



IMAGINATION大学计划

RVfpga

入门指南

致谢

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

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/zubairlk/>)
- M. Hamza Liaqat (<https://www.linkedin.com/in/muhammad-hamza-liaqat-ab73a0195/>)

顾问

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

贡献者

- 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/>)

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/quanyang-he-5775ba109/>)
- Ataur Patwary教授 (<https://www.linkedin.com/in/ataurpatwary/>)

目录

致谢	2
0. 前言	4
1. 简介	5
2. 快速入门指南	9
3. RISC-V架构概述	18
4. RVFPGA系统概述	20
5. 安装软件工具	37
6. 运行RVfpgaNexys并对其编程	43
7. VERILATOR仿真	73
8. WHISPER仿真	79
9. 附录	81

更新历史:

- 版本1.0（2020年11月发布）：
 - RVfpga课程的原始版本。
 - 版本1.1（2021年6月发布）：
 - 在实验00中增加了实验11-20的说明。
 - 将SweRVolf版本更新为0.7.3，将Verilator版本更新为4.106。
 - 增加了引导ROM初始化程序。
 - 在说明RVfpga系统的GSG中新增了图1和表1
 - 在实验10中增加了UART练习。
 - 修正了一些文字错误。
 - 版本2.0（2021年11月发布）：
 - 增加了实验11-20：文档、图、软件源、练习和解答。
 - 补充了新增实验的幻灯片。
 - 在GSG和实验0-10中增加了一些细节，并修正了一些文字错误。
-



0. 前言

RVfpga计算机体系结构课程通过具体实验帮助用户了解商用RISC-V处理器、RISC-V SoC和RISC-V生态系统。本课程按照以下顺序介绍RISC-V系统：从基础数字设计和信号到指令集架构和处理器，再到编程环境、引导代码和编译器。用户可以通过RVfpga课程全面的了解RISC-V系统。用户不仅可以了解RISC-V SoC和RISC-V生态系统的工作状态，还能够掌握如何使用并扩展RISC-V处理器和系统来支持未来的项目和研究。

David Patterson教授（因对RISC的贡献而与John Hennessy共获ACM A.M.图灵奖）表示：“RISC-V正在推动处理器设计以及软件/硬件协同设计发生巨大变革。RISC-V是一种支持开源硬件实现的开放式架构。这种全新设计意味着软件开发可与硬件开发同步进行，从而加快设计速度。RVfpga课程可加强对RISC-V处理器、RISC-V生态系统和RISC-V SoC的了解。本课程可帮助用户深入了解日益普及的工业级处理器架构和系统，这将在他们的整个学术生涯和职业生涯中发挥巨大作用。

1. 简介

重要信息：在开始之前，将从Imagination大学计划下载的**RVfpga**文件夹复制到您的Ubuntu/Windows/macOS计算机上。我们将此**RVfpga**文件夹所在目录的绝对路径称为**[RVfpgaPath]**。

RVfpga包含两个文档（**幻灯片和本入门指南**）和五个文件夹：

- (1) **示例：**使用本指南时将运行的示例程序
- (2) **src：**包含**RVfpga**系统的源代码（Verilog和SystemVerilog）（参见下面的图1）
- (3) **verilatorSIM：**包含用于在Verilator中运行**RVfpgaSim**仿真的脚本
- (4) **driversLinux_NexysA7：**包含Nexys A7 FPGA开发板的Linux驱动程序
- (5) **实验：**包含**RVfpga**实验1-20期间将使用的程序、文档、图表、说明、任务、练习和解答。
实验0是一份介绍性文档，其中包含**RVfpga**实验概述，您应该在开始实验之前阅读该文档。

RISC-V FPGA（也称为**RVfpga**）是一个包含用户指导、工具和实验的软件包，用于将商用RISC-V处理器应用于现场可编程门阵列（Field Programmable Gate Array, FPGA）和仿真器，然后使用并扩展此处理器来学习关于计算机体系结构、数字设计、嵌入式系统和编程方面的知识。

“**RVfpga入门指南**”包含以下部分，简述如下：

- 快速入门指南（第2部分）
- 背景和概述
 - RISC-V架构（第3部分）
 - RVfpga系统（第4部分）
- 在硬件中使用**RVfpga**系统
 - 安装软件工具（第5部分）
 - 运行**RVfpga**系统并对其编程（第6部分）
- **RVfpga**系统仿真
 - 使用Verilator（一种HDL仿真器）（第7部分）
 - 使用Whisper（Western Digital的指令集仿真器）（第8部分）
- 附录
 - 使用配套的**RISC-V**工具链和**OpenOCD**（附录A）
 - 在**Windows**中安装并使用**PlatformIO**所需的驱动程序（附录B）
 - 在**Windows**中安装**Verilator**和**GTKWave**（附录C）
 - 在**macOS**中安装**Verilator**和**GTKWave**（附录D）
 - 使用**Vivado**将**RVfpga**系统加载到**FPGA**上（附录E）
 - 示例：在工业物联网应用中使用**RVfpga**（附录F）

“快速入门指南”（第2部分）介绍**RVfpga**所需安装的最少软件，并说明如何基于**RVfpga**系统下载和执行简单的示例程序。如需更全面地了解**RVfpga**，请跳过第2部分，从第3部分的完整指南开始。

第3部分简要介绍RISC-V计算机架构。第4部分介绍RVfpga系统（第4.A – 4.C部分）和构成系统的Verilog文件的组织结构（第4.D部分）。RVfpga系统是基于SweRVolf SoC（<https://github.com/chipsalliance/Cores-SweRVolf>），同时后者又使用Western Digital（WD）的开源RISC-V SweRV EH1内核（<https://github.com/chipsalliance/Cores-SweRV>）。图1和表1说明RVfpga系统的层级结构（从SweRV EH1内核开始，一直到RVfpgaNexys和RVfpgaSim）。

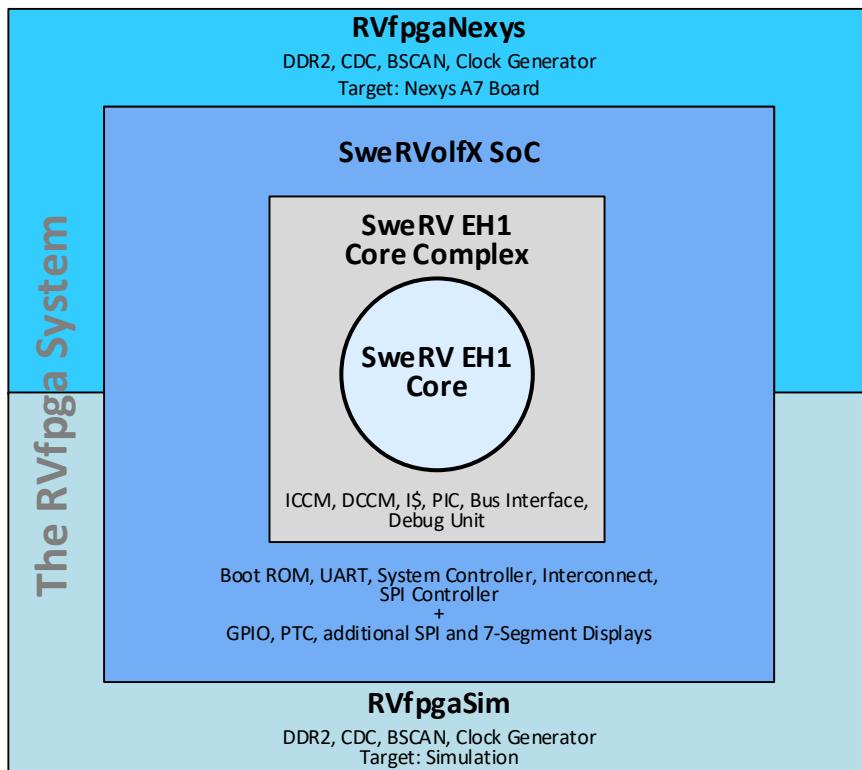


图1. RVfpga系统层级

表1. RVfpga系统层级

名称	说明
SweRV EH1内核	由Western Digital开发的开源商用RISC-V内核（ https://github.com/chipsalliance/Cores-SweRV ）。
SweRV EH1内核组合	一种增加了存储器（ICCM、DCCM和指令高速缓存）、可编程中断控制器（Programmable Interrupt Controller, PIC）、总线接口和调试单元的SweRV EH1内核（ https://github.com/chipsalliance/Cores-SweRV ）。
SweRVolfX (扩展SweRVolf)	我们在RVfpga课程中使用的片上系统。它是SweRVolf的扩展。 <u>SweRVolf</u> （ https://github.com/chipsalliance/Cores-SweRVolf ）：一种围绕SweRV EH1内核组合构建的开源SoC。它增加了引导ROM、UART接口、系统控制器、互连（AXI互连、Wishbone互连和AXI转Wishbone桥）以及SPI控制器。 <u>SweRVolfX</u> ：与SweRVolf相比增加了4个新外设：GPIO、PTC、一个额外的SPI以及用于8位7段显示屏的控制器。

RVfpgaNexys	以Nexys A7开发板及其外设为目标的SweRVolfX SoC。它增加了DDR2接口、CDC（时钟域交叉）单元、BSCAN逻辑（用于JTAG接口）和时钟发生器。 RVfpgaNexys与SweRVolf Nexys基本相同（ https://github.com/chipsalliance/Cores-SweRVolf ），只是后者基于SweRVolf。
RVfpgaSim	SweRVolfX SoC具有配套的测试平台和用于仿真的AXI内存。 RVfpgaSim与SweRVolf Sim基本相同（ https://github.com/chipsalliance/Cores-SweRVolf ），只是后者基于SweRVolf。

其余部分说明如何将RVfpga系统用于硬件（RVfpgaNexys）和仿真（RVfpgaSim）。第5部分说明如何安装并使用RVfpga所需的软件工具。第6部分说明如何使用PlatformIO将RVfpgaNexys下载到Nexys A7 FPGA开发板上（第6.A部分），以及基于RVfpgaNexys运行多个示例程序（第6.B-6.H部分）。第7部分和第8部分说明如何使用开源的HDL仿真器Verilator（第7部分）和Western Digital的RISC-V指令集仿真器（Instruction Set Simulator, ISS）Whisper（第8部分）仿真RVfpgaSim。

最后，附录说明了如何在Linux中命令行界面下使用RVfpga（附录A），如何在Windows和macOS计算机上安装所需的驱动程序和软件（附录B-D）以及如何使用Vivado将RVfpgaNexys下载到FPGA上（附录E）。附录F说明了如何在工业物联网应用中使用RVfpga。

表2列出了RVfpga所需的软件和硬件。本指南说明如何在Ubuntu 18.04操作系统（Operating System, OS）上安装和使用这些工具及硬件。对于其他操作系统（例如Windows或macOS），遵循类似（但是不完全相同）的步骤。若操作方式有所不同，我们会利用突出显示功能为Windows和macOS插入具体说明。

注：如果无法使用Nexys A7 FPGA开发板，仍然可以使用Western Digital的指令集仿真器（ISS）Whisper和开源HDL仿真器Verilator通过仿真完成实验。在这种情况下，无需安装Vivado（第5.A部分）；只需安装VSCode/PlatformIO（如第2.A部分所述）和Verilator/GTKWave（如第5.C部分所述）。

表2. RVfpga所需的软件和硬件

软件		
名称	网站	费用
Vivado 2019.2 WebPACK	https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html	免费
VSCode	https://code.visualstudio.com/Download	免费
PlatformIO	https://platformio.org/ 安装在VSCode中	免费
Verilator（一种HDL仿真器）和GTKWave	https://github.com/verilator/verilator http://gtkwave.sourceforge.net/	免费
Whisper（Western Digital的RISC-V指令集仿真器）	https://github.com/chipsalliance/SweRV-ISS 安装在PlatformIO中	免费

RISC-V工具链和OpenOCD	https://github.com/riscv/riscv-gnu-toolchain https://github.com/riscv/riscv-openocd 安装在PlatformIO中	免费
硬件		
名称	网站	费用
Nexys A7 FPGA开发板*	https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/	265美元 (学术优惠价: 199美元)
RISC-V内核和片上系统 (SoC) **		
名称	网站	费用
Western Digital的SweRV EH1内核	https://github.com/chipsalliance/Cores-SweRV	免费
SweRVolf	https://github.com/chipsalliance/Cores-SweRVolf	免费

*本指南中描述的所有步骤也适用于Digilent的Nexys4 DDR FPGA开发板。

**从Imagination Technologies下载RVfpga时一并提供。

知识储备要求:

在完成本RVfpga课程（包括本RVfpga入门指南和RVfpga实验）之前，用户至少应对以下内容有基本了解：

- 数字逻辑设计
- 高级编程（最好是C语言）
- 汇编编程
- 指令集架构
- 处理器微架构
- 存储器系统

这些内容被包含在以下教科书中：《数字设计和计算机体系结构》（RISC-V版），Harris & Harris, © Morgan Kaufmann（预计出版时间：2021年夏季）。其他教科书（包括《计算机组织结构和设计》（RISC-V版本），Patterson & Hennessy, © Morgan Kaufmann 2017）涵盖其中一些内容。

2. 快速入门指南

本部分依次说明了如何安装使用RVfpga所需的必须工具，以及如何使用PlatformIO将RVfpgaNexys下载到Nexys A7 FPGA开发板上，然后基于RVfpgaNexys运行程序。同时需要购买FPGA开发板（参见表2）。这些步骤也适用于早期的开发板型号Nexys4-DDR FPGA电路板。

- A. 最低安装要求：VSCode、PlatformIO和Nexys A7开发板驱动程序**
- B. 将RVfpgaNexys下载到FPGA上并在RVfpgaNexys上运行程序**

以下说明适用于Ubuntu 18.04系统。此外，也适用于Windows10和macOS操作系统 –若操作与Ubuntu不同，我们会针对Windows和macOS插入包含具体说明的文本框。如果使用的是Ubuntu，则可以忽略这些文本框。路径书写形式如下：Linux路径使用正斜杠（/），Windows路径通常相同，但带有反斜杠（\）。

A. 最低安装要求：VSCode、PlatformIO和Nexys A7开发板驱动程序

在这一步中，将安装使用RVfpga所需的最少软件和驱动程序。首先，将安装编程环境，然后安装Nexys A7 FPGA开发板的驱动程序。

VSCode和PlatformIO安装：将使用集成开发环境（Integrated Development Environment, IDE）PlatformIO将RVfpgaNexys下载到Nexys A7开发板上，然后在RVfpgaNexys上下载并运行程序。PlatformIO编译为Microsoft的Visual Studio Code（VSCode）的扩展。PlatformIO支持跨平台操作，包含一个内置调试器。

按照以下步骤安装VSCode和PlatformIO:

1. 安装VSCode:
 - a. 从以下链接下载安装文件：<https://code.visualstudio.com/Download>
 - b. 打开一个终端，然后安装并执行VSCode:

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

Windows/macOS: VSCode软件包也支持Windows (.exe文件) 和macOS (.zip文件)，可通过<https://code.visualstudio.com/Download>获取。请按照在这些操作系统中安装和执行应用程序使用的常规步骤操作。

2. 在VSCode软件上安装PlatformIO:

- a. 通过在终端中输入以下内容来安装python3实用程序:

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

Windows/macOS: Windows中不需要执行这一步（2.a）。对于macOS，可以使用homebrew来安装python3: `brew install python3`

- b. 如果VSCode尚未打开，请按照以下步骤将其启动：选择“Start”（开始）按钮并在搜索菜单中输入“VSCode”，然后选择VSCode，或者在Ubuntu终端中输入code。
- c. 在VSCode中，单击VSCode左侧栏中的“Extensions”（扩展）图标（参见图2）。



图2. VSCode的“Extensions”（扩展）图标

- d. 在搜索框中输入*PlatformIO*，然后单击*PlatformIO IDE*旁边的“Install”（安装）按钮进行安装（参见图3）。

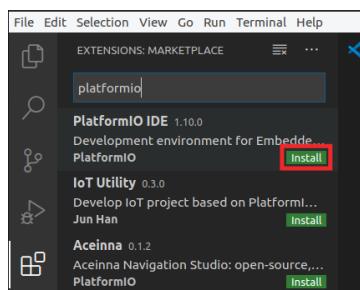


图3. PlatformIO IDE扩展

- e. 底部的“OUTPUT”（输出）窗口将通知用户有关安装进程的信息。安装完成后，单击窗口右下角的“Reload Now”（立即重新加载），PlatformIO将在VSCode内部完成安装（参见图4）。

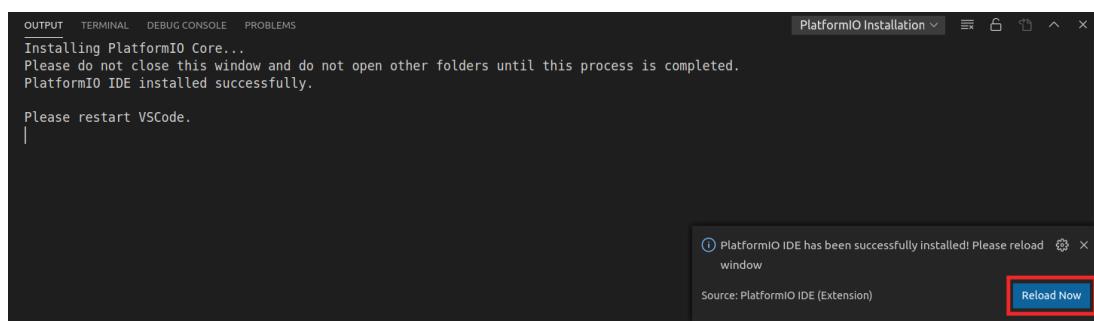


图4. 安装PlatformIO后立即重新载入

Nexys A7驱动程序安装：需要手动安装Nexys A7开发板的驱动程序。

- 打开一个终端。
- 转到目录[RVfpgaPath]/RVfpga/driversLinux_NexysA7。（为简便起见，我们在RVfpga文件夹内提供了这些驱动程序。此外，按照本指南的第5部分所述安装Vivado时，还可以在该部分所述的下载软件包内找到这些驱动程序。）

- 运行安装脚本：

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

- 从计算机上拔掉Nexys A7开发板，然后重启计算机，以使更改生效。

Windows: 按照附录B中的说明来安装Nexys A7开发板的驱动程序。

macOS: 不需要安装任何其他驱动程序。

B. 将RVfpgaNexys下载到FPGA并在RVfpgaNexys上运行程序

现在，将RVfpgaNexys（以FPGA为目标的RISC-V系统）下载到Nexys A7 FPGA开发板上。在本入门指南中，我们将不会对RVfpga系统进行修改，但[RVfpgaPath]/RVfpga/src中提供了RVfpga系统的Verilog。我们将在本指南的第4部分中介绍RVfpga系统的源代码，并在RVfpga实验6-20中加以详细说明。此外，用户还将通过这些实验中的一些练习修改RVfpga系统。

通过完成以下步骤，在Nexys A7 FPGA开发板上运行RVfpgaNexys：

- 将Nexys A7 FPGA开发板连接到计算机并接通开发板
- 打开PlatformIO和C程序
- 将RVfpgaNexys下载到Nexys A7开发板上
- 在RVfpgaNexys上下载并运行程序

第1步. 将Nexys A7 FPGA开发板连接到计算机并接通开发板

使用配套的USB电源线将Nexys A7开发板连接到计算机。图5显示了Nexys A7 FPGA开发板上的LED和开关，以及USB连接器、接通开关、按钮和7段显示屏的物理位置。将USB电源线把计算机和Nexys A7开发板上的USB连接器端口进行连接，然后打开开发板开关。

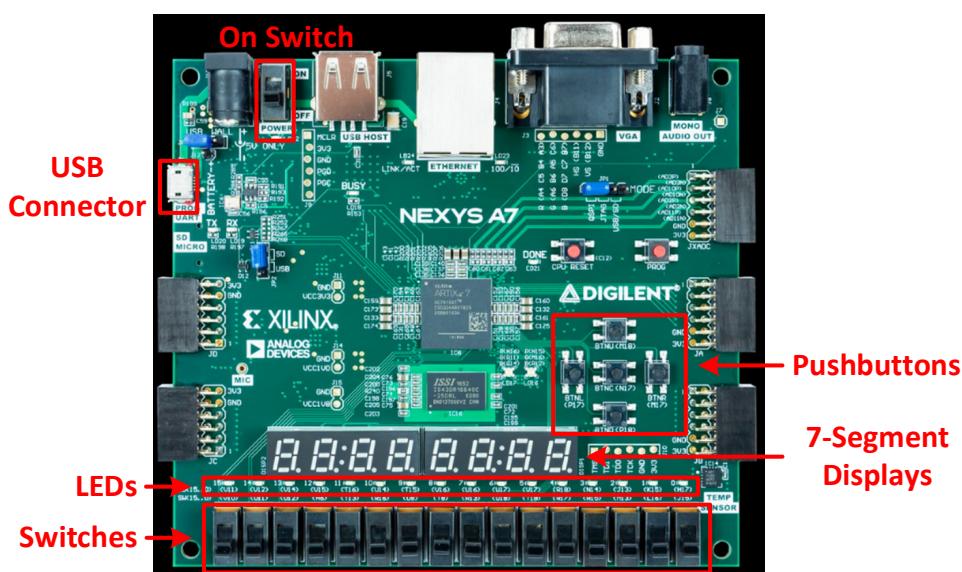


图5. Digilent的Nexys A7 FPGA开发板的I/O接口

(开发板图片来源: <https://reference.digilentinc.com/>)

第2步. 打开PlatformIO和C程序

现在，通过以下方式打开Visual Studio Code (VSCode)：在“Start”（开始）菜单中输入VSCode（参见图6）或者在终端中输入code。

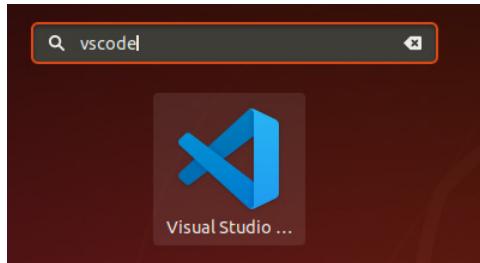


图6. 打开VSCode

如果PlatformIO主页（“PIO Home”（PIO主页））窗口没有自动打开，请单击左侧功能区菜单中的PlatformIO图标。然后展开“PIO Home”（PIO主页），并单击“Open”（打开）。此时，“PIO Home”（PIO主页）将打开并显示“Welcome”（欢迎）窗口（参见图7）。

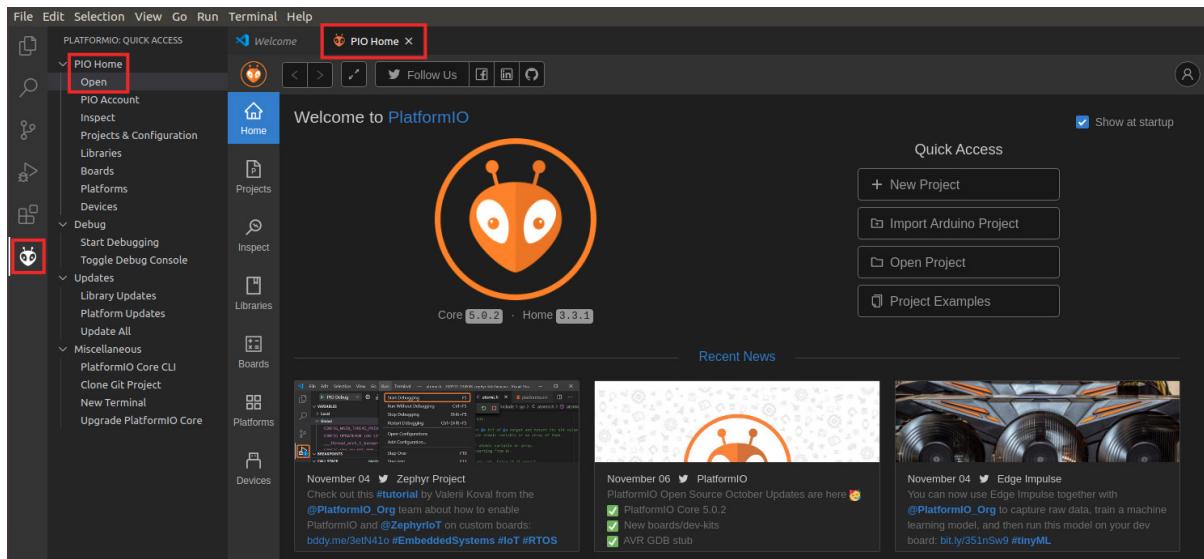


图7. 打开“PIO Home”（PIO主页）

现在单击顶部“File”（文件）菜单 → “Open Folder”（打开文件夹），然后选择：
`[RVfpgaPath]\RVfpga\examples\LedsSwitches_C-Lang`

选中文件夹，但不要将其打开（参见图8）。PlatformIO现在将打开程序LedsSwitches_C-Lang，该程序会读取Nexys A7开发板上的开关值并将这些值写入到开发板上的LED。

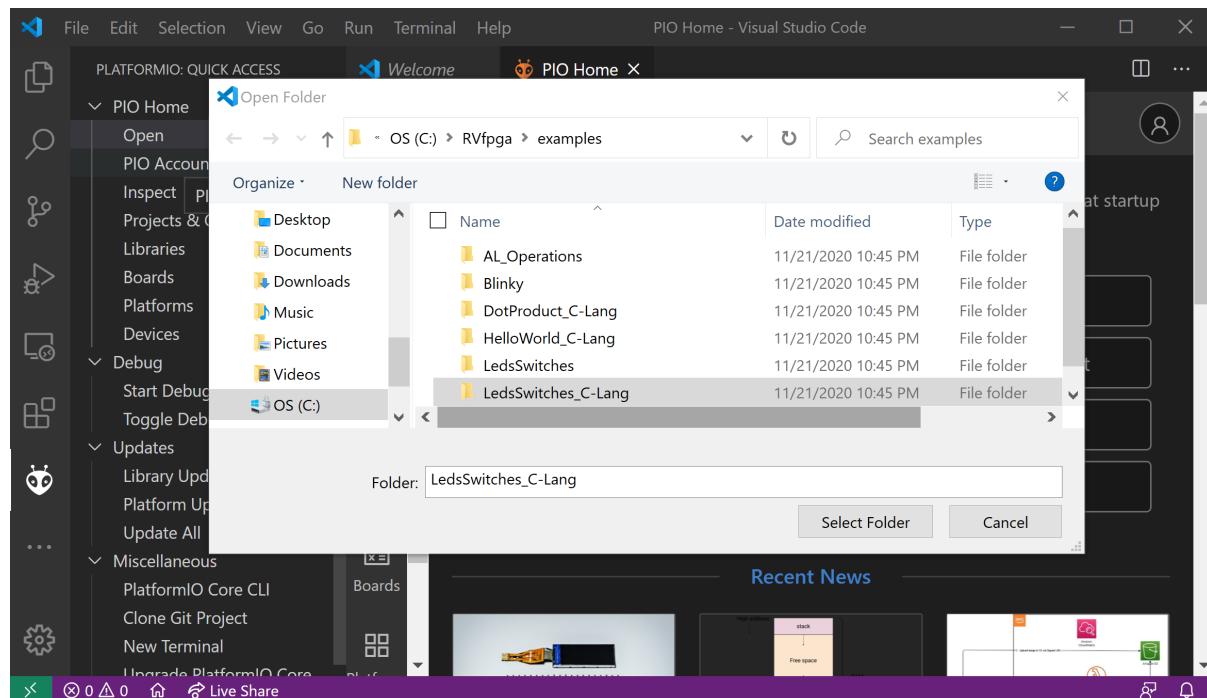


图8. 打开LedsSwitches_C-Lang示例

你可通过展开src文件夹，然后双击LedsSwitches_C-Lang.c来查看LedsSwitches_C-Lang程序（图9）。我们将在本入门指南的后面部分详细讨论该程序。在本“快速入门指南”部分，我们只将该程序下载到将在Nexys A7开发板上运行的RVfpgaNexys。

The screenshot shows the Visual Studio Code interface with the title bar 'LedsSwitches_C-Lang.c - LedsSwitches_C-Lang - Visual Studio Code'. The left sidebar shows the 'EXPLORER' view with files like '.pio', '.vscode', 'include', 'lib', and 'src'. The 'src' folder contains 'LedsSwitches_C-Lang.c', 'start.S', 'test', 'gitignore', '.travis.yml', 'platformio.ini', and 'README.rst'. The right pane shows the code editor with the following C code:

```

#define GPIO_SWs 0x80001400
#define GPIO_LEDs 0x80001404
#define GPIO_INOUT 0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main ( void )
{
    int En_Value=0xFFFF, switches_value;
    WRITE_GPIO(GPIO_INOUT, En_Value);

    while (1) {
        switches_value = READ_GPIO(GPIO_SWs);
        switches_value = switches_value >> 16;
        WRITE_GPIO(GPIO_LEDs, switches_value);
    }
    return(0);
}

