（延伸模組）圖示

- d. 在搜尋方塊中輸入PlatformIO，然後按一下PlatformIO IDE旁邊的「Install」（安裝）按鈕進行安裝（請參閱圖31）。

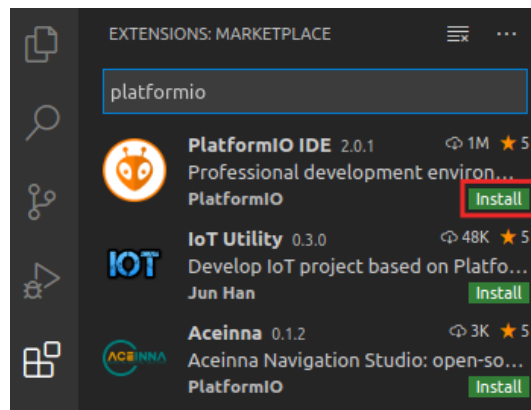


圖31. PlatformIO IDE延伸模組

- e. 底部的「OUTPUT」(輸出)視窗將通知使用者相關的安裝進度。安裝完畢後，按一下視窗右下角的「Reload Now」(立即重新載入)，PlatformIO便會安裝在VSCode內部(請參閱圖32)。

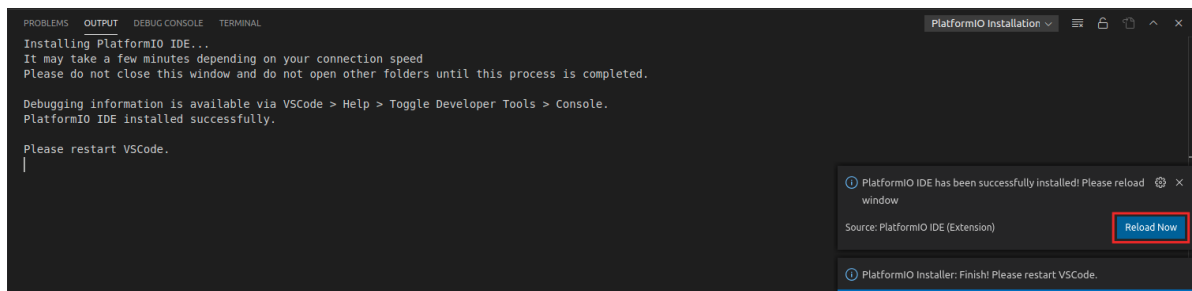


圖32. 安裝PlatformIO後立即重新載入

C. 在Ubuntu 18.04中安裝Verilator和GTKWave

本節中的說明僅適用於Linux系統。

Windows：使用附錄C替代本節所提供的說明。

macOS：使用附錄D替代本節所提供的說明。

按照以下步驟將Verilator(安裝說明可造訪<https://www.veripool.org/projects/verilator/wiki/Installing>查看，具體內容摘要如下)和GTKWave安裝在Ubuntu 18.04 Linux系統中。此程序需要很長時間。

- `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` (也可以使用`make -j$(nproc)`來加快執行速度)

- `sudo make install`
- `export PATH=$PATH:/usr/local/bin` (在系統中變更路徑)

要將`/usr/local/bin`永久新增至路徑，請將最後一行新增至`~/.bashrc`檔。

6. 執行RVfpgaNexys並對其進程式設計

在本部分中，我們將介紹如何在RVfpgaNexys（參見圖25）上執行七個簡單程式。

LINUX/Windows/macOS： 假設所有必需的工具和驅動程式均已按照第5節所述正確安裝，則本節介紹的所有說明均應適用於這三個作業系統。在某些情況下，可能需要修改一些小細節，例如Linux中使用的斜線，Windows中使用的反斜線。

我們通過展示如何執行表9中列出的七個程式範例來說明如何使用RVfpgaNexys。前三個程式是用RISC-V組合語言編寫的，後四個程式是用C語言編寫的。下面提供了在RVfpgaNexys上執行每個程式的說明。

表9. RVfpgaNexys範例程式

程式名稱	說明	語言
AL_Operations	練習算術和邏輯運算	RISC-V組合語言
Blinky	使Nexys A7電路板上的LED閃爍	RISC-V組合語言
LedsSwitches	讀取Nexys A7電路板上的開關值，並將該值寫入LED	RISC-V組合語言
LedsSwitches_C-Lang	讀取Nexys A7電路板上的開關值，並將該值寫入LED	C
HelloWorld_C-Lang	通過序列埠向shell列印一則短訊息	C
VectorSorting_C-Lang	從大到小對向量進行排序	C
DotProduct_C-Lang	計算兩個向量的點積	C

請注意，在能夠執行這七個範例中的任何一個範例之前，必須使用RVfpgaNexys對FPGA進程式設計，如以下部分所述。

A. 用RVfpgaNexys對FPGA進程式設計

在本部分中，我們介紹一種使用PlatformIO的建議方法，這種方法使用RVfpgaNexys對FPGA進程式設計。按照以下步驟使用RVfpgaNexys對FPGA進程式設計：

（如果您有興趣使用Vivado對FPGA進程式設計，則可以遵循本指南的附錄E中提供的說明，而不必遵循以下說明。但是，此處所述的方法僅適用於Linux和Windows系統（不適用於macOS），因此，總體而言不建議使用Vivado將RVfpgaNexys下載到FPGA上。反之，建議您遵循以下說明，而忽略附錄E。）

- a. 將Nexys A7電路板連接到電腦。
- b. 開啟左上方的開關開啟Nexys A7電路板。
- c. 如果VSCode和PlatformIO尚未開啟，請將其開啟。
- d. 在頂端功能表列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾）（請參閱圖33），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/

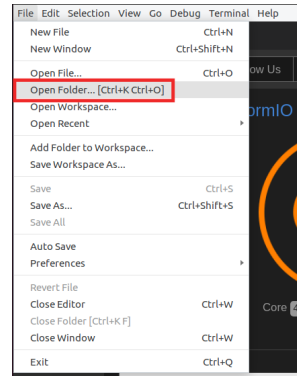


圖33. 開啟資料夾

- e. 選擇要使用的PlatformIO專案。在本節中，我們以表9中提到的第一個範例AL_Operations為例，將在下一節中對此範例進行偵錯，對任何其他範例可執行相同的步驟。為此，選擇目錄AL_Operations（不要開啟，只需將其選取 – 參見圖34），然後按一下視窗頂部的「OK」（確定）。PlatformIO現在將開啟AL_Operations範例。

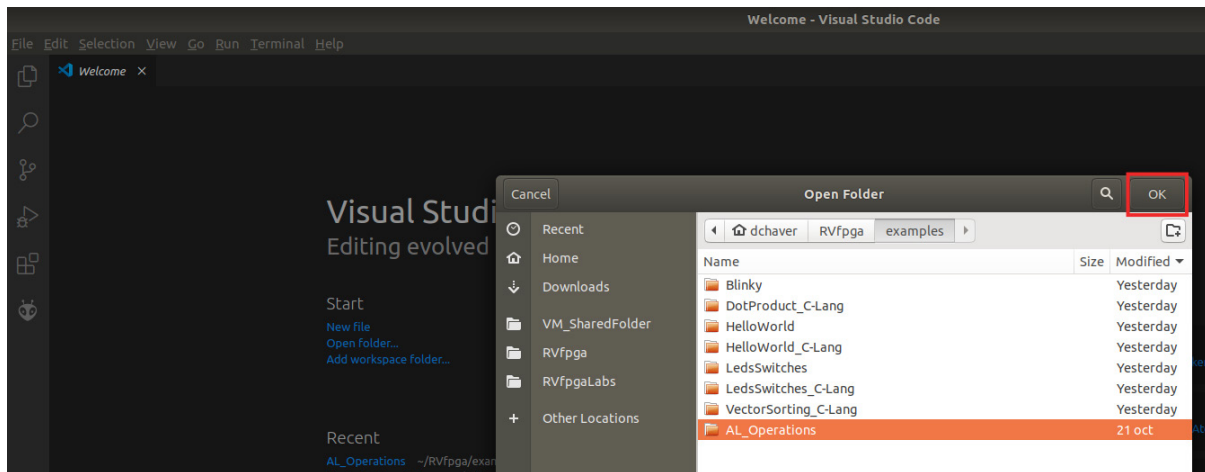


圖34. 開啟AL_Operations資料夾

- f. 按一下左側工具列的platformio.ini開啟platformio.ini檔（請參閱圖35）。通過編輯下行在系統中建立RVfpgaNexys位元串流的路徑（請參閱圖35）。請注意，預先綜合的RVfpgaNexys位元串流在RVfpga資料夾中提供，此資料夾位於：
[RVfpgaPath]/RVfpga/src/rvfpganexys.bit

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

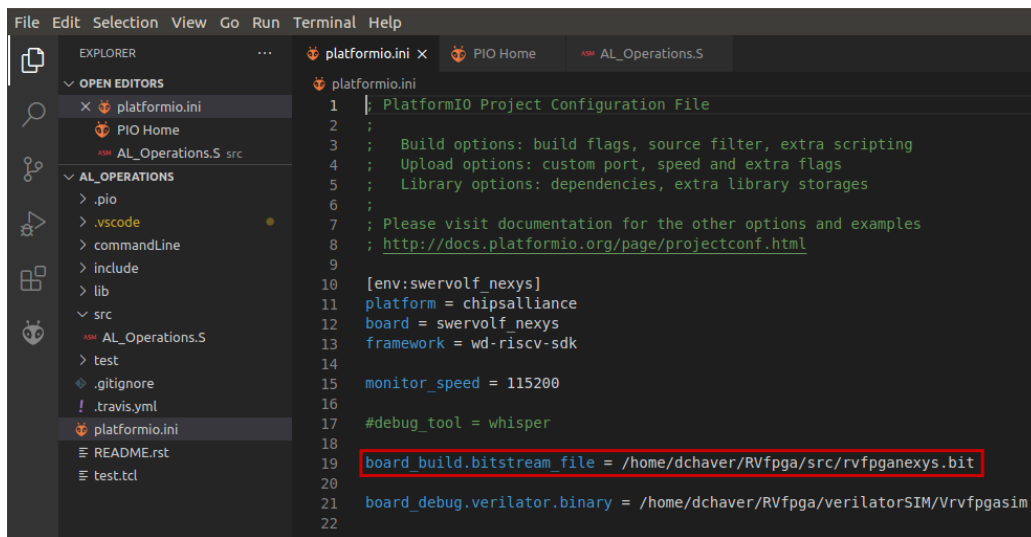



圖35. Platformio初始化檔：platformio.ini

可以在專案組態檔案（*platformio.ini*）中使用許多不同的命令，相關資訊請參見 <https://docs.platformio.org/en/latest/projectconf/>。

- g. 按一下左側功能表功能區中的PlatformIO圖示 （請參閱圖36）。



圖36. PlatformIO圖示

如果「Project Tasks」（專案任務）視窗為空（圖37），必須先按一下  來重新整理「Project Tasks」（專案任務）。此程序可能需要幾分鐘。

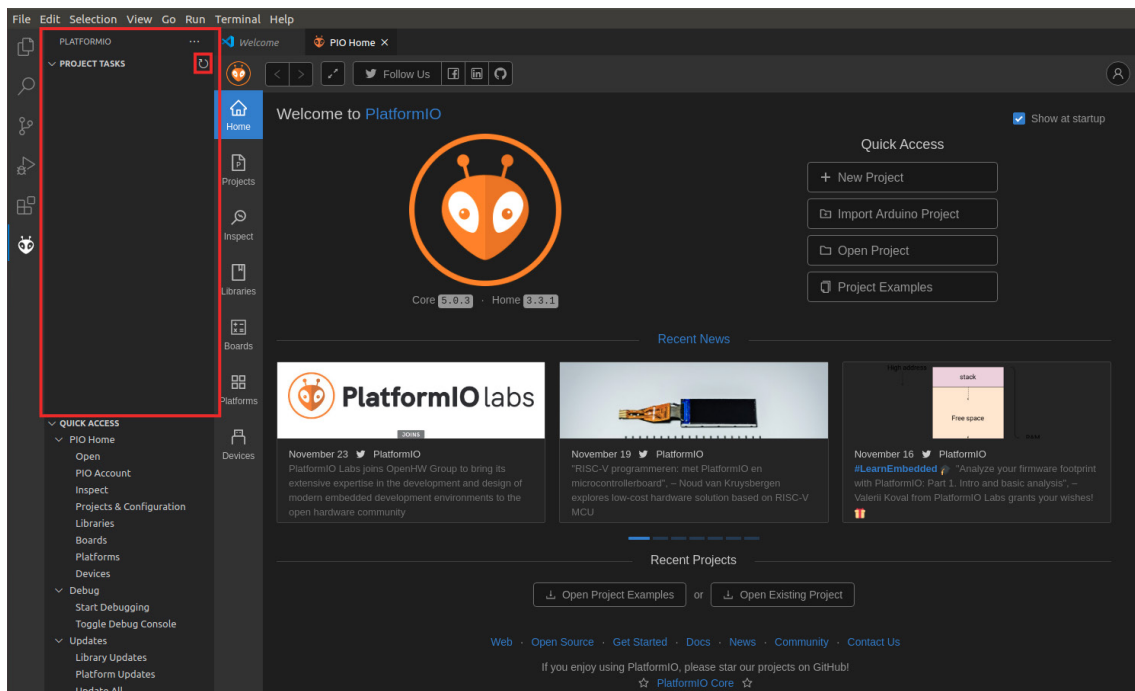


圖37.「PROJECT TASKS」（專案任務）視窗為空 – 重新整理

然後展開「Project Tasks」（專案任務）→ env:swervolf_nexys → 「Platform」（平台），再按一下「Upload Bitstream」（上傳位元串流），如圖38所示。一到兩秒後，將使用RVfpgaNexys對FPGA進行程式設計。

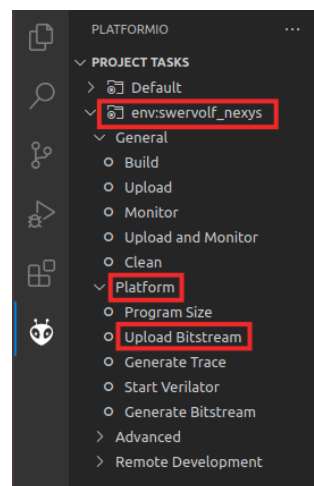


圖38. 上傳位元串流

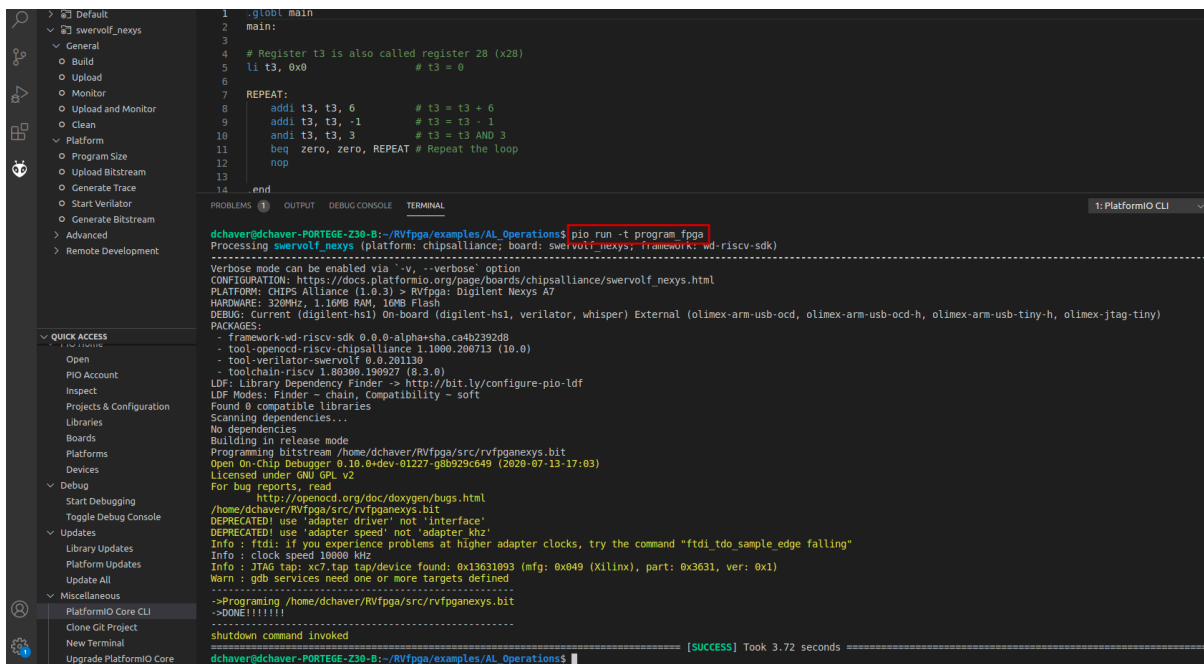
預設情況下，處理器從位址0x80000000處開始擷取指令，其中啟動ROM位於我們的SoC中（參見表6）。啟動ROM使用一個程式（*boot_main.mem*）進行初始化，該程式使LED和7段顯示器閃爍四次，然後熄滅所有LED，將0寫入8個7段顯示器並保持空迴圈。該程式位於以下資料夾：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*。

按下Nexys A7電路板（圖26）上的CPU重設按鈕，使該程式再次執行。

如果要對該程式進行變更和重新編譯，請按照附錄A – 第III部分中的說明進行操作（注意檔案`boot_main.mem`只是檔案`boot_main.vh`的副本）。在實驗1中，我們將介紹如何在建立位元串流時使用該程式初始化啟動ROM。

- h. 也可以通過PlatformIO終端機視窗下載RVfpgaNexys（如圖39所示）來替代上一步（步驟g）。按一下  按鈕（「PlatformIO: New Terminal」（PlatformIO：新建終端機）按鈕，該按鈕位於PlatformIO視窗的底部）開啟一個新的終端機視窗，然後在PlatformIO終端機中輸入（或複製）以下命令：

```
pio run -t program_fpga
```



```

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
12     nop
13
14 end

```

```

dchaver@dchaver-PORTGE-Z30-B:~/RVfpga/examples/AL_Operations$ pio run -t program_fpga
Processing swervolf_nexys (Platform: chipsalliance; board: swervolf_nexys; framework: rd-riscv-sdk)
Verbose mode can be enabled via '-v, --verbose' option
CONFIGURATION: https://docs.platformio.org/page/boards/chipsalliance/swervolf_nexys.html
PLATFORM: CHIPS Alliance (1.0.3) > RVfpga: Digilent Nexys A7
HARDWARE: 320MHz, 1.16MB RAM, 16MB Flash
DEBUG: current (digilent-hs1) on-board (digilent-hs1, verilator, whisper) External (olimax-arm-usb-ocd, olimex-arm-usb-ocd-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny)
PACKAGES:
- framework-rd-riscv-sdk 0.8.0-alpha+sha.ca4b2392d8
- tool-openocd-riscv-chipsalliance 1.1000.200713 (10.0)
- tool-verilator-swervolf 0.8.201130
- toolchain-riscv 1.80300.190927 (8.3.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder - chain, Compatibility - soft
Found 0 compatible libraries
Scanning dependencies...
No dependencies
Building in release mode
Programming bitstream /home/dchaver/RVfpga/src/rvfpganexys.bit
Open On-Chip Debugger 0.10.0+dev-01227-g80929c649 (2020-07-13-17:03)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
/home/dchaver/RVfpga/src/rvfpganexys.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
-->Programming /home/dchaver/RVfpga/src/rvfpganexys.bit
-->DONE!!!!!!
shut down command invoked
===== [SUCCESS] Took 3.72 seconds =====
dchaver@dchaver-PORTGE-Z30-B:~/RVfpga/examples/AL_Operations$

```

圖39. 使用PlatformIO將RVfpgaNexys上傳到Nexys A7 FPGA電路板上

請注意，首次在PlatformIO中開啟範例時，Chips Alliance平台會自動安裝（可以在「PIO Home」（PIO首頁）中檢視該平台，如圖40所示）。該平台包含以後將用到的幾個工具，例如預編譯的RISC-V工具鏈、用於RISC-V的OpenOCD、RVfpgaNexys bit檔和RVfpgaSim、JavaScript和Python指令碼以及一些範例。

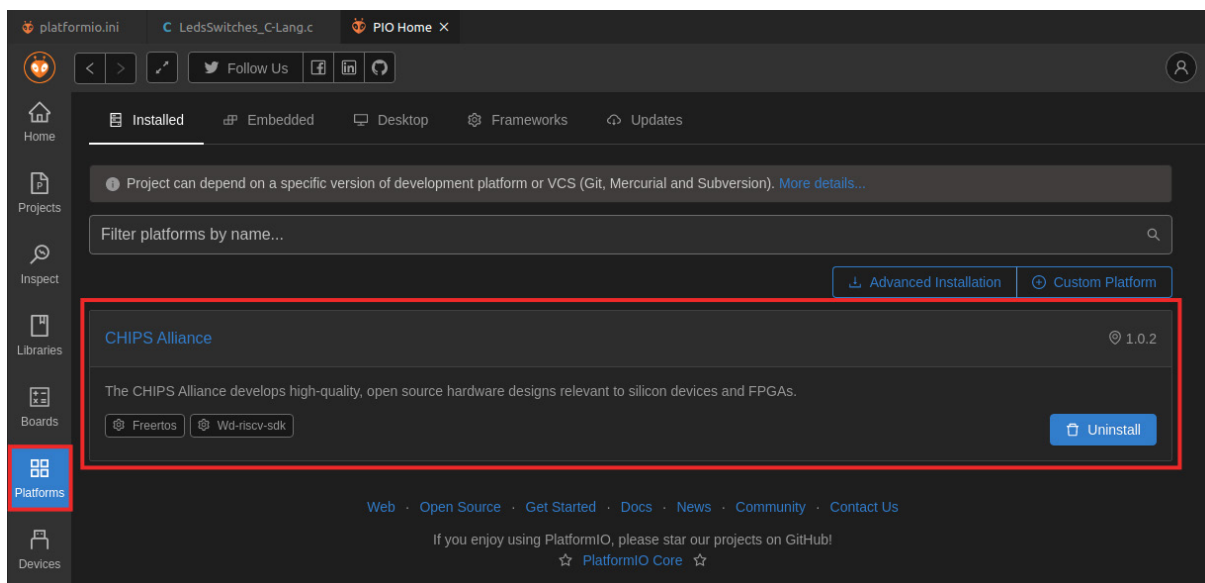


圖40. PlatformIO中安裝的Chips Alliance平台

如果由於某種原因未自動安裝Chips Alliance平台，則可以按照下列步驟手動安裝（通常只需跳過此程序，繼續B節即可）：

- 按一下左列的  按鈕以檢視「Quick Access」（快速存取）功能表（請參閱圖41）。接著在「PIO Home」（PIO首頁）中，依序按一下  按鈕和  索引標籤（圖41）。找到**Chipsalliance**（我們在RVfpga中使用的平台），然後按一下  按鈕將其開啟（圖41）。

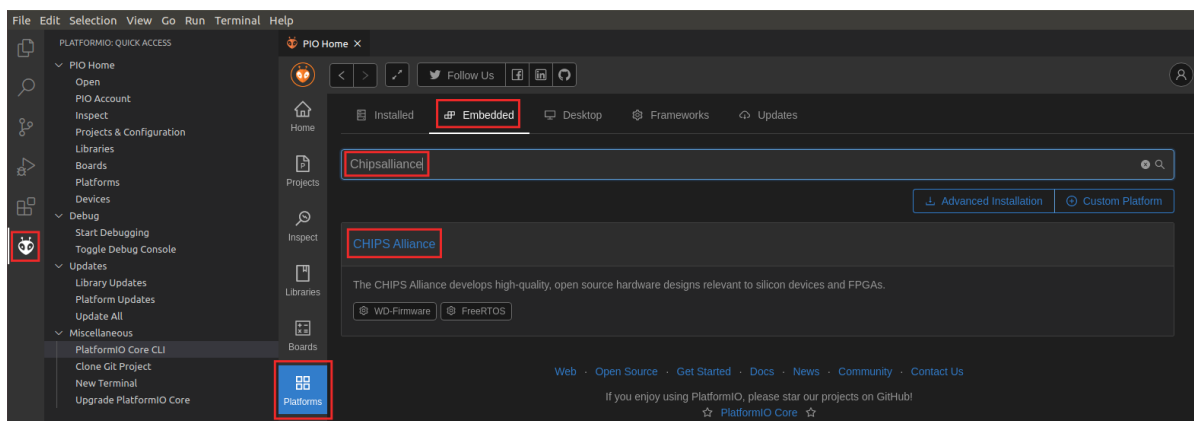
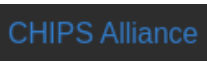