```

图9. LedsSwitches_C-Lang.c程序

请注意，首次在PlatformIO中打开RVfpga示例时，Chips Alliance平台会自动安装（可以在“PIO Home”（PIO主页）→“Platforms”（平台）中查看该平台，如图10所示）。该平台包含以后将用到的几个工具，例如预编译的RISC-V工具链、用于RISC-V的OpenOCD、RVfpgaNexys bit文件和RVfpgaSim、JavaScript和Python脚本以及一些示例。如果由于某种原因未自动安装Chips Alliance平台，则可以手动完成安装，具体过程请参见第6.A部分。

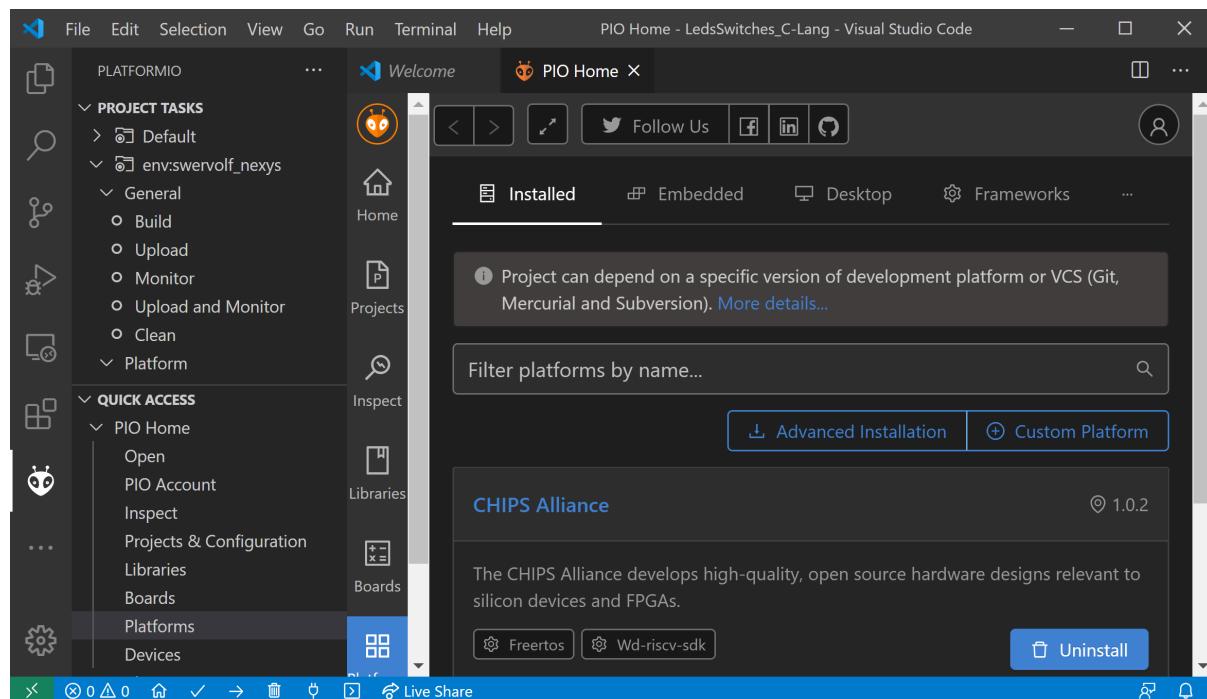


图10. PlatformIO中安装的Chips Alliance平台

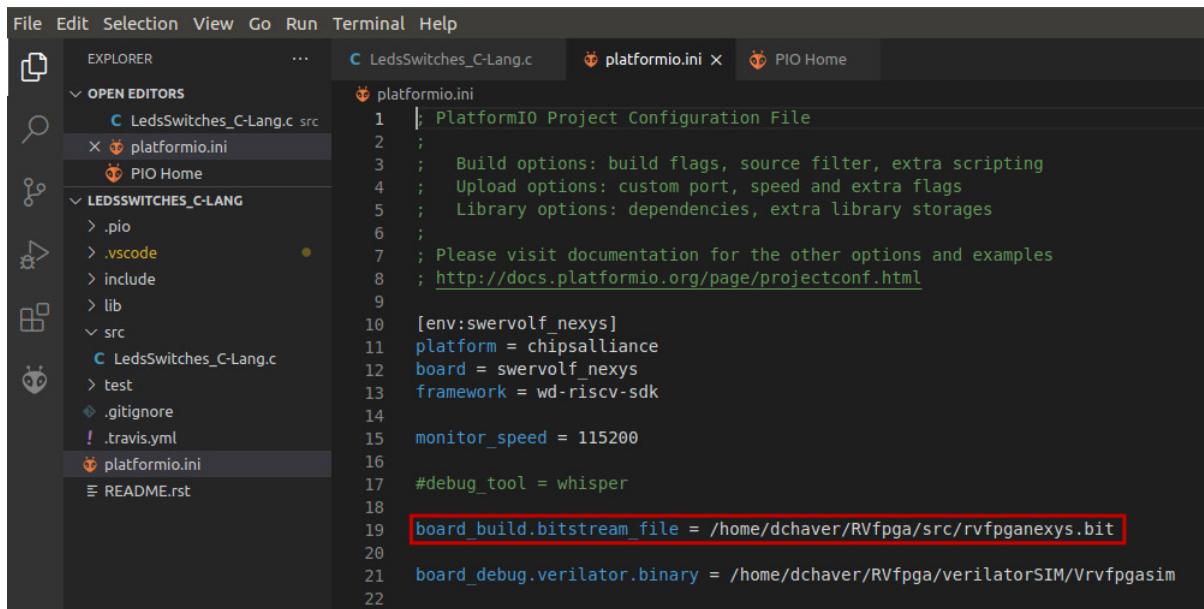
第3步. 将RVfpgaNexys下载到Nexys A7开发板上

现在可以下载RVfpgaNexys，它是一个可以支持外设的RISC-V处理器，属于RISC-V SoC的一种。在“EXPLORER”（资源管理器）窗口中双击platformio.ini（PlatformIO初始化文件）将其打开，如图11所示。（如果“EXPLORER”（资源管理器）窗口尚未打开，请通过单击左侧功能区菜单中的`□`将其打开。）现在，通过将board_build.bitstream_file路径替换为您自己的路径，来添加用于定义RVfpgaNexys的bit文件的位置路径（参见图11）：

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

按Ctrl-S保存platformio.ini文件。

有多条命令可用于项目配置文件（`platformio.ini`）；有关这些选项的更多信息，请访问：
<https://docs.platformio.org/en/latest/projectconf/>。



```

File Edit Selection View Go Run Terminal Help
EXPLORER ... C LedsSwitches_C-Lang.c platformio.ini PIO Home
OPEN EDITORS
  C LedsSwitches_C-Lang.c src
  X platformio.ini
  PIO Home
LEDSSWITCHES_C-LANG
  >.pio
  >.vscode
  >include
  >lib
  >src
  C LedsSwitches_C-Lang.c
  >test
  .gitignore
  !.travis.yml
  platformio.ini
  README.rst
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter, extra scripting
4 ; Upload options: custom port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ;
7 ; Please visit documentation for the other options and examples
8 ; http://docs.platformio.org/page/projectconf.html
9
10 [env:swervolf_nexys]
11 platform = chipsalliance
12 board = swervolf_nexys
13 framework = wd-riscv-sdk
14
15 monitor_speed = 115200
16
17 #debug_tool = whisper
18
19 board_build.bitstream_file = /home/dchaver/RVfpga/src/rvfpganexys.bit
20
21 board_debug.verilator.binary = /home/dchaver/RVfpga/verilatorSIM/Vrvfpgasim
22

```

图11. 在RVfpgaNexys bit文件中添加路径

将RVfpgaNexys（由该bit文件定义）下载到Nexys A7开发板上：

- 单击左侧功能区菜单中的PlatformIO图标  （参见图12）。



图12. PlatformIO图标

- 如果“Project Tasks”（项目任务）窗口为空（图13），必须先通过单击  来刷新“Project Tasks”（项目任务）。这可能需要几分钟。

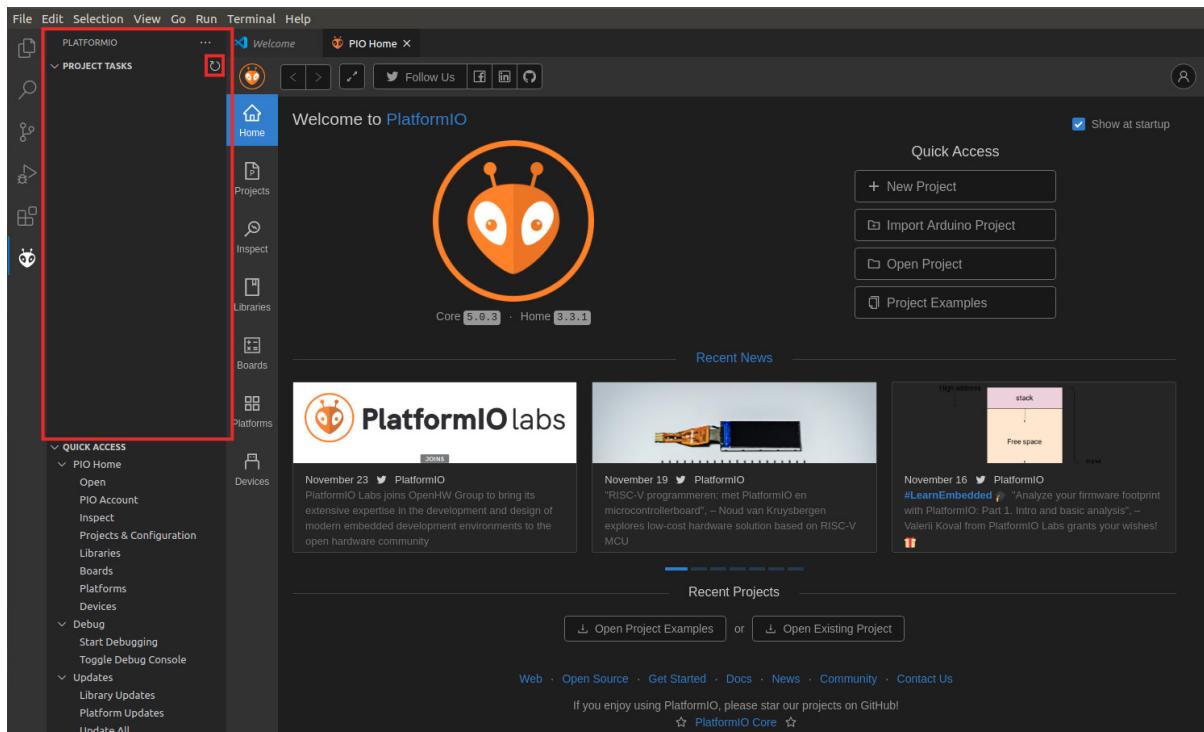


图13. “PROJECT TASKS”（项目任务）窗口为空 – 刷新

- 然后展开“Project Tasks”（项目任务）→ env:swervolf_nexys → “Platform”（平台），并单击“Upload Bitstream”（上传比特流），如图14所示。一到两秒后，将使用RVfpgaNexys SoC对FPGA进行编程。

默认情况下，处理器从地址0x80000000处开始取指，其中引导ROM位于我们的SoC中（参见表6）。引导ROM使用一个程序（*boot_main.mem*）进行初始化，该程序使LED和7段显示屏闪烁四次，然后熄灭所有LED，将0写入8个7段显示屏并保持空循环。该程序位于以下文件夹：*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw*。如果要进行更改和重新编译，请按照附录A – 第III部分中的说明进行操作（注意文件*boot_main.mem*只是文件*boot_main.vh*的副本）。在实验1中，我们将介绍如何在创建比特流时使用该程序初始化引导ROM。

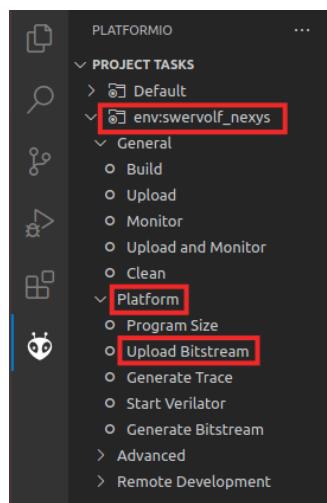


图14. 上传比特流

第4步. 在RVfpgaNexys上下载并运行程序

现在，RVfpgaNexys已下载到Nexys A7开发板上并运行，接下来会将程序下载到RVfpgaNexys的存储器中，然后运行/调试该程序。单击“Run and Debug”（运行和调试）按钮：，该按钮位于左侧栏中。单击播放按钮启动调试器（确保“PIO Debug”（PIO调试）选项已选中）。播放按钮靠近窗口顶部位置（参见图15）。程序将先编译，然后开始调试。

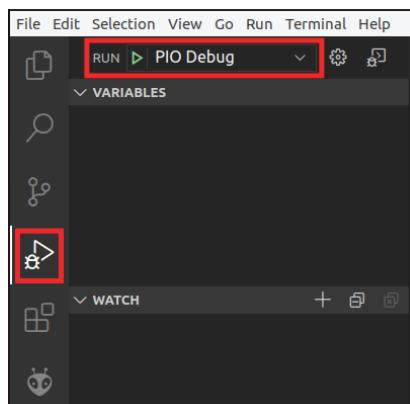


图15. 编译和下载程序，以及启动调试器

要控制调试会话，可以使用靠近编辑器顶部位置的调试工具栏（参见图16）。我们将在本入门指南的后面部分描述和测试所有选项。



图16. 调试工具

PlatformIO在main函数的开头设置一个临时断点。因此，单击“Continue”（继续）按钮运行程序。现在打开Nexys A7 FPGA开发板上的开关，并查看相应的LED是否点亮。

3. RISC-V架构概述

RISC-V是一种指令集架构（Instruction Set Architecture, ISA），于2011年在加州大学伯克利分校的Par实验室中创建。RISC-V旨在成为面向各种应用范围（涵盖小型、受限、低资源物联网设备到超级计算机等各种应用）的处理器的“通用ISA”。为实现这一目标，RISC-V架构师制定了五项架构原则：

- 架构必须与各种软件包和编程语言兼容。
- 架构必须能够通过各种技术实现，包括FPGA到ASIC（专用集成电路）以及新兴技术。
- 架构必须适用于各种微架构场景，包括实现微代码或硬连线控制、顺序或乱序流水线、各种类型的并行性等。
- 架构必须能够针对特定任务进行定制，以实现所需的最大性能，而不会因ISA本身而引起缺陷。
- 架构的基本指令集必须稳定且持久，从而为开发人员提供一个通用且稳固的框架。

RISC-V是一种开放标准，实际上，此规范隶属于公共领域，自2015年起一直由**RISC-V Foundation**（现称为**RISC-V International**，是一家促进RISC-V架构软硬件开发的非营利组织）管理。2018年，RISC-V Foundation与Linux Foundation开展了持续的合作；2020年3月，RISC-V Foundation更名为RISC-V International，总部设于瑞士。这种转变消除了社区对标准未来开放性的任何担忧。截至2020年，RISC-V International得到了来自研究机构、学术界和工业界的200多家关键企业的支持，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是过去10-20年间创建的少数全球性相关ISA之一，而且可能是唯一的全球相关ISA，因为它是一种开放标准并采用模块化，而不是增量式的设计。这种模块化使其既灵活又简洁。处理器实现基本ISA，并且仅实现所使用的扩展。这种模块化方法不同于采用增量式架构的传统ISA（例如x86或ARM），它对之前的ISA进行了扩展，并且每个新处理器必须实现所有指令（包括标记为“过时”的指令）以确保与旧版软件程序兼容。例如，x86从一开始具有80条指令到现在具有超过1300条指令，如果考虑机器代码中所有可用的不同操作码，则具有3600条指令。由于大多数短操作码或短指令已在使用中，因此为了处理大量指令和满足向后兼容性的要求，必须提供能够支持长指令的大型高能耗处理器。

RISC-V具有四个基本ISA选项：两个32位版本（整型和嵌入式版本，RV32I和RV32E）以及64位和128位版本（RV64I和RV128I），如表3所示。标记为“**Ratified**”（批准）的ISA模块现已经过批准。标记为“**Frozen**”（冻结）的模块预计在提交批准之前不会发生重大更改。标记为“**Draft**”（草稿）的模块预计在批准之前发生更改。对于成本、空间和资源有限的设备而言，尤其需要具备编译小型处理器的能力。可以在这些基本ISA的上层添加指令扩展，以实现特定任务，例如浮点运算、乘法和除法以及向量运算。这些专用的硬件扩展也包含在标准中，并且编译器知道这些标准，因此在编译器中使用所需的选项可以被允许生成目标二进制代码。其中每个扩展都用一个字母标识，该字母必须添加到内核ISA中以表示实现的硬件功能，如表4所示。例如，RVM是乘法/除法扩展，RVF是浮点扩展等等。

表3. RISC-V基本ISA

(表来自<https://riscv.org/technical/specifications/>)

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

表4. RISC-V标准ISA扩展

(表来自<https://riscv.org/technical/specifications/>)

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

字母**G**表示“通用”，用于指示包含所有**MAFD**扩展。请注意，公司或个人可能会使用标准模块中没有使用的操作码来开发专有扩展。这样有助于第三方加快实现，从而缩短其上市时间。

例如，包含全部通用ISA扩展以及位操作和用户级中断的64位RISC-V被称为RV64GBN ISA。所有这些模块都符合非特权或用户规范。此外，RISC-V基金会还规定了运行通用操作系统所需的特权操作的一组要求和指令。

4. RVFPGA系统概述

本部分介绍整个RVfpga系统，包括从内核到FPGA开发板接口的各个部分。图17说明嵌入式系统的典型层级结构，从处理器内核开始，然后是围绕内核编译的SoC，最后是系统和开发板接口。

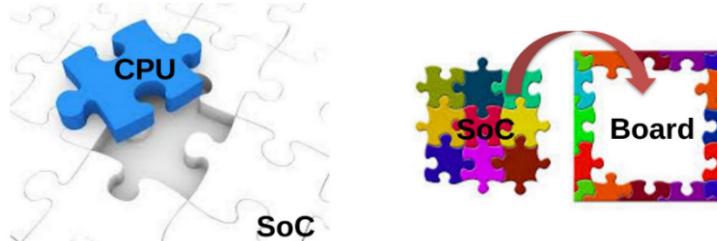


图17. 嵌入式系统结构

图1和表1说明系统的层级结构（从SweRV EH1内核开始，一直到RVfpgaNexys和RVfpgaSim）。在以下部分中我们首先介绍执行RISC-V指令的处理器内核（**Western Digital的SweRV EH1内核**）；然后，在B部分介绍**SweRVolfX SoC**，它集成了系统的硬件组件（内核、存储器和输入/输出）以及为在RVfpga内使用此SoC而执行的扩展；在C部分介绍在Nexys A7 FPGA开发板（**RVfpgaNexys**）上实现的SweRVolfX SoC以及在仿真（**RVfpgaSim**）中使用的SweRVolfX SoC。最后，在D部分介绍整个RVfpga系统的文件结构。

A. SweRV EH1内核和SweRV EH1内核组合

Western Digital在过去几年中开发了三种RISC-V内核：**SweRV EH1**（RVfpga系统中使用的内核）、SweRV EH2和SweRV EL2（RVfpga的未来版本中可能包含这些内核）。每个内核都有一个Apache 2.0许可证。SweRV EH1是一个32位内核，采用2路超标量和9级流水线设计。SweRV Core EH2建立在EH1内核基础上并对其进行了扩展，支持双线程功能，性能得到了提升。SweRV Core EL2是一种小型内核，可提供中等性能。**RISC-V**页面（网址为<https://www.westerndigital.com/company/innovations/risc-v>）概述了这三种可用的内核，它们的主要特性如表5所示。

表5. 三种WD RISC-V内核的主要特性
(表来自<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

在这三种内核中，**SweRV EH1内核**（随RVfpga软件包一起提供，也可从<https://github.com/chipsalliance/Cores-SweRV>获取）可提供高性能/MHz，并且采用简单的线程结构，是首选内核。此外，致力于提供开源硬件的组织Chips Alliance提供了一个完整且经过验证的SoC，称为SweRVolf（随RVfpga软件包提供，也可从

<https://github.com/chipsalliance/Cores-SweRVolf>获取）。RVfpga系统使用SweRVolf SoC的扩展，而后者又使用Western Digital的SweRV EH1内核版本1.8。

SweRV EH1内核是一种仅限机器模式（M模式）的32位CPU内核，支持RISC-V的整数（I）、压缩指令（C）以及整数乘法和除法（M）扩展。参考手册（https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf）详细描述了内核从结构到定时信息和存储器映射的各个方面。

SweRV EH1是超标量内核，采用双发射9级流水线设计（参见图18），支持四个算术逻辑单元（Arithmetic Logic Unit, ALU），这些单元在两条流水线I0和I1中标记为EX1至EX4。两路流水线均支持ALU运算。其中一路流水线支持装载/存储，另一路具有3周期延时乘法器。处理器还具有一个非流水线式34周期的延时除法器。流水线中存在四个停顿点：“取指1”、“对齐”、“译码”和“提交”。 “取指1”级包括Gshare分支预测器。在“对齐”级，从三个缓冲区中进行取指。在“译码”级，最多对四个指令缓冲区中的两条指令进行译码。在“提交”级，每个周期最多提交两条指令。最后，在“回写”级，更新架构寄存器。

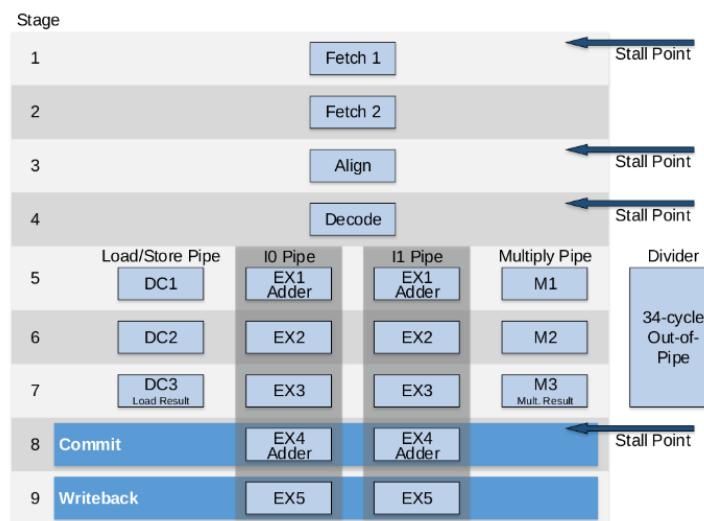


图18. SweRV EH1内核微架构

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

图19对目前的各种内核和处理器进行了比较。SweRV EH1内核的性能非常高，可达4.9 CM/MHz（CoreMark/MHz）：其运行速度为ARM Cortex A8的两倍，性能甚至超过ARM Cortex A15。

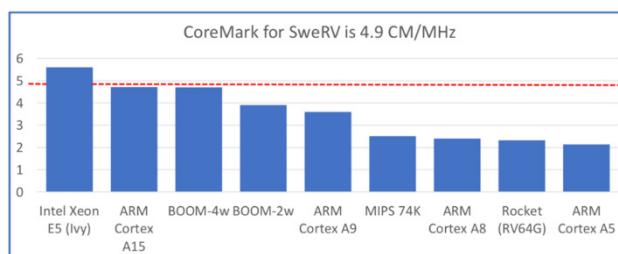


图19. 单线程和频率的基准比较

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

Western Digital还提供了SweRV EH1内核的扩展，称为**SweRV EH1内核组合**（参见图20），此扩展在上述EH1内核的基础上增加了以下元件，如图中蓝色部分所示：

- 两个专用存储器，一个用于存储指令（ICCM），另一个用于存储数据（DCCM），它们与内核紧密耦合。这两个存储器提供低延时访问和SECDED ECC（单一误差校正和双重误差检测纠错码）保护。每个存储器可以配置为4、8、16、32、48、64、128、256或512KB。
- 可选的4路组相连指令高速缓存，具有奇偶校验或ECC保护。
- 可选的可编程中断控制器（Programmable Interrupt Controller, PIC），最多支持255个外部中断。
- 四个系统总线接口，用于取指（IFU总线主控）、数据访问（LSU总线主控）、调试访问（调试总线主控）和外部DMA访问（DMA从端口）紧密耦合的存储器（可配置为64位AXI4或AHB-Lite总线）。
- 符合RISC-V调试规范的内核调试单元。

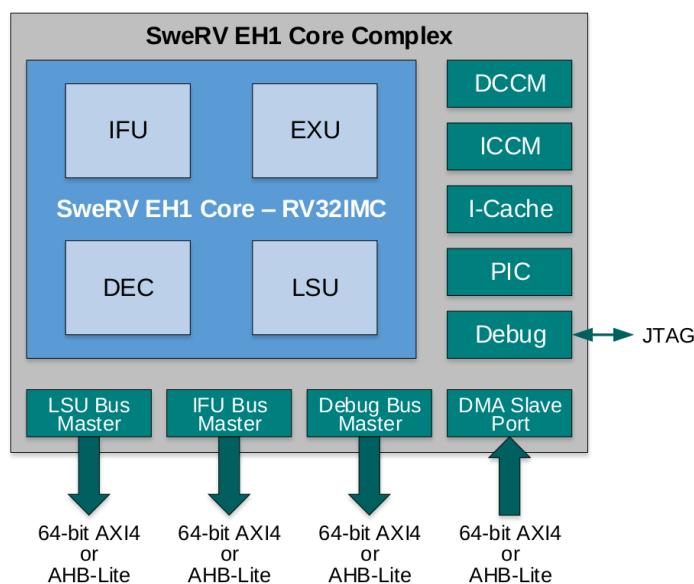


图20. SweRV EH1内核组合

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

B. SweRVolfX SoC

此RVfpga软件包中使用的片上系统（SoC）称为SweRVolfX（如图21所示），它基于SweRVolf0.7.3版本（<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>），而后者基于SweRV EH1内核组合构建。除了SweRV EH1内核组合（参见图20）外，SweRVolf SoC还包括一个引导ROM、一个UART、一个系统控制器和一个SPI控制器（这些元件如图21中的白色部分所示）。如果SweRV EH1内核使用AXI总线，而外设使用Wishbone总线，则SoC也具有AXI-Wishbone桥。

在RVfpga中，SweRVolf SoC扩展了一些功能，例如一个额外SPI控制器（SPI2）、GPIO（通用输入/输出）控制器、PTC（PWM/定时器/计数器）模块以及与8位7段显示屏接口的控制器。这些新外设如图21中红色部分所示，但是7段显示屏控制器除外，它被包含在系统控制器当中。我们把这个系统称为片上系统**SweRVolfX**（X代表扩展）。

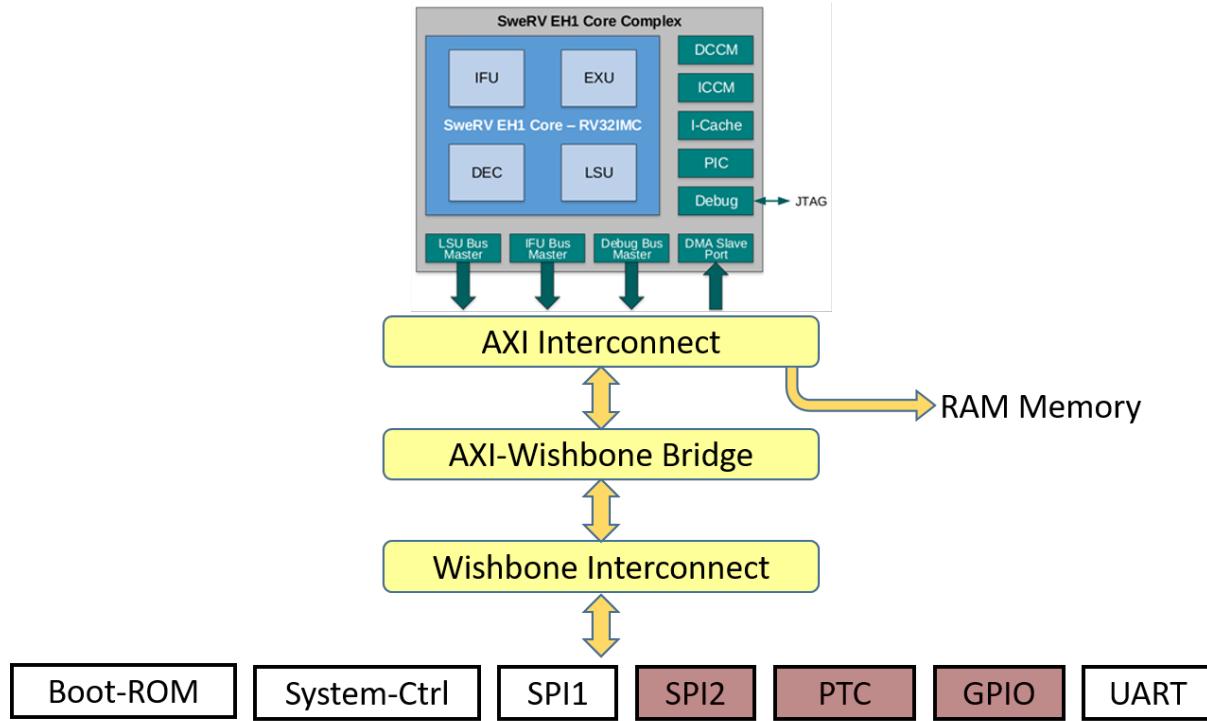


图21. SweRVolfX（使用4个新外设扩展SweRVolf）片上系统

表6列出了通过Wishbone互连连接到内核的外设的存储器映射地址。

表6. 扩展SweRVolfX SoC外设的存储器映射地址

系统	地址
引导ROM	0x80000000 - 0x80000FFF
系统控制器	0x80001000 - 0x8000103F
SPI1	0x80001040 - 0x8000107F
SPI2	0x80001100 - 0x8000113F
PTC	0x80001200 - 0x8000123F
GPIO	0x80001400 - 0x8000143F
UART	0x80002000 - 0x80002FFF

i. 输入/输出

SweRVolfX SoC使用两种硬件控制器与外设通信：用Verilog编写的自定义控制器和来自OpenCores [<https://opencores.org/>]的开源控制器，OpenCores是一个基于免费和开源合作的原则来开发gateware（门件）IP（知识产权）内核的在线社区。我们在本课程中使用的SweRVolfX SoC包括下面列出的I/O接口，我们将在RVfpga的实验6-20中使用、详述这些接口，甚至对它们进行扩展。