圖41. 選擇CHIPS Alliance平台

- 按一下  按鈕後，將看到有關Chips Alliance平台的詳細資訊（如圖42所示）。按一下  按鈕進行安裝（圖42）。

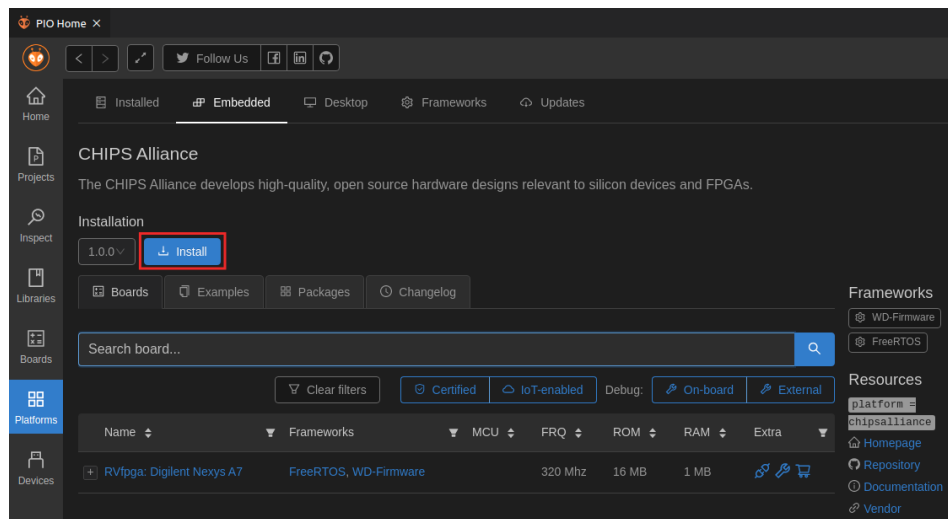



圖42. 安裝CHIPS Alliance平台

- 安裝完畢後，將顯示已安裝工具的摘要資訊，如圖43所示。按一下  以關閉相應視窗。

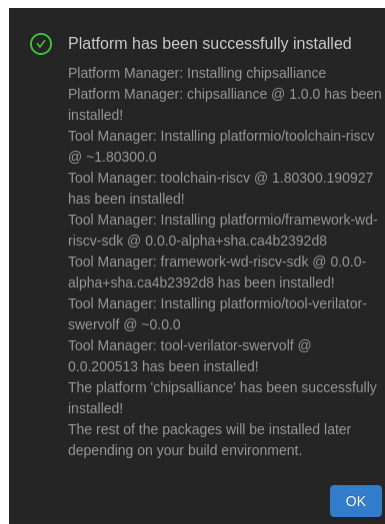


圖43. CHIPS Alliance平台安裝成功

B. AL_Operations程式

第一個範例程式AL_Operations.s（參見圖44）是一個組合語言程式，它在一個無限迴圈內對同一暫存器t3（也稱作x28）執行三條算術邏輯指令（加、減和邏輯與）。


```
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
```

```
12    nop
13
14    .end
```

圖44. AL_Operations程式：AL_Operations.S

按照以下步驟，使用PlatformIO在Nexys A7 FPGA電路板上執行和偵錯此程式碼：

1. 按上節所述對FPGA進行程式設計。請注意，*AL_Operations*專案已在PlatformIO中開啟。
2. 通過以下方式開啟組合程式AL_Operations.S：按一下左側功能表功能區中的「Explorer」

（檔案總管）圖示，展開左側工具列中AL_OPERATIONS下的src，然後按一下AL_Operations.S（請參閱圖45）。

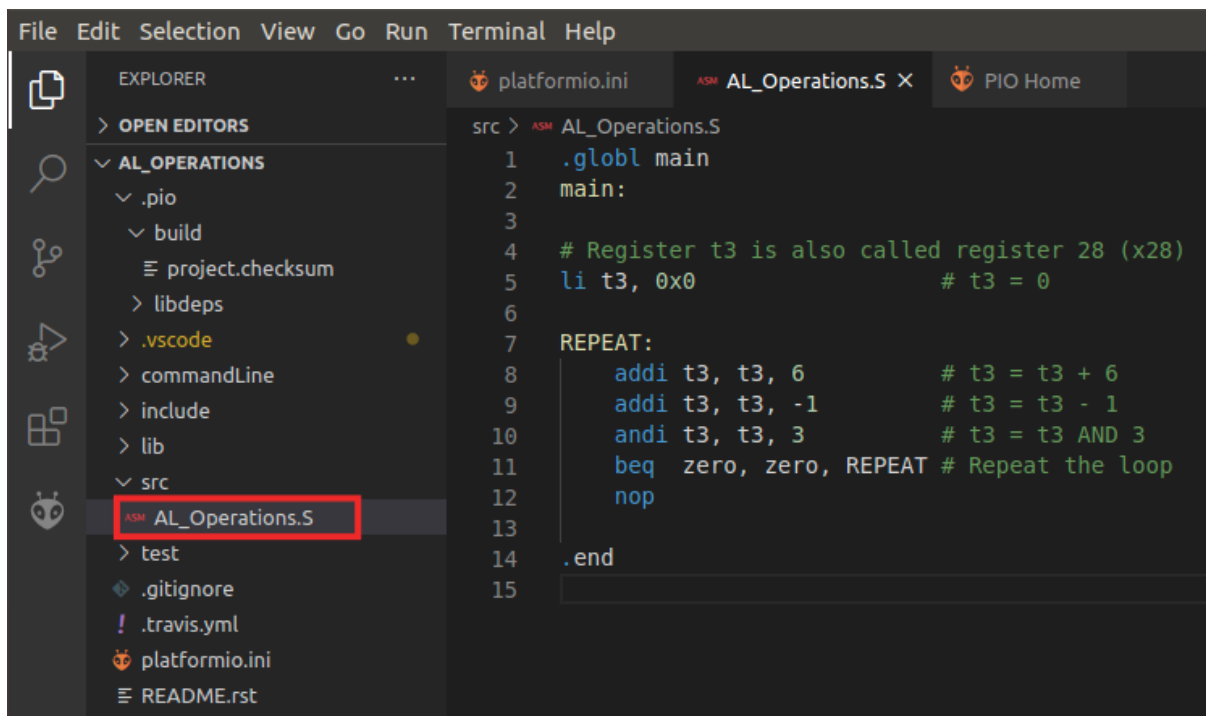








圖45. 檢視組件檔AL_Operations.S

3. VSCode和PlatformIO提供了多種編譯、清除和偵錯程式的方法。在VSCode的底部，可以找到一些提供實用功能的按鈕：。例如，可用於編譯專案，或可用於清除專案。在左列（請參閱圖30）中，「Run」（執行）按鈕可用於編譯程式，然後開啟偵錯工具。

4. 按一下「Run」（執行）按鈕。按一下播放按鈕 **PIO Debug** 啟動偵錯工具（確定已選擇「PIO Debug」（PIO偵錯）選項）。播放按鈕靠近視窗頂端位置（請參閱圖46）。程式將先編譯，然後開始偵錯。PlatformIO在main函數的開頭設定了一個臨時中斷點，因此將在此處停止執行。

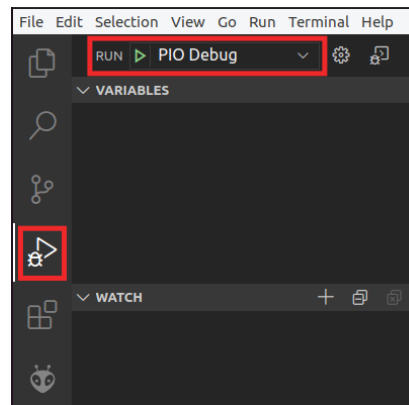


圖46. 啟動偵錯工具

5. 要控制偵錯工作階段，可以使用靠近編輯器頂端位置的偵錯工具列（請參閱圖47）。選項如下：

- **Continue（繼續）**：執行程式，直至達到下一個中斷點。
- **Breakpoints（中斷點）**：可在編輯器中按一下行號左側予以新增。
- **Step Over（單步跳過）**：執行當前行，然後停止。
- **Step Into（單步進入）**：執行目前行，如果目前行包含函數呼叫，將跳轉到該函數並停止。
- **Step Out（單步跳出）**：執行所在函數中的所有程式碼，然後在該函數返回結果時停止。
- **Restart（重新啟動）**：從程式的開頭重新啟動偵錯工作階段。
- **Stop（停止）**：停止偵錯工作階段並返回正常編輯模式。
- **Pause（暫停）**：暫停執行。程式執行時，「Continue」（繼續）按鈕將替換為「Pause」（暫停）按鈕。



圖47. 偵錯工具

6. 在左側提要欄位中，可以檢視偵錯工具選項。提供以下選項：

- **Variables（變數）**：列出程式中存在的區域、全域和靜態變數及其變數值。
- **Call Stack（呼叫堆疊）**：顯示目前正在執行的函數、呼叫函數（如果存在），以及目前指令在記憶體中的位置。
- **「Breakpoints」（斷點）**：顯示所有設定的斷點並強調顯示其行號。可在此部分中管理中斷點。也可以暫時停用中斷點，而無需通過切換核取方塊將其刪除。

- **Peripherals (週邊設備)**：顯示設備的記憶體映射週邊設備的暫存器狀態（我們將在RVfpga實驗中對此進行更詳細的說明）。
- **Registers (暫存器)**：列出存在於處理器的每個暫存器中的目前值。
- **Memory (記憶體)**：顯示特定記憶體位址的內容。
- **Disassembly (反組合語言程式碼)**：顯示特定函數的組合語言程式碼，對於C語言等較高階的程式碼，此選項支援檢視組合語言程式碼以逐行偵錯指令。

7. 展開偵錯工具提要欄位中的「Registers」（暫存器）選項，然後繼續逐步執行



。將觀察到暫存器x28（也稱作t3，如「REGISTERS」（暫存器）部分所示）儲存以下三種算術邏輯運算的結果：加、減和邏輯與。請參閱圖48。

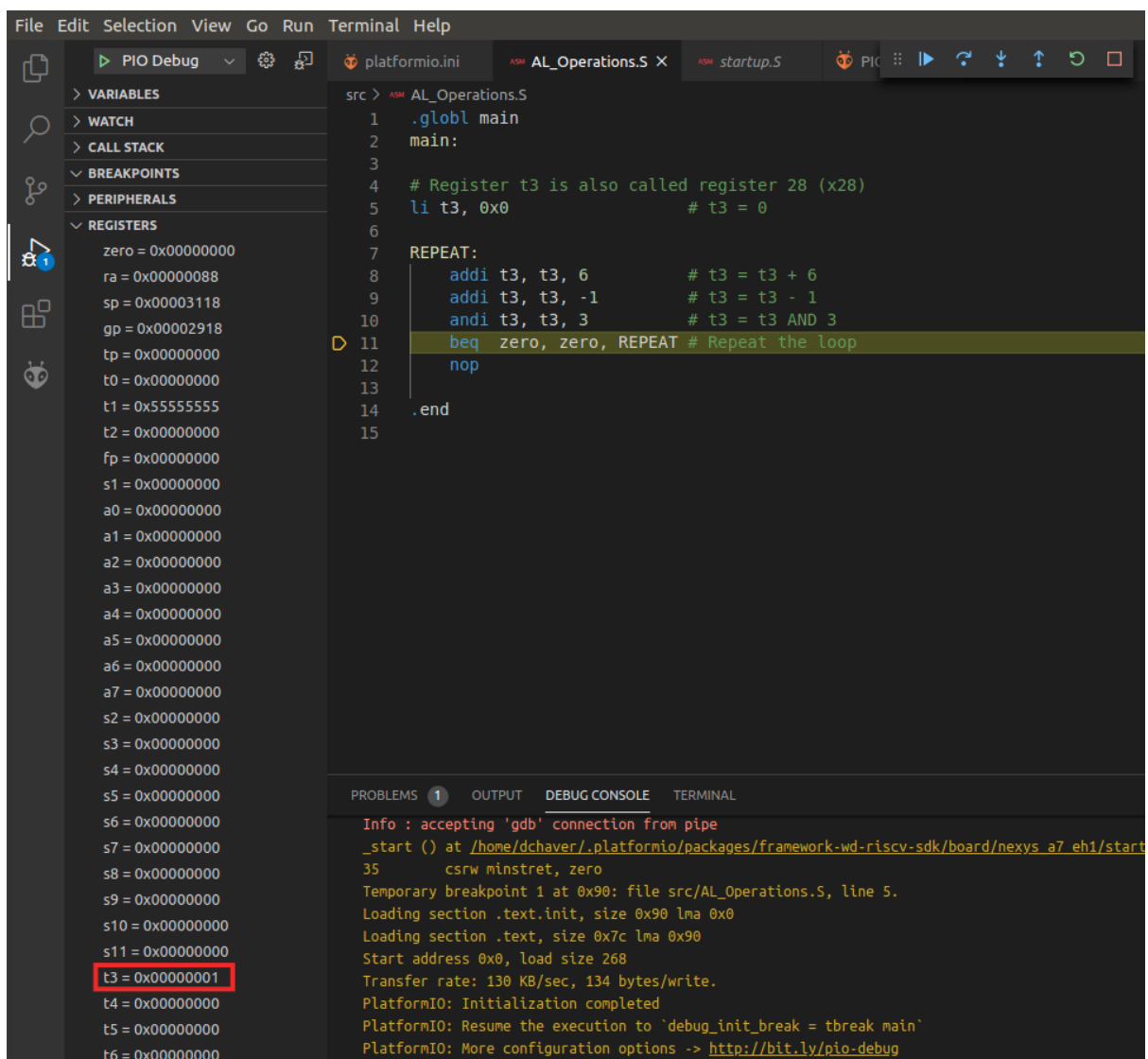


圖48. 檢視暫存器內容

8. 在呼叫main函數之前，將執行Western Digital提供的啟動檔案，該檔案位於
~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S。該檔案

可配置核心：指令快取設定、暫存器初始化（例如**SP**或**gp**）等。啟動偵錯後，該檔案將在主視窗中開啟（請參閱圖48），使用者可在其中檢查該檔案。

Windows： **.platformio**資料夾位於使用者資料夾（**C:\Users\<USER>**）內。請注意，可能需要啟用系統管理才能檢視隱藏的檔案/資料夾。

macOS： 與在Linux中一樣，**.platformio**資料夾位於主資料夾（**~/platformio**）內。

9. 我們還應強調一點，同一目錄（**~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/**）會提供**link.lds**檔，該檔案構成我們將在所有專案中使用的連結器指令碼。該檔案確定組譯部分（文字、資料、**bss...**）在記憶體中的位置。

10. 最後，停止除錯 （這將使啟動ROM程式再次執行），並透過按

一下最左側列頂部的  返回到「**Explorer**」（資源管理器）視窗。在頂端功能表列上，按一下「**File**」（檔案）→「**Close Folder**」（關閉資料夾）。

C. Blinky程式

第二個程式範例**blinky.S**是一個組合語言程式，可使Nexys A7電路板最右側的LED閃爍（請參閱圖49）。該程式會重複地反轉最右側的LED相關值，且每次反轉都會延遲。

```

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 b11:
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 timel:                          # Delay loop
24     addi t2, t2, 1
25     bne t1, t2, timel
26     j b11

```

圖49. blinky.S

按照以下步驟在RVfpgaNexys（已載入到FPGA電路板上的RISC-V SoC）上執行和偵錯此程式碼：

1. 如果執行了第一個範例（**AL_Operations**），則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進行程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用的是**Blinky**範例而非AL_Operations範例。
2. 在頂端列上，按一下「**File**」（檔案）→「**Open Folder**」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/

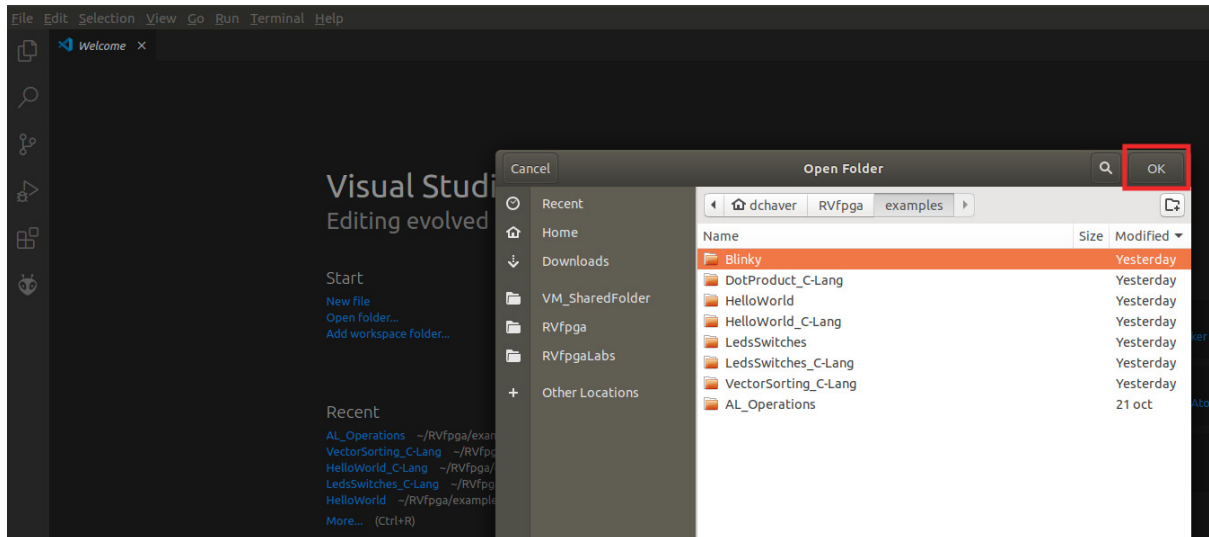


圖50. Blinky程式資料夾

3. 選擇目錄**Blinky**，然後按一下「OK」（確定）（圖50）。
4. 在編輯器中按一下**blinky.S**檔（圖51）以開啟其組合語言程式碼範例。

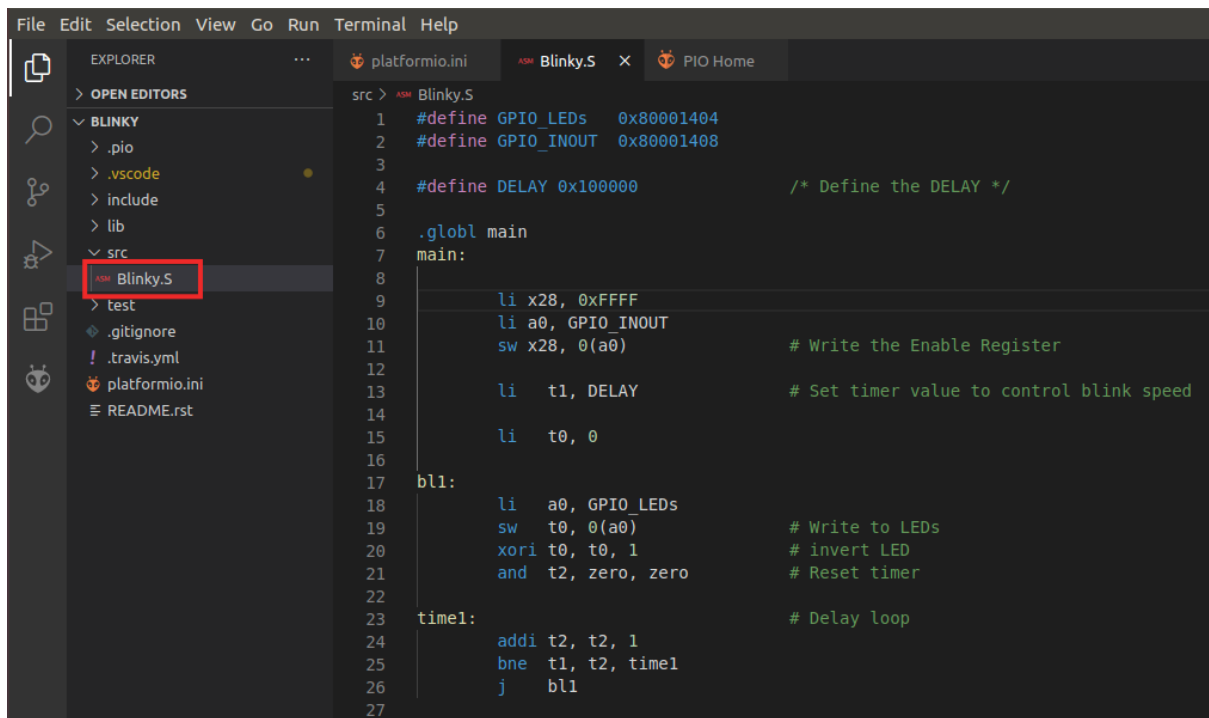


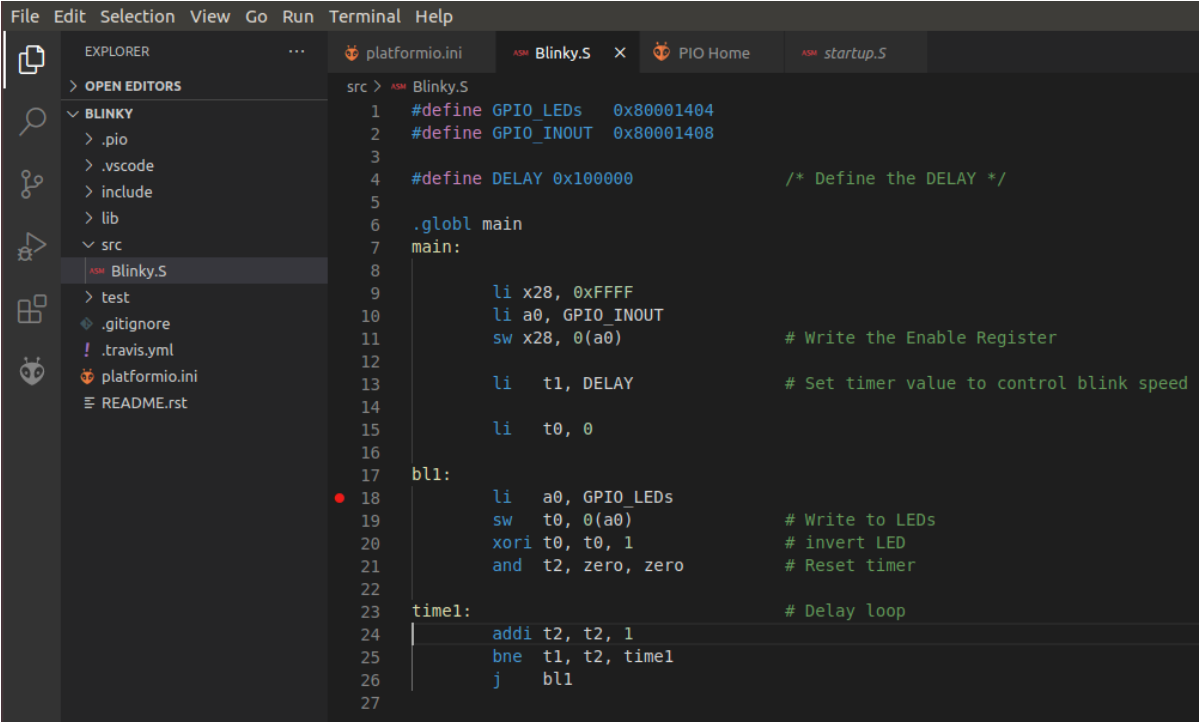


圖51. PlatformIO中的blinky.S

5. 按一下  執行和偵錯程式；然後通過按一下播放按鈕  **PIO Debug** 開始偵錯。PlatformIO在main函數的開頭設定一個臨時中斷點。因此，按一下「Continue」（繼續）按鈕  可執行程式。
6. 在電路板上，將看到最右側的LED開始閃爍。
7. 透過按一下「Pause」（暫停）按鈕  暫停執行。將在無限迴圈內的某個位置停止執行（可能在time1延遲迴圈內）。
8. 通過按一下第18行的左側設定一個中斷點。將出現一個紅點，並且將中斷點新增到「BREAKPOINTS」（中斷點）索引標籤（請參閱圖52）。




```

File Edit Selection View Go Run Terminal Help
EXPLORER
  > OPEN EDITORS
  > BLINKY
    > .pio
    > .vscode
    > include
    > lib
    > src
    > Blinky.S
    > test
    > .gitignore
    > !.travis.yml
    > platformio.ini
    > README.rst
src > ASM Blinky.S
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     sw t0, 0(a0)          # Write to LEDs
20     xori t0, t0, 1        # invert LED
21     and t2, zero, zero    # Reset timer
22
23 timel:                    # Delay loop
24     addi t2, t2, 1
25     bne t1, t2, timel
26     j bl1
27

```

圖52. 在blinky.S中設定中斷點

9. 然後，通過按一下「Continue」（繼續）按鈕  繼續執行。隨後將一直執行到儲存字（sw）指令後停止，該指令會將1（或0）寫入最右側的LED。

10. 繼續執行幾次；將看到驅動到最右側LED的值每次都會發生改變。

11. 停止偵錯 ，然後按一下  返回到「Explorer」（檔案總管）視窗。通過選擇「File」（檔案）→「Close Folder」（關閉資料夾）關閉程式。

D. LedsSwitches程式

第三個組合程式範例將與電路板上的LED和開關通訊（請參閱圖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

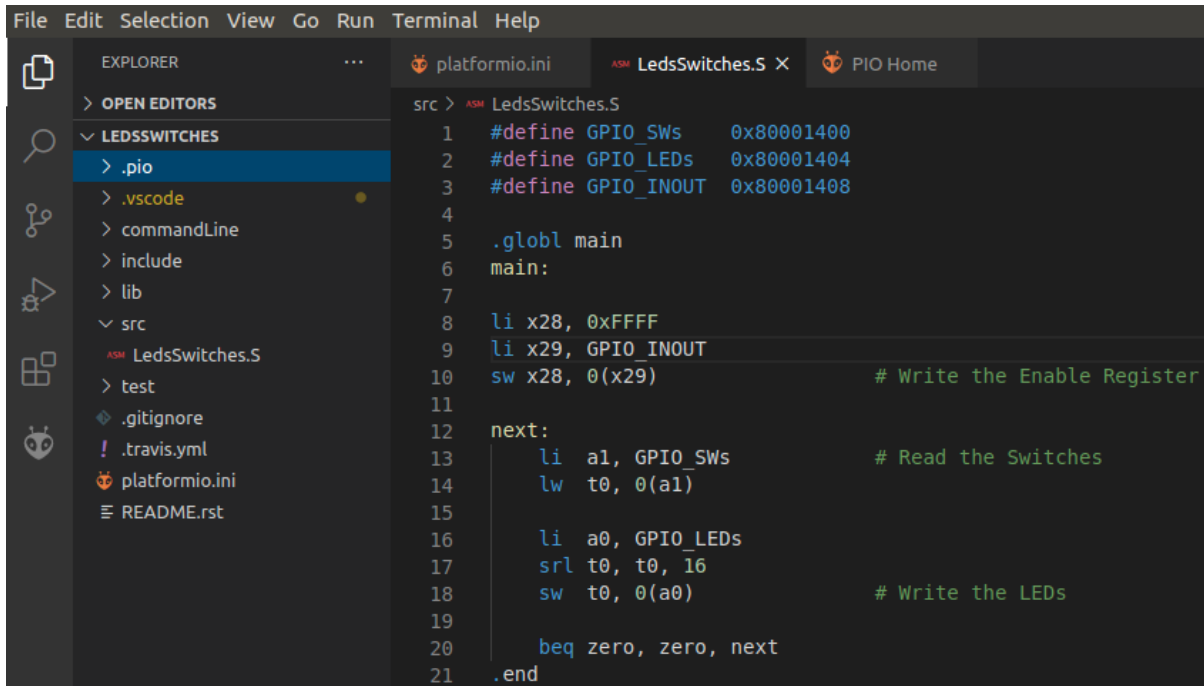
```

```
20    beq zero, zero, next
21 .end
```

圖53. LedsSwitches.S

按照下面的步驟在FPGA電路板上執行和偵錯此程式碼：


1. 如果執行了前幾個範例，則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進行程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用的是LedsSwitches範例而非AL_Operations範例。
2. 在頂端列上，按一下「**File**」（檔案）→「**Open Folder**」（開啟資料夾），然後導覽至目錄 `[RVfpgaPath]/RVfpga/examples/`。選擇目錄 `LedsSwitches`，然後按一下「**OK**」（確定）。
3. 程式 `LedsSwitches.S` 有一個無限迴圈，在其中讀取開關，然後將開關狀態顯示在LED上（圖54）。



```
File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
LEDSSWITCHES
> .pio
> .vscode
> commandLine
> include
> lib
src
  LedsSwitches.S
test
.gitignore
.travis.yml
platformio.ini
README.rst

src > ASM LedsSwitches.S
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
```

圖54. PlatformIO中的LedsSwitches.S

4. 按照先前程式的說明啟動偵錯工具後，程式開始執行。PlatformIO在main函數的開頭設定一個臨時中斷點。因此，按一下「**Continue**」（繼續）按鈕  可執程式。
5. 撥動Nexys A7電路板底部的開關。將立即看到電路板上的LED顯示開關的新值。可以按上文所述暫停執行，逐步執行並檢查暫存器。完成後，按一下「**File**」（檔案）→「**Close Folder**」（關閉資料夾）關閉專案。
6. 有時，檢查記憶體儲存的值可能非常有用。為此，PlatformIO提供了「**Memory**」（記憶體）顯示。

- a. 暫停執行，並跳到`next`迴圈的開始處。展開視窗左側的「Memory」（記憶體）顯示（請參閱圖55），然後按一下「Enter address...」（輸入位址...）

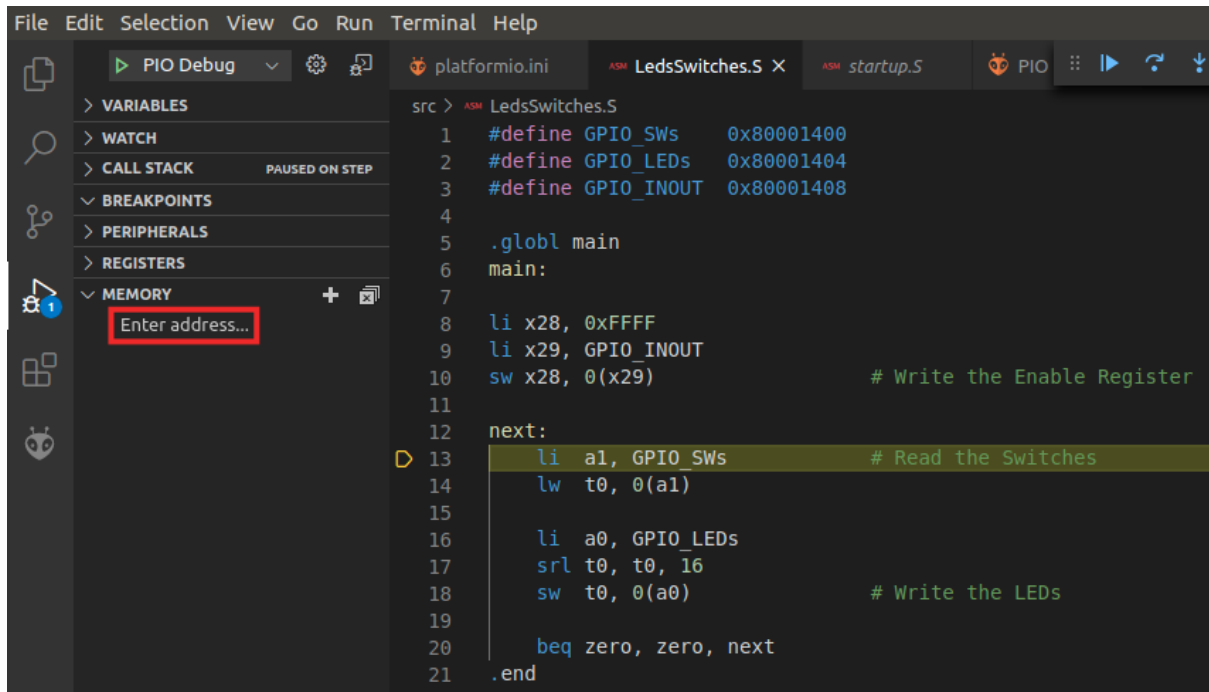


圖55. 記憶體顯示

- b. 將請求初始記憶體位址（請參閱圖56）。在開關的映射位置插入初始位址，在本例中為0x80001400。

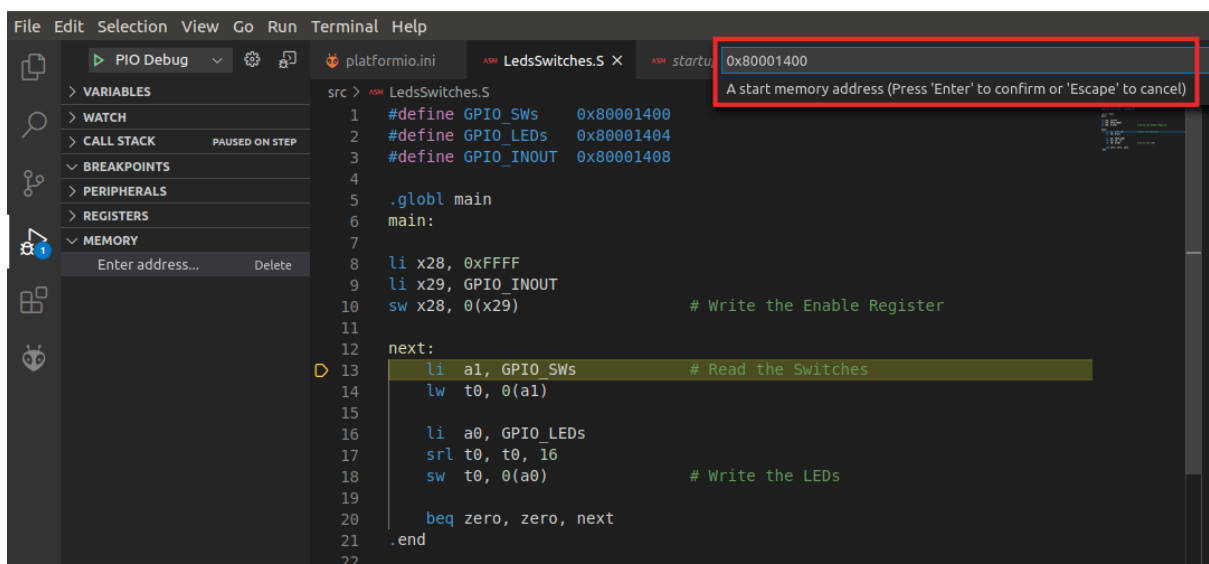


圖56. 要顯示的初始記憶體位址

- c. 然後，請求要檢查的位元組數（請參閱圖57），為此插入一個值0xc（我們要檢查三個4位元組的I/O暫存器，因此需要12個位元組）。

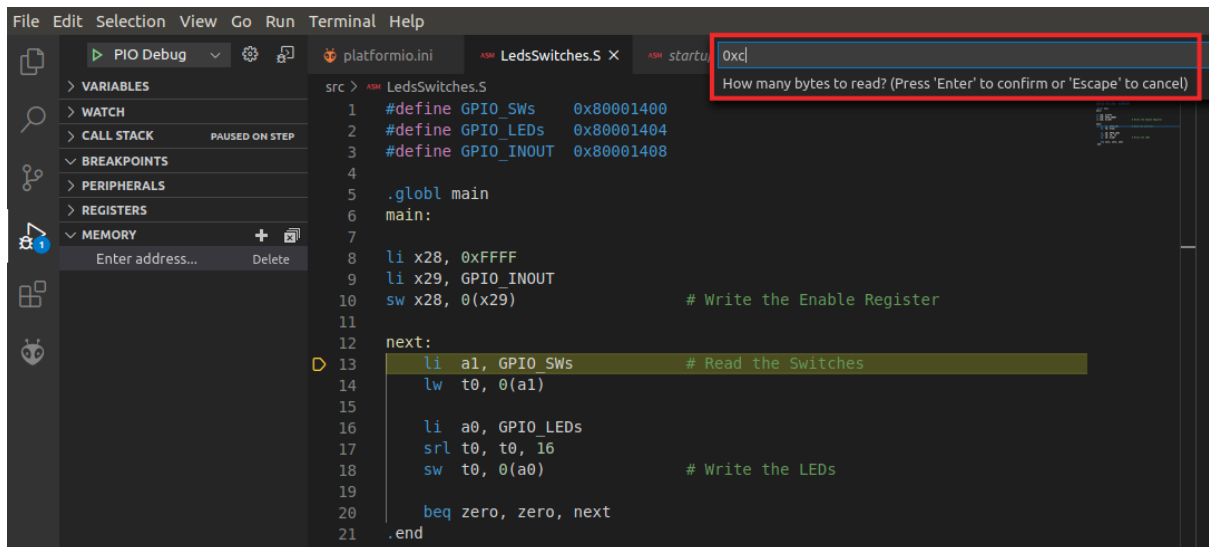


圖57. 要顯示的位元組數

- d. 「Memory」（記憶體）顯示將在右側開啟，其中顯示我們已請求的12個位元組（請參閱圖 58）。我們在 16個開關中的值為0x123C（請參閱位址 0x80001402和0x80001403處的位元組）。考慮到RISC-V架構採用小端模式，因此圖中所示的值與之一致。16個LED（儲存在位址 0x80001404 和 0x80001405處）顯示相同的值。

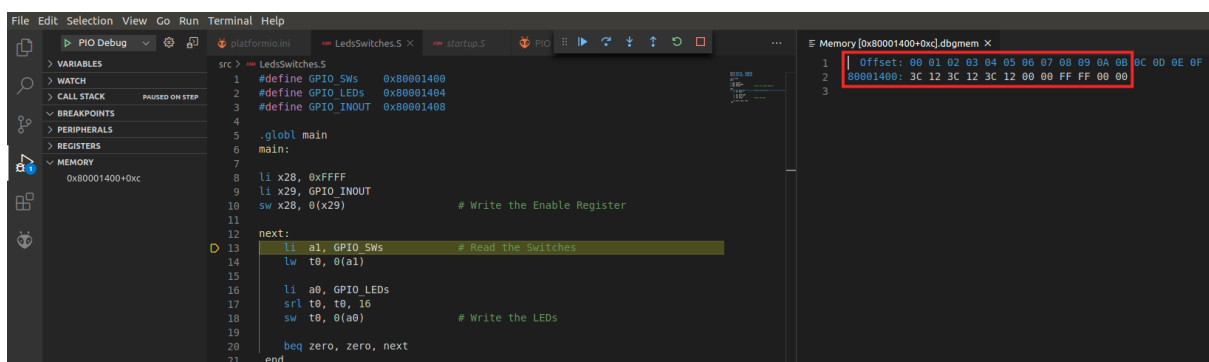
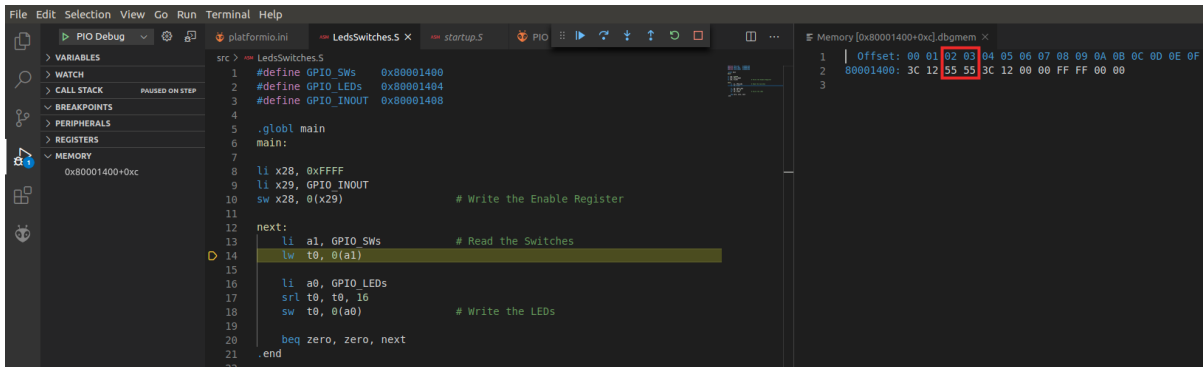


圖58. 記憶體位址0x80001400-0x8000140B

- e. 例如，將電路板上的開關的值變更為0x5555，然後再逐步執行一次迴圈迭代。記憶體中的開關值應在執行第一行指令後立即變更（圖59，上圖），而LED的值應在執行sw指令後相應地變更（圖59，下圖）。



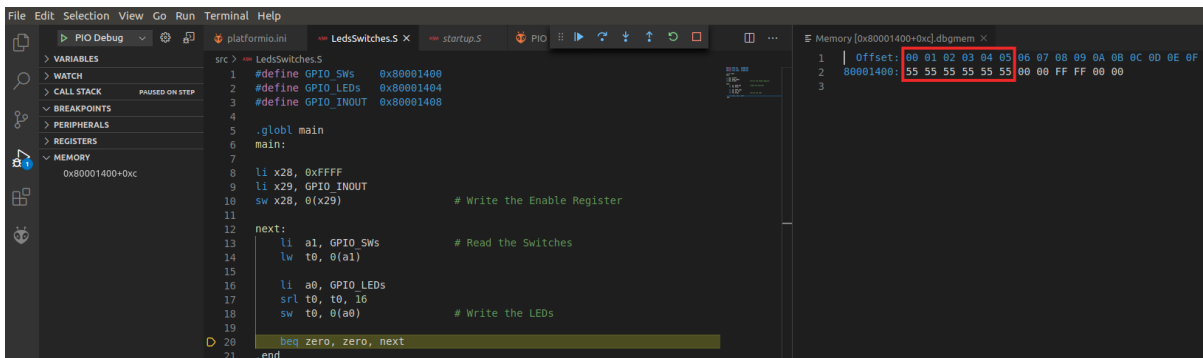
```

src > LedsSwitches.S
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

```

Memory [0x80001400+0xc].dbgmem X

1	Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	00001400:	3C	12	55	55	3C	12	00	00	FF	FF	00	00	00	00	00	00
3																	



```

src > LedsSwitches.S
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

```

Memory [0x80001400+0xc].dbgmem X

1	Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	00001400:	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55	55
3																	

圖59. 開關和LED變更

- f. 也可以檢視其他記憶體位置，例如用於儲存程式的機器指令的RAM位址。開啟另一個記憶體範圍，其起始位址為0x0（分配給RAM記憶體的初始位址），佔用0x100個位元組（圖60）。在啟動程式（Startup.S）之後，將立即看到LedsSwitches程式中儲存在位址範圍0x90-0xC4內的指令。

Memory [0x0+0x100].dbgmem X

1		Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2		00000000:	73	10	20	B0	73	10	20	B8	81	40	01	41	81	41	01	42
3		00000010:	81	42	01	43	81	43	01	44	81	44	01	45	81	45	01	46
4		00000020:	81	46	01	47	81	47	01	48	81	48	01	49	81	49	01	4A
5		00000030:	81	4A	01	4B	81	4B	01	4C	81	4C	01	4D	81	4D	01	4E
6		00000040:	81	4E	01	4F	81	4F	37	53	55	55	13	03	53	55	73	10
7		00000050:	03	7C	97	31	00	00	93	81	E1	8E	17	31	00	00	13	01
8		00000060:	61	0E	17	25	00	00	13	05	E5	0D	97	25	00	00	93	85
9		00000070:	65	0D	63	77	B5	00	23	20	05	00	11	05	E3	6D	B5	FE
10		00000080:	91	20	01	45	81	45	29	20	01	A0	00	00	00	00	00	00
11		00000090:	37	0E	01	00	13	0E	FE	FF	B7	1E	00	80	93	8E	8E	40
12		000000a0:	23	A0	CE	01	B7	15	00	80	93	85	05	40	83	A2	05	00
13		000000b0:	37	15	00	80	13	05	45	40	93	D2	02	01	23	20	55	00
14		000000c0:	E3	02	00	FE	41	11	22	C4	4A	C0	17	04	00	00	13	04
15		000000d0:	64	F3	17	09	00	00	13	09	E9	F2	33	09	89	40	06	C6
16		000000e0:	26	C2	13	59	29	40	63	09	09	00	81	44	1C	40	85	04
17		000000f0:	11	04	82	97	E3	1C	99	FE	17	04	00	00	13	04	84	F0
18																		

圖60. 記憶體位址0x0至0x100

- g. 通過開啟程式的反組譯版本，可以檢視程式指令的機器程式碼，程式的反組譯版本位於：

`[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/firmware.dis`（請參閱圖61）。將兩幅圖進行比較，嘗試確定程式的指令。

```

65 Disassembly of section .text:
66
67 00000090 <main>:
68   90: 00010e37      lui t3,0x10
69   94: fffe0e13      addi t3,t3,-1 # ffff <_sp+0xcebf>
70   98: 80001eb7      lui t4,0x80001
71   9c: 408e8e93      addi t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
72   a0: 01cea023      sw t3,0(t4)
73
74 000000a4 <next>:
75   a4: 800015b7      lui a1,0x80001
76   a8: 40058593      addi a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
77   ac: 0005a283      lw t0,0(a1)
78   b0: 80001537      lui a0,0x80001
79   b4: 40450513      addi a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
80   b8: 0102d293      srli t0,t0,0x10
81   bc: 00552023      sw t0,0(a0)
82   c0: fe0002e3      beqz zero,a4 <next>
83

```

圖61. LedsSwitches程式的反組譯版本

E. LedsSwitches_C-Lang程式

程式LedsSwitches_C-Lang.c（圖62）與先前顯示的LedsSwitches.s程式（圖53）的功能相同，但它是用C語言而不是組合語言編寫的。