- 系统控制器：**系统控制器包含常见的系统功能，例如在寄存器中保存SoC版本信息、RAM初始化状态和RISC-V机器定时器（可以在<https://github.com/chipsalliance/Cores-SweRVolf>下找到完整的存储器映射）。我们对此模块进行了如下修改：

- 添加了一个新的控制器（称为**SevSegDisplays_Controller**）来与Nexys A7开发板上提供的8位7段显示屏通信，同时还为此控制器添加了两个新的寄存器（映射到地址0x80001038和0x8000103C）。
- 添加了两个1位寄存器（映射到地址0x80001018）来处理来自GPIO和PTC的中断。
- 删除了SweRVolf提供的简单GPIO寄存器（映射到地址0x80001010–0x80001017）。请注意，我们添加了一个更完整的GPIO控制器，如下所述。
- **SPI:** 在SweRVolfX中实现了两个开源SPI控制器（从https://opencores.org/projects/simple_spi获取，命名为SPI1和SPI2）。这些控制器的公开的寄存器（SPI_SPCR、SPI_SPSR、SPI_SPDR、SPI_SPER、SPI_SPSS）映射到地址0x80001040和0x8000107F（对于SPI1）之间以及地址0x80001100和0x8000113F（对于SPI2）之间。
- **PTC:** 我们使用的定时器模块来自<https://opencores.org/projects/ptc>。其寄存器映射到地址范围0x80001200至0x800012FF。
- **GPIO:** 我们使用的GPIO控制器来自<https://opencores.org/projects/gpio>。该控制器包括32个I/O端口，这些端口映射到地址范围0x80001400至0x800014FF。每个引脚连接一个三态缓冲区，因此可以将其配置为输入或输出。
- **UART:** SweRVolfX中提供了开源UART控制器（从<https://opencores.org/projects/uart16550>获取）。该控制器的公开的寄存器映射到地址0x80002000和0x80002FFF之间。

ii. 存储器

SweRVolfX SoC包括引导ROM存储器以及允许用户将RAM和SPI闪存包含在内的必要硬件。

- **引导ROM:** 引导ROM包含第一阶段的引导程序。系统复位后，SweRVolfX SoC将开始从此区域获取初始指令，该区域占用的地址为0x80000000至0x80000FFF。
- **RAM:** SweRVolfX SoC不包含存储控制器，但保留了其存储器映射（0x00000000–0x07FFFFFF）的前128 MiB，并且显示AXI总线，以便用户可以使用存储控制器访问RAM存储器。
- **SPI闪存:** 也可以使用上一部分中介绍的SPI1控制器包含SPI闪存（地址范围：0x80001040–0x8000107F）。

iii. 互连

SweRV EH1内核使用AXI4总线连接内核和存储器。该总线也可以配置为AHB-Lite总线，但是在本指南中我们不使用这一选项。所有外设（I/O设备）都连接到Wishbone总线，这是一种在OpenCore CPU和外设中被广泛使用的开源总线。系统包括AXI-Wishbone桥（如图21所示），用于将内核连接到外设。

在本节中，我们简要介绍了AXI4总线和Wishbone总线的操作。如果您有兴趣了解有关这些总线规范的更多信息，可以使用下面提供的参考。

AXI4总线

SweRV EH1内核组合使用AXI4互连与外界通信（参见图20）。高级可扩展接口（Advanced eXtensible Interface, AXI）是许多处理器的通用总线，并且是ARM高级微处理器总线架构片上互连规范的一部分。

在下面的小节中，我们将简要说明AXI4互连的一些主要方面。有关完整的AXI规范，请参见以下文档：

https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf

• AXI总线的主要特性

AXI总线技术的主要特性如下：

- 适用于高带宽和低延时设计
- 无需使用复杂的桥就能实现高频操作
- 可以满足各种组件的接口要求
- 适用于初始访问时具有高延时的存储控制器
- 在互连架构实现方面具有灵活性
- 与现有AHB和APB接口向后兼容
- 分离的地址/控制和数据阶段
- 支持非对齐数据的传输（使用字节选通）
- 允许仅发出起始地址的突发型事务
- 提供支持低功耗DMA的单独读写数据通道
- 允许在实际数据传输之前进行寻址
- 支持多个未完成传输的地址进行操作和乱序传输
- 易于添加寄存器级来进行时序收敛

• AXI架构

AXI协议定义了以下独立的传输通道：

- 读地址通道
- 读数据通道
- 写地址通道
- 写数据通道
- 写响应通道

图22显示了数据读取如何使用读地址和读数据通道。首先从主设备发送地址和控制信号，然后从设备用读数据通道上的数据进行响应。

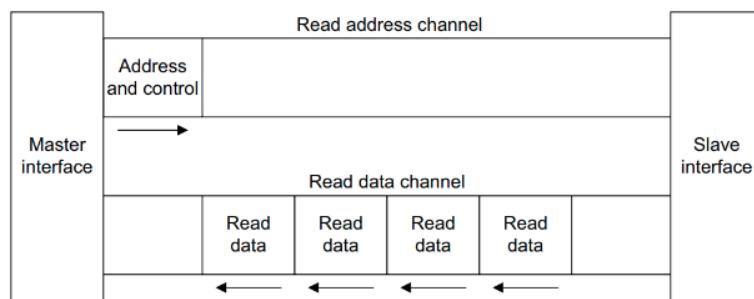


图22. 读通道架构

(图来自https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

图23显示了数据写入如何使用写地址、写数据和写响应通道。与数据读取类似，主设备发送地址和控制信号。然后，主设备发送写数据通道上的数据，从设备发送写响应。

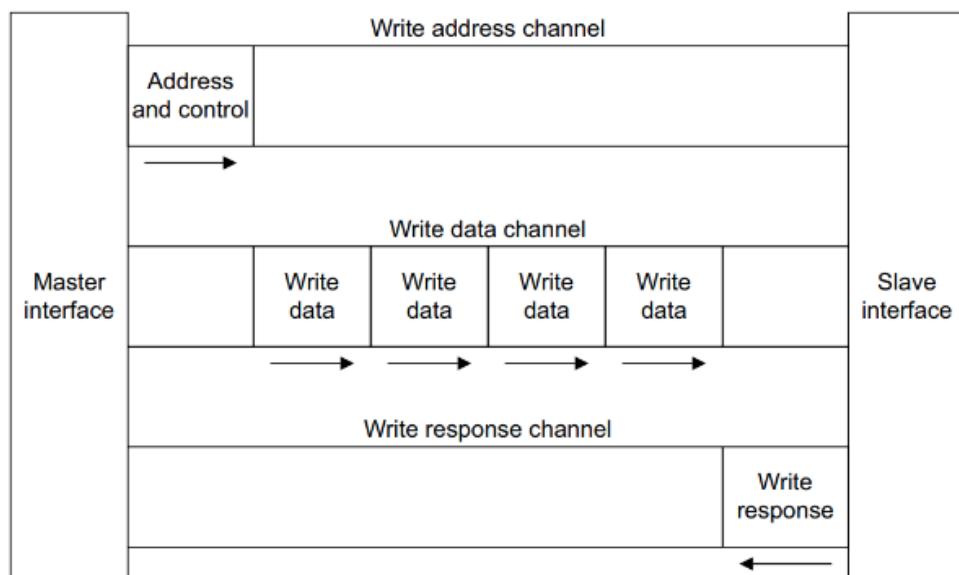


图23. 写通道架构

(图来自https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

AXI地址通道带有地址和控制信息，这些信息描述了要传输数据的性质。在主从设备之间使用以下任一方式传输数据：

- 使用读数据通道将数据从从设备传输到主设备（图22）。
- 使用写数据通道将数据从主设备传输到从设备（图23）。在数据写入过程中，从设备使用写响应通道向主设备发送传输完成信号（图23）。

• AXI信号

表7列出了AXI总线中使用的主要信号以及每个信号的简要说明。这些信号分为五组，分别对应于上文所述的五个通道：

- 写地址通道信号，名称以**AW**开头
- 写数据通道信号，名称以**W**开头

- 写响应通道信号，名称以**B**开头
- 读地址通道信号，名称以**AR**开头
- 读数据通道信号，名称以**R**开头

表7. AXI信号

(表来自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总线

SweRVolfX外设将Wishbone片上系统（SoC）互连架构用于可移植的IP内核

(<https://opencores.org/howto/wishbone>)。该总线主要用于通过解决片上系统集成问题来促进设计重复使用。以前，IP内核采用非标准互连方案，因此难以集成。这些非标准互连要求创建自定义胶合逻辑，以将每个内核连接在一起。通过采用Wishbone总线等标准互连方案，最终用户可以更快速、更轻松地集成内核。

• Wishbone主要特性

这种Wishbone总线技术的主要特性如下：

- 支持大型项目团队使用的结构化设计方法。
- 包括一整套常用的数据传输总线协议，具体如下：
 - 读/写周期
 - 块传输周期

iii. 读/修改/写周期

- 提供高达64位的模块化数据总线宽度和操作数大小。
- 支持大端和小端数据顺序。
- 支持各种内核互连方法，包括点对点、共享总线、交叉开关和交换结构互连。
- 包括握手协议，该协议允许每个IP内核限制其数据传输速度。
- 支持单时钟数据传输。
- 支持正常周期终止、重试终止和由于错误终止。
- 包括模块化地址宽度。
- 为从设备提供部分地址译码方案。这有助于高速地址译码，使用较少的冗余逻辑，并支持可变地址大小的调节和互连。
- 提供用户自定义标签。这些标签有助于将信息应用于地址、数据总线或总线周期。在修改总线周期以标识如下信息时，用户自定义标签特别有用：
 - i. 数据传输
 - ii. 奇偶校验或纠错位
 - iii. 中断向量
 - iv. 高速缓存控制操作
- 包括具有灵活系统设计的主/从架构。
- 具有多重处理（multi-MASTER）功能。这可实现各种SoC配置。
- 包括可由最终用户定义的仲裁方法（优先级仲裁器、循环仲裁器等）。

• Wishbone架构和信号

图24说明了主设备（本例中为SweRV EH1内核）和从设备（本例中为一个外设，如GPIO、SPI等）之间通过Wishbone总线实现的标准连接。Wishbone总线比AXI4总线简单得多，而且，如表8所示，Wishbone总线使用更少的信号。

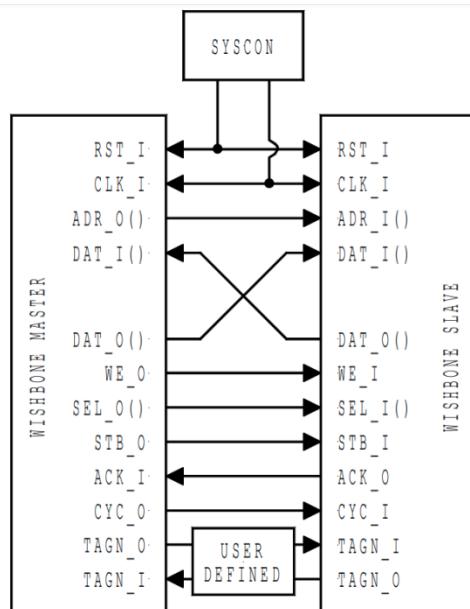


图24. Wishbone架构
(图来自<https://opencores.org/howto/wishbone>)

表8. Wishbone信号
(表来自<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上进行仿真

SweRVolfX SoC (图21) 可以(1)在Nexys A7 (或Nexys4 DDR) FPGA开发板上运行, 本例中其配置称为**RVfpgaNexys**, 也可以(2)在仿真中运行, 本例中称为**RVfpgaSim**。

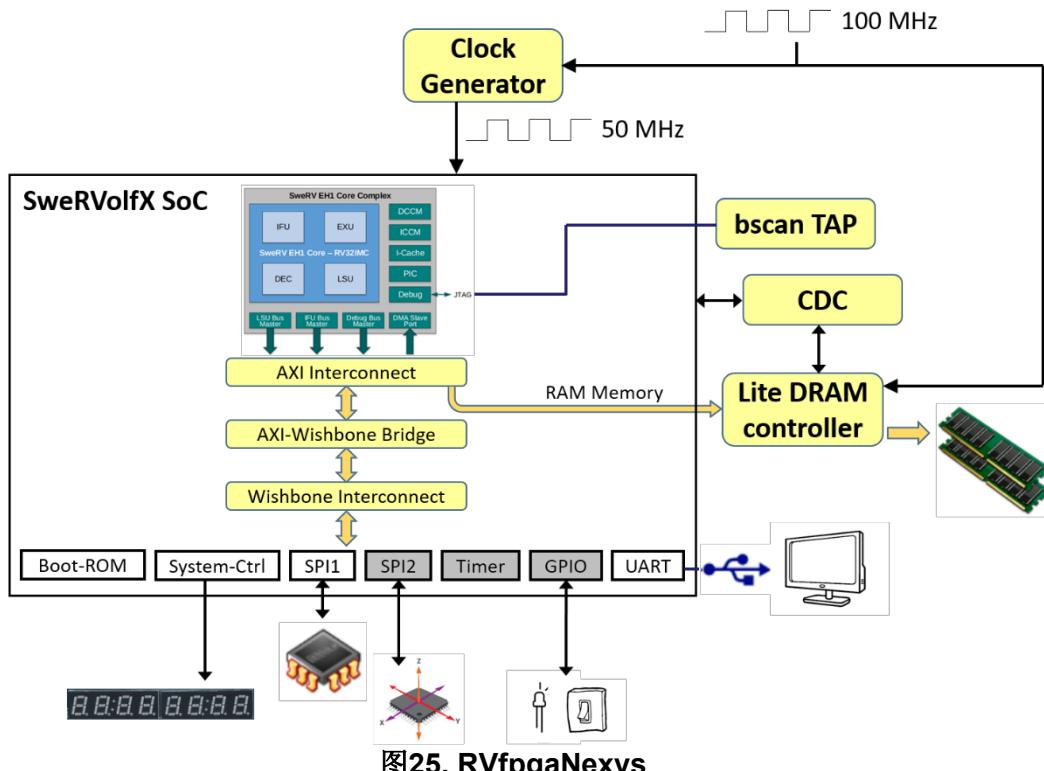
i. RVfpgaNexys

RVfpgaNexys是以Digilent Nexys A7 FPGA开发板为目标的SweRVolfX SoC (图25)。

RVfpgaNexys与SweRVolf Nexys基本相同 (<https://github.com/chipsalliance/Cores-SweRVolf>) , 只是后者基于SweRVolf。 **RVfpgaNexys**使用的主要元件如图25所示:

- 编程到FPGA中的硬件:
 - **SweRVolfX SoC** (如图21所示)
 - **Lite DRAM控制器**
 - **时钟发生器:** Nexys A7开发板包括一个由**Lite DRAM控制器**使用的**100 MHz**晶振。此时钟的频率按比例缩小至**50 MHz**, 以在**SweRVolfX SoC**中使用。
 - **跨时钟域模块:** 连接2个时钟域: SweRVolfX SoC和Lite DRAM。
 - **JTAG的BSCAN逻辑:** 有关此模块的更多信息, 请访问<https://github.com/chipsalliance/Cores-SweRVolf/issues/29>。

- Nexys A7（或Nexys4 DDR）FPGA开发板上的RVfpgaNexys中使用的存储器/外设：
 - **DDR2存储器**（通过上述Lite DRAM控制器访问）
 - **USB连接**
 - **SPI闪存**
 - **SPI加速计**
 - **16个LED和16个开关**
 - **8位7段显示屏**



建议使用Nexys A7开发板（图26）来开展电气与计算机工程课程的培训。此开发板的价格为265美元（学术优惠价为198.75美元 – 使用.edu电子邮箱地址注册一个Digilent帐户）。Digilent在以下位置提供有关Nexys A7开发板的丰富参考手册：

https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf。此开发板可以由5V壁式电源适配器（未随开发板提供）供电，也可以通过板上的microUSB连接器由PC供电。Microchip PIC24微处理器管理装载到FPGA的过程，因此该开发板非常简单易用。此开发板可以使用Xilinx的Vivado Design Suite或OpenOCD进行编程。可以使用下列四种不同的来源之一将所需配置下载到FPGA：FAT32格式的MicroSD卡、FAT32格式的U盘、内部闪存或JTAG接口。

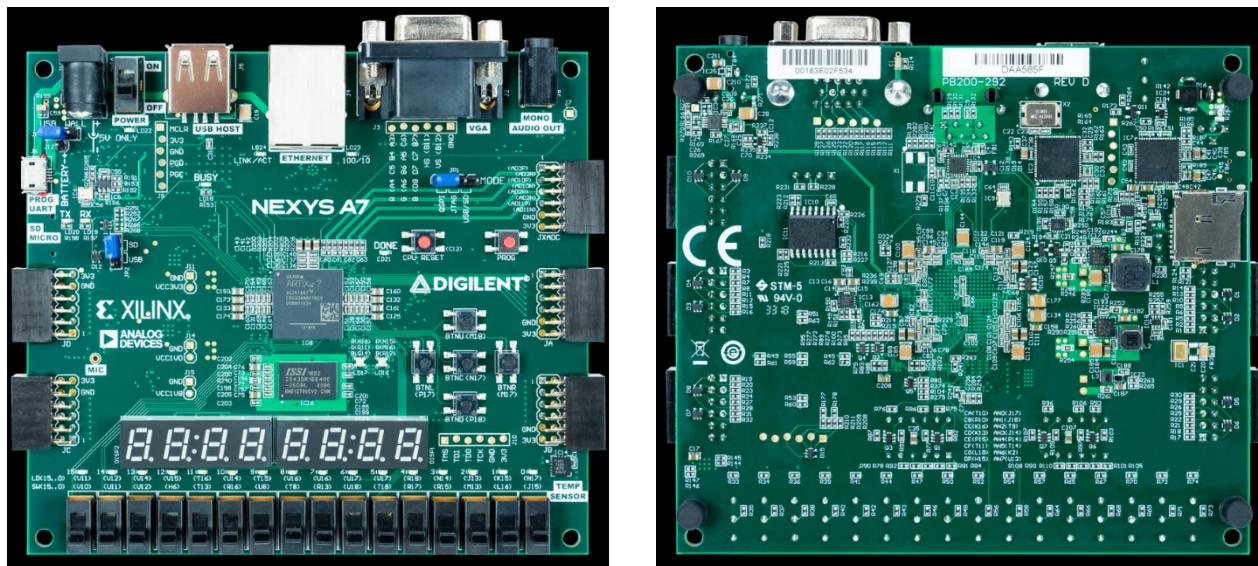


图26. Digilent的Nexys A7 FPGA开发板

(图来自<https://reference.digilentinc.com/>)

Nexys A7-100T FPGA开发板包括以下接口和设备：

- 128 MiB DDR RAM
- 128 Mbit SPI闪存
- 8位7段显示屏
- 16个开关
- 16个LED
- 传感器和连接器，包括麦克风、音频插孔、VGA 25端口、USB主机端口、RGB-LED、I2C温度传感器和SPI加速计等。
- Xilinx Artix-7 FPGA，具有以下功能：
 - 由四个6路输入LUT和8个触发器组成的15.850逻辑片。
 - 总块RAM大小为4.860 Kb
 - 6个时钟管理图块（Clock Management Tile, CMT）
 - 170个I/O引脚
 - 450 MHz内部时钟频率

ii. RVfpgaSim

SweRVolfX SoC（图21）还可以包括Verilog包装程序以使能仿真。RVfpgaSim是包装在HDL仿真器所用测试平台中的SweRVolfX SoC。RVfpgaSim与SweRVolf Sim基本相同（<https://github.com/chipsalliance/Cores-SweRVolf>），只是后者基于SweRVolf。

虽然存在许多开源HDL仿真器，但我们使用Verilator（<https://www.veripool.org/wiki/verilator>）这是一款开放且免费的HDL仿真器，接受可综合的Verilog或SystemVerilog，并且宣称是最快的Verilog/SystemVerilog仿真器。这款仿真器广泛应用于工业和学术领域；它提供现成的支持，只需提供ARM和RISC-V供应商IP；它以Chips Alliance和Linux Foundation为指导。

D. 文件结构

在前面的部分中，我们已经介绍了在这些资料中使用的系统高级组织，从SweRV EH1内核组合（图20）到SweRVolfX SoC（图21），最后到RVfpgaNexys（图25）和RVfpgaSim实现。

在本部分中，我们介绍整个系统的文件结构。阅读这些说明时，请打开文件并在PC上查看。这些文件位于 **[RVfpgaPath]/RVfpga/src** 下。

i. SweRV EH1内核组合

图27显示了**SweRV EH1**内核组合（图20）的文件结构。内核分为三个主要模块：**SweRV** 包装程序（以灰色突出显示），其中又包括**SweRV EH1**内核（以绿色突出显示）和其他元件（例如中断控制器或调试单元），以及数据/指令存储器和指令高速缓存（以红色突出显示）。

SweRV EH1 Core Complex (swerv_wrapper_dmi.sv)

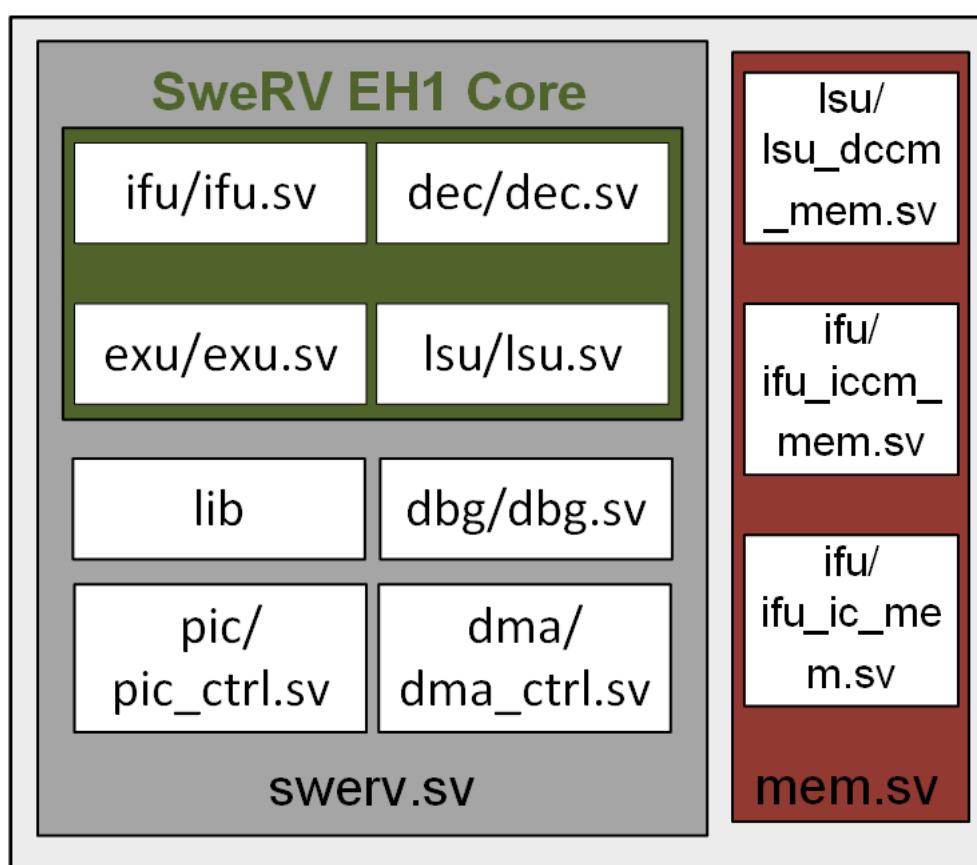


图27. SweRV EH1内核组合

SweRV EH1内核组合的Verilog文件位于以下文件夹中：

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex

在您的PC上找到该目录，以查看我们在本部分中引用的文件。

SweRV EH1内核组合的顶层文件位于以下文件中：`swerv_wrapper.sv`；顶层模块称为**swerv_wrapper**，它可实例化与图27中以灰色和红色突出显示的两个方框相对应的两个模块：

- **mem**（在 *mem.sv* 内实现）：该模块实例化用于实现 DCCM（**lsu_dccm_mem**，在 *lsu/lsu_dccm_mem.sv* 文件中实现）、ICCM（**ifu_iccm_mem**，在 *ifu/ifu_iccm_mem.sv* 文件中实现）和指令高速缓存（**ifu_ic_mem**，在 *ifu/ifu_ic_mem.sv* 文件中实现）模块。
- **swerv**（在 *swerv.sv* 内部实现）：该模块实例化组成内核的单元。

SweRV EH1 内核（在图27中以绿色突出显示）由以下四个单元组成：

 - 文件夹 **ifu**（取指单元）：该文件夹包含用于 **lcache**（指令高速缓存）、取指、分支预测器和对齐器的 Verilog 文件（*ifu.sv* 内部提供的顶层模块）。
 - 文件夹 **dec**（译码单元）：该文件夹包含用于指令译码、依赖性记分牌和寄存器文件的 Verilog 文件（*dec.sv* 内部提供的顶层模块）。
 - 文件夹 **exu**（执行单元）：该文件夹包含用于内核中算术/逻辑单元的 Verilog 文件（*exu.sv* 内部提供的顶层模块）：两个流水线式 ALU、一个流水线式乘法器和一个非流水线式除法器。
 - 文件夹 **lsu**（装载存储单元）：该文件夹包含用于流水线式装载/存储单元的 Verilog 文件（*lsu.sv* 内部提供的顶层模块）。

该模块中还包含其他单元，具体如下：

- 文件夹 **dbg**（调试单元）：该文件夹包含调试单元的 Verilog 文件（*dbg.sv* 内部提供的顶层模块），调试单元负责将内核的其余部分置于静止模式，发送命令/地址，发送写数据和接收读数据，然后恢复内核以进入正常模式。
- 文件夹 **lib**：该文件夹包含用于 AXI 和 AHB-Lite 总线的 Verilog 文件。
- 文件夹 **pic**：该文件夹包含文件 *pic_ctrl.sv*，用于在模块 **pic_ctrl** 中实现可编程中断控制器。
- 文件夹 **dma**：该文件夹包含文件 *dma_ctrl.sv*，用于在模块 **dma_ctrl** 中实现直接存储器访问。

ii. SweRVolfX SoC

图28显示图21所示 **SweRVolfX SoC** 的文件结构。SoC 被划分为多个模块，这些模块对应于图21中显示的方框。

SweRVolfX SoC (swervolf_core.v)