```

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 }

```

圖62. LedsSwitches_C-Lang.c

按照下面的步驟在FPGA電路板上執行和偵錯此程式：

1. 如果執行了前幾個範例，則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進行程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用LedsSwitches_C-Lang範例而非AL_Operations範例。
2. 在頂端功能表列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/。選擇目錄LedsSwitches_C-Lang，然後按一下「OK」（確定）。
3. 在呼叫偵錯工具之前，先在C程式碼的第15行設定一個中斷點。
4. 然後，開始偵錯。程式將開始執行，並將在中斷點處停止（圖63）。

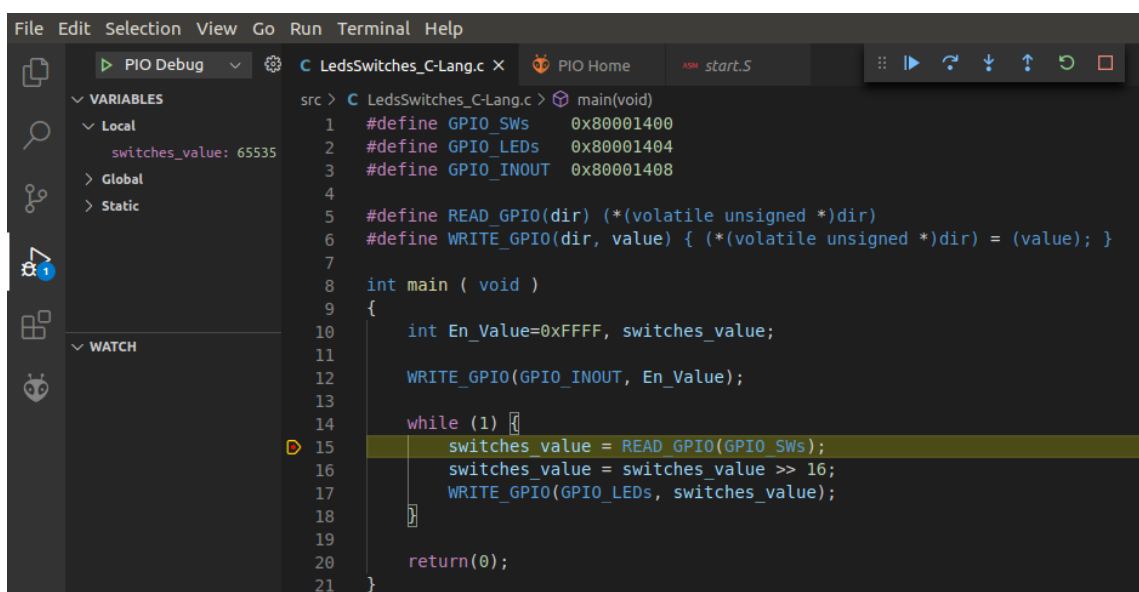



圖63. 在中斷點處停止執行

5. 使程式繼續執行  幾次，但在每次按一下時變更開關。LED應顯示開關的值。
6. 可以檢視上述C語言程式的執行情況，也可以通過按一下圖64中突出顯示的「Switch to assembly」（切換到組合語言）來檢視編譯器所產生的組合語言程式的執行情況。

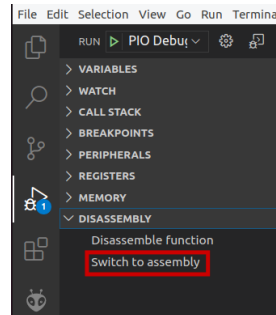


圖64. 切換到組合語言

7. 組合語言中的程式（圖65）首先使用載入指令（`lw a5,1024(a4)`）讀取開關值，然後使用儲存指令（`sw a5,1028(a4)`）將該值寫入LED。逐步執行程式，變更開關並驗證LED發生變化以反映新的開關值。

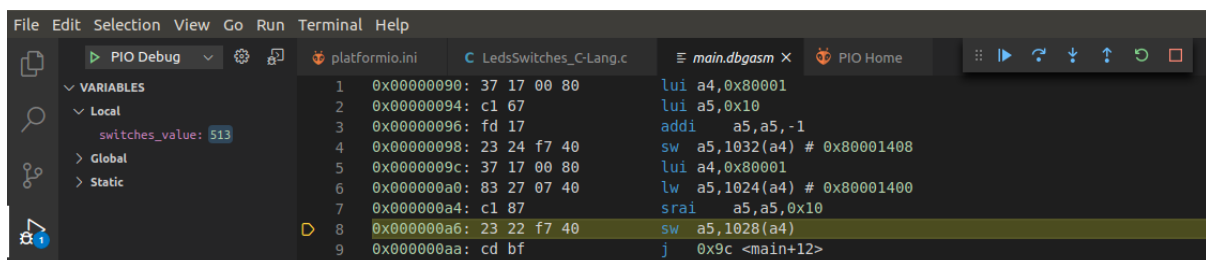


圖65. 組合語言程式

F. HelloWorld_C-Lang程式

第二個C程式範例通過序列埠向shell列印一則短訊息。要檢視此訊息，可以使用任何終端機模擬器，例如`gtkterm`、`minicom`等；但是，PlatformIO提供了自己的序列監視器，因此我們在這裡展示如何使用此監視器。

為了配置PlatformIO序列監視器，必須配置一些參數；具體地說，必須確定用於序列資料傳輸的資料速率（以每秒位元數或波特數為單位），為此，我們可以使用`platformio.ini`檔（請注意，此檔案是PlatformIO專案的一部分）中的參數`monitor_speed`來確定該值。請參閱圖66。

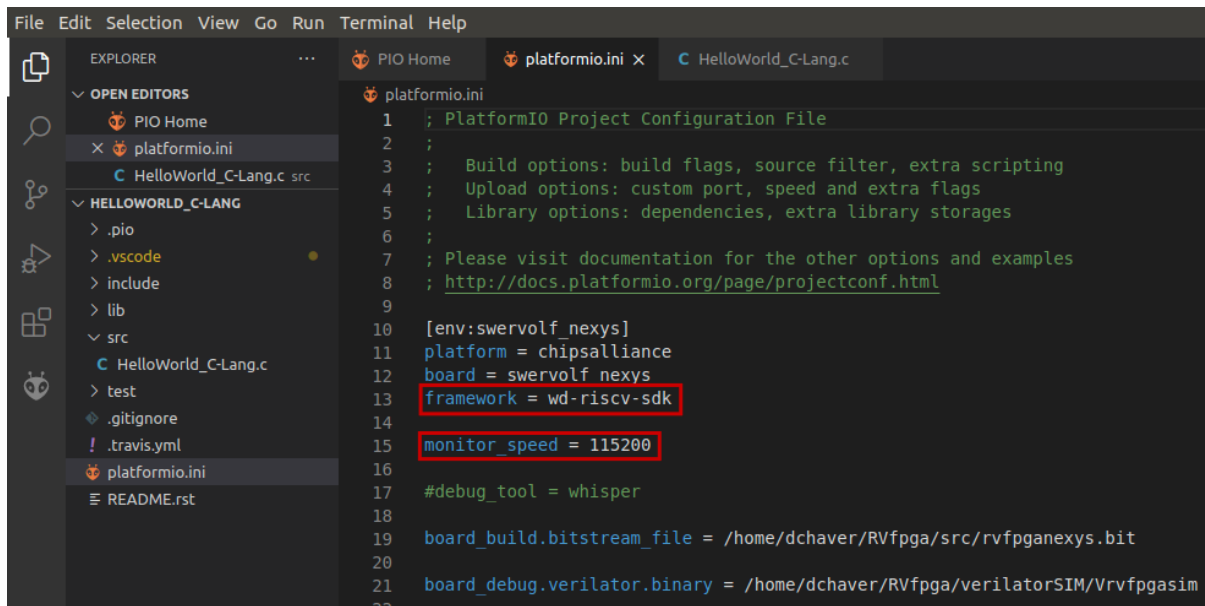


圖66. 序列監視器配置

此外，需要在終端機中輸入以下命令，將您自己新增到dialout、tty和uucp群組中：

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

執行這三行命令後，重新啟動電腦，以使群組變更生效。

Windows/macOS：Windows和macOS使用者不需要完成上述步驟。

此外，該程式使用WD在其韌體套件（<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>）中提供的處理器支援套件（Processor Support Package，PSP）和電路板支援套件（Board Support Package，BSP）。通過在platformio.ini中使用特定命令（framework = wd-riscv-sdk）（如圖66所示），並在C程式的開頭包括正確的檔案（如圖67所示）將這些函數庫包括在專案中。有關係統中的完整函數庫，請造訪以下路徑：

- PSP：~/platformio/packages/framework-wd-riscv-sdk/psp/
- BSP：~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/

這些函數庫提供許多函數和巨集，使用者可借此執行許多操作，例如使用中斷、列印字串、讀取/寫入各個暫存器等。在本例中，我們將使用printfNexys函數在序列監視器上列印一則訊息。在後面的範例和實驗中，我們將展示如何使用其他函數和巨集來實現不同的目的。

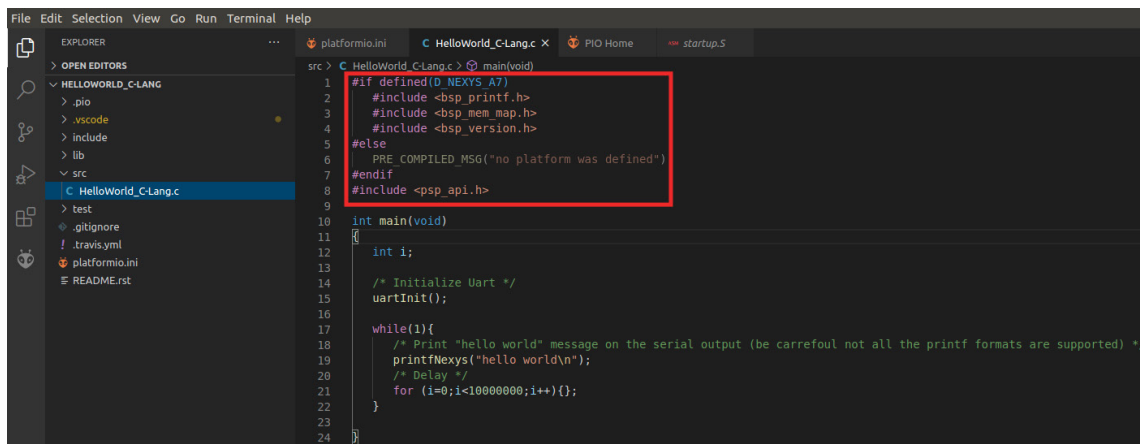


圖67. 在HelloWorld_C-Lang.c中包括.h檔案

按照下面的步驟在FPGA電路板上執行和偵錯此程式碼：

1. 如果執行了前幾個範例，則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進行程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用的是HelloWorld_C-Lang範例而非AL_Operations範例。
2. 開啟VSCode。開啟VSCode時，PlatformIO應在VSCode中自動開啟。在頂端列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/。選擇HelloWorld_C-Lang資料夾，然後按一下「OK」（確定）。
3. 程式HelloWorld_C-Lang.C（圖68）初始化UART（函數uartInit），然後使用函數printfNexys通過序列埠傳送字串（可以在檔案~/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/bsp_printf.c中找到這些函數的實作）。該程式接著會延遲一段時間，之後再返回迴圈的開始處。

```

1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <psp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<10000000;i++){
22             ;
23         }
24     }

```

圖68. HelloWorld_C-Lang.C的main函數

4. 在PlatformIO中啟動偵錯工具。當程式開始執行時，通過按一下VS Code底部的插頭按鈕（圖69）開啟序列監視器。

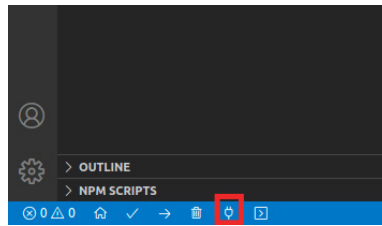


圖69. 開啟序列終端機

5. 序列監視器重複列印訊息「HELLO WORLD !!!」，如圖70所示。

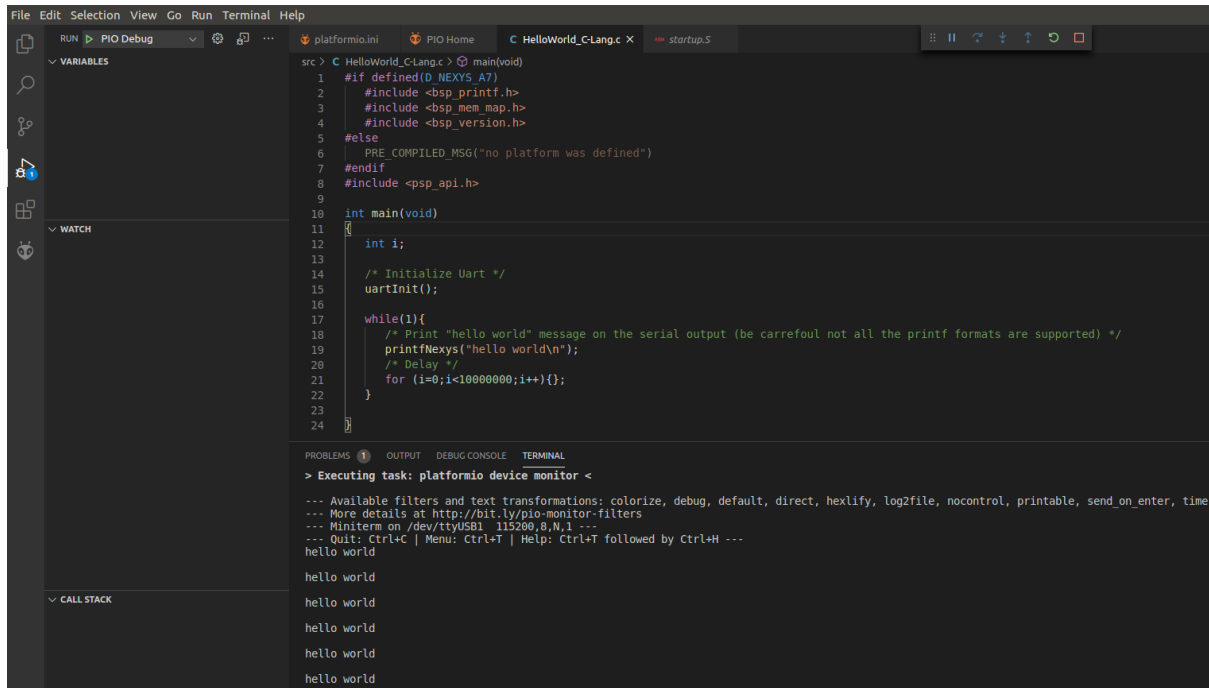


圖70. 程式的執行

G. VectorSorting_C-Lang程式

最後，我們展示另一個C程式，該程式從大到小對向量A的各元素進行排序，並將排序後的值放在向量B中。向量A的值替換為零。圖71顯示了該程式。

```

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 }

```

圖71. VectorSorting_C-Lang.c

按照下面的步驟在FPGA電路板上執行和偵錯此程式：

1. 如果執行了前幾個範例，則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用的是VectorSorting_C-Lang範例而非AL_Operations範例。
2. 在頂端功能表列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/。選擇VectorSorting_C-Lang資料夾，然後按一下「OK」（確定）。
3. 在第10行放置一個中斷點並開始偵錯。執行將在for迴圈開始處停止（圖72）。展開偵錯工具提要欄位中的「VARIABLES」（變數）部分，然後分析A和B陣列的值（如圖72中紅色突出顯示部分所示）。

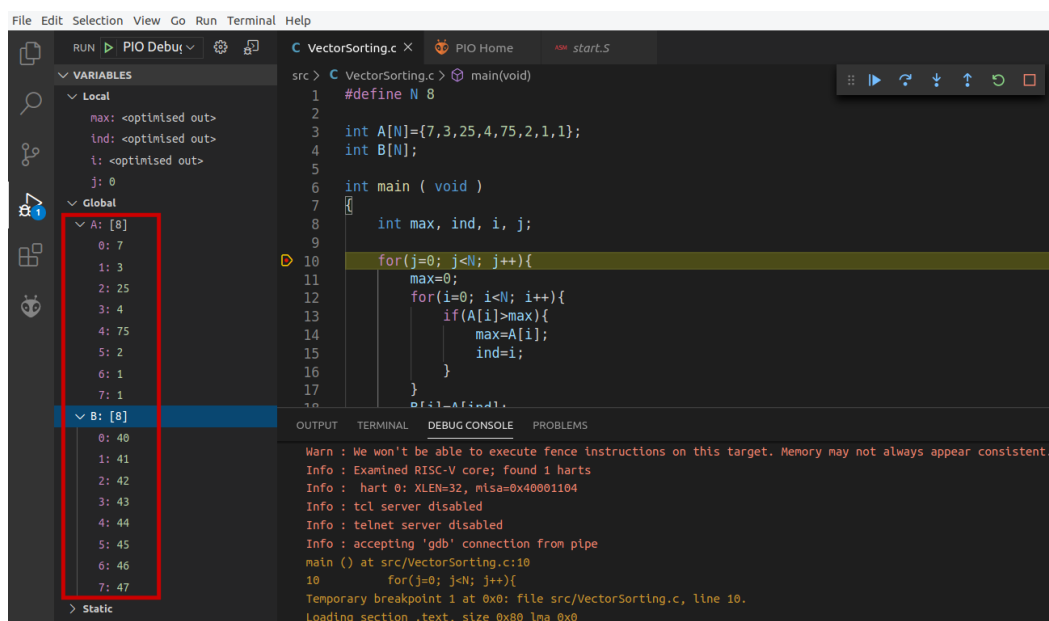



圖72. 在程式開始處停止執行

4. 現在在第18行放置另一個中斷點，並通過按一下  繼續執行（請參閱圖73）。開啟「Memory」（記憶體）顯示（如LedsSwitches程式部分所述，圖55），從位址0x2148（該位址為此程式的記憶體中用於儲存向量A的位址）開始顯示0x50個位元組（請參閱圖73）。可以檢視向量A（範圍為0x2148-0x2167）和B（範圍為0x2178-0x2197）的初始值。

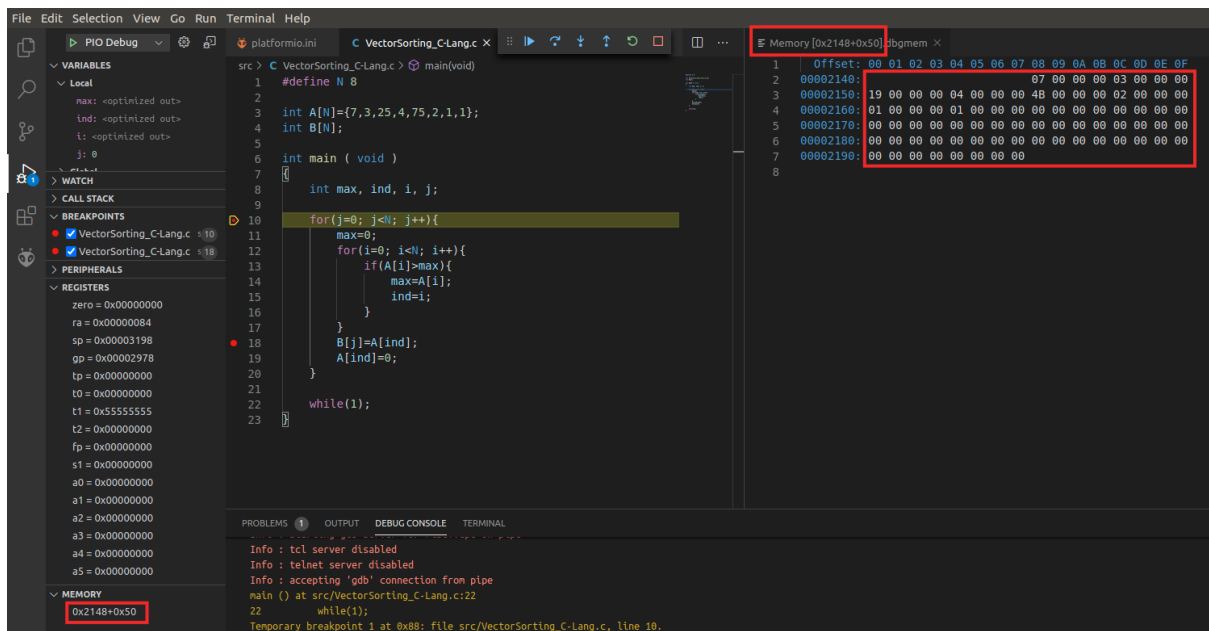


圖73. 陣列A和陣列B的記憶體顯示 – 初始狀態

請注意，通過切換到組合語言（如圖64所示），並分析存取向量A和向量B的任何指令，可以輕鬆找到記憶體中用於儲存這些向量的位址（圖74）。如圖中所示，在大多數情況下，圖形都提供此資訊；但也可以繼續執行這些指令，並檢視儲存在暫存器中的值。

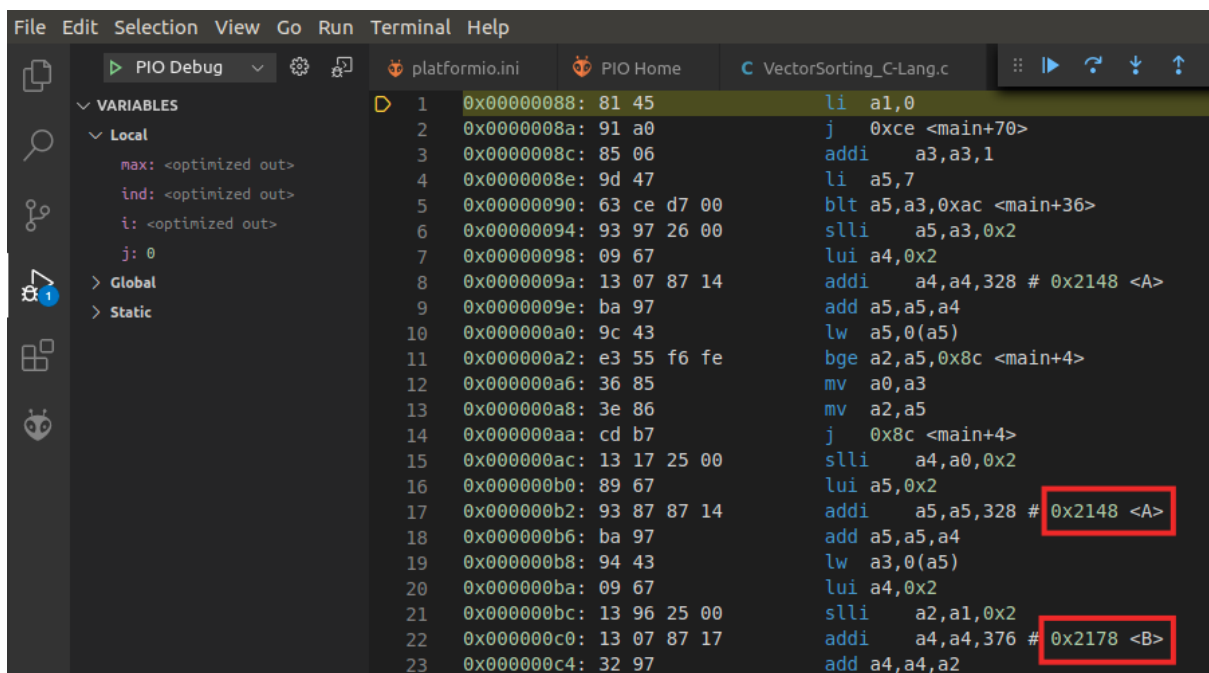
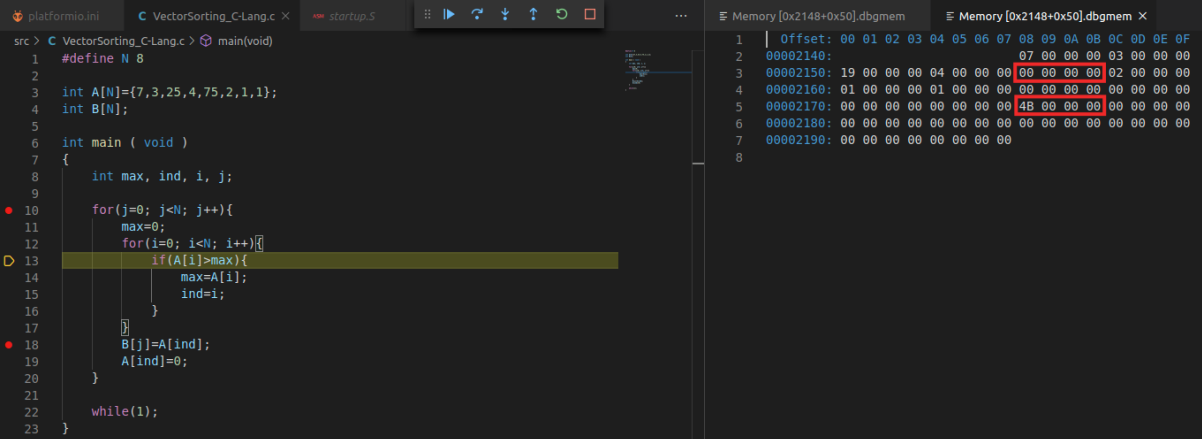


圖74. 記憶體中用於儲存A和B的位址。

按一下「Step Over」（單步跳過）按鈕（）兩次，將看到B的第一個元件儲存在記憶體中，並將A中的對應值設定為0（請參閱圖75）。



```

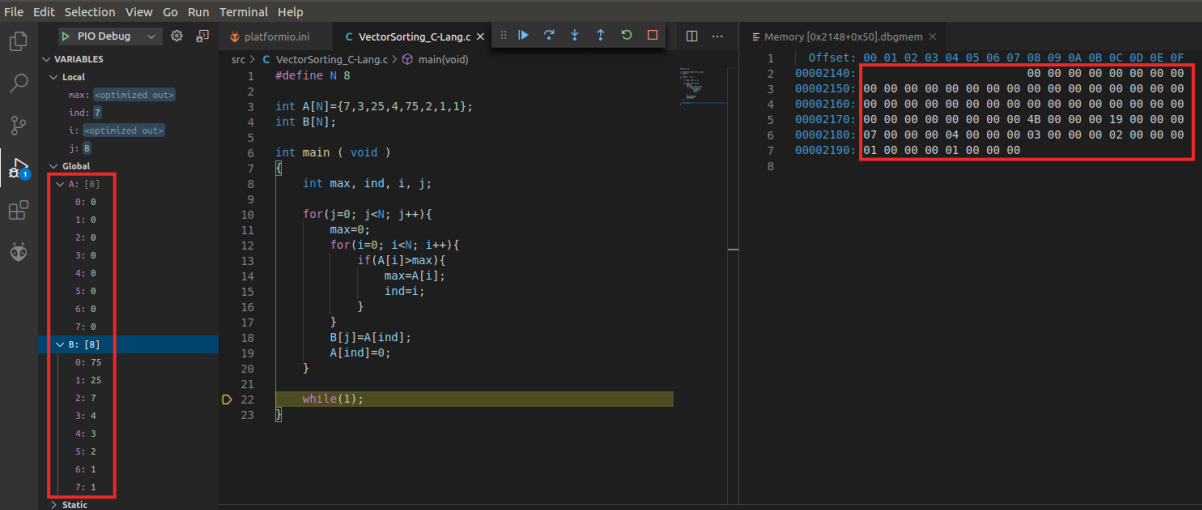
src > C VectorSorting_C-Lang.c > main(void)
1  #define N 8
2
3  int A[N]={7,3,25,4,75,2,1,1};
4  int B[N];
5
6  int main ( void )
7  {
8      int max, ind, i, j;
9
10     for(j=0; j<N; j++){
11         max=0;
12         for(i=0; i<N; i++){
13             if(A[i]>max){
14                 max=A[i];
15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

```

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00002140:	19	00	00	00	04	00	00	00	00	00	00	00	00	00	00	00
00002150:	01	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00
00002160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002180:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002190:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

圖75. 陣列A和陣列B的記憶體顯示 – 儲存B的第一個元件並重設A中的相應元件。

5. 移除所有中斷點，繼續執行，並在幾秒鐘後暫停執行 – 此時程式將已完成執行。再次分析儲存在陣列A和陣列B中的值。如圖76所示，向量B保留原始向量A的值（從大到小排序），向量A全為零（在左側的變數清單和右側的記憶體控制台中均可檢視這一點）。



```

src > C VectorSorting_C-Lang.c > main(void)
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 }

```

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00002140:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002150:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002160:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002170:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00002180:	07	00	00	00	04	00	00	00	03	00	00	00	02	00	00	00
00002190:	01	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00

圖76. 在程式結束處停止執行

H. DotProduct_C-Lang程式

最後一個程式範例DotProduct_C-Lang.c（圖77）計算兩個向量的點積。該程式具有兩個函數：*main*和*dotproduct*。*main*函數使用以下三個輸入引數呼叫*dotproduct*函數：向量大小和兩個向量的初始位址。然後，*dotproduct*函數計算兩個向量的點積並傳回結果。

```

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 }

```

圖77. DotProduct_C-Lang.c

在本範例中，我們使用實數進行運算（請注意，變數x、y和dot的資料類型是double）。但是，SweRV EH1處理器不支援浮點數。因此，該範例通過gcc

（<https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html>）提供的軟體浮點數函數庫來使用浮點數模擬。只要包含-msoft-float，該函數庫即可禁止產生浮點數指令。

按照下面的步驟在FPGA電路板上執行和偵錯此程式碼：

1. 如果執行了前幾個範例，則已在FPGA電路板上設計了RVfpgaNexys的程式，因此無需再次對其進行程式設計。但是，如果確實需要在電路板上重新設計RVfpgaNexys的程式，請按照A節所述進行操作，注意使用的是DotProduct_C-Lang範例而非AL_Operations範例。
2. 在頂端功能表列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/。選擇目錄DotProduct_C-Lang，然後按一下「OK」（確定）。
3. 在呼叫偵錯工具之前，先在第10行和第19行分別設定一個中斷點（請參閱圖78）。
4. 然後，開始偵錯。該程式將開始執行；並在第一個中斷點處停止（請參閱圖78）。
5. 在偵錯工具提要欄位上，展開「Variables」（變數）部分（請參閱圖78）。這兩個向量包含在main中分配的初始值。dot變數初始化為0。

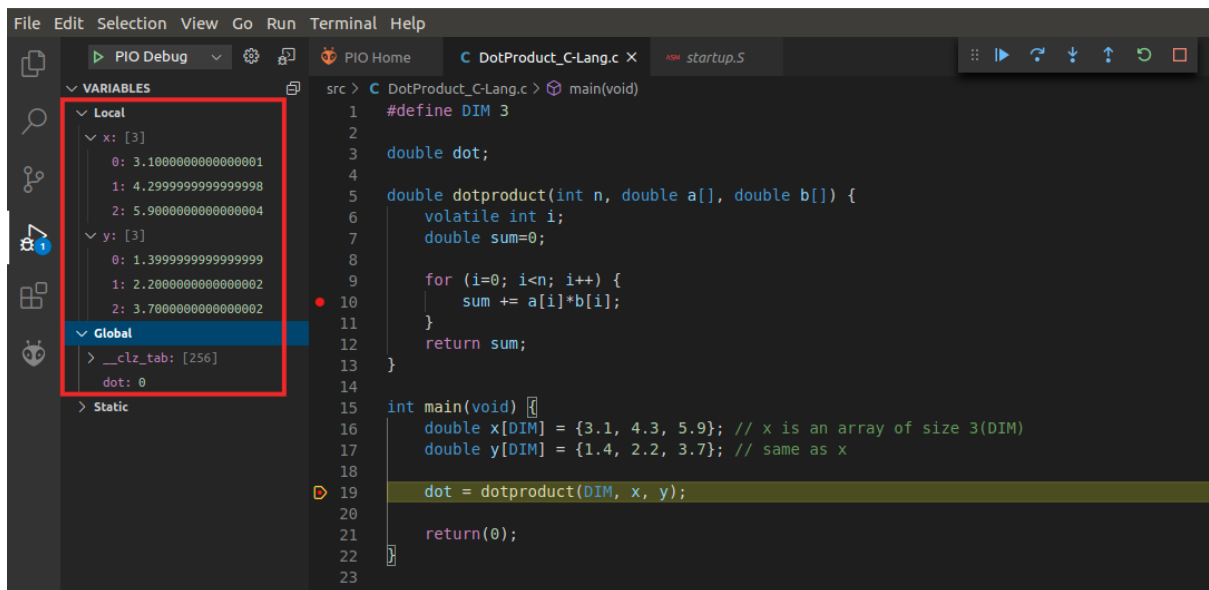



圖78. DotProduct_C-Lang程式：第一個斷點處的變數值

6. 使程式繼續執行 。程式在第二個中斷點處停止（第10行）。
7. 切換到組合語言（如圖64所示操作）。可以檢視浮點數模擬常式，並通過單步進入這些常式來對其進行深入分析（請參閱圖79）。

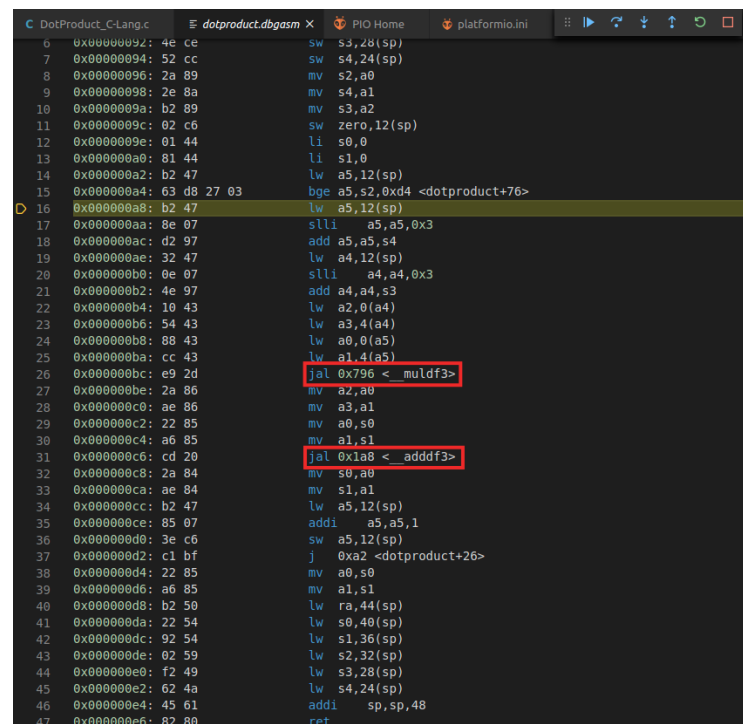
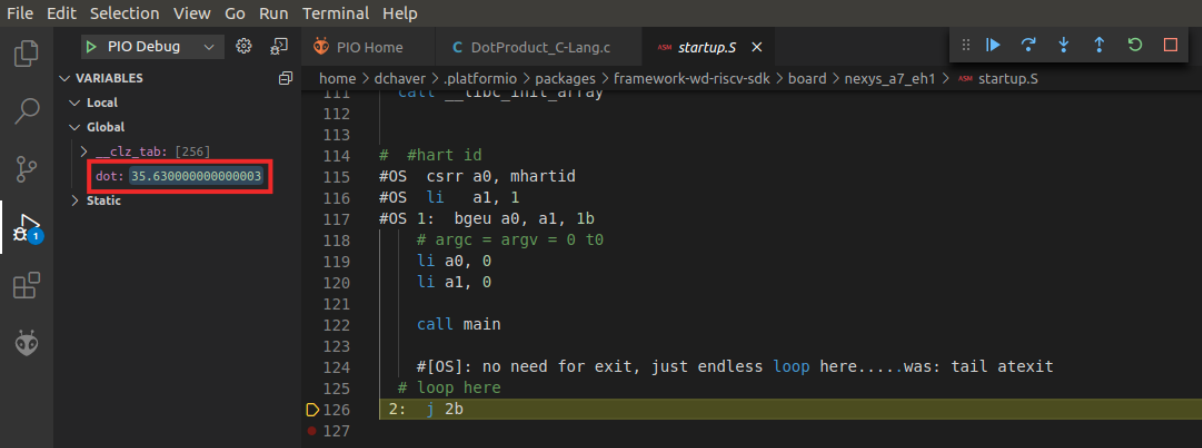


圖79. DotProduct_C-Lang程式：第二個斷點處的組合語言程式碼

8. 切換回C程式並刪除兩個中斷點。繼續執行，然後將其暫停。將看到`dot`變數的值變為兩個向量的點積（圖80）。



The screenshot shows the PIO IDE interface. On the left, the 'VARIABLES' window is open, showing a list of variables. The variable 'dot' is highlighted with a red box, and its value is '35.630000000000003'. The main editor shows assembly code for 'startup.S'. The code includes comments and instructions for setting up the hardware and running the main function. The line numbers 111 through 127 are visible on the left side of the code editor.

圖80. DotProduct_C-Lang程式：點積的結果

9. 完成對此程式的探索後，按一下「*File*」（檔案）→「*Close Folder*」（關閉資料夾）來關閉專案。

7. VERILATOR模擬

在本節中，將在RVfpgaSim上使用Verilator執行上一節中使用的第一個程式（*AL_Operations*）。Verilator是一種硬體描述語言（Hardware Description Language，HDL）模擬器，用於模擬定義SoC的Verilog（可從[RVfpgaPath]/RVfpga/src取得）。通過這種執行SoC的方式，使用者能夠分析系統的內部訊號，這對於後續的實驗和練習（在這些實驗和練習中我們將向SoC新增內部操作或新硬體）特別有用。

在此，我們展示了如何使用Verilator來檢視逐週期的指令和*AL_Operations*的暫存器值，*AL_Operations*是在第5節中執行和偵錯的第一個簡單的組合語言程式（圖44）。將使用PlatformIO產生模擬軌跡，然後將時脈、兩路超標量處理器的指令和暫存器x28（即暫存器t3）訊號加到模擬波形上，並通過GTKWave檢視指令和暫存器訊號隨程式執行的變化情況。

產生模擬二進位檔案，Vrvfpgasim：

目錄[RVfpgaPath]/RVfpga/verilatorSIM包含Makefile和用於產生RVfpgaSim模擬器二進位檔的指令碼（*swervolf_0.7.vc*）。該指令碼包含Verilator需要瞭解的資訊，以及SoC來源的位置資訊，在本例中，SoC來源位於[RVfpgaPath]/RVfpga/src中。接下來，我們展示如何產生RVfpgaSim的二進位檔，該檔案稍後將用於建立在RVfpgaSim上執行的*AL-Operations*程式的模擬軌跡。

1. 在終端機視窗中，通過執行以下命令來產生模擬器二進位檔：

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

檔案*Vrvfpgasim*（RVfpgaSim模擬二進位檔）應在目錄[RVfpgaPath]/RVfpga/verilatorSIM內產生。

Windows：如果使用Windows，則必須在Cygwin終端機內執行這些相同的步驟（有關詳細說明，請參閱附錄C）。請注意，C: Windows資料夾位於Cygwin中的以下位置：*/cygdrive/c*。本節中的所有其他說明與Linux的說明相同。

macOS：有關詳細說明，請參閱附錄D。

使用Vrvfpgasim通過PLATFORMIO產生模擬軌跡：

產生模擬器二進位檔（*Vrvfpgasim*）後，將在PlatformIO內使用該檔案來產生程式*AL_Operations*的模擬軌跡（*trace.vcd*）。

2. 在電腦中依序開啟VSCode和PlatformIO。
3. 在頂端列上，按一下「File」（檔案）→「Open Folder...」（開啟資料夾...）（圖81），然後導覽至目錄[RVfpgaPath]/RVfpga/examples/

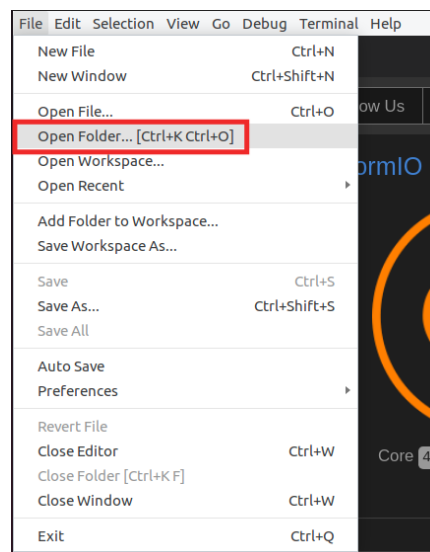


圖81. 開啟AL_Operations.S範例

4. 選擇目錄AL_Operations（不要開啟，只需選擇它），然後按一下「OK」（確定）。該範例將在PlatformIO中開啟。
5. 開啟platformio.ini檔。通過編輯下行，建立到第一步（Vrvfpgasim）產生的RVfpgaSim模擬二進位檔的路徑（請參閱圖82）。

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

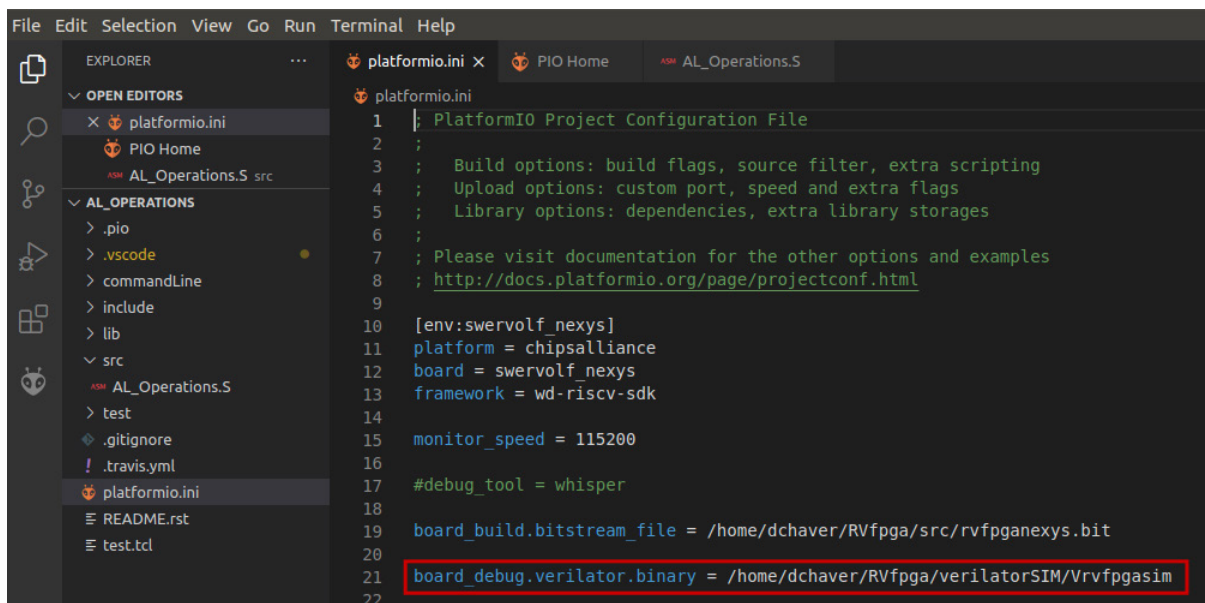


圖82. PlatformIO初始化檔：platformio.ini

Windows： 在Windows中，RVfpgaSim模擬可執行檔案稱為Vrvfpgasim.exe。因此：

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

- 按一下左側功能表功能區中的PlatformIO圖示來執行模擬，然後展開「Project Tasks」（專案任務）→ env:swervolf_nexys → 「Platform」（平台），並按一下「Generate Trace」（產生軌跡），如圖83所示。

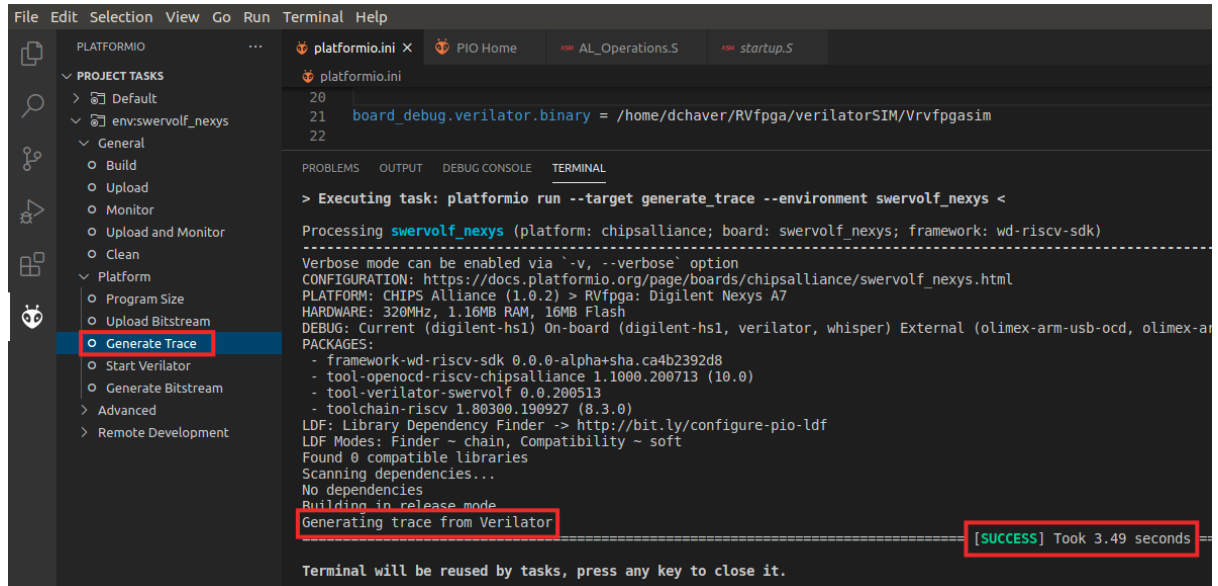



圖83. 通過Verilator產生軌跡

或者，可以通過PlatformIO終端機視窗產生軌跡。為此，請按一下按鈕（PlatformIO: New Terminal」（PlatformIO：新建終端機）按鈕，該按鈕位於PlatformIO視窗的底部）開啟一個新的終端機視窗，然後在PlatformIO終端機中輸入（或複製）以下命令：`pio run --target generate_trace`

- 執行上一步後幾秒鐘，檔案`trace.vcd`應已在
`[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys`內產生，可以使用GTKWave將其開啟。

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

WINDOWS：已下載的資料夾`gtkwave64`包括一個稱作`gtkwave.exe`的應用程式，該應用程式位於`bin`資料夾內。按兩下該應用程式啟動GTKWave。在應用程式的頂端，按一下「File」（檔案）–「Open New Tab」（開啟新索引標籤），然後開啟在資料夾
`[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys`中產生的`trace.vcd`檔。

在GTKWAVE中分析模擬軌跡：

- 現在，將新增時脈、指令和暫存器訊號。在GTKWave的左上方窗格中展開SoC的階層，將訊號新增到圖形中。將階層展開為「TOP」（上層）→ `rvfpgasim` → `swervolf` → `swerv_eh1` → `swerv`，然後按一下模組`ifu`（如圖84中突出顯示部分所示），選擇訊號`clk`（用於核心的時脈），並將其拖曳到右側的白色「Signals」（訊號）窗格或黑色「Waves」（波形）窗格中。

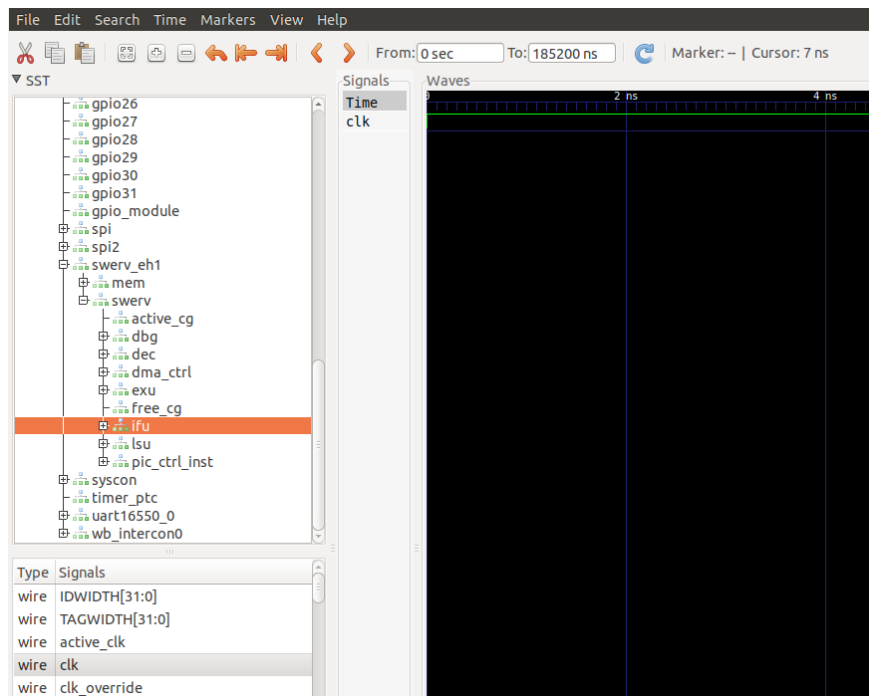


圖84. 將訊號`clk`新增至圖形

9. 放大幾次，以便可以查看時脈訊號的變化（圖85）。

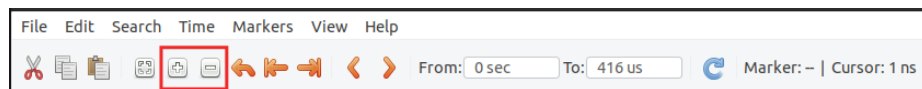


圖85. 放大

10. 現在新增訊號，這些訊號顯示在雙路超標量RISC-V核心中每一路執行的指令。在同一模組（`ifu`）中尋找訊號`ifu_i0_instr[31:0]`和`ifu_i1_instr[31:0]`（圖86），然後將這些訊號拖曳到黑色「Waves」（波形）窗格中。字首`ifu`表示指令擷取單元，`i0`表示超標量通路0，`i1`表示超標量通路1；`instr[31:0]`表示32位指令。

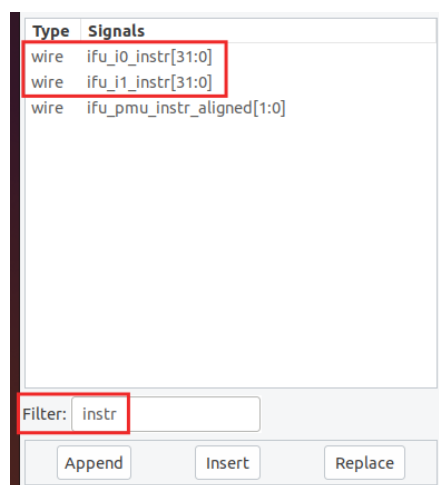


圖86. 將訊號`ifu_i0_instr[31:0]`和`ifu_i1_instr[31:0]`加到時序波形上

11. 現在新增用於儲存暫存器t3（即，暫存器編號28，x28）值的訊號。在**swerv**下將階層展開為**dec** → **arf** → **gpr_banks(0)** → **gpr(28)**，然後按一下模組**gprff**（如下圖突出顯示部分所示），選擇訊號**dout[31:0]**（該訊號顯示AL_Operations.S範例中使用的暫存器x28的內容）並將其拖曳到黑色「Waves」（波形）窗格中（圖87）。

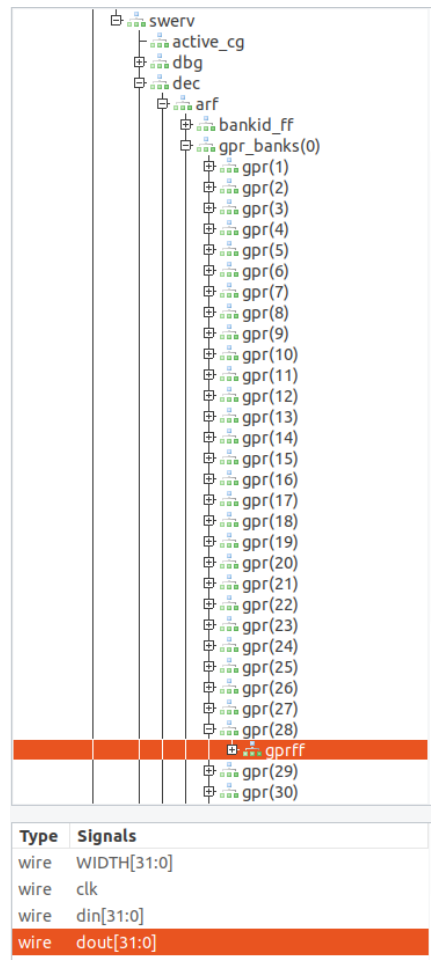


圖87. 將訊號**dout[31:0]**加到圖形上

12. 在GTKWave中顯示訊號的另一種方法是使用**.tcl**檔案。檔案**test.tcl**位於 **[RVfpgaPath]/RVfpga/examples/AL_Operations**中。開啟該檔案並對其進行分析。在每一行中，您將看到我們想要在圖中顯示的每個訊號的路徑和名稱。

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

要在GTKWave上使用**.tcl**檔案，只需按一下「**File – Read Tcl Script File**」（檔案 – 讀取Tcl指令碼檔案）並選擇 **[RVfpgaPath]/RVfpga/examples/AL_Operations/test.tcl** 檔案即可。

13. 圖88顯示AL_Operations.S程式及其等效機器碼。

# 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
  