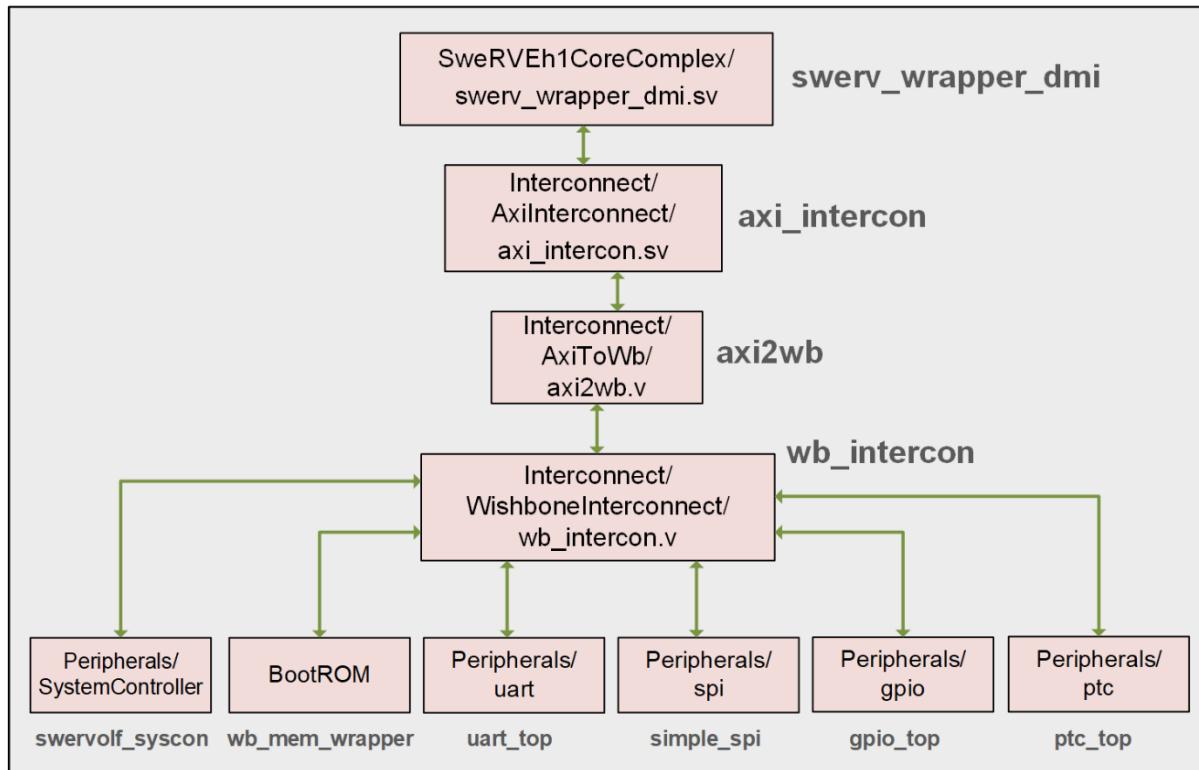


图28. SweRVolfX SoC

SweRVolfX SoC的文件位于:

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`

在您的PC上找到该目录，以查看我们在本部分中引用的文件。

SweRVolfX SoC的顶层模块位于:

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v`。打开该文件，注意其中包括SweRVolfX SoC（图21）内包含的模块，具体如下：

- **axi_intercon**（在`Interconnect/AxilInterconnect/axi_intercon.v`内部提供）：该模块通过另一个文件的第100行（`include "axi_intercon.vh"`) 包含在内。它将SweRV EH1内核组合与AXI-Wishbone桥相连。
- **axi2wb**（在`Interconnect/AxiToWb/axi2wb.v`内部提供）：该模块在`swervolf_core.v`的第153行中实例化，它是AXI-Wishbone桥，允许在基于AXI的EH1内核与基于Wishbone的外设之间进行通信。
- **wb_intercon**（在`Interconnect/WishboneInterconnect/wb_intercon.v`内部提供）：该模块通过另一个文件的第145行（`include "wb_intercon.vh"`) 包括在内。它通过一个多路复用器将AXI-Wishbone桥与不同的外设相连，我们稍后将对此多路复用器进行分析和修改。

- **wb_mem_wrapper**（在*BootROM/wb_mem_wrapper.v*内部提供）：上述引导存储器的包装程序在*swervolf_core.v*的第197行实例化。此外，它实例化了**dpram64**模块（在*BootROM/dpram64.v*内部提供），该模块是一个基本RAM模块。
- **swervolf_syscon**（在*Peripherals/SystemController/swervolf_syscon.v*内部提供）：该模块在*swervolf_core.v*的第215行实例化，用于定义系统控制器。
- **simple_spi**：从OpenCores获得的SPI控制器，在*Peripherals/spi/simple_spi_top.v*内部提供。它在*swervolf_core.v*的第246（SPI）和387（spi2）行实例化。
- **uart_top**：从OpenCores获得的UART控制器，在*Peripherals/uart/uart_top.v*内部提供。它在*swervolf_core.v*的第272行实例化。
- **gpio_top**：从OpenCores获得的GPIO控制器，在*Peripherals/gpio/gpio_top.v*内部提供。它在*swervolf_core.v*的第338行实例化。
- **ptc_top**：从OpenCores获得的PTC控制器，在*Peripherals/ptc/ptc_top.v*内部提供。它在*swervolf_core.v*的第361行实例化。
- **swerv_wrapper_dmi**（在*SweRVEh1CoreComplex/swerv_wrapper_dmi.v*内部提供）：Western Digital的SweRV EH1内核组合的实例化（*swervolf_core.v*的第407行），如前面的部分（图27）所述。

iii. 用于板上运行和仿真的包装程序

仿真：

RVfpgaSim是一个仿真目标，用于将**SweRVolfX SoC**（图21）包装在HDL仿真器使用的测试平台中。它位于`[RVfpgaPath]/RVfpga/src/rvfpgasim.v`中。

板上运行：

RVfpgaNexys（位于`[RVfpgaPath]/RVfpga/src/rvfganexys.v`中）将**SweRVolfX SoC**（图21）包装在以Nexys A7 FPGA开发板及其外设（参见图25）作为目标的包装程序中。该模块除了实例化一些其他模块（例如时钟发生器模块**clk_gen_nexys**、跨时钟域模块**axi_cdc_intf**或JTAG端口的BSCAN模块**bscan_tap**）外，还实例化两个主要SoC结构：

- **swervolf_core**：上一小节（图28）所述**SweRVolfX SoC**的实例化。此外，还需要一个称为*rvfganexys.xdc*的约束文件（位于`[RVfpgaPath]/RVfpga/src/`中），该文件定义SoC与开发板之间的连接。
- **litedram_top**：LiteDRAM DDR2控制器的包装程序，用于连接SweRVolfX SoC与DDR2存储器，此包装程序在文件`[RVfpgaPath]/RVfpga/src/LiteDRAM/litedram_top.v`中实现。此外，还需要一个称为*litedram.xdc*的约束文件（位于`[RVfpgaPath]/RVfpga/src/LiteDRAM`中），该文件定义存储器控制器与板上DDR2存储器之间的连接。

总结，图29显示了Nexys A7 FPGA开发板上整个RVfpgaNexys实现的层级。

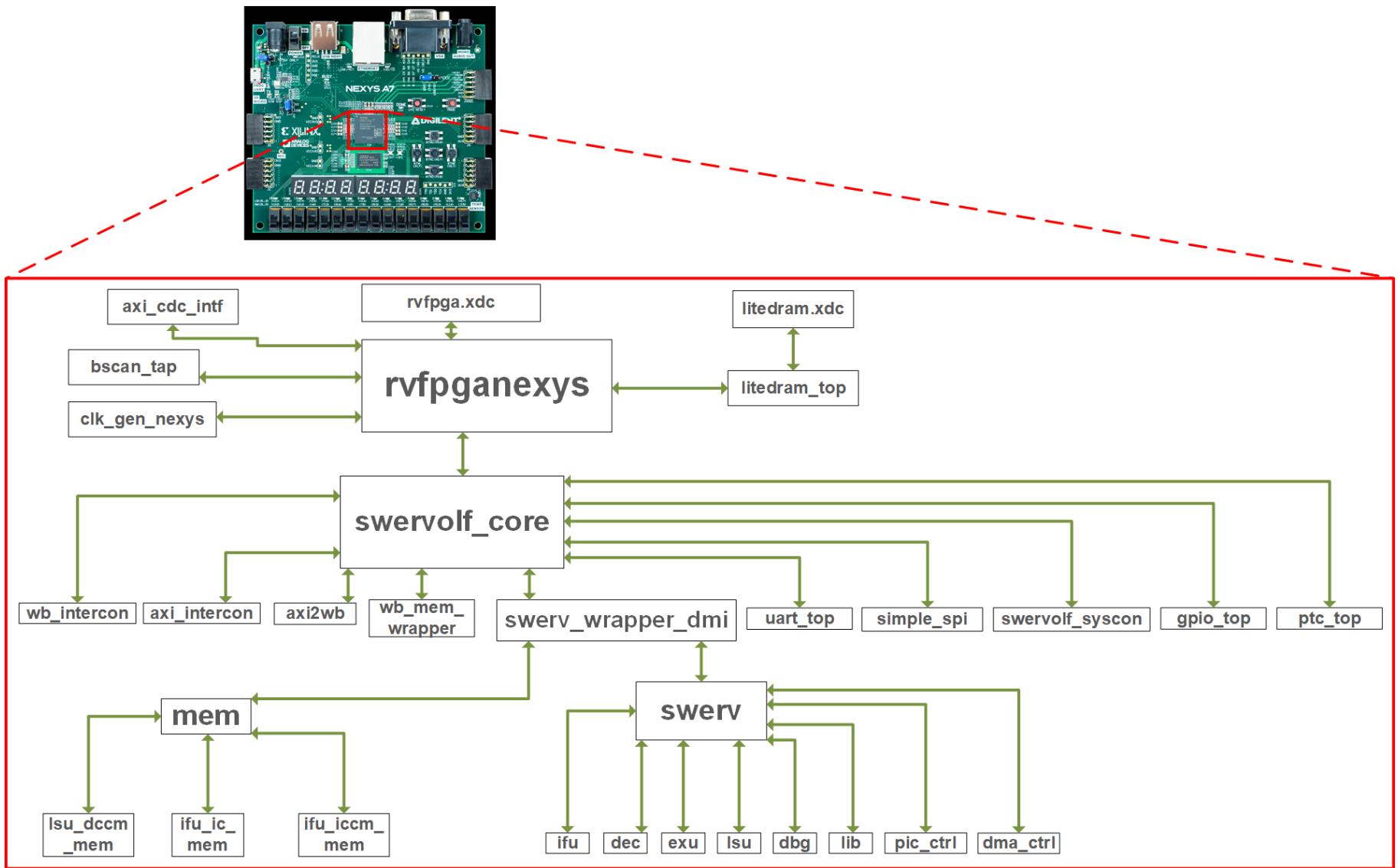


图29. Nexys A7 FPGA开发板实现的模块层级

5. 安装软件工具

以下说明适用于**Ubuntu 18.04 OS**，但其他Linux操作系统以及Windows或macOS遵循类似（但是不完全相同）的步骤。在某些情况下，我们会插入文本框来针对这些不同的操作系统提供具体说明。如果使用的是Ubuntu，则可以忽略这些文本框。

这些说明显示了如何安装以下工具：

- A. **Vivado**: 重新综合片上系统所需的工具。在实验6-20中，将主要完成此操作，并且不同的特性将包含在基线SoC中。
- B. **VSCode (Visual Studio Code)** 和 **PlatformIO**: 这些是GSG和实验中使用的主要工具。这些工具用于对FPGA进行编程以及在FPGA上运行/调试程序。
- C. **Verilator**和**GTKWave**: 仿真SoC和分析不同信号时所需的工具。同样，将主要在实验6-20中使用这些工具。

请注意，对于将在此GSG和实验中进行的大多数操作，安装VSCode和PlatformIO已经足够。但现在，我们建议用户同时安装其他工具（Vivado、Verilator和GTKWave），这样以后便无需再安装。

此过程可能需要几个小时（或更长时间，具体取决于下载速度），但大部分时间都花在了等待程序下载和安装上。

A. 安装Vivado

Vivado是一种Xilinx工具，用于查看、修改和综合RISC-V FPGA的Verilog代码。您将在以后的实验中广泛使用此工具。安装说明可访问<https://reference.digilentinc.com/vivado/installing-vivado/start>查看，具体内容总结如下。

Windows: 上面引用的网页 (<https://reference.digilentinc.com/vivado/installing-vivado/start>) 还包括在Windows中安装Vivado的详细说明。在下文中，我们将在Windows需要具体说明时插入文本框。

macOS: macOS不支持Vivado；因此，需要Linux/Windows虚拟机才能在此操作系统中运行Vivado。

1. 导航到<https://reference.digilentinc.com/vivado/installing-vivado/start>
2. 您将被引导至Xilinx下载页面：<https://www.xilinx.com/support/download.html>
3. 建议安装“Self Extracting Web Installer”。在撰写本文档时，可通过下载页面上的以下链接获取此程序：[Xilinx Unified Installer 2019.2: Linux Self Extracting Web Installer](https://www.xilinx.com/support/download.html#product=XILINX%20Unified%20Installer%202019.2&category=Software)

WINDOWS: 在编写本文档时，Windows版“自提取Web安装程序”的下载页面链接如下：
[Xilinx Unified Installer 2019.2: Windows Self Extracting Web Installer](https://www.xilinx.com/support/download.html#product=XILINX%20Unified%20Installer%202019.2&category=Software)

4. 在下载安装程序之前，系统将要求您登录到Xilinx帐户。如果您还没有帐户，则需要创建一个帐户。

5. 执行二进制文件。打开一个终端，将其设置为根（输入“`sudo su`”）。然后将二进制文件（`Xilinx_Unified_2019.2_1106_2127_Lin64.bin`）拖动到终端中。如果系统提示将文件设置为可执行并运行该文件，请选择“OK”（确定）。

- **故障处理：**如果终端显示权限被拒绝，则在终端中输入以下内容（与二进制文件位于同一目录中）：

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

WINDOWS: 在Windows中，只需双击执行在步骤3和4中下载的.exe文件。

6. Vivado安装程序将引导您完成安装过程。重要注意事项：

- 选择**Vivado**（非**Vitis**）作为要安装的产品。
- 选择**Vivado HL Webpack**（非**Vivado HL**系统版）；**Webpack**是免费的。
- 其他选项应为默认值。

提示：如果更改了Vivado的安装目录，将需要在以下步骤中适当地修改路径。

WINDOWS: 在Windows中不需要第7步和第8步。只需忽略这两个步骤，直接转到第9步即可。

7. Vivado安装完毕后，需要设置环境。打开一个终端并输入以下内容：

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

将上述内容（`source /tools/Xilinx/Vivado2019.2/settings64.sh`）添加到`~/.bashrc`文件，以便该文件在每次启动终端时运行。

8. 通过在终端中输入以下内容来测试Vivado：

```
vivado
```

故障处理：

- 如果系统找不到该可执行文件，则需要在路径中添加以下内容：

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

- 如果收到一条错误消息，例如“`application-specific initialization failed...`”（应用程序特定的初始化失败…），则在终端输入以下内容：

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

9. 将需要手动安装**Nexys A7 FPGA**开发板的驱动程序。在终端窗口中输入以下内容：

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

```
sudo ./install_drivers
```

WINDOWS: 在Windows中安装Vivado时会自动安装Nexys A7开发板的驱动程序，这些驱动程序与PlatformIO不兼容。因此，如果使用的是Windows，必须按照附录B中的说明更新驱动程序。务必执行此操作，即使已经在“快速入门指南”部分中完成了此操作时也如此，因为驱动程序已被Vivado安装覆盖。

10. 此外，还需要手动安装Digilent Board Files。

- 从Github存储库下载Vivado开发板的[存档](#)并解压缩文件。
- 打开从存档中解压的文件夹，导航至其*new/board_files*目录。选择并复制此目录中的所有文件夹。
- 打开安装Vivado的文件夹（默认为*/tools/Xilinx/Vivado*）。在此文件夹下，导航至*<version>/data/boards/board_files*目录，然后将开发板文件粘贴到该目录中。
- 此外，也可以使用终端，方式是转到*new/board_files*目录并输入以下内容：

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

WINDOWS: 按照第10步中的说明复制/粘贴下载的文件夹。在Windows中，可以在下列位置找到Vivado的*board_files*文件夹：C:\Xilinx\Vivado\2019.2\data\boards\board_files

B. 安装VSCode和PlatformIO

现在，将安装VSCode和PlatformIO。如果已经在第1部分“快速入门指南”中完成此操作，则无需再次重复此过程，可以直接转到C部分。

PlatformIO是一种面向嵌入式系统的集成开发环境（IDE），它在Microsoft的Visual Studio（VS）Code基础上进行编译。借助此开发环境，用户可以使用C语言或汇编语言对RISC-V处理器（位于FPGA上）进行编程。PlatformIO支持跨平台操作，包含一个内置调试器。

按照以下步骤安装VSCode和PlatformIO：

LINUX命令行：尽管建议使用VSCode+PlatformIO，但附录A为有兴趣在Linux中安装和使用本机RISC-V工具链和OpenOCD来代替PlatformIO的用户提供了解说。如果要使用PlatformIO，可直接忽略附录A。

1. 安装VSCode：

按照以下步骤安装VSCode：

- 从以下链接下载.deb文件：<https://code.visualstudio.com/Download>
- 打开一个终端，然后在此终端中输入以下内容来安装并执行VSCode：
cd ~/Downloads

```
sudo dpkg -i code*.deb
code
```

Windows/macOS: VSCode软件包也支持Windows (.exe文件) 和macOS (.zip文件)，可通过<https://code.visualstudio.com/Download>获取。请遵循在这些操作系统中安装和执行应用程序的通用步骤。

2. 在VSCode软件上安装PlatformIO:

按照以下步骤安装PlatformIO:

- 通过在终端中输入以下内容来安装python3实用程序:

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

Windows/macOS: Windows 中不需要执行这一步 (2.a)。对于macOS，可以使用homebrew来安装python3: brew install python3

- 如果VSCode尚未打开，请通过以下方式将其启动：选择“Start”（开始）按钮并在搜索菜单中输入“VSCode”，然后选择VSCode；或者在终端中输入code。

- 在VSCode中，单击VSCode左侧栏中的“Extensions”（扩展）图标  (参见图30)。



图30. VSCode的“Extensions”（扩展）图标

- 在搜索框中输入PlatformIO，然后单击PlatformIO IDE旁边的“Install”（安装）按钮进行安装（参见图31）。

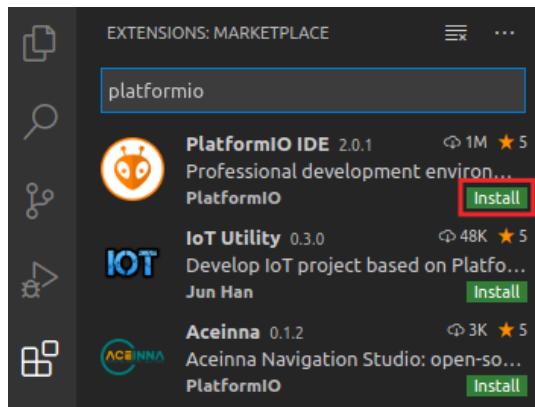


图31. PlatformIO IDE扩展

- e. 底部的“OUTPUT”（输出）窗口将通知用户有关安装进程的信息。安装完成后，单击窗口右下角的“Reload Now”（立即重新载入），PlatformIO便会安装在VSCode内部（参见图32）。

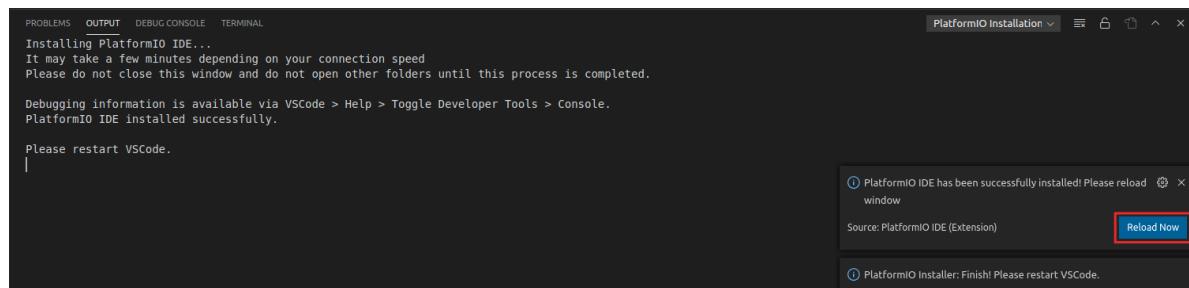


图32. 安装PlatformIO后立即重新载入

C. 在Ubuntu 18.04中安装Verilator和GTKWave

本节中的说明仅适用于Linux系统。

Windows: 使用附录C代替本部分中提供的说明。

macOS: 使用附录D代替本部分中提供的说明。

按照以下步骤将Verilator（安装说明可访问

<https://www.veripool.org/projects/verilator/wiki/Installing>查看，具体内容总结如下）和 GTKWave安装在Ubuntu 18.04 Linux系统中。此过程需要很长时间。

```
> sudo apt-get install git make autoconf g++ flex bison libfl12 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) 来加快运行速度)
> sudo make install
```

➤ `export PATH=$PATH:/usr/local/bin` (在系统中更改路径)

要将`/usr/local/bin`永久添加到路径中，将最后一行添加到`~/.bashrc`文件。

6. 运行RVfpgaNexys并对其编程

在本部分中，我们将介绍如何在RVfpgaNexys（参见图25）上运行七个简单程序。

LINUX/Windows/macOS: 假设所有必需的工具和驱动程序均已按照第5部分所述正确安装，则本部分中介绍的所有说明均应适用于这三个操作系统。在某些情况下，可能需要修改一些小细节，例如Linux中使用的斜杠，Windows中使用的反斜杠。

我们通过展示如何运行表9中列出的七个示例程序来说明如何使用RVfpgaNexys。前三个程序是用RISC-V汇编语言编写的，后四个程序是用C语言编写的。下面给出了在RVfpgaNexys上运行每个程序的说明。

表9. RVfpgaNexys示例程序

程序名称	说明	语言
AL_Operations	练习算术和逻辑运算	RISC-V汇编语言
Blinky	使Nexys A7开发板上的LED闪烁	RISC-V汇编语言
LedsSwitches	读取Nexys A7开发板上的开关值，并将该值写入LED	RISC-V汇编语言
LedsSwitches_C-Lang	读取Nexys A7开发板上的开关值，并将该值写入LED	C
HelloWorld_C-Lang	通过串行端口向shell打印一条短消息	C
VectorSorting_C-Lang	从大到小对向量进行排序	C
DotProduct_C-Lang	计算两个向量的点积	C

请注意，在能够执行这七个示例中的任何一个示例之前，必须使用RVfpgaNexys对FPGA进行编程，如以下部分所述。

A. 用RVfpgaNexys对FPGA进行编程

在本部分中，我们介绍一种使用PlatformIO的建议方法，这种方法使用RVfpgaNexys对FPGA进行编程。按照以下步骤使用RVfpgaNexys对FPGA编程：

（如果您有兴趣使用Vivado对FPGA进行编程，则可以遵循本指南的附录E中提供的说明，而不必遵循以下说明。但是，此处所述的方法仅适用于Linux和Windows系统（不适用于macOS），因此，总体而言不建议使用Vivado将RVfpgaNexys下载到FPGA上。相反，建议您遵循以下说明，而忽略附录E。）

- 将Nexys A7开发板连接到计算机。
- 打开左上方的开关接通Nexys A7开发板。
- 如果VSCode和PlatformIO尚未打开，请将其打开。
- 在顶部菜单栏上，单击“File”（文件）→“Open Folder”（打开文件夹）（参见图33），然后导航至目录[RVfpgaPath]/RVfpga/examples/

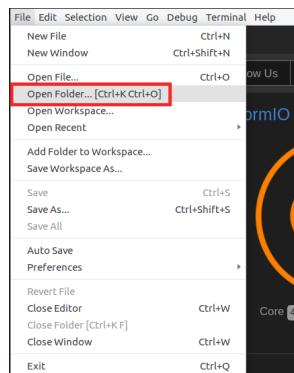


图33. 打开文件夹

- 选择要使用的PlatformIO项目。在本部分中，我们以表9中提到的第一个示例AL_Operations为例，将在下一部分中对此示例进行调试，对任何其他示例可执行相同的步骤。为此，选择目录AL_Operations（不要打开，只需将其选中 – 参见图34），然后单击窗口顶部的“OK”（确定）。PlatformIO现在将打开AL_Operations示例。

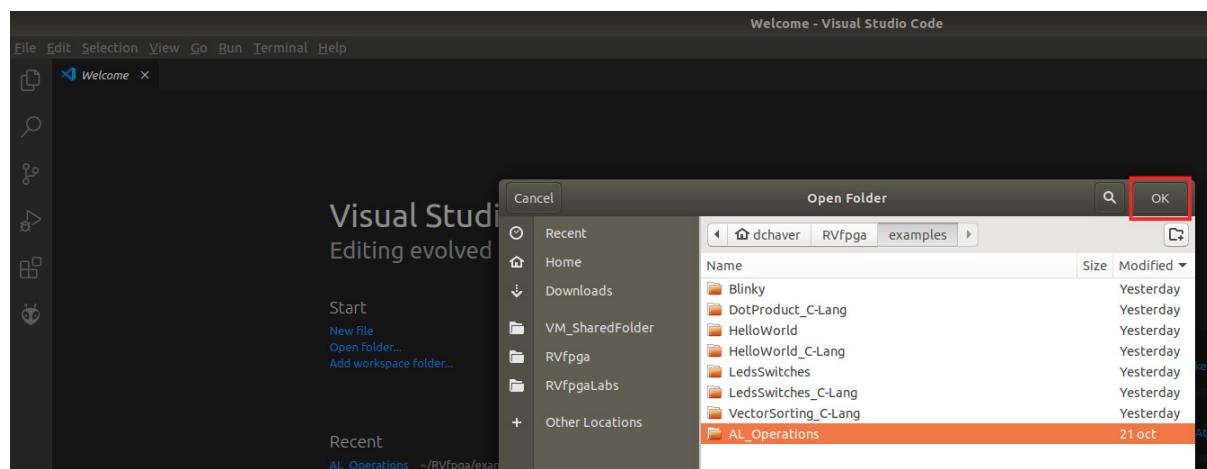
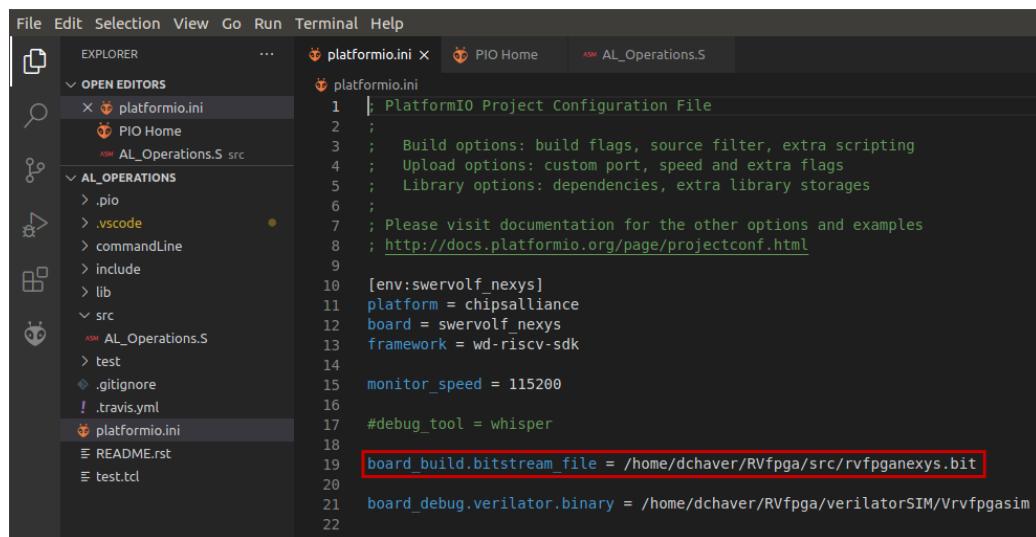


图34. 打开AL_Operations文件夹

- 单击左侧工具栏中的platformio.ini打开文件platformio.ini（参见图35）。通过编辑以下行在系统中建立RVfpgaNexys比特流的路径（参见图35）。请注意，预综合的RVfpgaNexys比特流在RVfpga文件夹中提供，此文件夹位于：[RVfpgaPath]/RVfpga/src/rvfganexys.bit.