```

圖88. AL_Operations.S及其等效機器碼

現在檢視訊號隨程式執行的變化情況。隨著程式執行，預計指令和t3（暫存器x28）將變為圖89所示的值：

```

                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
                ...
  
```

圖89. AL_Operations執行期間的指令流動和暫存器t3（x28）的值

14. 在10100 ns附近放大，將在此時間內分析前兩次迴圈迭代的三行算術邏輯指令的執行情況（圖90）。前兩條指令（li t3, 0x0 = 0x00000E13和addi t3, t3, 6 = 0x006E0E13）先擷取，超標量RISC-V處理器的兩路各一條（如訊號ifu_i0_instr[31:0]和ifu_i1_instr[31:0]所示）。接下來的兩條指令（addi t3, t3, -1 = 0xFFFE0E13和andi t3, t3, 3 = 0x003E7E13）在下一個週期擷取。最後兩條指令（beq zero, zero, REPEAT = 0xFE000CE3和nop = 0x00000013）在接下來的下一個週期擷取。

基於SweRV核心的9階段管線處理器和相關性，指令的結果在指令擷取後的八個或更多週期才能展現出來。第一條和第二條指令擷取完成8個週期後，由於第一條指令li t3, 0x0（0x00000E13）的原因，x28（t3）將變為0（原本就是0）。一個週期後，由於下一條指令addi t3, t3, 6（0x006E0E13）的原因，x28將更新為0x6。接下來，由於下一條指令addi t3, t3, -1（0xFFFE0E13）的原因，x28將更新為5。最後，由於下一條指令andi t3, t3, 3（0x003E7E13）的原因，x28將更新為1。接著擷取以下兩行指令：beq zero, zero, REPEAT（0xFE000CE3）和nop（0x00000013），將發生分支並重複迴圈。情況正如圖89所預計的那樣。使用類似的推理，可以分析第二次迭代，情況如圖90和圖89所示。

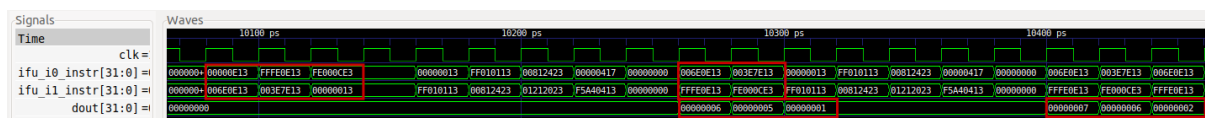


圖90. 範例中三行算術邏輯指令的執行

8. WHISPER模擬

Whisper (<https://github.com/chipsalliance/SweRV-ISS>) 是Western Digital開發的一款RISC-V指令集模擬器 (ISS)，用於驗證SweRV微處理器。借助這款模擬器，使用者無需底層的RISC-V硬體即可執行RISC-V程式碼。利用Whisper，可以使用PlatformIO測試、執行和偵錯C或組合語言譯程式，而無需Nexys A7 FPGA電路板。

Windows：本節中描述的所有說明都應適用於Windows（我們要感謝Jean-François Monestier，他率先將Whisper移植到Windows：<https://jean-francois.monestier.me/porting-western-digital-swerv-iss-to-windows/>）。請注意，快顯視窗可能會要求使用者允許Whisper通過Windows防火牆。

macOS：本節中描述的所有說明也適用於macOS。

使用命令列和使用Eclipse或PlatformIO之類的IDE（整合開發環境）均可執行Whisper。在本節中，我們演示一個範例以展示如何在PlatformIO中使用Whisper模擬程式。之後，可以使用此處所述的步驟來模擬其他程式。

我們首先使用Whisper ISS來模擬AL_Operations，它是在第5節中執行和偵錯的第一個簡單組合語言程式（請參閱圖44）。按照下面的步驟在Whisper上執行和偵錯此程式碼：

1. 開啟VSCode（和PlatformIO）。在頂端功能表列上，按一下「File」（檔案）→「Open Folder」（開啟資料夾），導覽至目錄[RVfpgaPath]/RVfpga/examples/，選擇（但不開啟）目錄AL_Operations，然後按一下「OK」（確定）。
2. 按一下「File」（檔案）→「Open File」（開啟檔案），按兩下[RVfpgaPath]/RVfpga/examples/AL_Operations/platformio.ini，然後通過取消第17行的註解將whisper設定為偵錯工具（請參閱圖91）。儲存檔案（按Ctrl-S）。

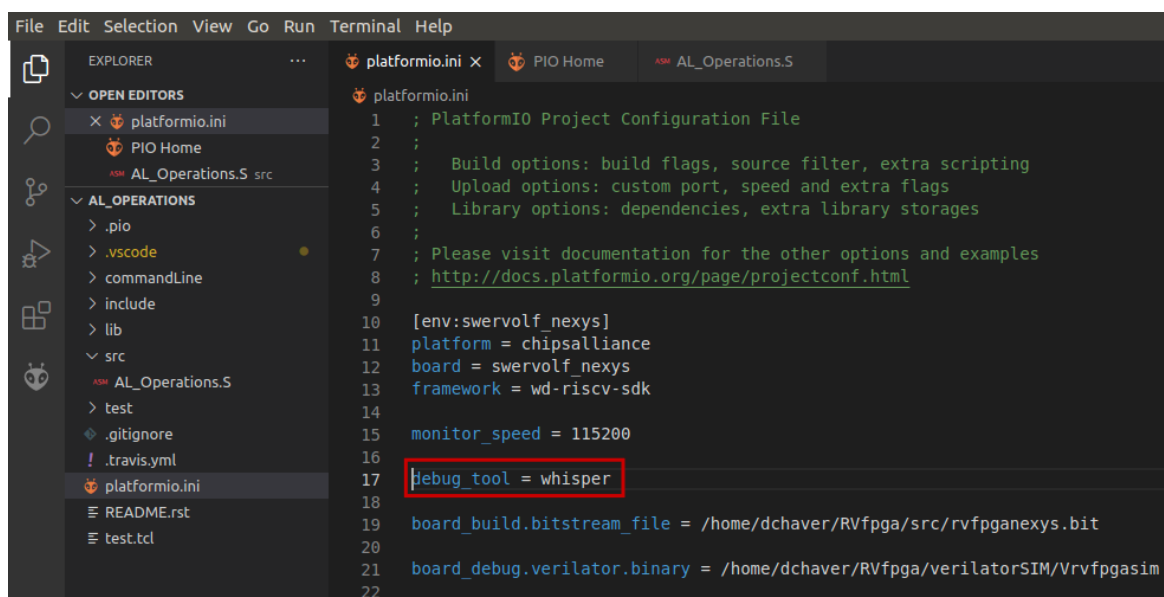

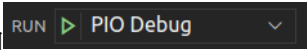


圖91. 取消第17行的註解

3. 如常依序按一下  和  來啟動偵錯工具
4. 現在，可以像在第6.B節中一樣偵錯該程式，但該程式此次是在Whisper而非Nexys A7 FPGA電路板上模擬執行。
5. 如果程式在Whisper中使用`printfNexys`函數，例如HelloWorld_C-Lang範例（第6.F節），則不應開啟PlatformIO序列監視器，因為訊息將顯示在偵錯控制台中（請參閱圖92）。

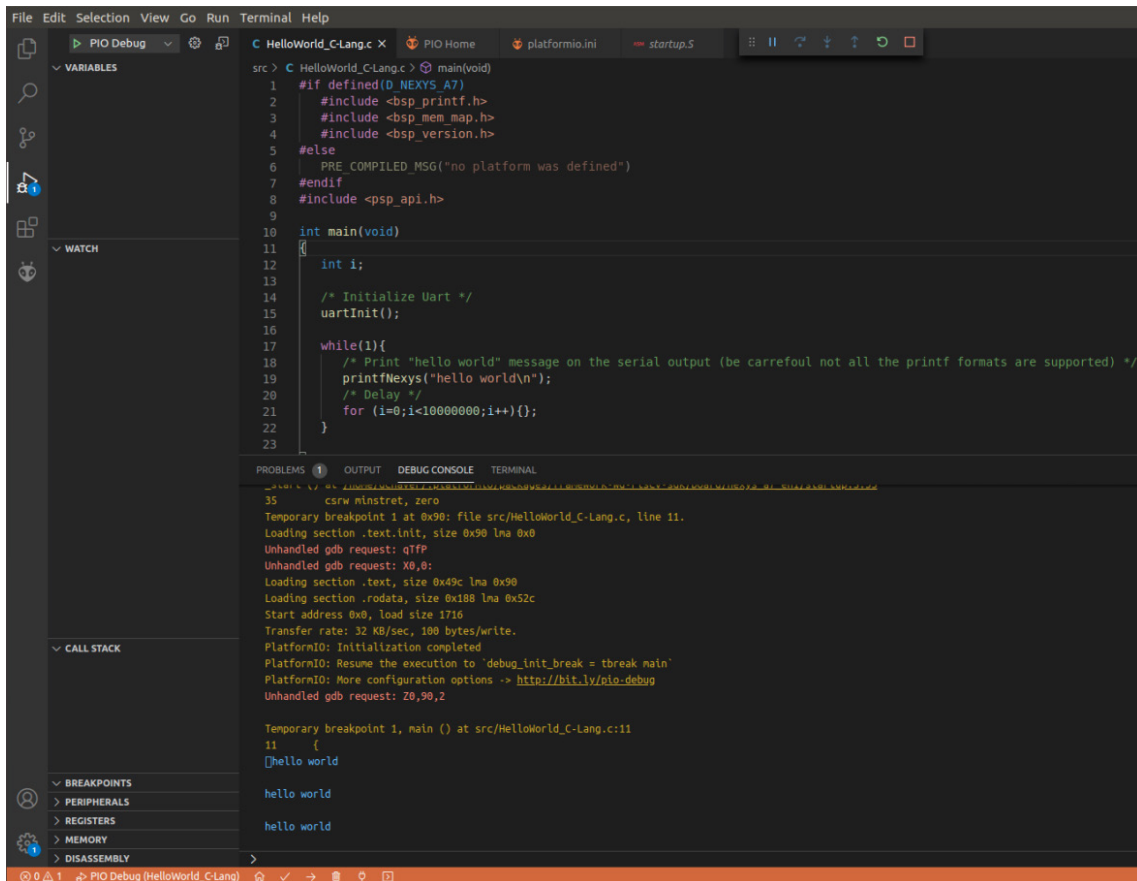


圖92. 在Whisper中執行HelloWorld_C-Lang範例

9. 附錄

以下附錄顯示了如何在Linux中使用開放原始碼RISC-V工具鏈和OpenOCD（而不是PlatformIO）、如何在Windows中安裝使用PlatformIO下載位元串流的驅動程式、如何在Windows和Mac OS電腦上安裝Verilator和GTKWave，以及如何使用Vivado對RVfpgaNexys進行程式設計。表10列出了本「RVfpga入門指南」中所有可用的附錄。

表10. 附錄清單

附錄	說明	作業系統
A	在Ubuntu 18.04中將開放原始碼RISC-V工具鏈和OpenOCD用於RVfpga	Linux
B	在Windows中安裝使用PlatformIO所需的驅動程式	Windows
C	在Windows中安裝Verilator和GTKWave	Windows
D	在macOS中安裝Verilator和GTKWave	macOS
E	使用Vivado將RVfpgaNexys下載到FPGA上	Windows和Linux
F	在工業物聯網應用中使用RVfpga	所有

附錄A適用於想要使用開放原始碼gcc/gdb工具和OpenOCD，以原生方式編譯和執行/偵錯程式的使用者。但是，**建議RVfpga使用者使用PlatformIO來代替**，如本入門指南所述。

Windows使用者必須遵循附錄B和附錄C中的說明。附錄B說明了如何下載驅動程式，方便Windows系統使用PlatformIO來下載程式，以及如何將RVfpgaNexys下載到Nexys A7 FPGA電路板上。附錄C展示了如何安裝Verilator和GTKWave以便Windows使用者可以模擬RVfpgaSim。

macOS使用者必須遵循附錄D中的說明以使用Verilator和GTKWave來模擬RVfpgaSim。

建議使用PlatformIO將RVfpgaNexys系統下載（由bit檔rvfpganexys.bit定義）到Nexys A7 FPGA電路板上。該bit檔案（rvfpganexys.bit）可以由Vivado或PlatformIO產生。此外，還可以使用Vivado將RVfpgaNexys系統下載到Nexys A7 FPGA電路板上，如附錄E所述。但是，**不建議使用Vivado將RVfpgaNexys下載到電路板上** – 特別是**Windows**使用者，因為此操作需要不斷地交換驅動程式。

附錄A：在Ubuntu 18.04中使用原生RISC-V工具鏈和OpenOCD

盡管我們建議使用PlatformIO，但在本節中，我們將說明如何安裝、執行和使用開放原始碼RISC-V工具鏈；如何使用OpenOCD將RVfpgaNexys下載到Nexys A7 FPGA電路板上；以及如何使用gdb在RVfpgaNexys上執行和偵錯程式。該工具鏈由gnu編譯器、偵錯工具、組譯器等組成。我們展示了如何在Ubuntu 18.04作業系統（OS）上安裝RISC-V工具鏈和OpenOCD，但是此程序也應適用於其他Linux發行版本。這些說明假定使用全新的Ubuntu系統。

如果使用的是PlatformIO，則無需執行以下步驟，如本指南前面所述。建議使用PlatformIO、Vivado和Verilator或Whisper來執行、偵錯和模擬RISC-V程式，但是，對於有興趣使用開放原始碼RISC-V工具鏈和OpenOCD來替代PlatformIO和Vivado硬體管理員的任何使用者，提供了以下說明。

I. 在Linux Ubuntu OS上進行安裝

在本節中，我們介紹了如何在Ubuntu 18.04電腦中以原生方式安裝RISC-V工具鏈、OpenOCD和Whisper。這些工具僅替代PlatformIO；如本GSG第5節所述，仍然需要安裝Vivado和Verilator。

RISC-V工具鏈

在此我們展示如何在您的電腦上安裝完整的RISC-V工具鏈，即gnu編譯器、偵錯工具等。RISC-V International在以下位置提供了安裝說明：<https://github.com/riscv/riscv-gnu-toolchain> 這些說明摘要如下。

附註：安裝RISC-V工具鏈和OpenOCD可能需要幾個小時，大部分時間花在了等待工具鏈下載、編譯和安裝上。

在終端機輸入以下內容（該程序可能需要一個多小時，但大部分時間都花在了等待程式下載和安裝上）：

- `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`（如有可能，請使用`sudo make -j$(nproc)`，此指令可以顯著縮短編譯時間）
- `export PATH=$PATH:/opt/riscv/bin`（在系統中變更路徑）

OpenOCD

OpenOCD是一個開放的晶片上偵錯工具，允許使用者對嵌入式目標裝置進程式設計和偵錯。按照下面的步驟將RISC-V OpenOCD安裝到電腦上：

- `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

按照下面的步驟將Whisper安裝到電腦上（有關說明，請參見：

<https://github.com/chipsalliance/SweRV-ISS>，下面也對此進行了摘要）：

- 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 （根據需要替換 [RVfpgaPath]）。

II. 透過OpenOCD在RVfpgaNexys上執行程式

步驟A. 將RVfpgaNexys下載（圖25）到Nexys A7

- 移至包含RVfpgaNexys bit檔案的專案目錄：
`cd [RVfpgaPath]/RVfpga/src`
- 使用OpenOCD將RVfpgaNexys下載到電路板中：
`riscv-openocd -c "set BITFILE rvfpganexys.bit" -f
OtherSources/ConfigFiles/swervolf_nexys_program.cfg`

步驟B. 執行LedsSwitches，該程式讀取開關並將其狀態呈現在LED上

- 移至LedsSwitches/commandLine目錄：
`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`
- 在該目錄中，可找到用於編譯原始程式碼的Makefile、連結指令碼、python指令碼以及LedsSwitches.S程式。
- 編譯elf檔案：
`make clean
make LedsSwitches.elf`
- 將OpenOCD連接到SoC：
`riscv-openocd -
f ../../../../src/OtherSources/ConfigFiles/swervolf_nexys_debug.cf
g`

OpenOCD開始執行後，將出現幾則訊息，其中一則顯示：

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

- 開啟一個新終端機，移至程式目錄（`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`），並執行以下命令：
`telnet localhost 4444`

接著在telnet連線中輸入：

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

這三行命令的作用如下：(1)將LedsSwitches.elf程式載入到RVfpgaNexys上，(2)將程式計數器（Program Counter，PC）設定為0（程式第一行指令的位址）(3)恢復執行。

該程式將開始在RVfpgaNexys上執行，RVfpgaNexys是已在步驟2中下載到Nexys A7 FPGA電路板上的RISC-V SweRVofX SoC。該程式使LED顯示開關的狀態。撥動開關時，LED會立即改變以反映開關的值。

步驟C. 對執行簡單算術邏輯運算的AL_Operations_CommandLine程式進行偵錯

現在，我們展示如何使用OpenOCD和gdb偵錯另一個程式（AL_Operations_CommandLine）。

7. 維持OpenOCD連線開啟（請參閱步驟5）。
8. 在執行telnet的其他終端機中（從步驟6開始），通過輸入以下內容結束telnet連線：
`exit`

9. 移至包含AL_Operations/commandLine的專案目錄：
`cd ../../AL_Operations/commandLine`

在該目錄中，可找到用於編譯原始程式碼的Makefile、連結指令碼、python指令碼以及AL_Operations.S程式。

10. 編譯.elf檔案：
`make clean`
`make AL_Operations.elf`
11. 然後，在此終端機中，通過輸入以下內容來啟動gdb：
`riscv32-unknown-elf-gdb AL_Operations.elf`
12. 在gdb控制台中，輸入：
`target remote localhost:3333`
`load`

這將連接到OpenOCD，並將AL_Operations.elf程式載入到記憶體中。

13. 現在即可偵錯程式。輸入以下序列並分析輸出：

```
i. disas 0,20
```


該序列顯示從位址0到20（不包括位址20）的組合語言程式碼。請參閱圖93。

```
(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.
```

圖93. 檢視組合語言程式

ii. `i r t3`

該序列顯示暫存器t3的內容。或者，可以輸入更長的版本：`info reg t3`。請參閱圖94。

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

圖94. 列印暫存器t3包含的值

iii. `i r pc`

該序列顯示程式計數器（pc）的內容。請參閱圖95。

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

圖95. 列印指向第一行指令的暫存器PC中包含的值

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

`stepi`使程式執行一條指令。`i r t3`隨後顯示暫存器t3的內容。請參閱圖96。

```
(gdb) stepi
0x00000004 in REPEAT ()
(gdb) i r t3
t3                0x0        0
(gdb) stepi
0x00000008 in REPEAT ()
(gdb) i r t3
t3                0x6        6
(gdb) stepi
0x0000000c in REPEAT ()
(gdb) i r t3
t3                0x5        5
(gdb) stepi
0x00000010 in REPEAT ()
(gdb) i r t3
t3                0x1        1
```

圖96. 逐一執行幾行指令並檢視t3暫存器

使用gdb完成對程式和暫存器的偵錯和探索後，通過在gdb終端機中輸入**quit**來退出gdb，在OpenOCD終端機中輸入**^C**來退出OpenOCD。

III. 使用Verilator在RVfpgaSim上模擬程式

1. 在Ubuntu中開啟終端機
2. 在終端機視窗中，通過執行以下命令來產生模擬器二進位檔：

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

檔案*Vrvfpgasim*（RVfpgaSim模擬二進位檔）應在目錄*[RVfpgaPath]/RVfpga/verilatorSIM*內產生。

3. 移至包含程式範例的資料夾：


```
cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine
```
4. 建立用於模擬的十六進位程式。


```
make clean
make AL_Operations.elf
make AL_Operations.bin
make AL_Operations.vh
```
5. 執行模擬器。


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

幾秒鐘後，通過在終端機中輸入**^C**停止模擬。*trace.vcd*檔應已產生，可通過GTKWave將其開啟。

```
gtkwave trace.vcd
```

6. 按照第7節步驟8至12所提供的說明，將訊號新增到圖形並進行分析。

IV. 在Whisper上模擬程式

1. 在Ubuntu中開啟終端機

2. 移至包含程式範例的資料夾：

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

3. 建立反組譯程式。

```
make AL_Operations.dis
```

4. 在編輯器中開啟**AL_Operations.dis**。將看到如下內容：

```
<_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. 在互動模式下執行模擬器。