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



```

platformio.ini x PIO Home AL_Operations.S
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter, extra scripting
4 ; Upload options: custom port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ;
7 ; Please visit documentation for the other options and examples
8 ; http://docs.platformio.org/page/projectconf.html
9
10 [env:swervolf_nexys]
11 platform = chipsalliance
12 board = swervolf_nexys
13 framework = wd-riscv-sdk
14
15 monitor_speed = 115200
16
17 #debug_tool = whisper
18
19 board_build.bitstream_file = /home/dchaver/RVfpga/src/rvfpaganexys.bit
20
21 board_debug.verilator.binary = /home/dchaver/RVfpga/verilatorSIM/Vrvfpgasim
22

```

图35. PlatformIO初始化文件：platformio.ini

可以在项目配置文件（*platformio.ini*）中使用许多不同的命令，相关信息请参见 <https://docs.platformio.org/en/latest/projectconf/>。

- g. 单击左侧功能区菜单中的PlatformIO图标 （参见图36）。



图36. PlatformIO图标

如果“Project Tasks”（项目任务）窗口为空（图37），必须先通过单击  来刷新“Project Tasks”（项目任务）。这可能需要几分钟。

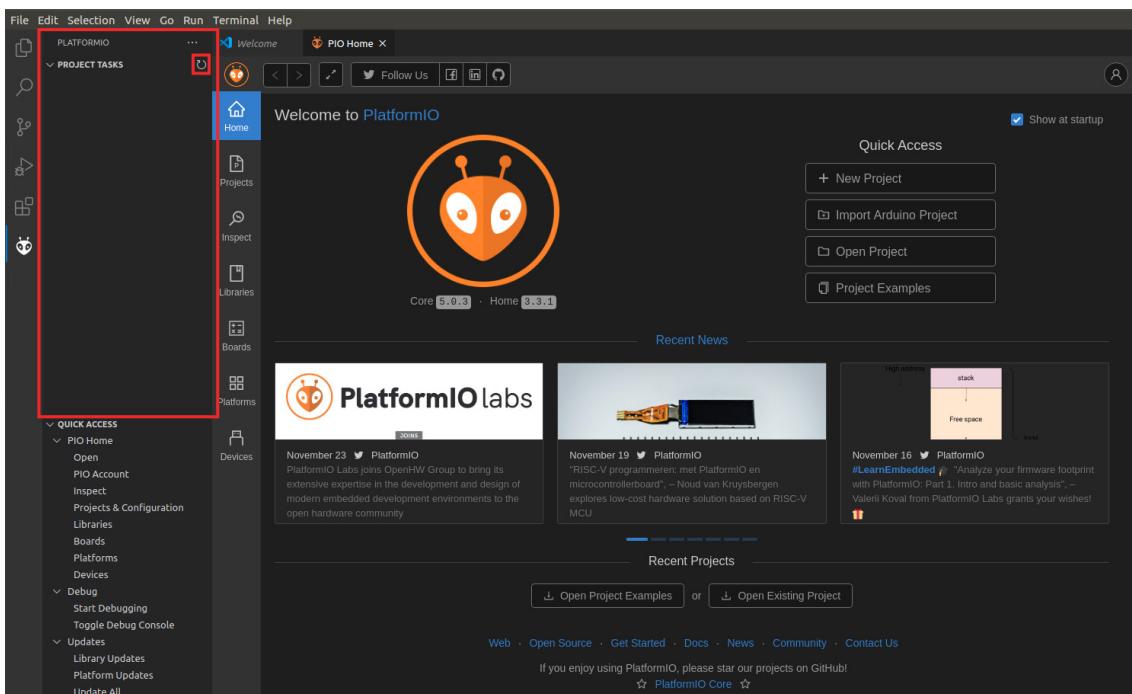


图37. “PROJECT TASKS”（项目任务）窗口为空 – 刷新

然后展开“Project Tasks”（项目任务）→ env:swervolf_nexys → “Platform”（平台），并单击“Upload Bitstream”（上传比特流），如图38所示。一到两秒后，将使用 RVfpgaNexys 对 FPGA 进行编程。

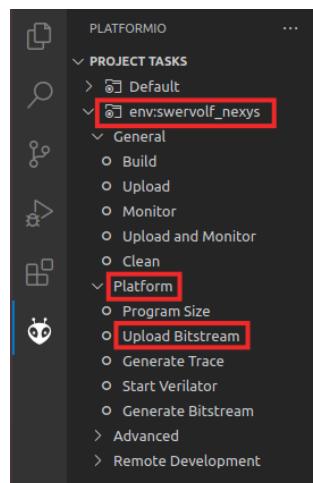


图38. 上传比特流

默认情况下，处理器从地址 0x80000000 处开始取指，其中引导 ROM 位于我们的 SoC 中（参见表6）。引导 ROM 使用一个程序（boot_main.mem）进行初始化，该程序使 LED 和 7 段显示屏闪烁四次，然后熄灭所有 LED，将 0 写入 8 个 7 段显示屏并保持空循环。该程序位于以下文件夹：[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw。

按下 Nexys A7 开发板（图26）上的 CPU 复位按钮，使该程序再次执行。

如果要对该程序进行更改和重新编译，请按照附录A – 第III部分中的说明进行操作（注意文件*boot_main.mem*只是文件*boot_main.vhd*的副本）。在实验1中，我们将介绍如何在创建比特流时使用该程序初始化引导ROM。

- h. 也可以通过PlatformIO终端窗口下载RVfpgaNexys（如图39所示）来替代上一步（步骤g）。单击按钮（“PlatformIO: New Terminal”（PlatformIO: 新建终端）按钮，该按钮位于PlatformIO窗口的底部）打开一个新的终端窗口，然后在PlatformIO终端中输入（或复制）以下命令：

```
pio run -t program fpga
```

The screenshot shows the PlatformIO IDE interface with the following details:

- Project Structure:** Default, swervolt_nexys (selected), General, Build, Upload, Monitor, Upload and Monitor, Clean, Platform (selected), Program Size, Upload Bitstream, Generate Trace, Start Verilator, Generate Bistream, Advanced, Remote Development.
- Terminal Output:**

```
dchaver@dchaver-PORTGE-Z3B-B:~/RVfpga/examples/AL_Operations$pio run -t program fpga
Processing swervolt_nexys [platform: chipsalliance; board: swervolt_nexys; framework: wd-riscv-sdk]
=====
Verbose mode can be enabled via '-v, --verbose' option
CONFIGURATION: https://docs.platformio.org/page/boards/chipsalliance/swervolt_nexys.html
PLATFORM: CHIPS Alliance (1.0.3) > RVfpga: digilent Nexys A7
HARDWARE: 320MHz, 1.14MB RAM, 16MB Flash
DEBUG: Current (digilent-hsi) On-board (digilent-hsi, verilator, whisper) External (olimex-arm-usb-ooc, olimex-arm-usb-ooc-h, olimex-arm-usb-tiny-h, olimex-jtag-tiny)
PACKAGES:
  - framework-wd-riscv-sdk 0.0.0+alpha+sha.ca4b2392d8
  - tool-openocd-riscv-chipsalliance 1.1000.200713 (10.0)
  - tool-verilator 0.18.0+sha.00092c649 (3.0)
  - tool-wcharch-riscv 1.0.0+sha.00092c649 (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/RVfpgas/src/rvfganexys.bit
Open On-Chip Debugger 0.10.0+dev-01227-gb0929c649 (2020-07-13-17:03)
Licensed under GNU GPL V2
For bug reports: read http://openocd.org/doc/doxygen/bugs.html
/home/dchaver/RVfpgas/src/rvfganexys.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 : clock speed 100000000
Info : JTAG tap: xcf7l.tap/device found: 0x13631093 (mfg: 0x0649 (Xilinx), part: 0x3631, ver: 0x1)
Warn : JTAG services need one or more targets defined
=====
->Programming /home/dchaver/RVfpgas/src/rvfganexys.bit
>DONE!!!!!!
=====
shutdown command invoked
=====
=====
```
- Terminal Status:** [SUCCESS] Took 3.72 seconds

图39. 使用PlatformIO将RVfpgaNexys上传到Nexys A7 FPGA开发板上

请注意，首次在PlatformIO中打开示例时，Chips Alliance平台会自动安装（可以在“PIO Home”（PIO主页）中查看该平台，如图40所示）。该平台包含以后将用到的几个工具，例如预编译的RISC-V工具链、用于RISC-V的OpenOCD、RVfpgaNexys bit文件和RVfpgaSim、JavaScript和Python脚本以及一些示例。

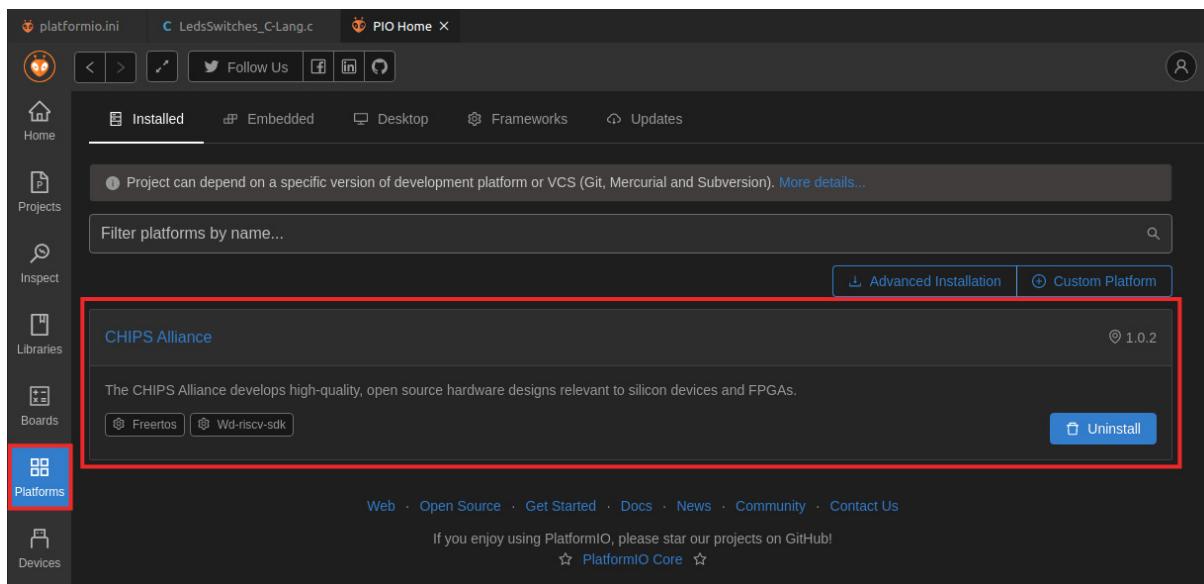


图40. PlatformIO中安装的Chips Alliance平台

如果由于某种原因未自动安装Chips Alliance平台，则可以按照下列步骤手动安装（通常，只需跳过此过程，继续B部分）：

- 通过单击左侧栏上的  按钮查看“Quick Access”（快速访问）菜单（参见图41）。
然后，在“PIO Home”（PIO主页）中，依次单击  按钮和  选项卡（图41）。找到**Chipsalliance**（我们在RVfpga中使用的平台），然后单击  按钮将其打开（图41）。

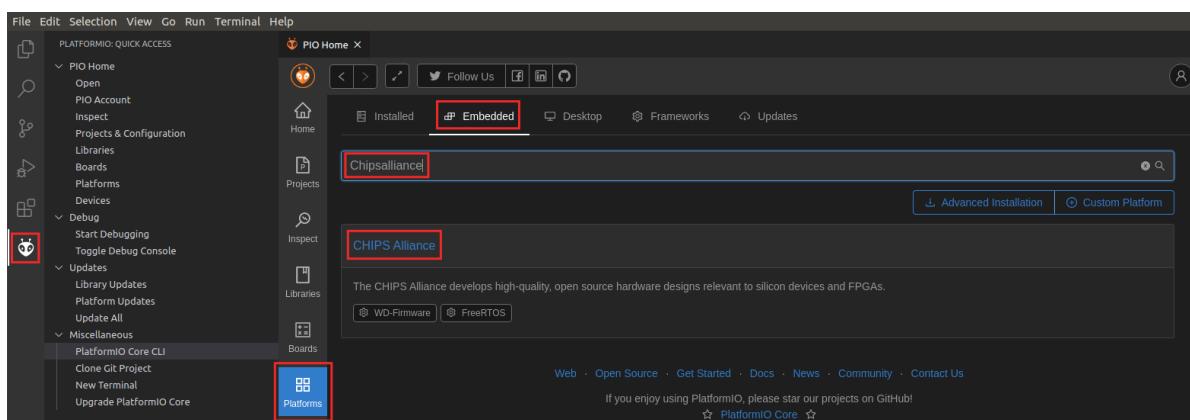


图41. 选择CHIPS Alliance平台

- 单击  按钮后，将看到有关Chips Alliance平台的详细信息（如图42所示）。单击  按钮进行安装（图42）。

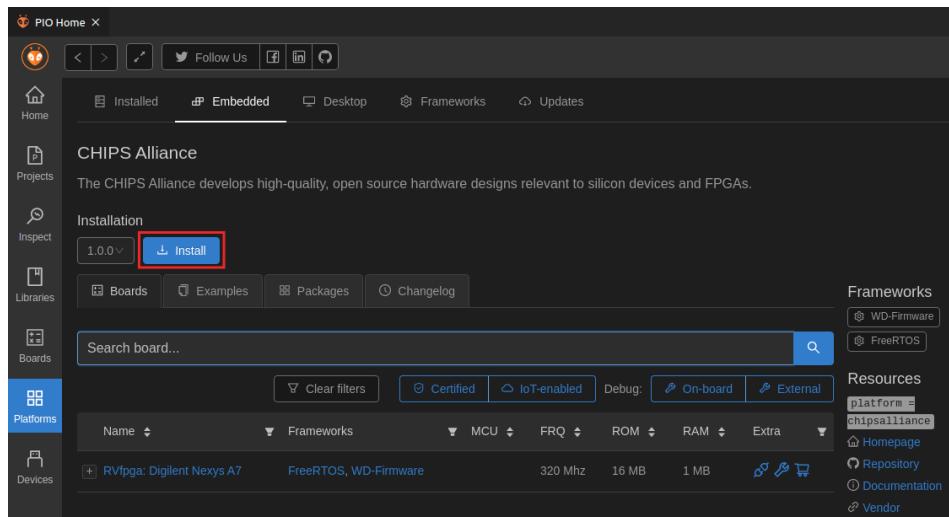


图42. 安装CHIPS Alliance平台

- 安装完成后，将显示已安装工具的汇总信息，如图43所示。单击  以关闭相应窗口。

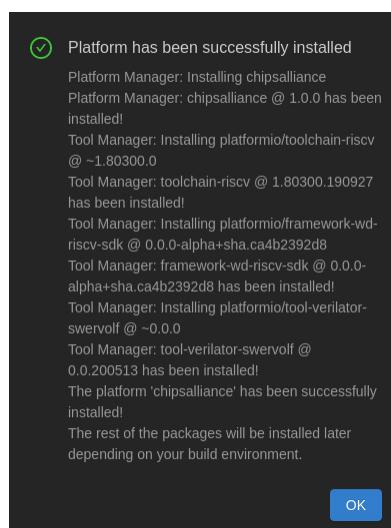


图43. CHIPS Alliance平台安装成功

B. AL_Operations程序

第一个示例程序AL_Operations.s（参见图44）是一个汇编程序，它在一个无限循环内对同一寄存器t3（也称作x28）执行三条算术逻辑指令（加、减和逻辑与）。

```

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

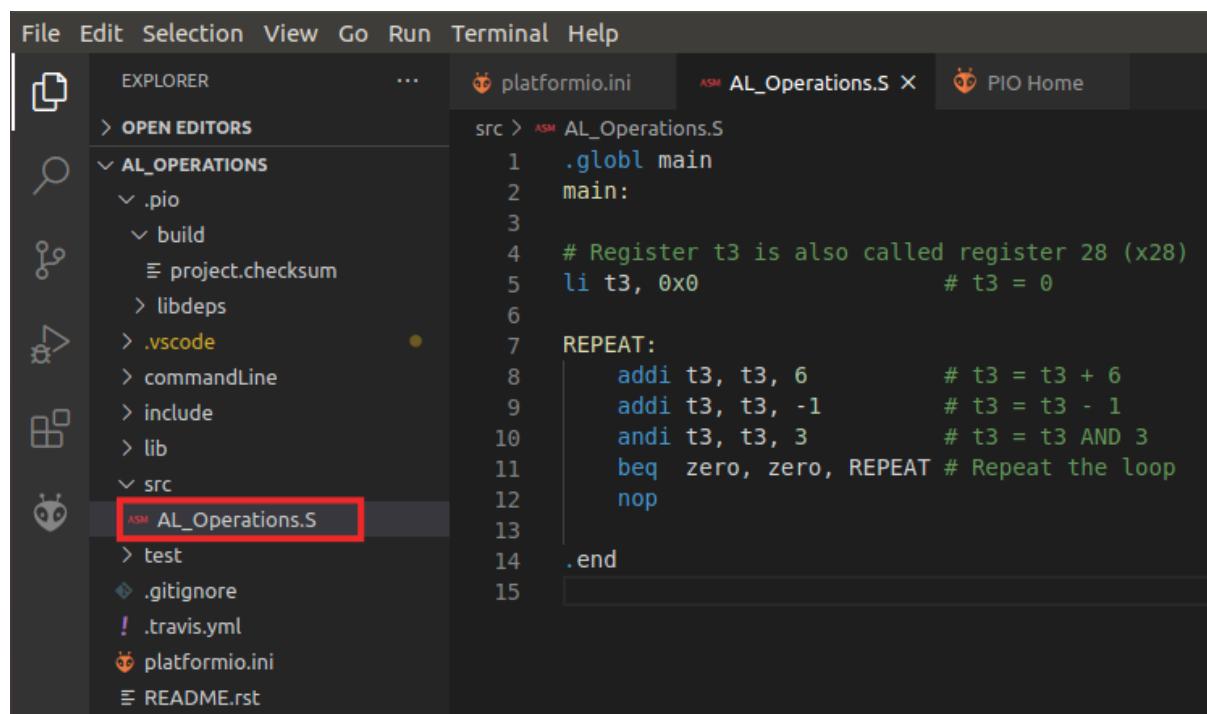
```

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

图44. AL_Operations程序：AL_Operations.S

按照以下步骤使用PlatformIO在Nexys A7 FPGA开发板上运行和调试此代码：

1. 按上一节中所述对FPGA进行编程。请注意，*AL_Operations*项目已在PlatformIO中打开。
2. 通过以下方式打开汇编程序*AL_Operations.S*：单击左侧功能区菜单中的“Explorer”（资源管理器）图标 ，展开左侧栏中*AL_OPERATIONS*下的*src*，然后单击*AL_Operations.S*（参见图45）。



The screenshot shows the PlatformIO IDE interface. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar has an 'EXPLORER' section with a tree view. Under 'AL_OPERATIONS', the 'src' folder is expanded, and 'AL_Operations.S' is selected and highlighted with a red box. Other items in 'src' include '.pio', 'build', 'project.checksum', 'libdeps', '.vscode', 'commandLine', 'include', 'lib', and 'test'. The main editor area displays the assembly code:

```
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
15
```

图45. 查看汇编文件AL_Operations.S

3. VSCode和PlatformIO提供了多种编译、清除和调试程序的方法。在VSCode的底部，可以找到一些提供实用功能的按钮：。例如，可用于编译项目，或可用于清除项目。在左侧栏（参见图30）中，“Run”（运行）按钮  可用于编译程序，然后打开调试器。

4. 单击“Run”（运行）按钮 。单击播放按钮  启动调试器（确保“PIO Debug”（PIO调试）选项已选中）。播放按钮靠近窗口顶部位置（参见图46）。程序将先编译，然后开始调试。PlatformIO在main函数的开头设置了一个临时断点，因此执行将在此处停止。

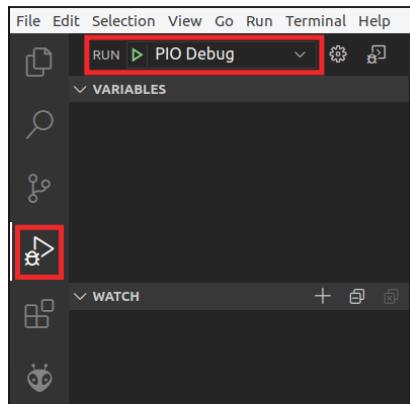


图46. 启动调试器

5. 要控制调试会话，可以使用靠近编辑器顶部位置的调试工具栏（参见图47）。选项如下：
- **Continue**（继续）：执行程序，直至达到下一个断点。
 - **Breakpoints**（断点）：可通过在编辑器中单击行号左侧区域进行添加。
 - **Step Over**（单步跳过）：执行当前行，然后停止。
 - **Step Into**（单步进入）：执行当前行，如果当前行包含函数调用，将跳转到该函数并停止。
 - **Step Out**（单步跳出）：执行所在函数中的所有代码，然后在该函数返回结果时停止。
 - **Restart**（重启）：从程序的开头重新启动调试会话。
 - **Stop**（停止）：停止调试会话并返回正常编辑模式。
 - **Pause**（暂停）：暂停执行。程序运行时，“Continue”（继续）按钮将替换为“Pause”（暂停）按钮。

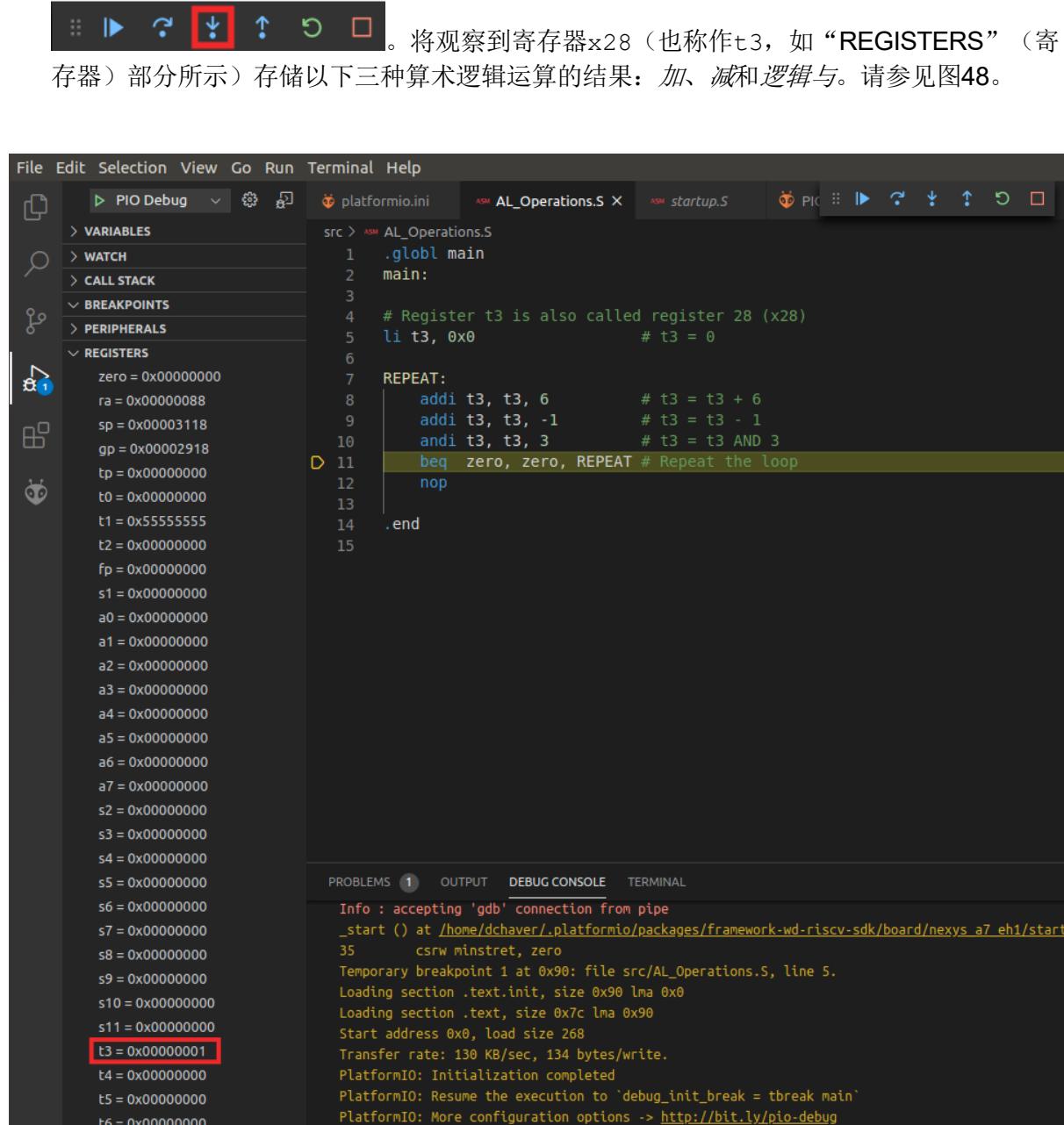


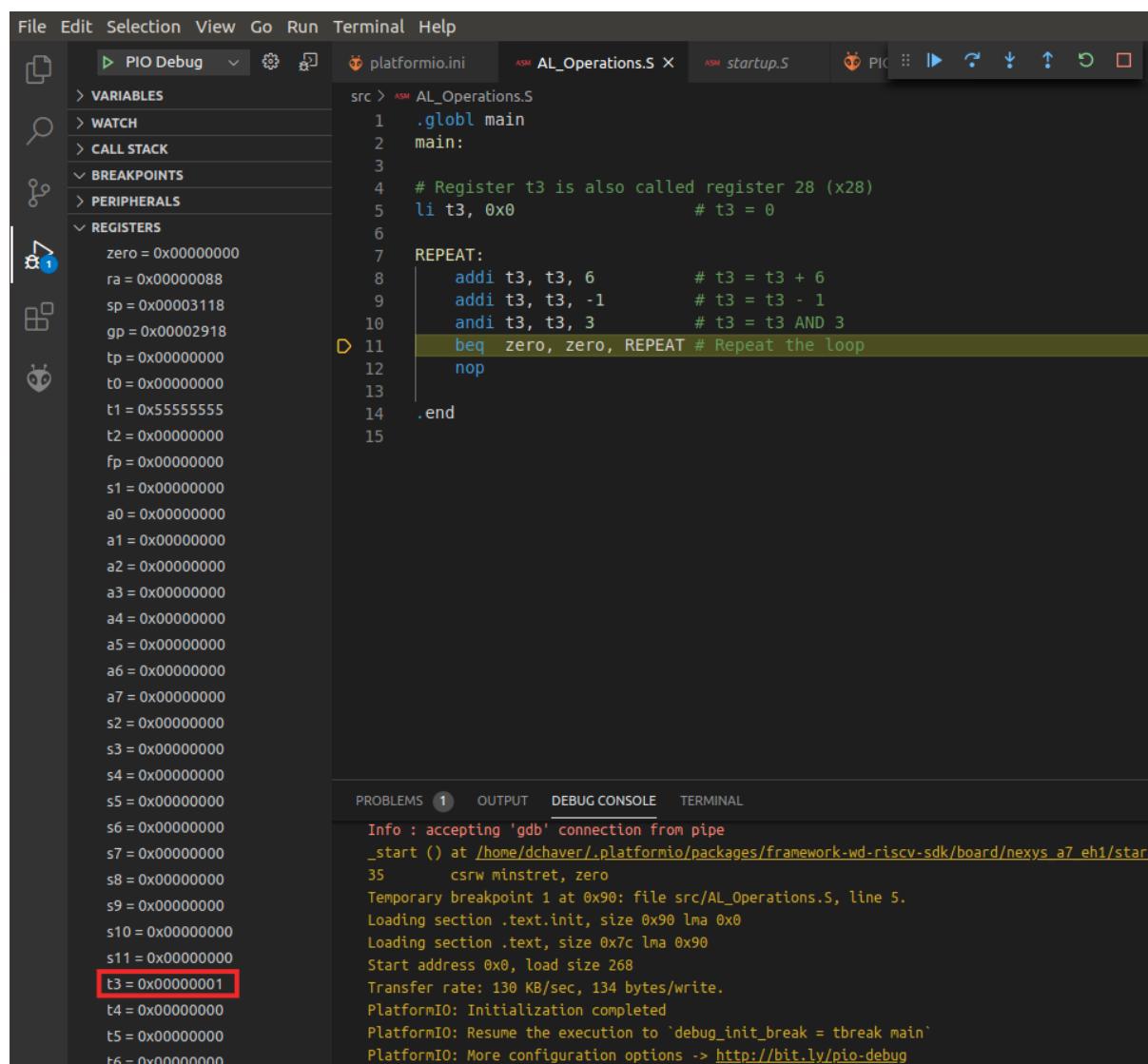
图47. 调试工具

6. 在左侧工具栏中可以查看调试器选项。提供以下选项：
- **Variables**（变量）：列出程序中存在的局部、全局和静态变量及其变量值。
 - **Call Stack**（调用堆栈）：显示当前正在运行的函数、调用函数（如果存在）以及当前指令在存储器中的位置。
 - “**Breakpoints**”（断点）：显示所有设置的断点并突出显示其行号。可在此部分中管理断点。也可以暂时停用断点，而无需通过切换复选框将其删除。

- **Peripherals**（外设）：显示设备的存储器映射外设的寄存器状态（我们将在RVfpga实验中对此进行更详细的说明）。
- **Registers**（寄存器）：列出存在于处理器的每个寄存器中的当前值。
- **Memory**（存储器）：显示特定存储器地址的内容。
- **Disassembly**（反汇编）：显示特定函数的汇编代码，对于C语言等较高级别的代码，此选项支持查看汇编代码以逐行调试指令。

7. 展开调试器侧边栏中的“Registers”（寄存器）选项，然后继续逐步执行

。将观察到寄存器x28（也称作t3，如“REGISTERS”（寄存器）部分所示）存储以下三种算术逻辑运算的结果：加、减和逻辑与。请参见图48。



The screenshot shows the PlatformIO IDE interface. The left sidebar has a tree view with sections like VARIABLES, WATCH, CALL STACK, BREAKPOINTS, PERIPHERALS, and REGISTERS. Under REGISTERS, there is a list of registers with their current values: zero = 0x00000000, ra = 0x00000088, sp = 0x00003118, gp = 0x00002918, tp = 0x00000000, t0 = 0x00000000, t1 = 0x55555555, t2 = 0x00000000, fp = 0x00000000, s1 = 0x00000000, a0 = 0x00000000, a1 = 0x00000000, a2 = 0x00000000, a3 = 0x00000000, a4 = 0x00000000, a5 = 0x00000000, a6 = 0x00000000, a7 = 0x00000000, s2 = 0x00000000, s3 = 0x00000000, s4 = 0x00000000, s5 = 0x00000000, s6 = 0x00000000, s7 = 0x00000000, s8 = 0x00000000, s9 = 0x00000000, s10 = 0x00000000, s11 = 0x00000000, t3 = 0x00000001 (highlighted with a red box). The main window displays assembly code for the file AL_Operations.S:

```

src > ASM AL_Operations.S
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
15

```

The bottom right corner shows the DEBUG CONSOLE tab with the following output:

```

Info : accepting 'gdb' connection from pipe
_start () at /home/dchaver/.platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/start
35      csrw minstret, zero
Temporary breakpoint 1 at 0x90: file src/AL_Operations.S, line 5.
Loading section .text.init, size 0x90 lma 0x0
Loading section .text, size 0x7c lma 0x90
Start address 0x0, load size 268
Transfer rate: 130 KB/sec, 134 bytes/write.
PlatformIO: Initialization completed
PlatformIO: Resume the execution to `debug_init_break = tbreak main'
PlatformIO: More configuration options -> http://bit.ly/pio-debug

```

图48. 查看寄存器内容

8. 在调用 *main* 函数之前，将执行 Western Digital 提供的启动文件，该文件位于 `~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S`。该文件可配置内核：指令高速缓存设置、寄存器初始化（例如 *SP* 或 *gp*）等。启动调试后，该文件将在主窗口中打开（参见图48），用户可在其中检查该文件。

Windows: `.platformio` 文件夹位于用户文件夹（`C:\Users\<USER>`）内。请注意，可能需要启用系统管理才能查看隐藏的文件/文件夹。

macOS: 与在 Linux 中一样，`.platformio` 文件夹位于主文件夹（`~/platformio`）内。

9. 我们还应强调一点，在同一目录（`~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/`）中，提供了文件 `link.lds`，该文件构成我们将在所有项目中使用的链接器脚本。该文件确定汇编部分（文本、数据、*bss*…）在存储器中的位置。

10. 最后，停止调试  (这将使引导 ROM 程序再次执行)，并通过单击最左侧栏顶部的  返回到“Explorer”（资源管理器）窗口。在顶部菜单栏上，单击“File”（文件）→“Close Folder”（关闭文件夹）。

C. Blinky 程序

第二个示例程序 `blinky.S` 是一个汇编程序，可使 Nexys A7 开发板最右侧的 LED 闪烁（参见图49）。该程序会重复地对与最右侧 LED 相关联的值取反，并且每次取反都有延迟。

```

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    xorl t0, t0, 1          # invert LED
21    andl t2, zero, zero    # Reset timer
22
23 timel:                         # Delay loop
24    addi t2, t2, 1
25    bne t1, t2, timel
26    j b11

```

图49. `blinky.S`

按照以下步骤在 RVfpgaNexys（已加载到 FPGA 开发板上的 RISC-V SoC）上运行和调试此代码：

1. 如果执行了第一个示例（*AL_Operations*），则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用的是Blinky示例而不是AL_Operations示例。
2. 在顶部栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录[RVfpgaPath]/RVfpga/examples/

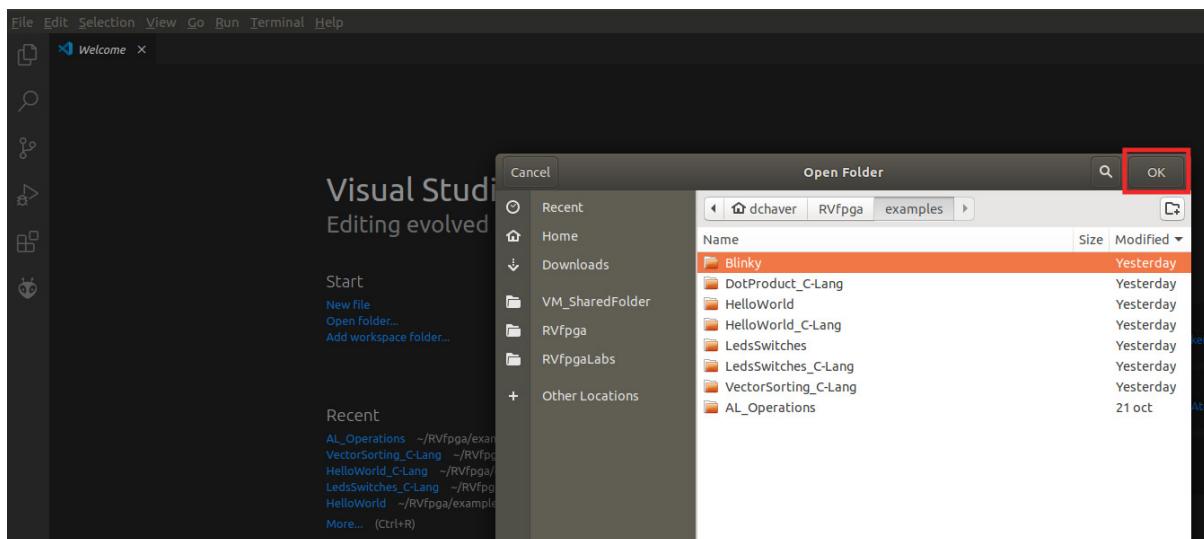
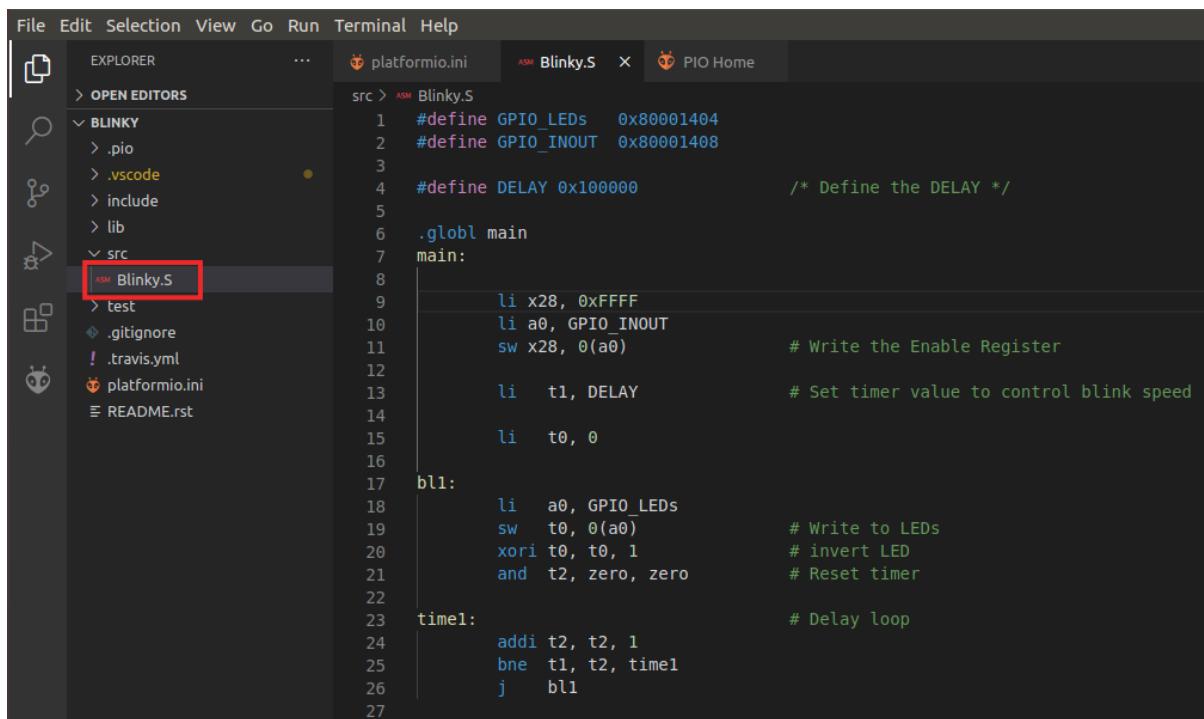


图50. Blinky程序文件夹

3. 选择目录*Blinky*，然后单击“OK”（确定）（图50）。
4. 在编辑器中通过单击打开示例的汇编代码，即文件*blink.y.S*（图51）。

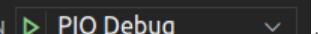


```

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
bl1:
17    li a0, GPIO_LEDs
18    sw t0, 0(a0)            # Write to LEDs
19    xor t0, t0, 1            # invert LED
20    and t2, zero, zero      # Reset timer
21
time1:                                # Delay loop
22    addi t2, t2, 1
23    bne t1, t2, time1
24    j bl1
25
26
27

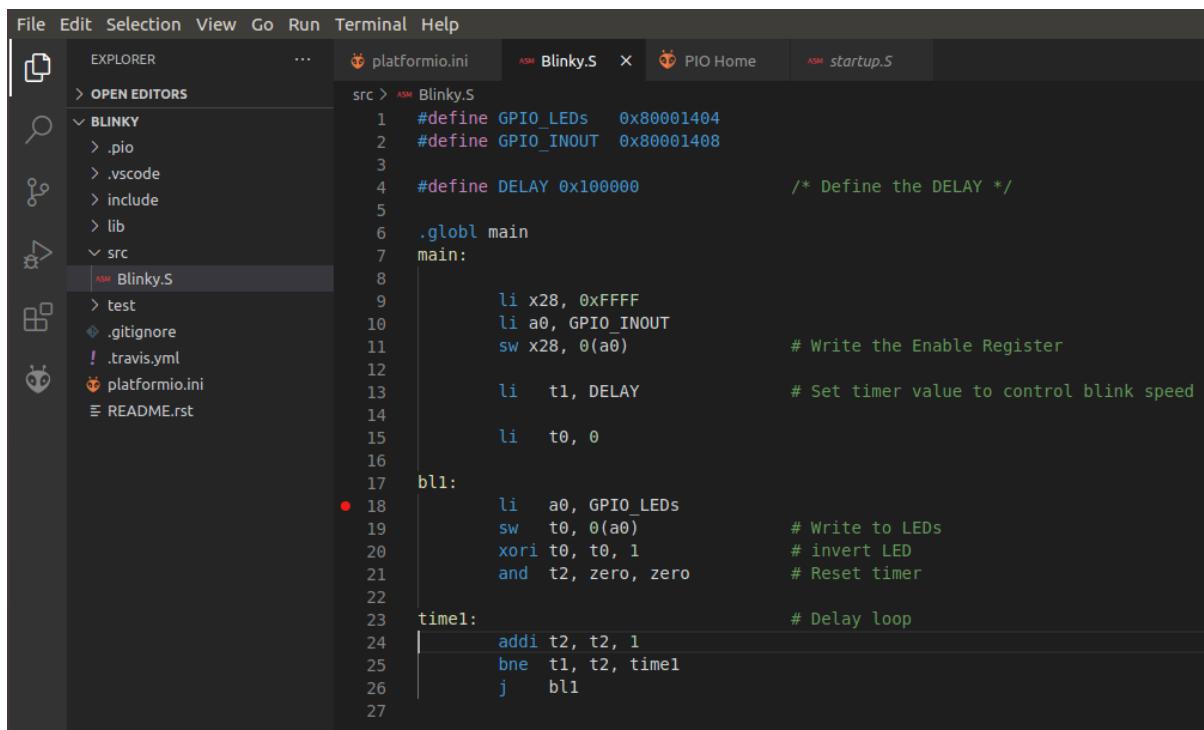
```

图51. PlatformIO中的blinky.S

5. 单击  运行和调试程序；然后通过单击播放按钮   开始调试。PlatformIO在main函数的开头设置一个临时断点。因此，单击“Continue”（继续）按钮  运行程序。

6. 在开发板上，将看到最右侧的LED开始闪烁。

7. 通过单击“Pause”（暂停）按钮  暂停执行。执行将在无限循环内的某处停止（可能在time1延时循环内）。
8. 通过单击第18行的左侧设置一个断点。将出现一个红点，并且断点将添加到“BREAKPOINTS”（断点）选项卡（参见图52）。



```

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
bl1:
17
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
time1:
23     addi t2, t2, 1          # Delay loop
24     bne t1, t2, time1
25     j bl1
26
27

```

图52. 在blinky.S中设置断点

9. 然后，通过单击“Continue”（继续）按钮  继续执行。随后将一直执行到存储字（sw）指令后停止，该指令会将1（或0）写入最右侧的LED。
10. 继续执行几次；将看到驱动到最右侧LED的值每次都会发生改变。

11. 停止调试 ，然后通过单击  返回到资源管理器窗口。通过选择“File”（文件）→“Close Folder”（关闭文件夹）关闭程序。

D. LedsSwitches程序

第三个汇编示例与开发板上的LED和开关通信（参见图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
14     lw t0, 0(a1)           # Read the Switches
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

```

图53. LedsSwitches.S

按照下面的步骤在FPGA开发板上运行和调试此代码：

1. 如果执行了前面的示例，则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用的是LedsSwitches示例而不是AL_Operations示例。
2. 在顶部栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录[RVfpgaPath]/RVfpga/examples/。选择目录LedsSwitches，然后单击“OK”（确定）。
3. 程序LedsSwitches.S有一个无限循环，在其中读取开关，然后将开关状态显示在LED上（图54）。

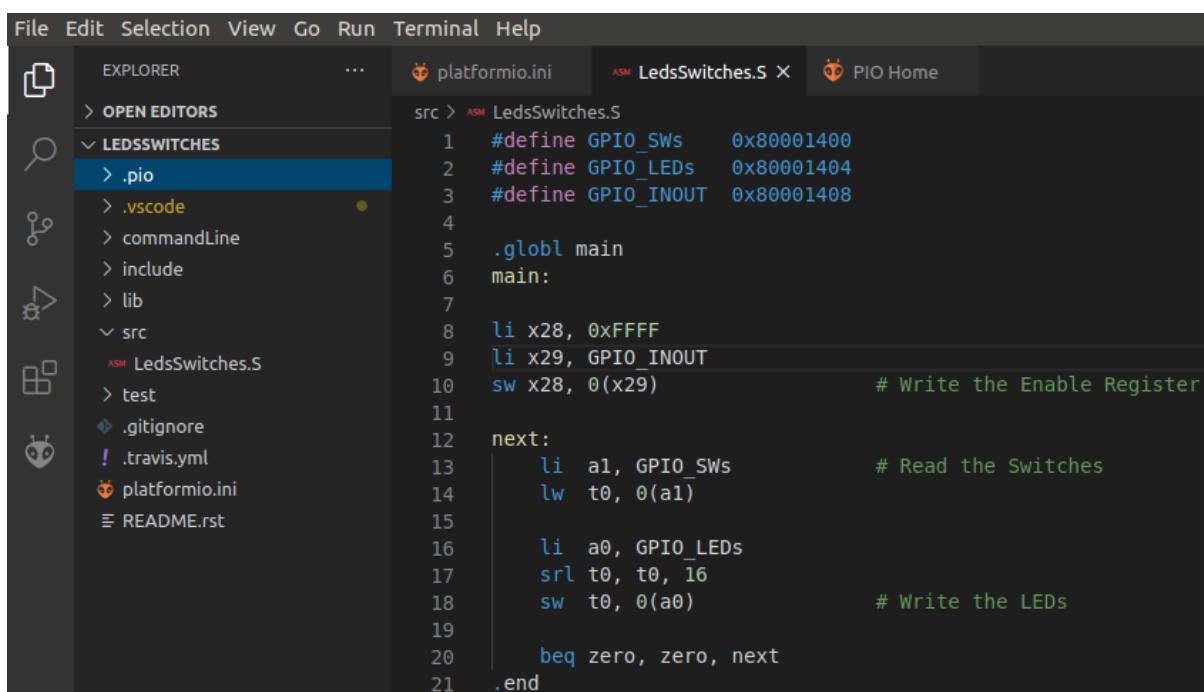
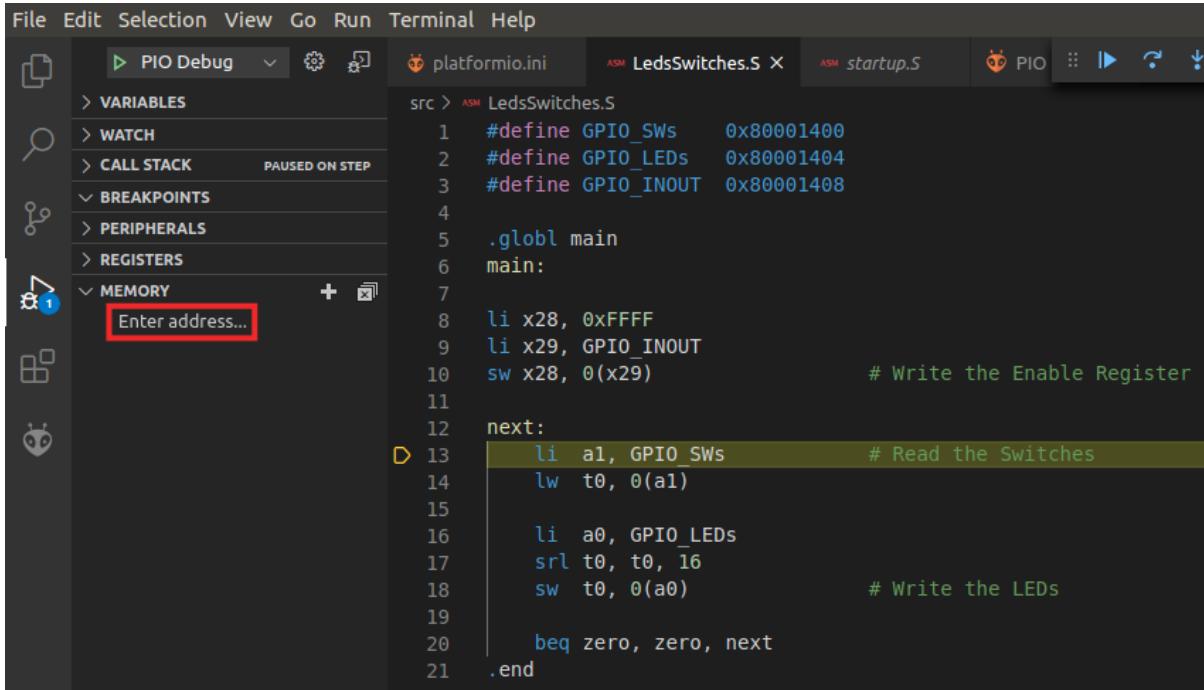


图54. PlatformIO中的LedsSwitches.S

4. 按照先前程序的说明启动调试器后，程序开始运行。PlatformIO在main函数的开头设置一个临时断点。因此，单击“Continue”（继续）按钮▶运行程序。
5. 拨动Nexys A7开发板底部的开关。将立即看到开发板上的LED显示开关的新值。可以按上文所述暂停执行，逐步运行并检查寄存器。完成后，单击“File”（文件）→“Close Folder”（关闭文件夹）关闭项目。
6. 有时，检查存储器中存储的值可能非常有用。为此，PlatformIO提供了“Memory”（存储器）显示。

- a. 暂停执行，并跳到`next`循环的开始处。展开窗口左侧的“Memory”（存储器）显示（参见图55），然后单击“Enter address...”（输入地址...）



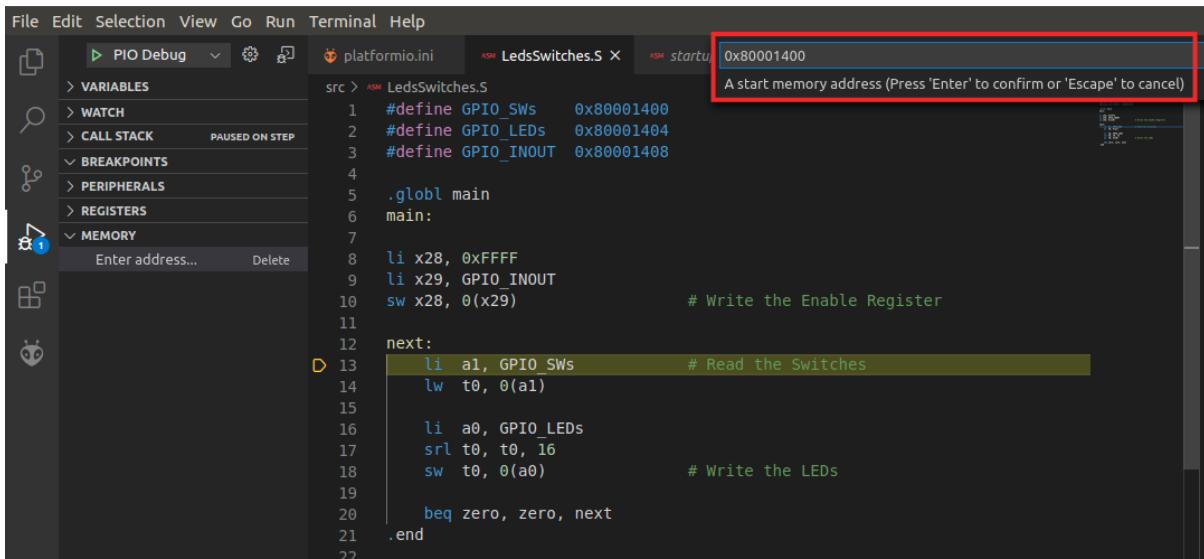
```

File Edit Selection View Go Run Terminal Help
File PIO Debug Variables Watch Call Stack PAUSED ON STEP Breakpoints Peripherals Registers Memory Enter address...
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

```

图55. 存储器显示

- b. 将请求初始存储器地址（参见图56）。在开关的映射位置插入初始地址，在本例中为`0x80001400`。



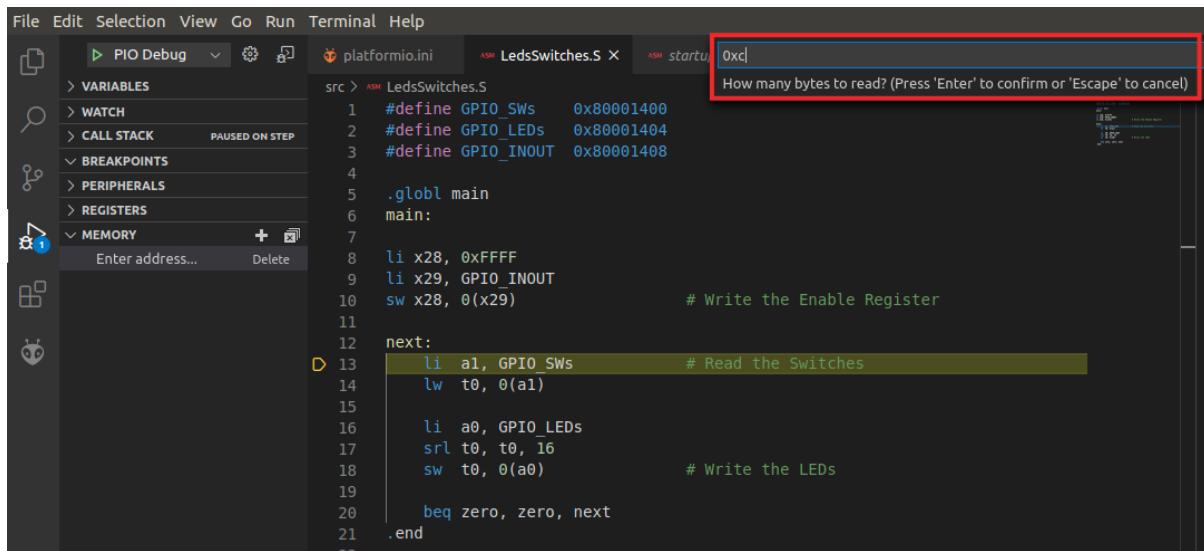
```

File Edit Selection View Go Run Terminal Help
File PIO Debug Variables Watch Call Stack PAUSED ON STEP Breakpoints Peripherals Registers Memory Enter address... Delete
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

```

图56. 要显示的初始存储器地址

- c. 然后，请求要检查的字节数（参见图57），为此插入一个值`0xc`（我们要检查三个4字节的I/O寄存器，因此需要12个字节）。



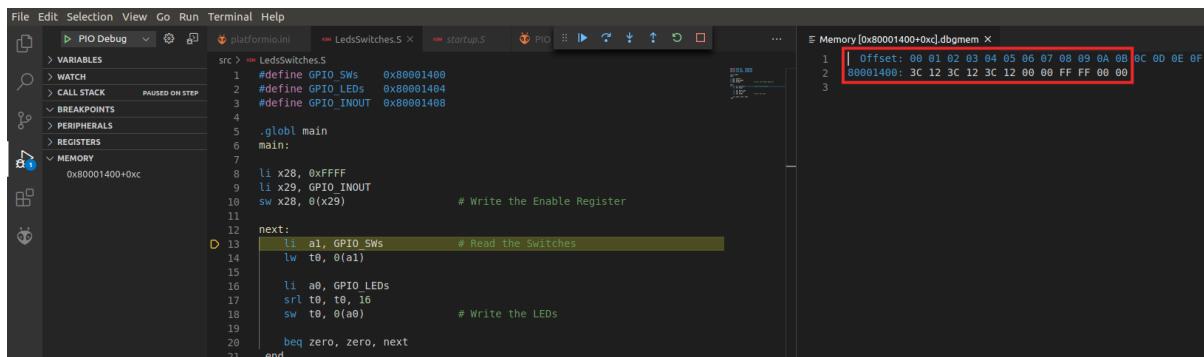
```

File Edit Selection View Go Run Terminal Help
  PIO Debug  platormio.ini LedsSwitches.S startup.S 0xc
  > VARIABLES
  > WATCH
  > CALL STACK PAUSED ON STEP
  > BREAKPOINTS
  > PERIPHERALS
  > REGISTERS
  > MEMORY Enter address... Delete
  How many bytes to read? (Press 'Enter' to confirm or 'Escape' to cancel)
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
22

```

图57. 要显示的字节数

- d. “Memory”（存储器）显示将在右侧打开，其中显示我们已请求的12个字节（参见图58）。我们在16个开关中的值为0x123C（参见地址0x80001402和0x80001403处的字节）。考虑到RISC-V架构采用小端模式，因此图中所示的值与之一致。16个LED（存储在地址0x80001404和0x80001405处）显示相同的值。



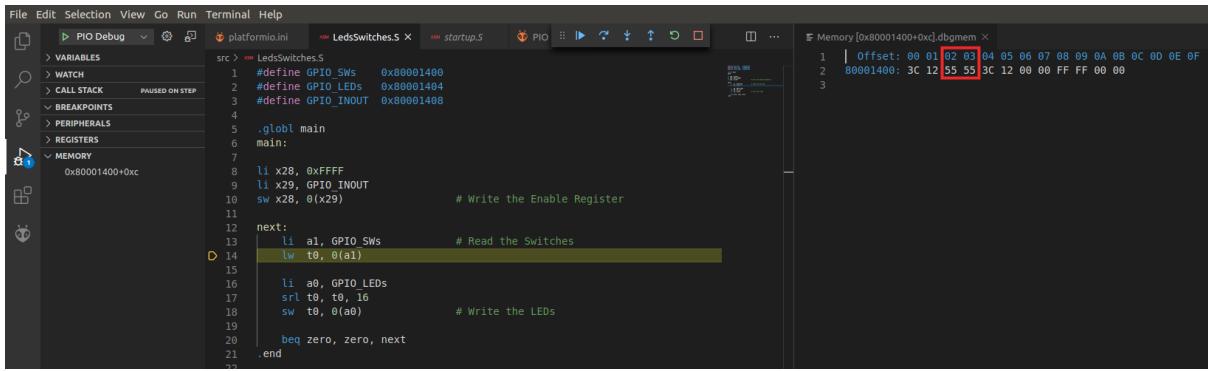
```

File Edit Selection View Go Run Terminal Help
  PIO Debug  platormio.ini LedsSwitches.S startup.S  Memory [0x80001400+0xc].dbgmem X
  > VARIABLES
  > WATCH
  > CALL STACK PAUSED ON STEP
  > BREAKPOINTS
  > PERIPHERALS
  > REGISTERS
  > MEMORY 0x80001400+0xc
  1 | Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
  2 80001400: 3C 12 3C 12 3C 12 00 00 FF FF 00 00
  3

```

图58. 存储器地址0x80001400-0x8000140B

- e. 例如，将开发板上的开关的值更改为0x5555，然后再逐步执行一次循环迭代。存储器中的开关值应在执行第一条指令后立即更改（图59，上图），而LED的值应在执行sw指令后相应地更改（图59，下图）。

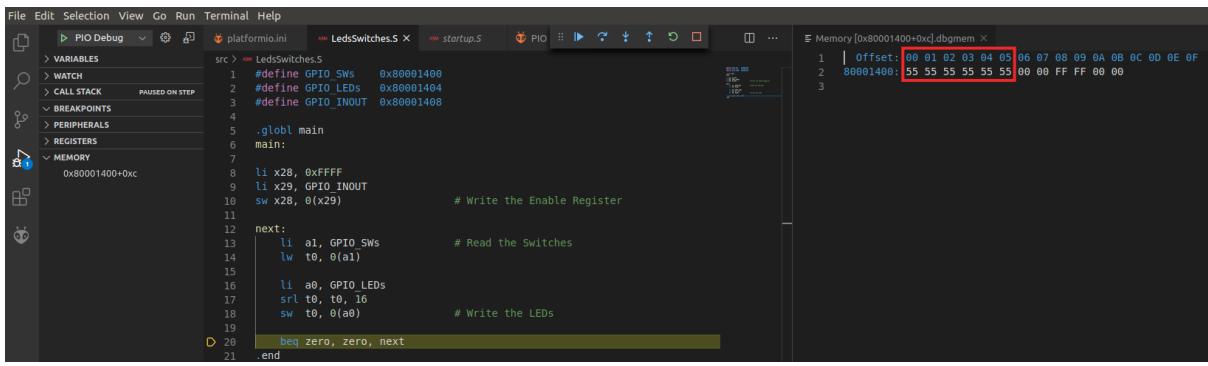