```
whisper --interactive AL_Operations.elf
```

6. 偵錯程式。

```
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
```

使用**whisper**完成對程式和暫存器的偵錯和探索後，通過在終端機中輸入**quit**來退出。

附錄B：在Windows中安裝使用PlatformIO所需的驅動程式

要下載Zadig可執行檔，請瀏覽至以下網站（請參閱圖97）：

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

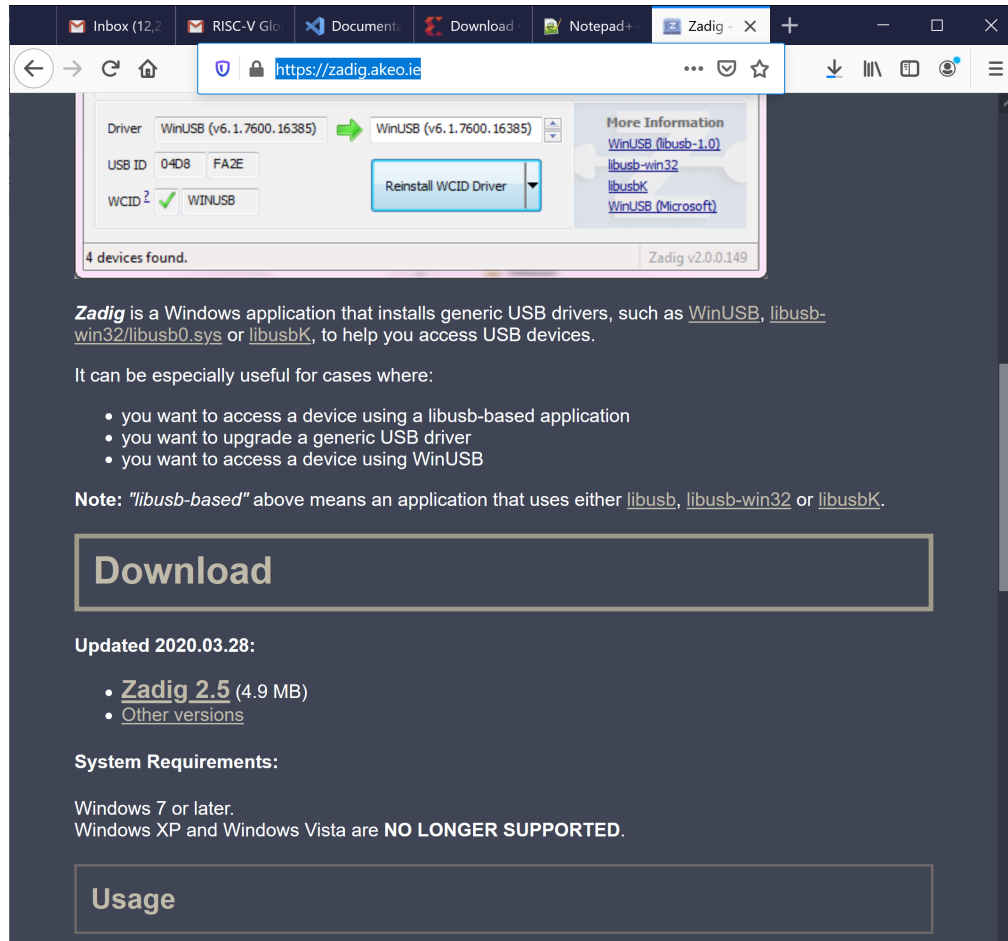


圖97. 安裝PlatformIO使用的Nexys A7電路板驅動程式

按一下Zadig 2.5並儲存可執行檔。然後執行該檔案（zadig-2.5.exe），該檔案位於其下載位置。此外，也可以在「Start」（開始）功能表中輸入zadig來找到該檔案。系統可能會詢問您是否要允許Zadig對電腦進行變更，以及是否允許其檢查更新。兩次都按一下「Yes」（是）。

將Nexys A7電路板連接到電腦並開機。在Zadig中，按一下「Options」（選項）→「List All Devices」（列出所有裝置）（請參閱圖98）。

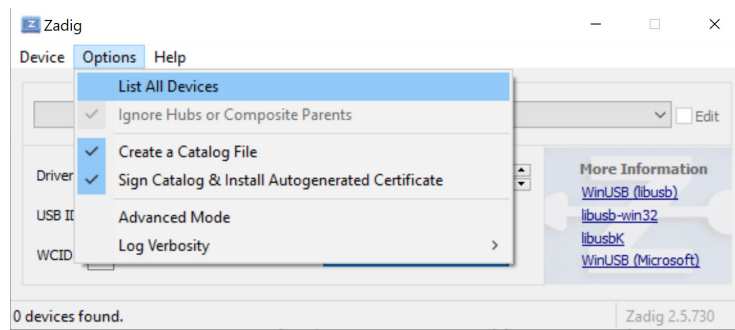


圖98. Zadig中的「List All Devices」（列出所有裝置）

如果按一下下拉式功能表，將看到列出的「Digilent USB Device (Interface 0)」(Digilent USB裝置備(介面0))和「Digilent USB Device (Interface 1)」(Digilent USB裝置(介面1))。將僅為「Digilent USB Device (Interface 0)」(Digilent USB裝置(介面0))安裝新的驅動程式(請參閱圖99)。

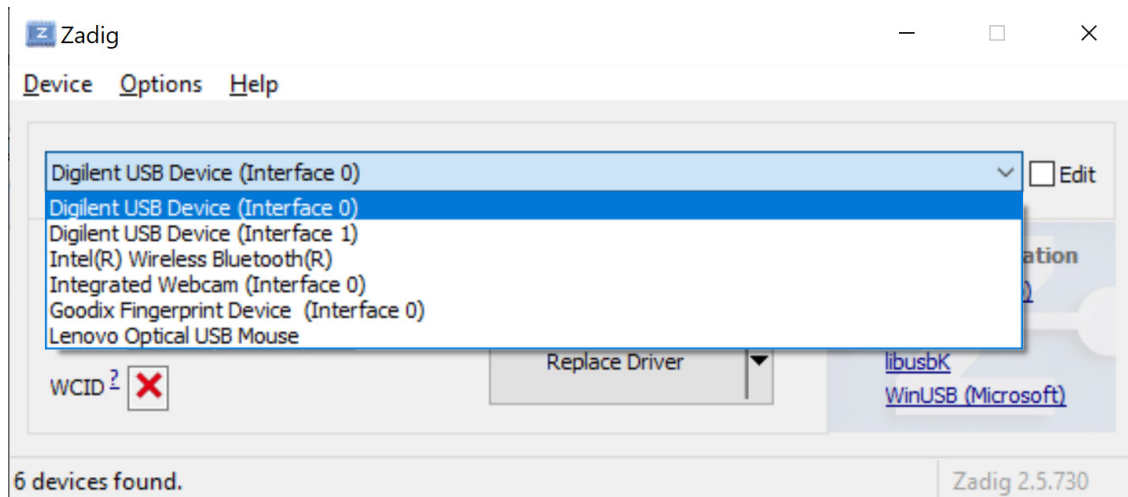


圖99. 為Digilent USB裝置(介面0)安裝WinUSB驅動程式

現在，將使用WinUSB驅動程式替換FTDI驅動程式，如圖100所示。按一下「Digilent USB Device (Interface 0)」(Digilent USB裝置(介面0))對應的「Replace Driver」(取代驅動程式)(或「Install Driver」(安裝驅動程式))。將安裝Nexys A7電路板的驅動程式，或者，如果先前已安裝Vivado，則會用PlatformIO使用的WinUSB驅動程式替換Vivado使用的FTDI驅動程式。

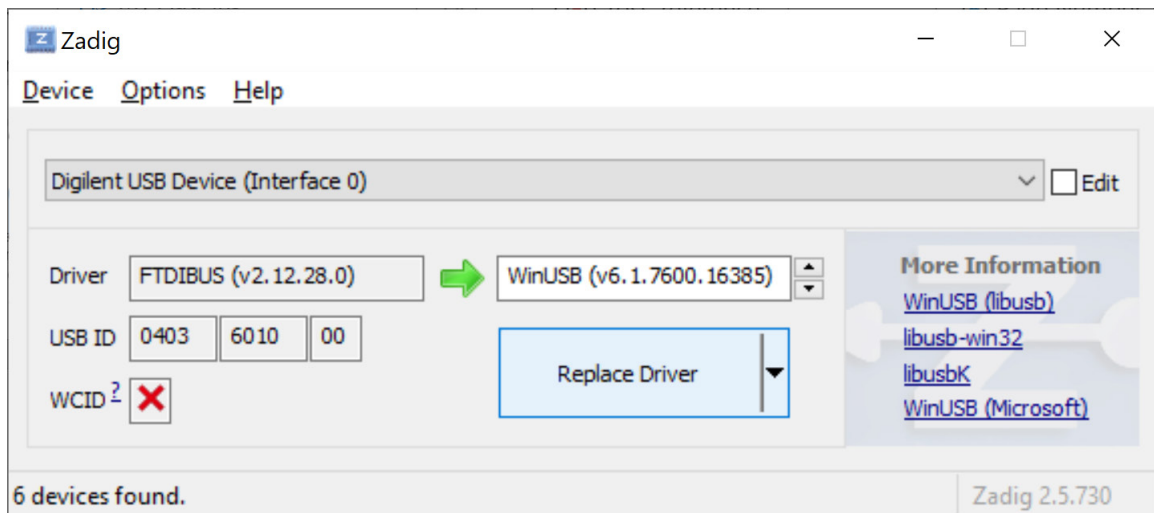


圖100. 更換Nexys A7電路板的驅動程式

一段時間（通常為幾分鐘）後，Zadig將指示驅動程式已正確安裝。按一下「Close」（關閉），然後關閉Zadig視窗。

下次使用PlatformIO時，無需重新安裝該驅動程式。但請注意，在Windows中，該驅動程式與Vivado不相容。因此，不能再使用Vivado將bit檔下載到FPGA電路板上。如果要使用Vivado下載bit檔（不建議），則需要將驅動程式還原為隨Vivado一起安裝的原始驅動程式，如附錄E中所述。

附錄C：在Windows中安裝Verilator和GTKWave

在此附錄中，我們說明如何在Windows 10中安裝Verilator和GTKWave。在Windows中，必須使用Cygwin來安裝Verilator，因此我們首先說明如何安裝此程式設計/執行階段環境。

Cygwin安裝：

如網頁（<https://www.cygwin.com>）所述，Cygwin由GNU和開放原始碼工具組成，這些工具可在Windows上提供與Linux發行版類似的功能。按照下面的步驟在Windows 10上安裝Cygwin。

1. 導覽至安裝網頁（<https://cygwin.com/install.html>），然後下載名為`setup-x86_64.exe`的安裝檔（圖101）。

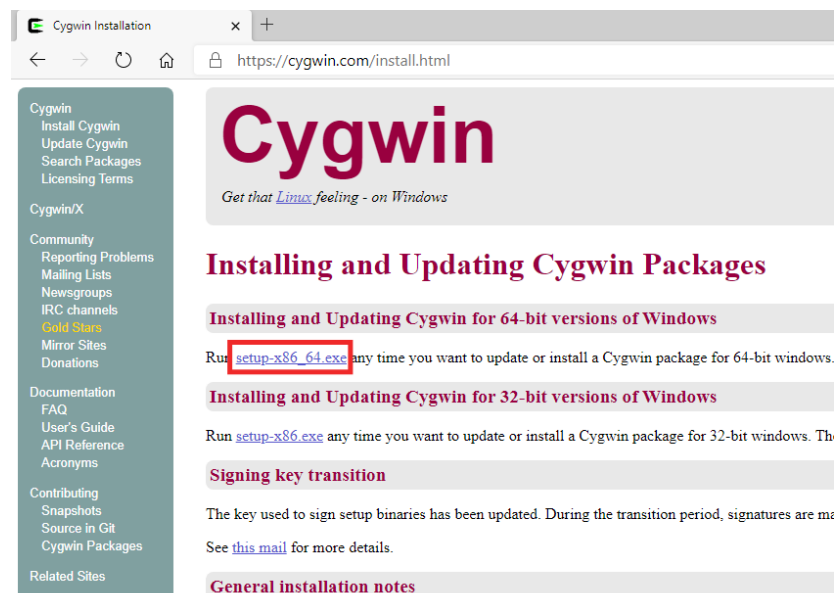


圖101. Cygwin安裝網頁

2. 按兩下安裝檔案以在電腦上執行此檔案（圖102）。按「Next」（下一步）幾次，保留預設選項。安裝程式會要求您「Choose a Download Site」（選擇下載網站）（圖103），可以選擇其中任一選項。

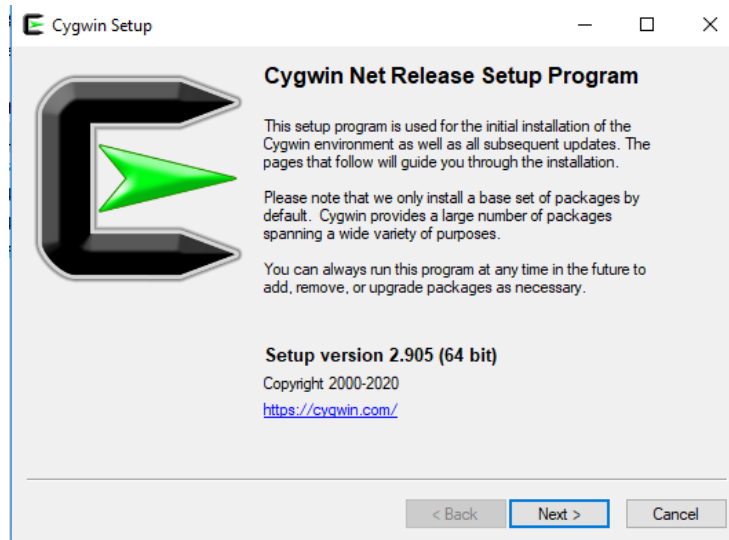


圖102. Cygwin安裝視窗

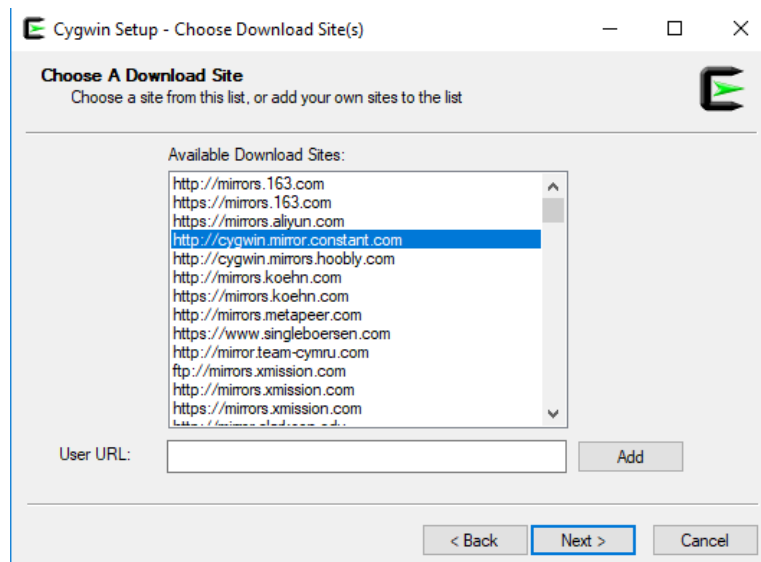


圖103. 選擇下載網站

3. 幾個步驟後，將進入「**Select Packages**」（選擇套件）視窗（圖104）。選擇「**Full**」（完整）視圖，如圖104所示。

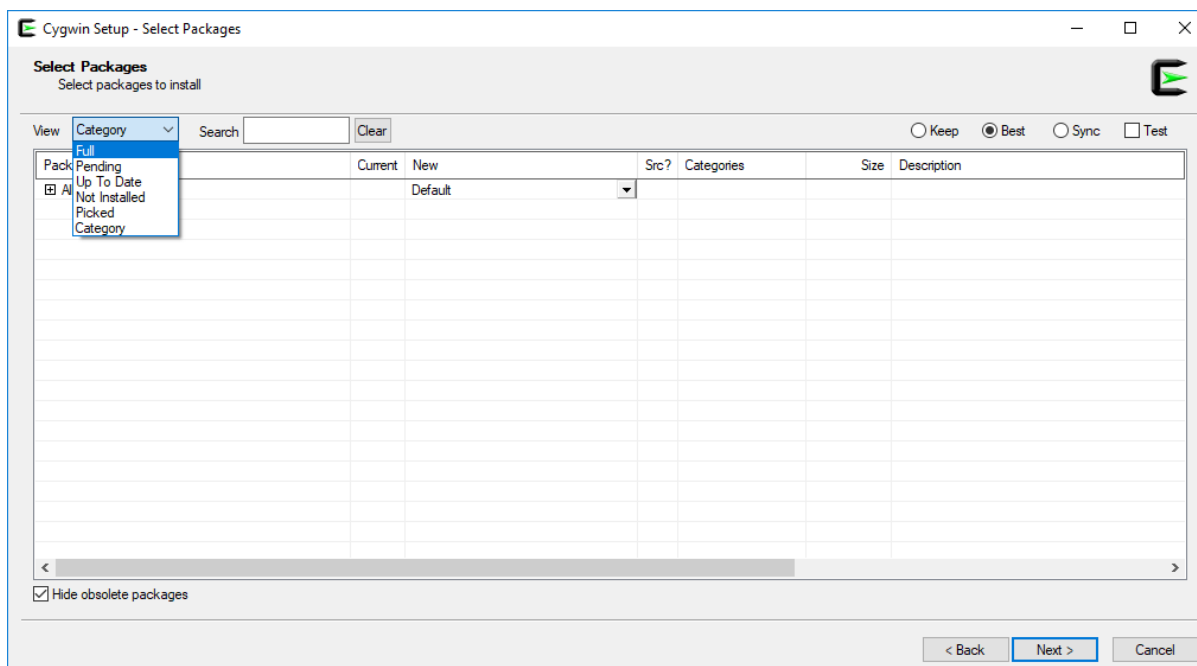


圖104. 「Select Packages」（選擇套件）視窗

- 將顯示可以安裝的套件的完整清單（圖105）。在「**Search**」（搜尋）方塊中，選擇要安裝的特定套件。

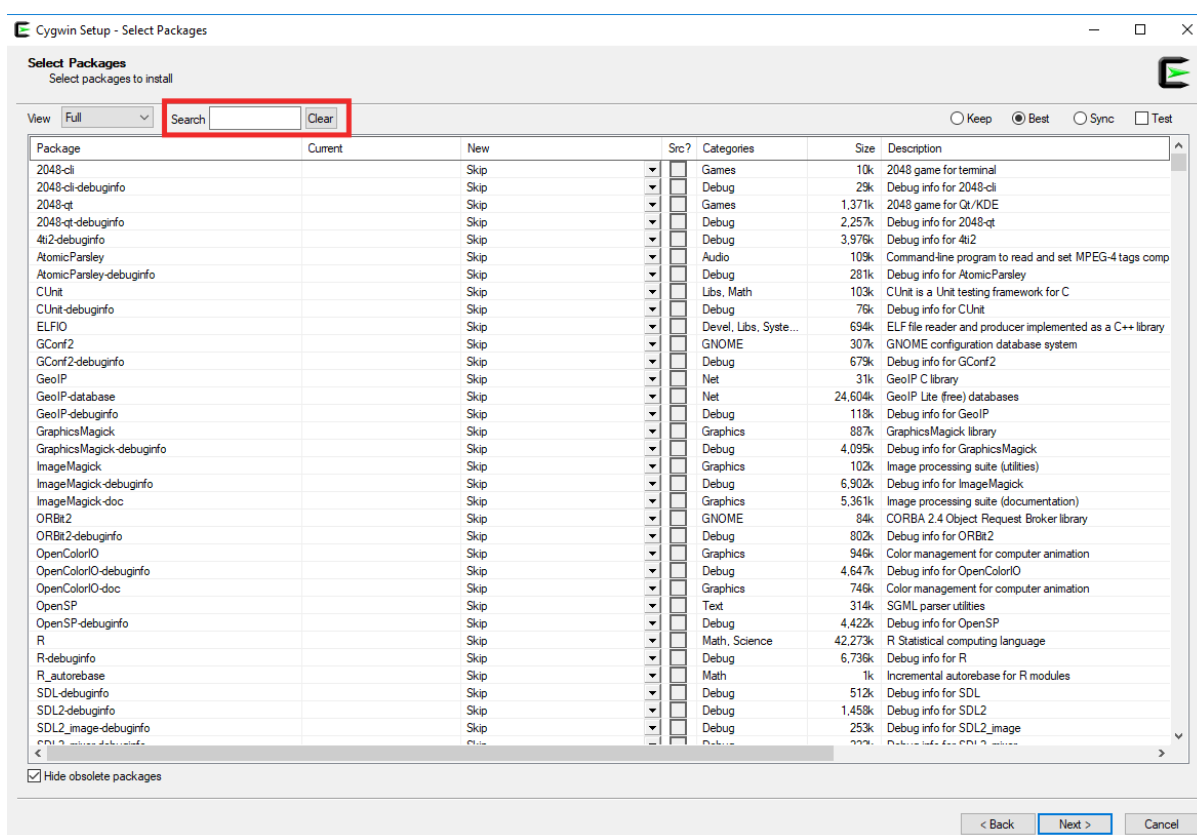


圖105. 「Select Packages」（選擇套件）視窗 – 「Full」（完整）視圖

為了能夠編譯Verilator並產生新的模擬器二進位檔，需要安裝以下套件：

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

在Cygwin安裝中至少包括這些套件。按照以下步驟逐一選擇這些套件（我們只顯示清單中第一個套件git的詳細步驟；其他套件的程序與此相同）：

- 在「**Search**」（搜尋）方塊中尋找git套件（圖106）。

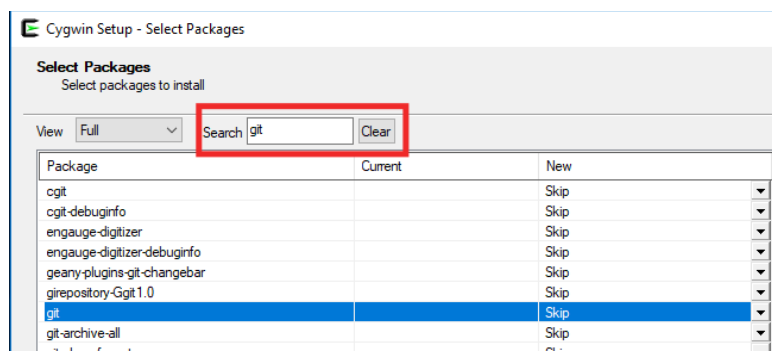


圖106. 尋找git套件

- 在下拉式功能表中選擇最新版本並勾選對應的方塊（圖107）。



圖107. 選擇最新版本並勾選對應的方塊

- 對上述清單的其餘套件執行相同的操作。在大多數情況下，可以使用最新版本，但對於套件gcc-core和gcc-g++，應當使用版本10.2.0，因為最新版本（在撰寫本文件時為11.2.0）與Verilator存在一些衝突。
5. 選擇九個套件後，在隨後的視窗中按「**Next**」（下一步）以在Cygwin安裝（對於安裝程序，請參閱圖108，該程序可能需要幾分鐘）中包括這些套件，然後按一下「**Finish**」（完成）完成安裝（圖109）。

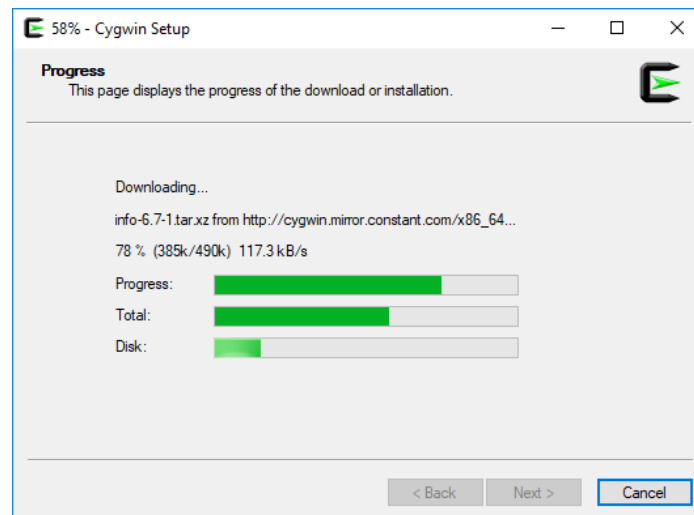


圖108. Cygwin安裝

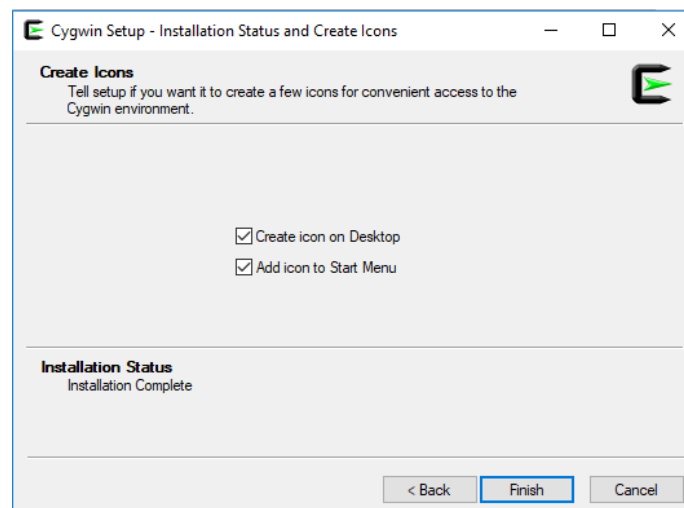


圖109. 完成安裝

6. 如果需要將某個套件新增到Cygwin安裝中，請對該套件重複步驟2-5。

Verilator安裝：

按照下面的步驟在Windows 10上安裝Verilator。

1. 開啟Cygwin終端機（圖110），可在Windows桌面或「Start」（開始）功能表中找到該終端機。



圖110. Cygwin終端機

2. 按照以下步驟編譯並安裝Verilator。這可能需要一些時間（甚至幾個小時），具體取決於電腦的速度：

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

GTKWave安裝：

GTKWave可以作為預編譯套件從<https://sourceforge.net/projects/gtkwave/files/>下載。尋找最新的Windows套件（在編寫本文件時，它稱為**gtkwave-3.3.100-bin-win64**），然後下載該套件並將其解壓縮。可在**bin**資料夾內找到一個名為**gtkwave**的可執行檔，可以在Windows電腦中執行和使用該檔案。

附錄D：在macOS中安裝Verilator和GTKWave

在本附錄中，我們說明如何在macOS中安裝Verilator和GTKWave。這些說明使用macOS Catalina 10.15.6進行了測試，但預計可以在其他版本的作業系統中使用。Homebrew（<https://brew.sh/>）套件管理員用於安裝。對於macOS中另一個廣泛使用的套件管理員MacPorts，可以找到類似的步驟（<https://www.macports.org/>）。

gcc安裝：

為了使用Verilator編譯新的模擬器，需要在系統中安裝編譯器工具鏈。有多種方法可用於安裝有效的編譯器工具鏈。下面介紹其中兩種方法：

1. 安裝XCode命令列工具。請注意，這將安裝LLVM，但在安裝後，**gcc**命令將始終可用。為此，請在「Terminal」（終端機）視窗中輸入以下命令：

- `xcode-select -install`

2. 使用Homebrew安裝**gcc**。輸入如下命令：

- o `brew install gcc@9`

Verilator安裝：

使用Homebrew安裝Verilator就像在開啟的終端機中輸入以下命令一樣簡單：

- `brew install verilator`

gtkwave安裝：

我們將再次使用Homebrew安裝**gtkwave**。但這次是使用GUI macOS應用程式，因此我們需要使用**cask**。在開啟的終端機中輸入以下命令：

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

安裝完畢後，「Application」（應用程式）資料夾中應出現**gtkwave.app**圖示。為了通過命令列使用該應用程式，可能需要安裝Perl的開關模組：

- `cpan install Switch`

附錄E：使用Vivado將RVfpgaNexys下載到FPGA上

按照以下步驟使用RVfpgaNexys對FPGA進行程式設計（透過Vivado）：

WINDOWS：在Windows中執行下列步驟前，需要按照本附錄（附錄E）末尾所述，將驅動程式還原為Vivado使用的驅動程式。

- 將Nexys A7電路板連接到電腦。
- 使用左上方的開關開啟Nexys A7電路板。
- 開啟Vivado 2019.2。
- 在Vivado中開啟可用的硬體管理員，如圖111中的突出顯示部分所示。

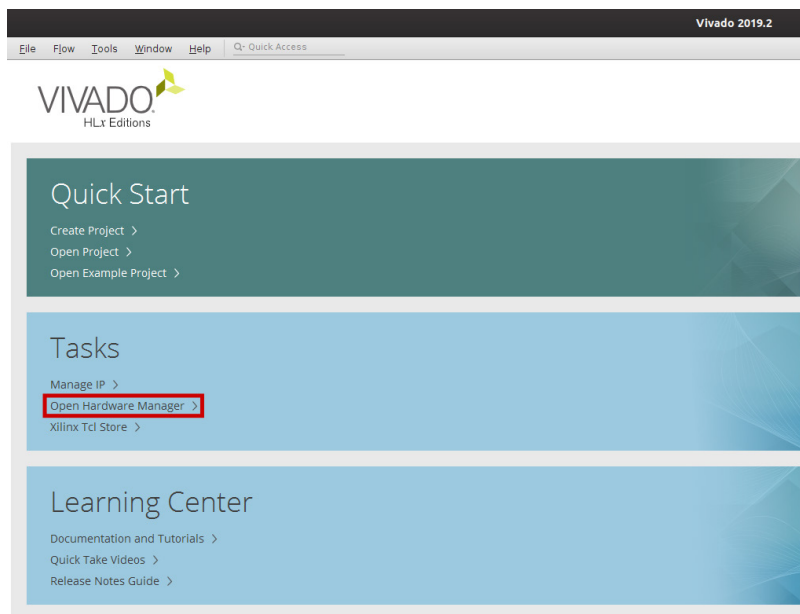


圖111. 開啟硬體管理員

- 「Hardware Manager」（硬體管理員）將開啟，並通知使用者沒有開啟任何硬體目標。通過按一下「Open target」（開啟目標）–「Auto connect」（自動連接）開啟目標（圖112）。

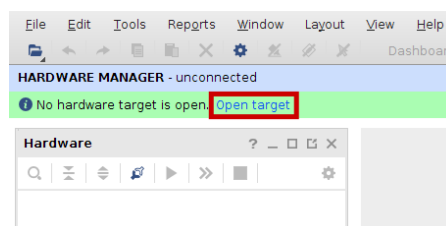


圖112. 開啟目標

- 選擇「Program device」（程式裝置），如圖113所示。現在，需要將RVfpgaNexys載入到FPGA上。在新視窗中，從[RVfpgaPath]/RVfpga/src/rvfpganexys.bit中選擇「Bitstream file」（位元串流檔）。按一下「Program」（程式）。

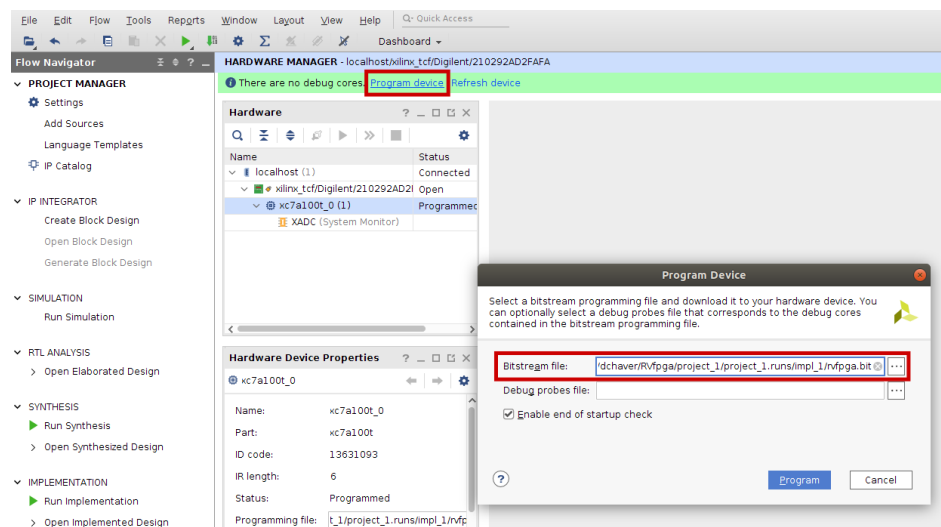


圖113. 程式裝置

- g. 幾秒鐘後，將使用RVfpgaNexys（以FPGA為目標的SweRVolfX SoC，請參閱圖25）對FPGA進行程式設計。
- h. 最後，在Vivado中按一下「Hardware Manager」（硬體管理員）窗格右上方的X按鈕關閉「Hardware Manager」（硬體管理員）（圖114），以便Vivado釋放電路板。

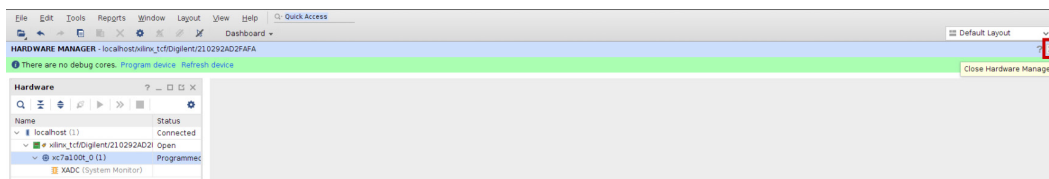


圖114. 關閉硬體管理員

如何將驅動程式還原為Windows中Vivado使用的驅動程式

遺憾的是，在Windows中，Nexys A7 FPGA電路板的驅動程式在Vivado和PlatformIO上有所不同。強烈建議您使用PlatformIO對FPGA進行程式設計，如本GSG第5.A節所述。但是，如果要使用Vivado下載bit檔，必須將在附錄B中安裝的驅動程式還原為Nexys A7 FPGA電路板的Vivado（FTDI）驅動程式。為此，通過以下方式開啟「Device Manager」（裝置管理員）：按一下「Start」（開始）功能表，在搜尋方塊中輸入「device manager」，然後按一下「Device Manager」（設備管理員）（請參閱圖115）。

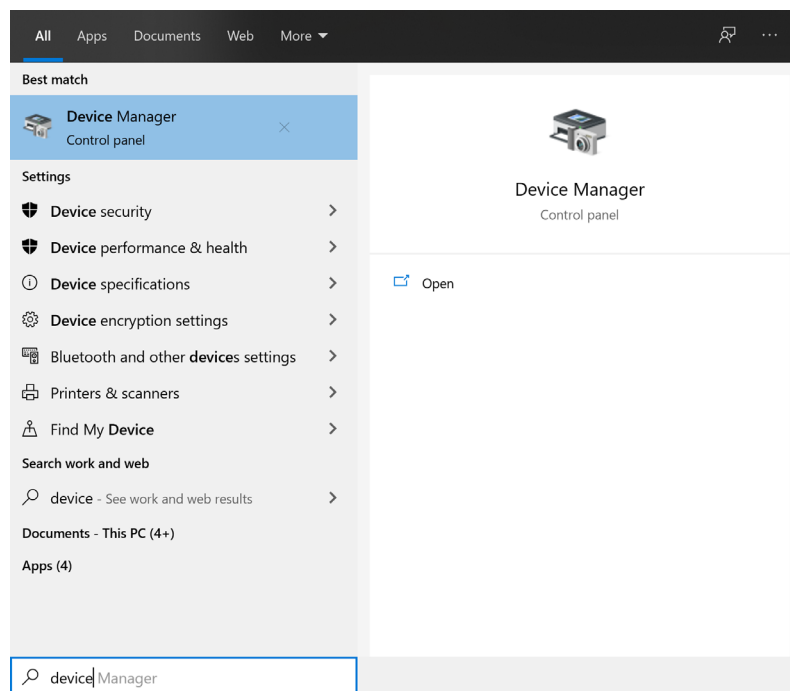


圖115. 開啟「Device Manager」（設備管理員）

接下來，展開「Universal Serial Bus Devices」（通用序列匯流排裝置），以滑鼠右鍵按一下「Digilent USB Device」（Digilent USB裝置），然後選擇「Properties」（屬性）（請參閱圖116）。

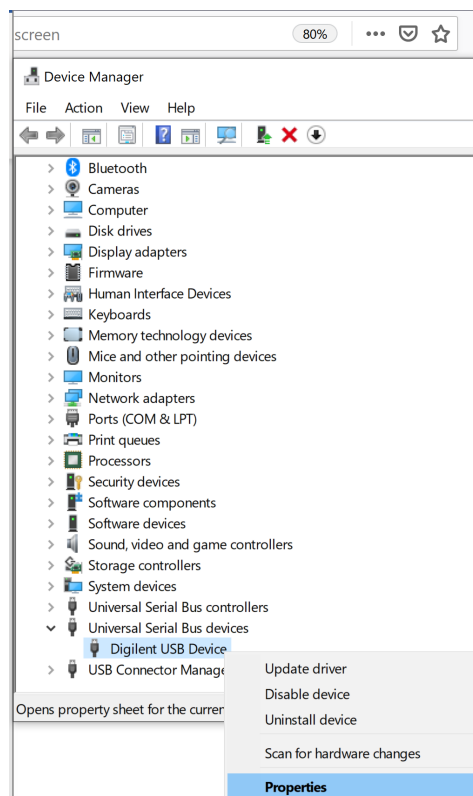


圖116. 開啟Digilent的Nexys A7 FPGA電路板的驅動程式屬性

在「Properties」（屬性）視窗中，按一下「Driver」（驅動程式）索引標籤，然後選擇「Roll Back Driver」（恢復驅動程式）（請參閱圖117）。

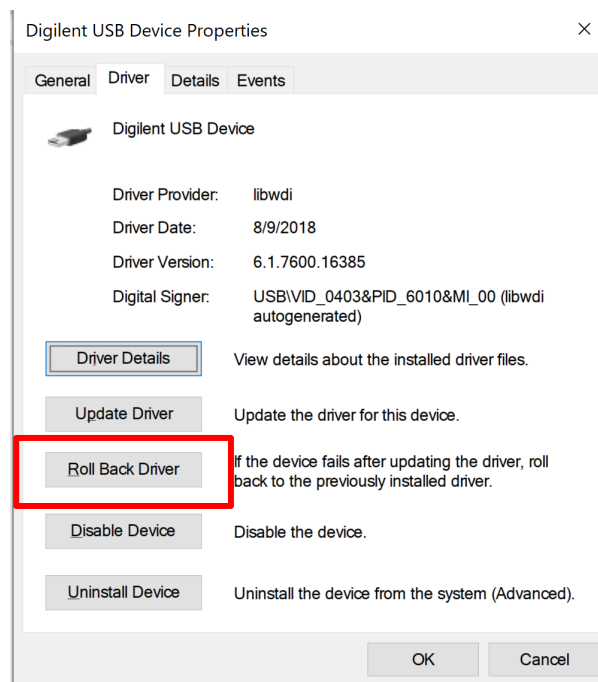


圖117. 恢復驅動程式

將彈出一個視窗，詢問恢復驅動程式的原因。選擇一個原因，然後按一下「Yes」（是）（請參閱圖118）。

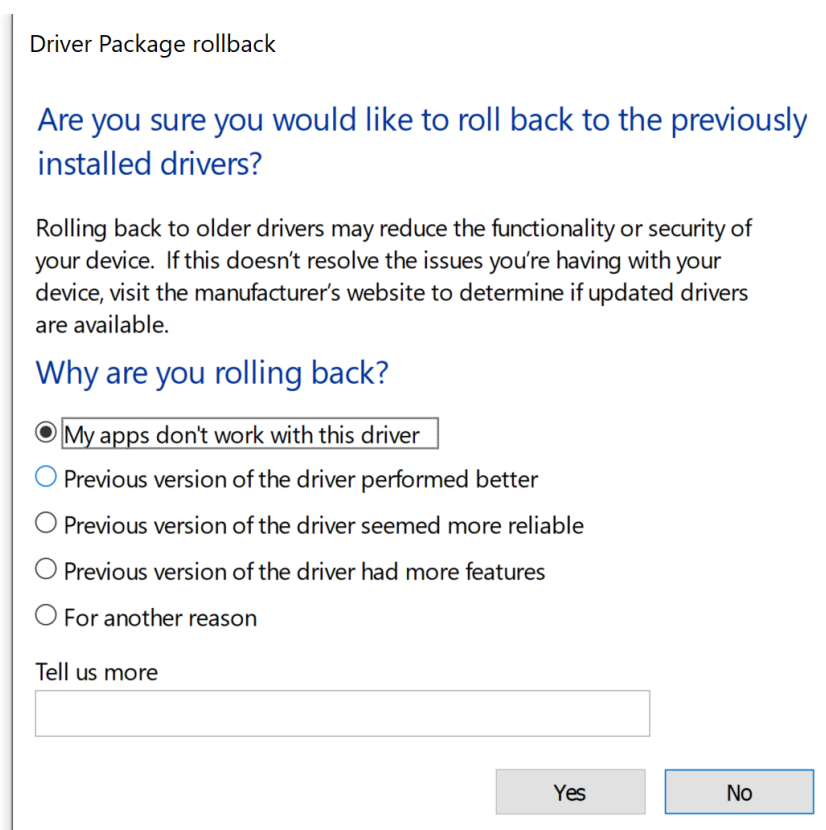


圖118. 確認恢復

驅動程式恢復為先前的驅動程式後，「Driver Provider」（驅動程式提供者）應顯示為FTDI（請參閱圖119）。

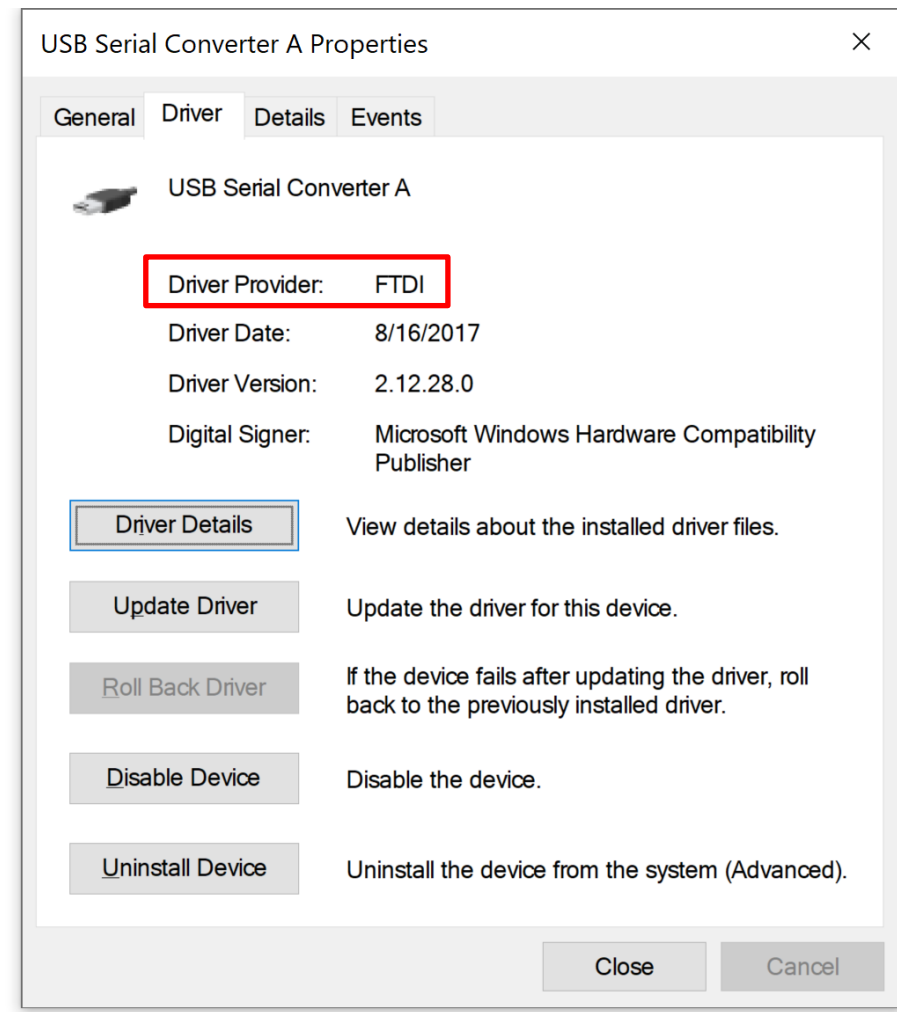


圖119. FTDI驅動程式顯示為提供的驅動程式

現在，可以使用Vivado將位元串流檔載入到FPGA電路板上。但是，仍需要使用Zadig替換Nexys A7電路板的驅動程式，以便PlatformIO可以將程式下載到RVfpgaNexys上。因此，同樣建議使用PlatformIO下載bit檔（而不使用Vivado）– 這樣便不必一直交換驅動程式。

附錄F：在工業物聯網應用中使用RVfpga

2020年7月，馬德里康普頓斯大學碩士研究生Daniel León González完成了他的碩士學位論文《適合工業物聯網的專用RISC-V晶片上系統的FPGA實作》。這篇論文展示了RVfpga在實際工業物聯網應用中的使用。我們在下面提供了專案摘要，可通過以下連結獲得完整論文：
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。

適合工業物聯網的專用RISC-V晶片上系統的FPGA實作

摘要：物聯網的節點裝置需要具備能源效率和成本效益，但在很多情況下它們並不需要很高的計算能力。但是在工業物聯網環境中，這種情況明顯不同，大量感應器的採用和事件的快速處理需要更高的處理能力。為了平衡這些要求並提供最佳解決方案，可以使用一種採用高效率處理器以及高效能的全功能作業系統的自訂開發節點。該專案利用Artix-7 FPGA解決了基於RISC-V處理器架構的原型物聯網節點的硬體實作問題，並展示了實作的自訂SoC及所需Zephyr OS驅動程式的開發程序，從而支援自訂邊界路由器周圍的星型網路所部署的概念驗證應用環境。在節點與ThingSpeak雲端平台之間可以傳送和接收端到端訊息。本文包括對現有RISC-V處理器實作的分析、必要元素的描述，以及環境配置和專案設計的詳細指南。