```

File Edit Selection View Go Run Terminal Help
src > LedsSwitches.S
platformio.ini LedsSwitches.S startup.S PIO ... Memory [0x80001400+0x4].dbgmem X
1 | Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2 80001400: 3C 12 55 55 3C 12 00 00 FF FF 00 00
3

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
22

```



```

File Edit Selection View Go Run Terminal Help
src > LedsSwitches.S
platformio.ini LedsSwitches.S startup.S PIO ... Memory [0x80001400+0x4].dbgmem X
1 | Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2 80001400: 55 55 55 55 55 55 00 00 FF FF 00 00
3

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
22

```

图59. 开关和LED更改

- f. 也可以查看其他存储单元，例如用于存储程序的RAM地址。打开另一个存储器范围，其起始地址为0x0（分配给RAM存储器的初始地址），占用0x100个字节（图60）。在启动程序（Startup.S）之后，将立即看到LedsSwitches程序中存储在地址范围0x90-0xC4内的指令。

```

≡ Memory [0x0+0x100].dbgmem ×

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

```

图60. 存储器地址0x0至0x100

- g. 通过打开程序的反汇编版本，可以查看程序指令的机器代码，程序的反汇编版本位于：
`[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/firmware.dis`（参见图61）。将两幅图进行比较，尝试确定程序的指令。

```

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       srlt t0,t0,0x10
81 bc: 00552023       sw t0,0(a0)
82 c0: fe0002e3       beqz zero,a4 <next>
83

```

图61. LedsSwitches程序的反汇编版本

E. LedsSwitches_C-Lang程序

程序LedsSwitches_C-Lang.c（图62）与先前显示的LedsSwitches.s程序（图53）的功能相同，但它是用C语言而不是汇编语言编写的。

```

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 }

```

图62. LedsSwitches_C-Lang.c

按照下面的步骤在FPGA开发板上运行和调试此程序：

1. 如果执行了前面的示例，则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用的是LedsSwitches_C-Lang示例而不是AL_Operations示例。
2. 在顶部菜单栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录 [RVfpgaPath]/RVfpga/examples/。选择目录 LedsSwitches_C-Lang，然后单击“OK”（确定）。
3. 在调用调试器之前，先在C代码的第15行设置一个断点。
4. 然后，开始调试。程序将开始执行，并将在断点处停止（图63）。

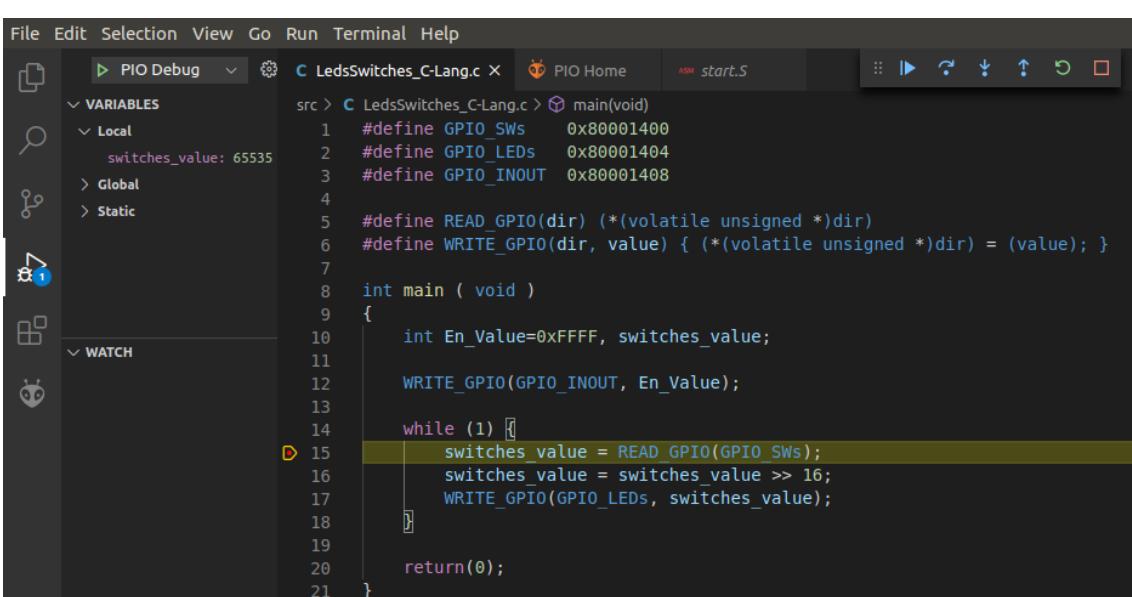


图63. 执行在断点处停止

5. 使程序继续执行  几次，但在每次单击时更改开关。LED应显示开关的值。
6. 可以查看上述C语言程序的执行情况，也可以通过单击图64中突出显示的“Switch to assembly”（切换到汇编）来查看编译器生成的汇编程序的执行情况。

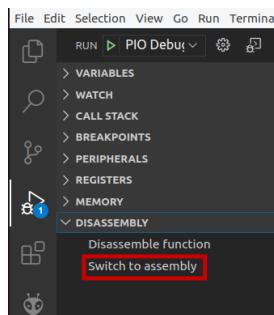
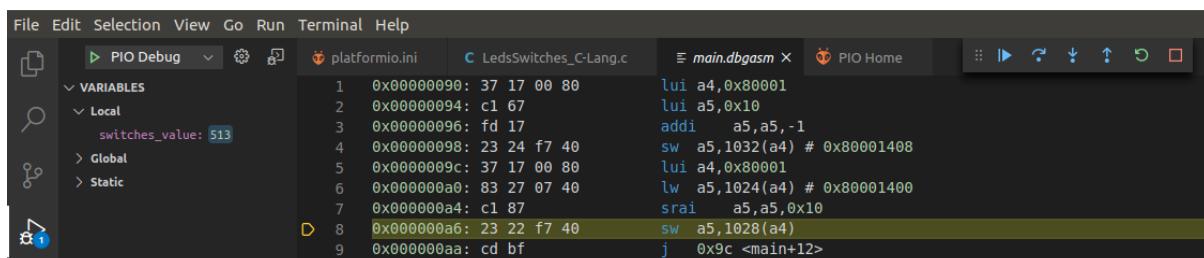


图64. 切换到汇编

7. 汇编程序（图65）首先使用装载指令（`lw a5,1024(a4)`）读取开关值，然后使用存储指令（`sw a5,1028(a4)`）将该值写入LED。逐步执行程序，更改开关并验证LED发生变化以反映新的开关值。



```

File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal
platformio.ini C LedsSwitches_C-Lang.c main.dbgasm X PIO Home
Disassemble function
Switch to assembly

1 0x00000090: 37 17 00 80    lui a4,0x80001
2 0x00000094: c1 67          lui a5,0x10
3 0x00000096: fd 17          addi a5,a5,-1
4 0x00000098: 23 24 f7 40    sw a5,1032(a4) # 0x80001408
5 0x0000009c: 37 17 00 80    lui a4,0x80001
6 0x000000a0: 83 27 07 40    lw a5,1024(a4) # 0x80001400
7 0x000000a4: c1 87          srai a5,a5,0x10
8 0x000000a6: 23 22 f7 40    sw a5,1028(a4)
9 0x000000aa: cd bf          j 0x9c <main+12>

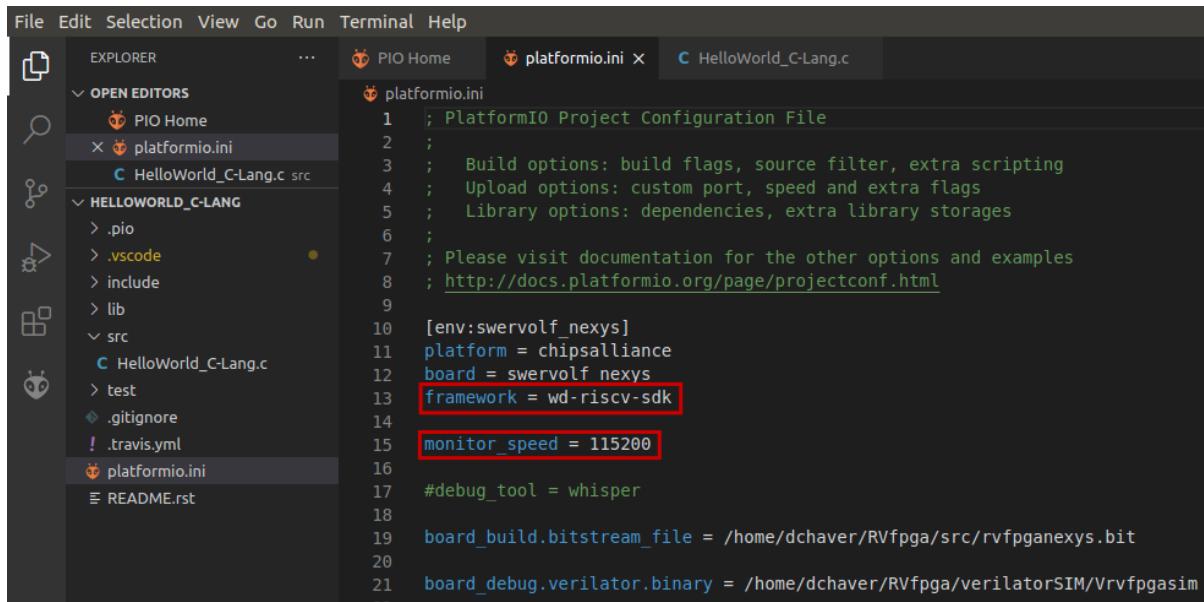
```

图65. 汇编程序

F. HelloWorld_C-Lang程序

第二个C程序示例通过串行端口向shell打印一条短消息。要查看此消息，可以使用任何终端仿真器，例如`gtkterm`、`minicom`等；但是，PlatformIO提供了自己的串行监视器，因此我们在这里展示如何使用此监视器。

为了配置PlatformIO串行监视器，必须配置一些参数；具体地说，必须确定用于串行数据传输的数据速率（以每秒位数或波特数为单位），为此，我们可以使用`platformio.ini`文件（请注意，此文件是PlatformIO项目的一部分）中的参数`monitor_speed`来确定该值。请参见图66。



```

1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter, extra scripting
4 ; Upload options: custom port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ;
7 ; Please visit documentation for the other options and examples
8 ; http://docs.platformio.org/page/projectconf.html
9
10 [env:swervolf_nexys]
11 platform = chipsalliance
12 board = swervolf nexys
13 framework = wd-riscv-sdk
14
15 monitor_speed = 115200
16
17 #debug_tool = whisper
18
19 board_build.bitstream_file = /home/dchaver/RVfpga/src/rvfpaganexys.bit
20
21 board_debug.verilator.binary = /home/dchaver/RVfpga/verilatorSIM/Vrvfpgasim
22

```

图66.串行监视器配置

此外，需要通过在终端中输入以下命令来将您自己添加到dialout、tty和uucp组中：

```

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

```

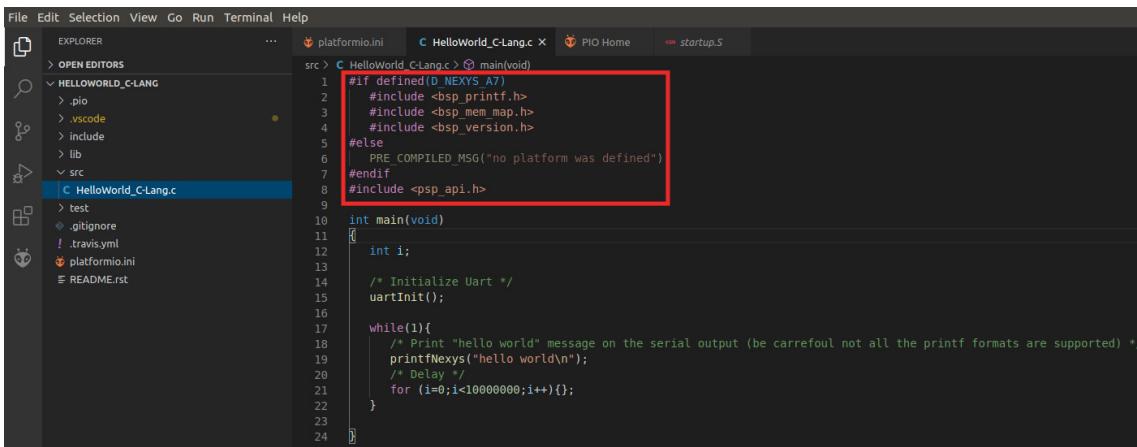
在执行这三条命令之后，重启计算机，以使组更改生效。

Windows/macOS: Windows和macOS用户不需要完成上述步骤。

此外，该程序使用WD在其固件包（<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>）中提供的处理器支持包（Processor Support Package, PSP）和开发板支持包（Board Support Package, BSP）。通过在platformio.ini中使用特定命令（framework = wd-riscv-sdk）（如图66所示），并在C程序的开头包括正确的文件（如图67所示）将这些库包括在项目中。有关系统中的完整库，请访问以下路径：

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

这些库提供许多功能和宏，用户可借此执行许多操作，例如使用中断、打印字符串、读/写各个寄存器等。在本例中，我们将使用printfNexys函数在串行监视器上打印一条消息。在后面的示例和实验中，我们将展示如何使用其他函数和宏来实现不同的目的。



```

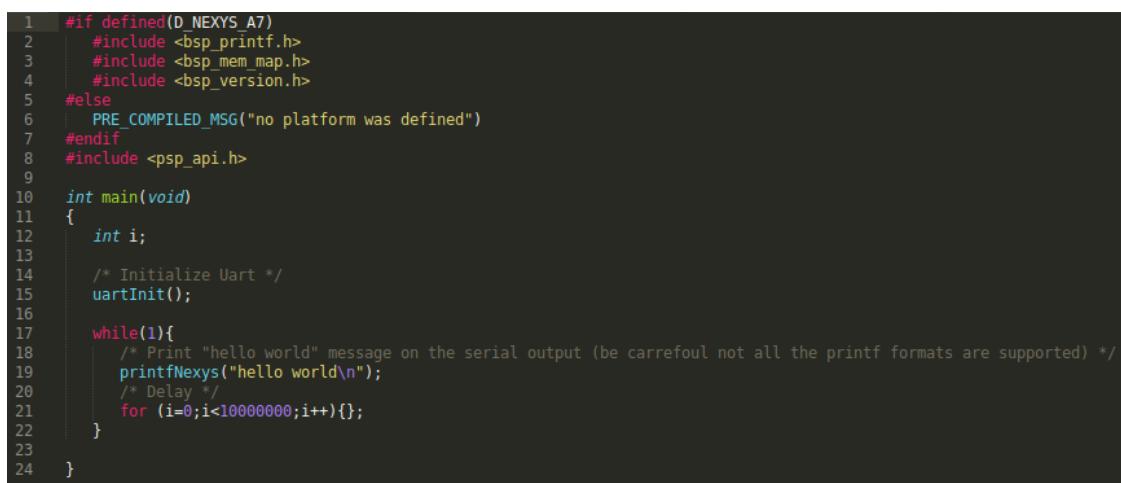
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
HELLOWORLD_C-LANG
> .pio
> .vscode
> include
> lib
> src
C HelloWorld_C-Lang.c
> test
.gitignore
.travis.yml
platformio.ini
README.rst
src > C HelloWorld_C-Lang.c X PIO Home
src > C HelloWorld_C-Lang.c main(void)
1 #if defined(D_NEXYS_A7)
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 <bsp_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 carrefoul not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<10000000;i++){};
22     }
23 }
24

```

图67. 在HelloWorld_C-Lang.c中包括.h文件

按照下面的步骤在FPGA开发板上运行和调试此代码：

1. 如果执行了前面的示例，则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用HelloWorld_C-Lang示例而不是AL_Operations示例。
2. 打开VSCode。打开VSCode时，PlatformIO应在VSCode中自动打开。在顶部栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录[RVfpgaPath]/RVfpga/examples/。选择HelloWorld_C-Lang文件夹，然后单击“OK”（确定）。
3. 程序HelloWorld_C-Lang.C（图68）初始化UART（函数uartInit），然后使用函数printfNexys通过串行端口发送字符串（可以在文件~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/bsp/bsp_printf.c中找到这些函数的实现）。然后，该程序会延迟一段时间，之后再返回到循环的开始处。



```

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 <bsp_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 carrefoul not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<10000000;i++){};
22     }
23 }
24

```

图68. HelloWorld_C-Lang.C的main函数

4. 在PlatformIO中启动调试器。当程序开始运行时，通过单击VS Code底部的插头按钮（图69）打开串行监视器。

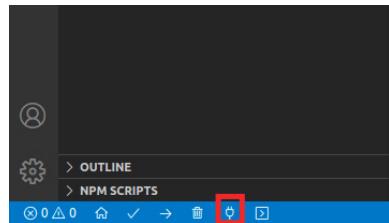


图69. 打开串行终端

5. 串行监视器反复打印消息“HELLO WORLD !!!”，如图70所示。

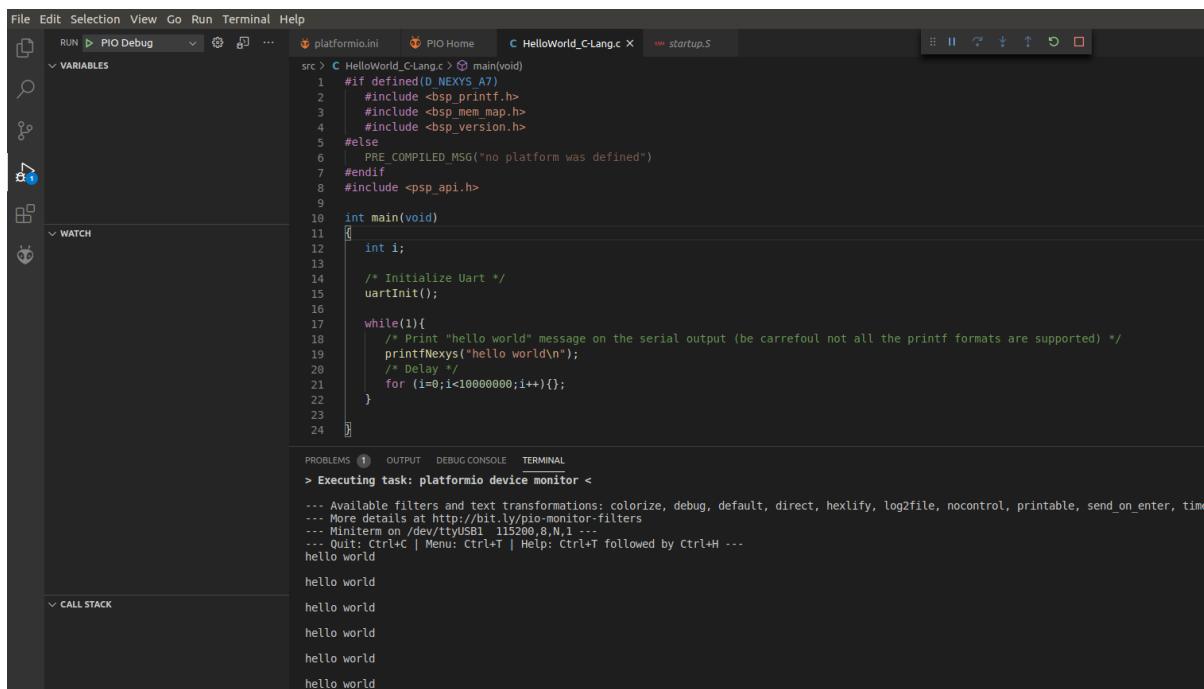


图70. 程序的执行

G. VectorSorting_C-Lang程序

最后，我们展示另一个C程序，该程序从大到小对向量A的各元素进行排序，并将排序后的值放在向量B中。向量A的值替换为零。图71给出了该程序。

```

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        A[ind]=0;
19        B[j]=max;
20    }
21
22    for(j=0; j<N; j++) {
23        printf("%d ", B[j]);
24    }
25 }
```

```

17         }
18         B[j]=A[ind];
19         A[ind]=0;
20     }
21
22     while(1);
23 }
```

图71. VectorSorting_C-Lang.c

按照下面的步骤在FPGA开发板上运行和调试此程序：

1. 如果执行了前面的示例，则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用VectorSorting_C-Lang示例而不是AL_Operations示例。
2. 在顶部菜单栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录[RVfpgaPath]/RVfpga/examples/。选择VectorSorting_C-Lang文件夹，然后单击“OK”（确定）。
3. 在第10行放置一个断点并开始调试。执行将在for循环开始时停止（图72）。展开调试器侧边栏中的“VARIABLES”（变量）部分，然后分析A和B数组的值（如图72中红色突出显示部分所示）。

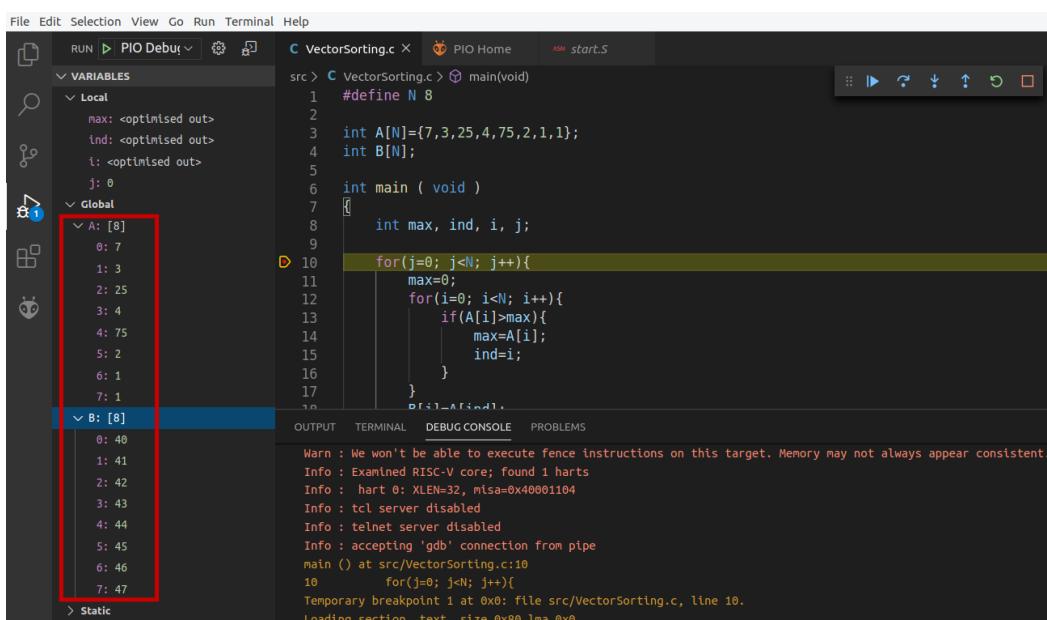


图72. 执行在程序开始时停止

4. 现在在第18行放置另一个断点，并通过单击继续执行（参见图73）。打开“Memory”（存储器）显示（如LedsSwitches程序部分所述，图55），从地址0x2148（该地址为此程序的存储器中用于存储向量A的地址）开始显示0x50个字节（参见图73）。可以查看向量A（范围为0x2148-0x2167）和B（范围为0x2178-0x2197）的初始值。

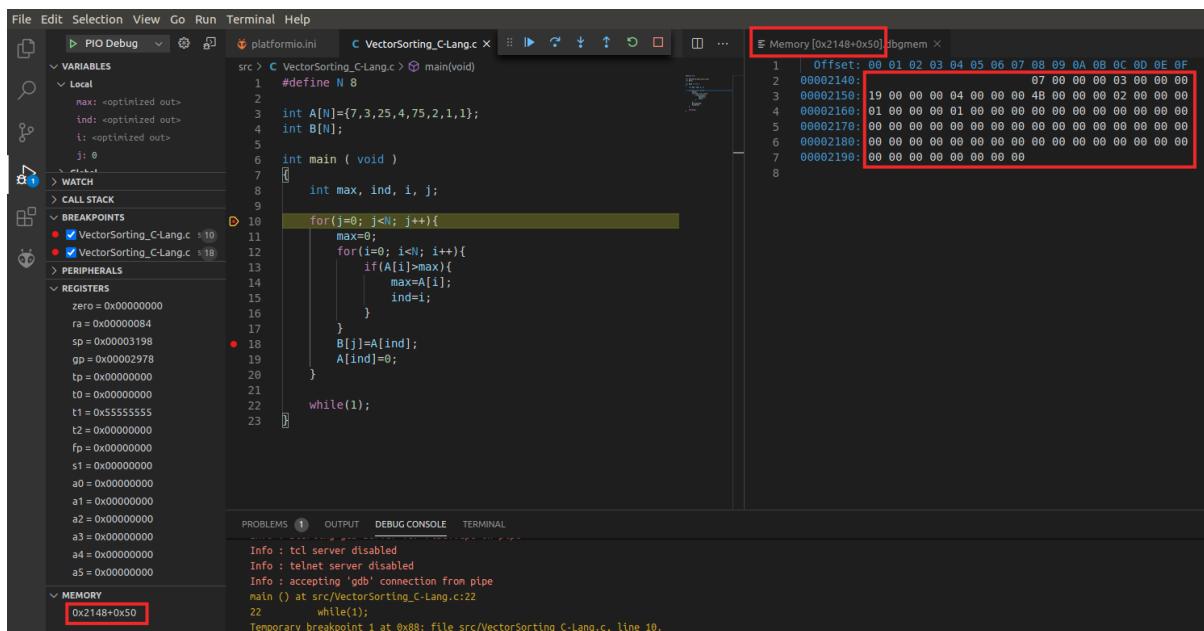


图73. 数组A和数组B的存储器显示 – 初始状态

请注意，通过切换到汇编（如图64所示），并分析访问向量A和向量B的任何指令，可以轻松找到存储器中用于存储这些向量的地址（图74）。如图中所示，在大多数情况下，注释都提供此信息；但是，也可以继续执行这些指令，并查看存储在寄存器中的值。

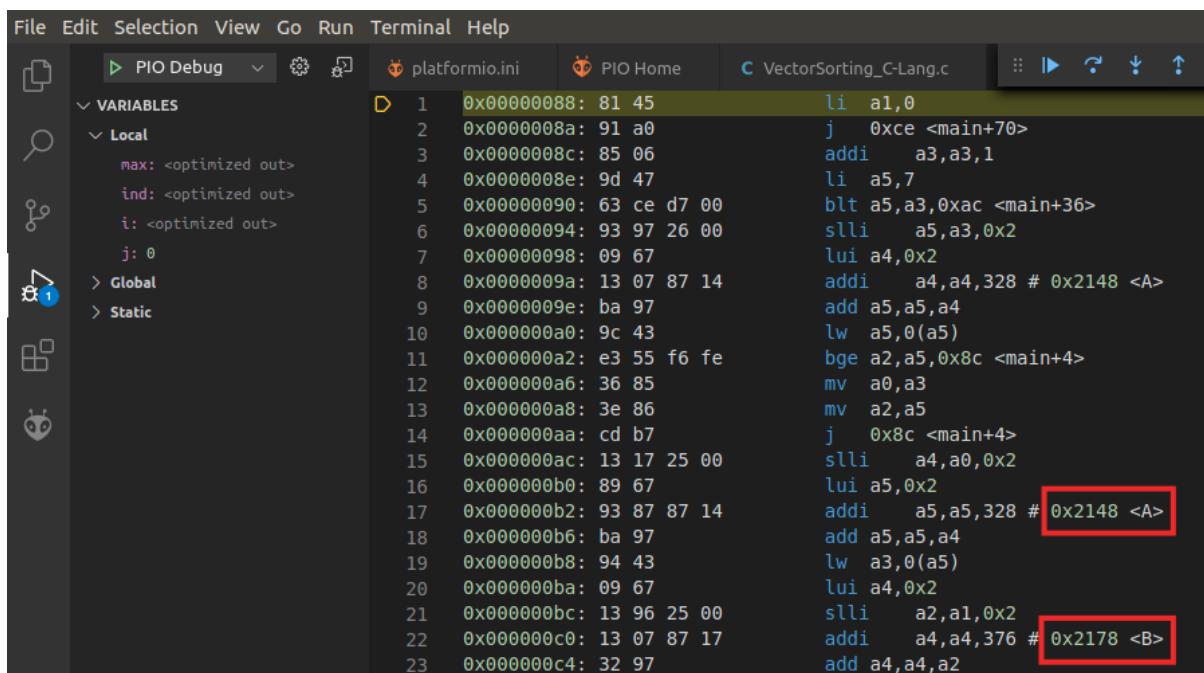
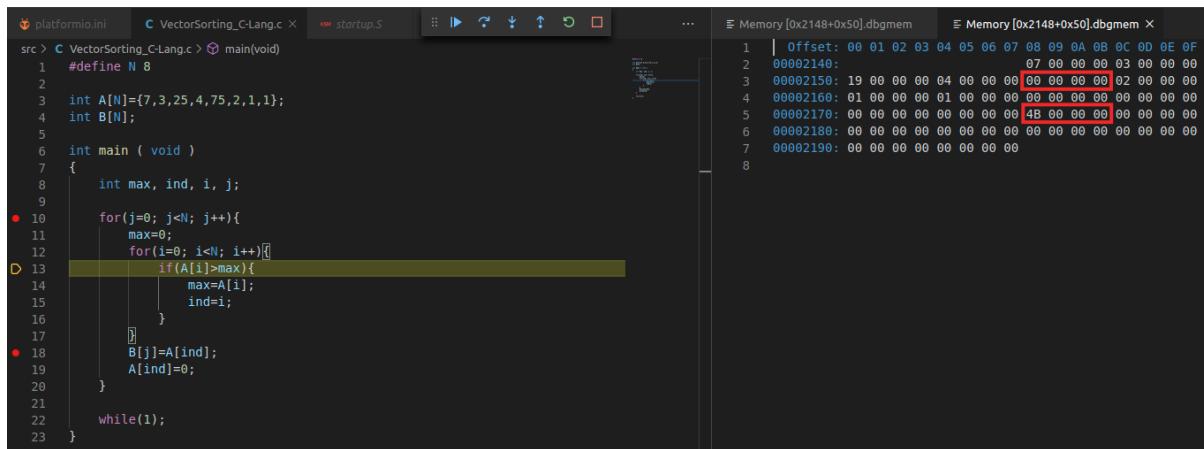


图74. 存储器中用于存储A和B的地址

单击“Step Over”（单步跳过）按钮 () 两次，将看到B的第一个分量存储在存储器中，并且A中的对应值设置为0（参见图75）。



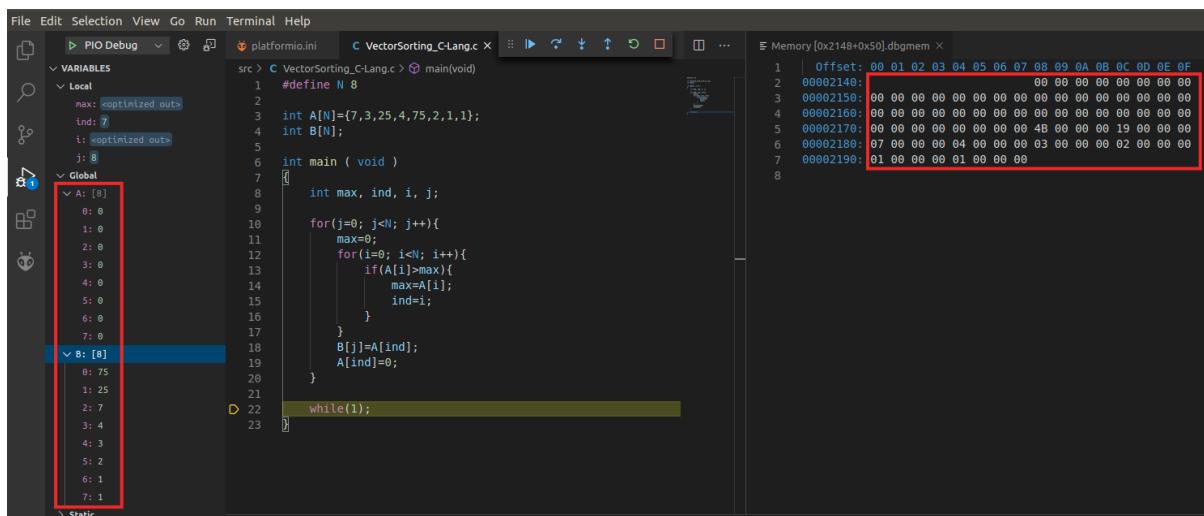
```

platformio.ini      C VectorSorting_C-Lang.c      startup.S ...
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 }

```

图75. 数组A和数组B的存储器显示 – 存储B的第一个分量并复位A中的相应分量

5. 删除所有断点，继续执行，并在几秒钟后暂停执行 – 此时程序已完成执行。再次分析存储在数组A和数组B中的值。如图76所示，向量B保留原始向量A的值（从大到小排序），向量A全为零（在左侧的变量列表和右侧的存储器控制台中均可查看这一点）。



Offset	Value
0	75
1	25
2	21
3	19
4	7
5	4
6	3
7	2

图76. 执行在程序结束时停止

H. DotProduct_C-Lang程序

最后一个示例程序DotProduct_C-Lang.c（图77）计算两个向量的点积。该程序具有两个函数：*main*和*dotproduct*。*main*函数使用以下三个输入参数调用*dotproduct*函数：向量大小和两个向量的初始地址。然后，*dotproduct*函数计算两个向量的点积并返回结果。

```

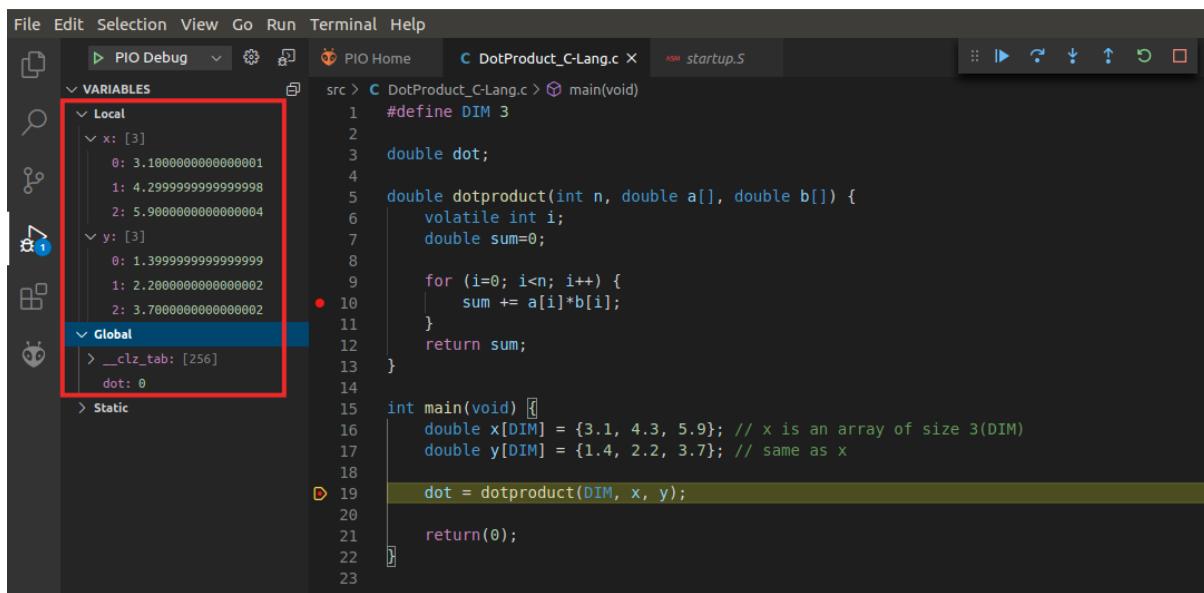
1 #define DIM 3
2
3 double dot;
4
5 double dotproduct(int n, double a[], double b[]){
6     volatile int i;
7     double sum=0;
8
9     for (i=0; i<n; i++) {
10         sum += a[i]*b[i];
11     }
12     return sum;
13 }
14
15 void main(void) {
16     double x[DIM] = {3.1, 4.3, 5.9};           // x is 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 }
```

图77. DotProduct_C-Lang.c

在本示例中，我们使用实数进行运算（请注意，变量x、y和dot的数据类型是double）。但是，SweRV EH1处理器不支持浮点。因此，该示例通过gcc（<https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html>）提供的软件浮点库使用浮点仿真。只要包含-msoft-float，该库就可用于禁止生成浮点指令。

按照下面的步骤在FPGA开发板上运行和调试此代码：

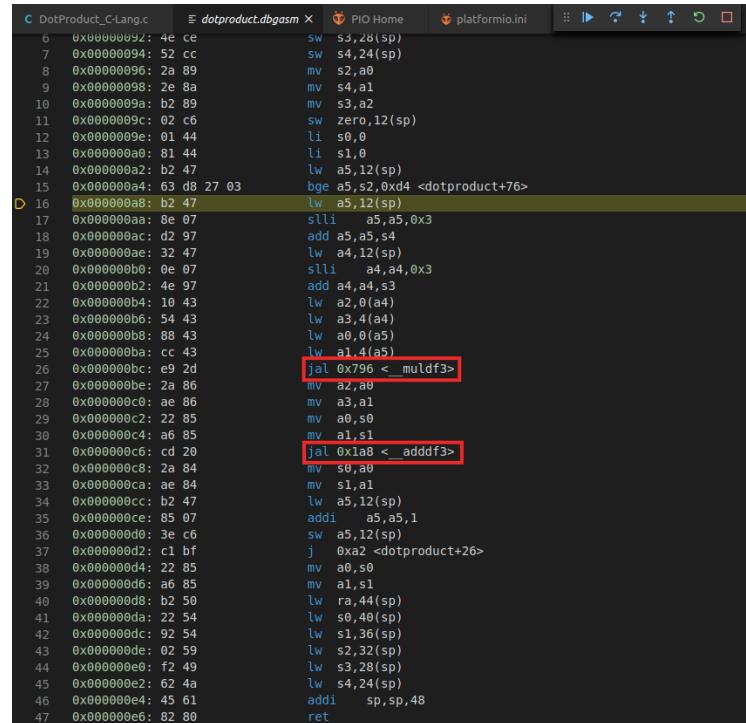
1. 如果执行了前面的示例，则RVfpgaNexys已编程到FPGA开发板上，因此无需再次对其进行编程。但是，如果确实需要将RVfpgaNexys重新编程到开发板上，请按照A部分所述进行操作，注意使用DotProduct_C-Lang示例而不是AL_Operations示例。
2. 在顶部菜单栏上，单击“File”（文件）→“Open Folder”（打开文件夹），然后导航至目录[RVfpgaPath]/RVfpga/examples/。选择目录DotProduct_C-Lang，然后单击“OK”（确定）。
3. 在调用调试器之前，先在第10行和第19行分别设置一个断点（参见图78）。
4. 然后，开始调试。该程序将开始执行；并在第一个断点处停止（参见图78）。
5. 在调试器侧边栏上，展开“Variables”（变量）部分（参见图78）。这两个向量包含在main中分配的初始值。dot变量初始化为0。



The screenshot shows the IUP IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows 'PIO Debug' and the file 'DotProduct_C-Lang.c'. The main area displays the C code for a dot product calculation. The left sidebar has icons for file operations like Open, Save, and Build. The bottom left shows the terminal output: 'src > C DotProduct_C-Lang.c > main(void)'. The bottom right shows the build status: 'ash startup.S'. The central part of the screen shows the variable watch window. A red box highlights the 'Local' section, which contains arrays 'x' and 'y' of size 3. The 'x' array values are 3.1, 4.29999999999998, and 5.9. The 'y' array values are 1.39999999999999, 2.2, and 3.7. Below 'Local' is the 'Global' section, which contains '_clz_tab' of size 256 and 'dot' at address 0. The code itself defines a 'dotproduct' function that calculates the dot product of two arrays and a 'main' function that calls it.

图78. DotProduct_C-Lang程序：第一个断点处的变量值

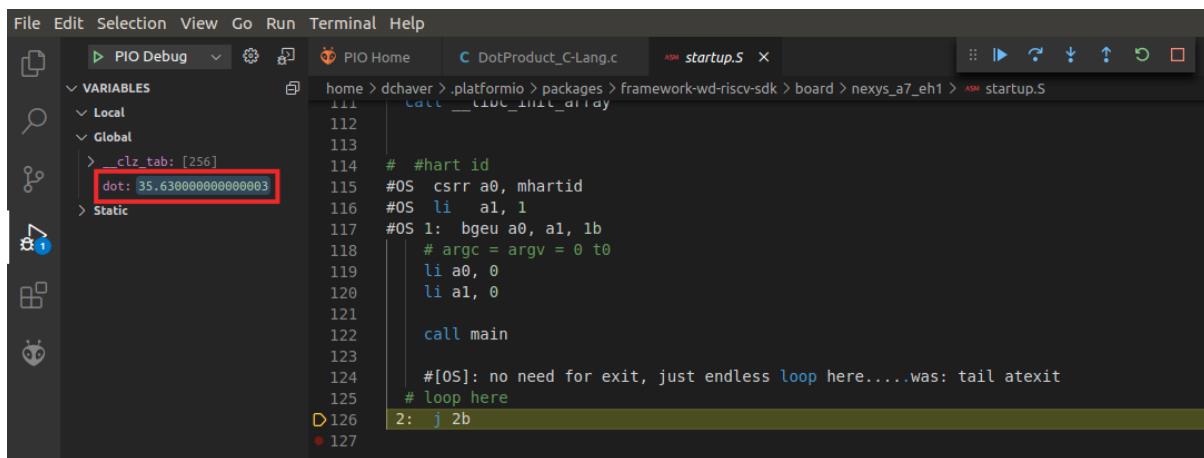
6. 使程序继续执行 。程序在第二个断点处停止（第10行）。
7. 切换到汇编（如图64所示操作）。可以查看浮点仿真程序，并通过单步进入这些例程来对其进行深入分析（参见图79）。



The screenshot shows the assembly code for the 'dotproduct' function. The assembly code uses floating-point registers (a0-a7) and memory locations (s0-s12). It performs a series of load (lw), add (add), and store (sw) operations. Two specific lines of assembly code are highlighted with red boxes: 'jal 0x796 < _muldf3>' and 'jal 0x1a8 < _addirf3>'. These lines correspond to the floating-point multiplication and addition operations in the C code.

图79. DotProduct_C-Lang程序：第二个断点处的汇编代码

8. 切换回C程序并删除两个断点。继续执行，然后将其暂停。将看到dot变量的值变为两个向量的点积（图80）。



```
home > dchaver > .platformio > packages > framework-wd-riscv-sdk > board > nexys_a7_eh1 > ASM startup.S
111    .text __main_start
112    .data
113
114    # #hart id
115    #OS csrr a0, mhartid
116    #OS li a1, 1
117    #OS 1: bgeu a0, a1, 1b
118    # argc = argv = 0 t0
119    li a0, 0
120    li a1, 0
121
122    call main
123
124    #[OS]: no need for exit, just endless loop here.....was: tail atexit
125    # loop here
126    2: j 2b
127
```

图80. DotProduct_C-Lang程序：点积的结果

9. 完成对此程序的探究后，单击“File”（文件）→“Close Folder”（关闭文件夹）来关闭项目。

7. 在VERILATOR中仿真

在本部分中，将在RVfpgaSim上使用Verilator运行上一节中使用的第一个程序（*AL_Operations*）。Verilator是一种硬件描述语言（Hardware Description Language, HDL）仿真器，用于仿真定义SoC的Verilog（可从*[RVfpgaPath]/RVfpga/src*获取）。通过这种运行SoC的方式，用户能够分析系统的内部信号，这对于以后的实验和练习（在这些实验和练习中我们将向SoC添加内部操作或新硬件）特别有用。

在此，我们展示了如何使用Verilator来查看逐周期的指令和*AL_Operations*的寄存器值，*AL_Operations*是在第6部分中执行和调试的第一个简单的汇编程序（图44）。将使用PlatformIO生成仿真轨迹，然后将时钟、两路超标量处理器的指令和寄存器x28（即寄存器t3）信号加到仿真波形上，并通过GTKWave查看指令和寄存器信号随程序执行的变化情况。

生成仿真二进制文件，**Vrvpgasim**：

目录*[RVfpgaPath]/RVfpga/verilatorSIM*包含*Makefile*和用于生成RVfpgaSim仿真器二进制文件的脚本（*swervolf_0.7.vc*）。该脚本包含Verilator需要了解的信息，以及SoC源的位置信息，在本例中，SoC源位于*[RVfpgaPath]/RVfpga/src*中。接下来，我们展示如何生成RVfpgaSim的二进制文件，该文件稍后将用于创建在RVfpgaSim上运行的*AL-Operations*程序的仿真轨迹。

1. 在终端窗口中，通过执行以下命令来生成仿真器二进制文件：

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

文件**Vrvpgasim**（RVfpgaSim仿真二进制文件）应在目录*[RVfpgaPath]/RVfpga/verilatorSIM*内生成。

Windows: 如果使用Windows，则必须在Cygwin终端内执行这些相同的步骤（有关详细说明，请参见附录C）。请注意，C: Windows文件夹位于Cygwin中的以下位置：*/cygdrive/c*。本节中的所有其他说明与Linux的说明相同。

macOS: 有关详细说明，请参见附录D。

使用Vrvpgasim通过PLATFORMIO生成仿真轨迹：

生成仿真器二进制文件（*Vrvpgasim*）后，将在PlatformIO内使用该文件来生成程序*AL_Operations*的仿真轨迹（*trace.vcd*）。

2. 在计算机中依次打开VSCode和PlatformIO。
3. 在顶部栏上，单击“File”（文件）→“Open Folder...”（打开文件夹...）（图81），然后导航至目录*[RVfpgaPath]/RVfpga/examples/*

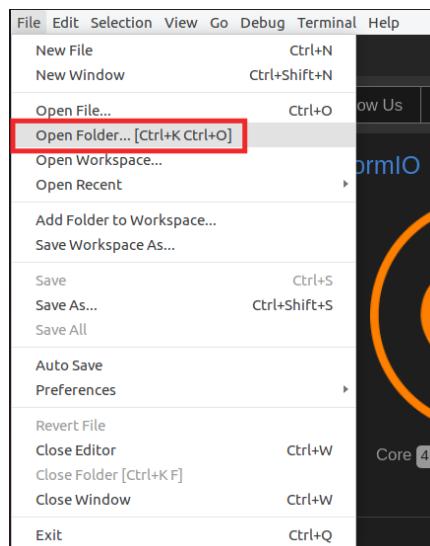


图81. 打开AL_Operations.S示例

4. 选择目录*AL_Operations*（不要打开，只需将其选中），然后单击“OK”（确定）。该示例将在PlatformIO中打开。
5. 打开文件*platformio.ini*。通过编辑以下行建立到在第一步（*Vrvfpgasim*）中生成的RVfpgaSim仿真二进制文件的路径（参见图82）。

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

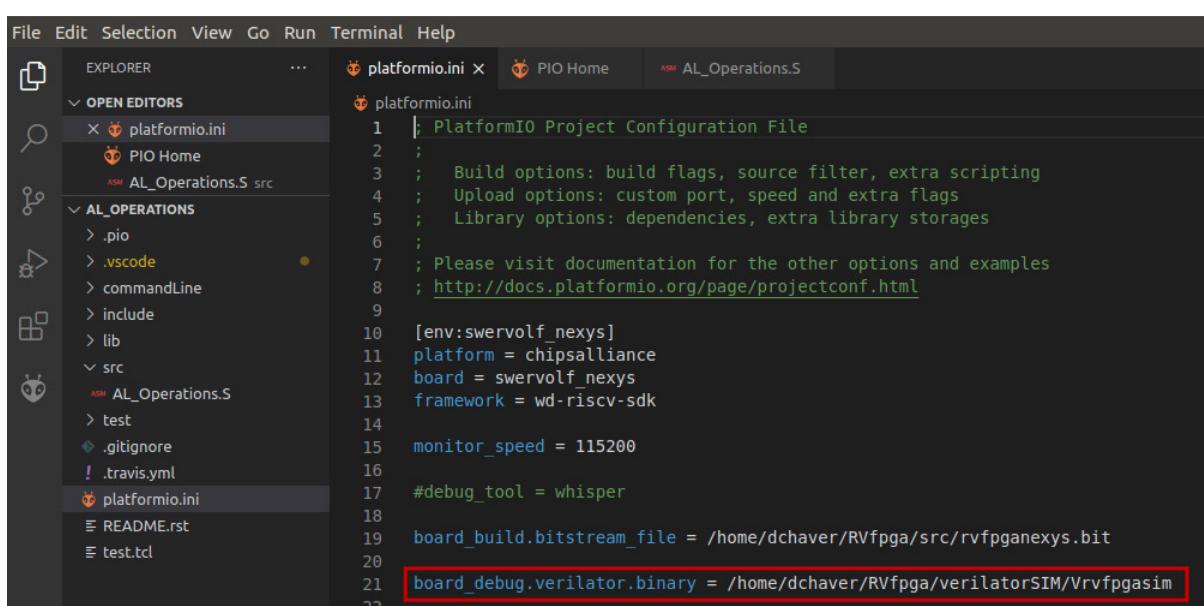


图82. PlatformIO初始化文件: platformio.ini

Windows: 在Windows中，RVfpgaSim仿真可执行文件称为*vrvfpgasim.exe*。因此：

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

6. 单击左侧功能区菜单中的PlatformIO图标来运行仿真，然后展开“Project Tasks”（项目任务）→ env:swervolf_nexys → “Platform”（平台），并单击“Generate Trace”（生成轨迹），如图83所示。

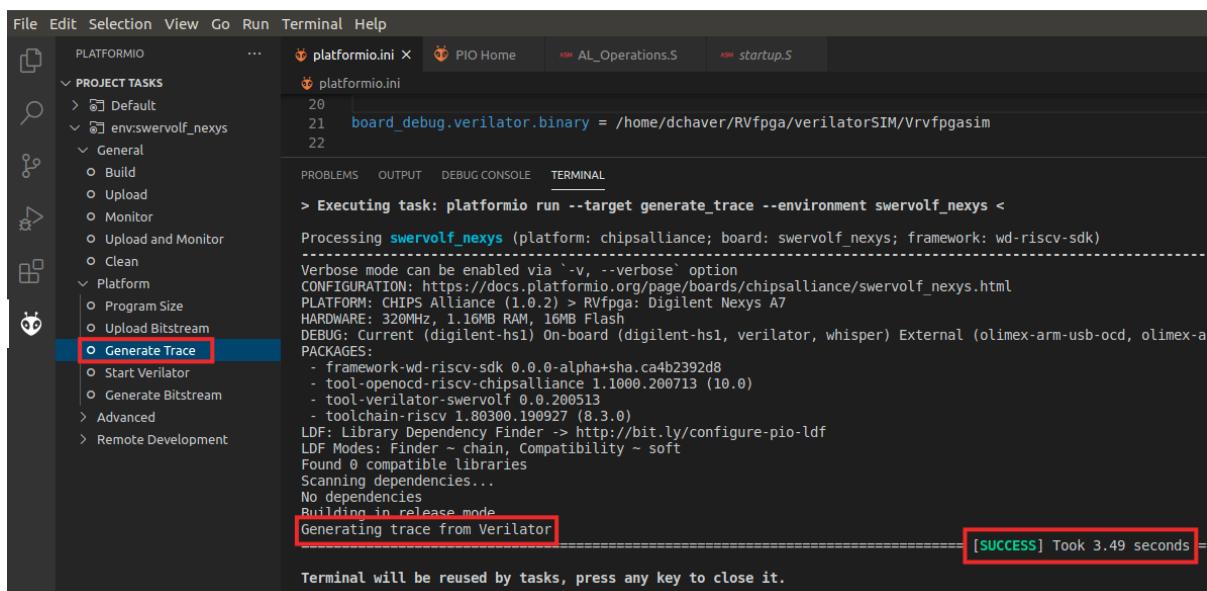


图83. 通过Verilator生成轨迹

或者，可以通过PlatformIO终端窗口生成轨迹。为此，请单击  按钮（PlatformIO: New Terminal”（PlatformIO: 新建终端）按钮，该按钮位于PlatformIO窗口的底部）打开一个新的终端窗口，然后在PlatformIO终端中输入（或复制）以下命令：pio run --target generate_trace

7. 上一步之后几秒钟，文件trace.vcd应已在 `[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys` 内生成，可以使用GTKWave将其打开。

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

WINDOWS: 已下载的文件夹gtkwave64包括一个称作 `gtkwave.exe` 的应用程序，该应用程序位于 `bin` 文件夹内。双击该应用程序启动GTKWave。在应用程序的顶部，单击“File”（文件） – “Open New Tab”（打开新选项卡），然后打开在文件夹 `[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys` 中生成的 `trace.vcd` 文件。

在GTKWAVE中分析仿真轨迹：

8. 现在，将添加时钟、指令和寄存器信号。在GTKWave的左上方窗格中，展开SoC的层级，以便可以将信号添加到图形中。将层级展开为“TOP”（上层）→ `rvfpgasim` → `swervolf` → `swerv_eh1` → `swerv`，然后单击模块 `ifu`（如图84中突出显示部分所示），

选择信号 **clk**（用于内核的时钟），并将其拖到右侧的白色“Signals”（信号）窗格或黑色“Waves”（波形）窗格中。

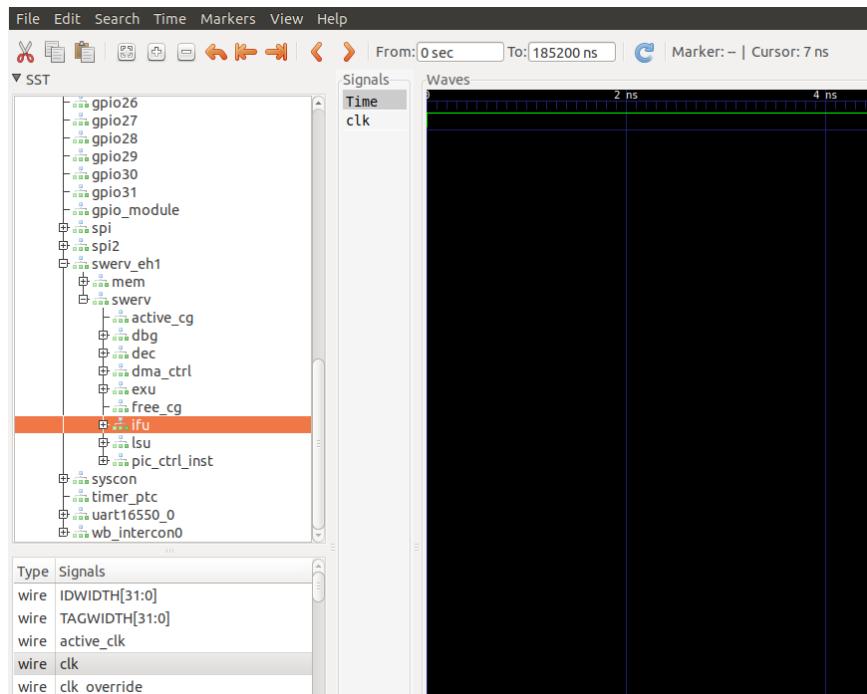


图84. 将信号 **clk** 加到图形上

9. 放大几次，以便可以查看时钟信号的变化（图85）。

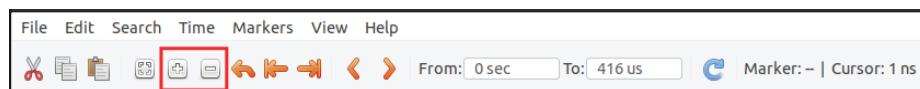


图85. 放大

10. 现在添加信号，这些信号显示在双路超标量RISC-V内核中每一路执行的指令。在同一模块（**ifu**）中查找信号 **ifu_i0_instr[31:0]** 和 **ifu_i1_instr[31:0]**（图86），然后将这些信号拖到黑色“Waves”（波形）窗格中。前缀 **ifu** 表示取指单元，**i0** 表示超标量通路0，**i1** 表示超标量通路1；**instr[31:0]** 表示32位指令。

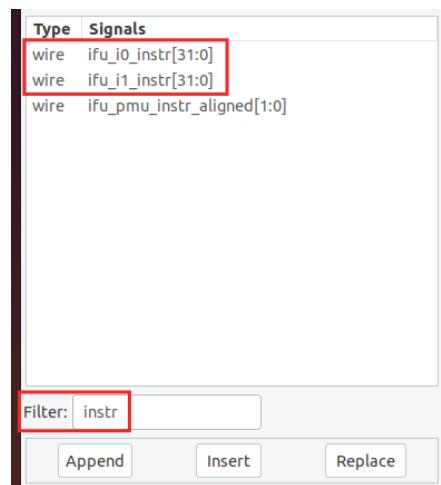


图86. 将信号 **ifu_i0_instr[31:0]** 和 **ifu_i1_instr[31:0]** 添加到时序波形

11. 现在添加用于保存寄存器 t_3 （即，寄存器编号28， x_{28} ）值的信号。将**swerv**下的层级展开为**dec → arf → gpr_banks(0) → gpr(28)**，然后单击模块**gprff**（将按下图所示突出显示），选择信号**dout[31:0]**（用于显示**AL_Operations.S**示例中使用的寄存器 x_{28} 的内容）并将其拖动到黑色“Waves”（波形）窗格中（图87）。

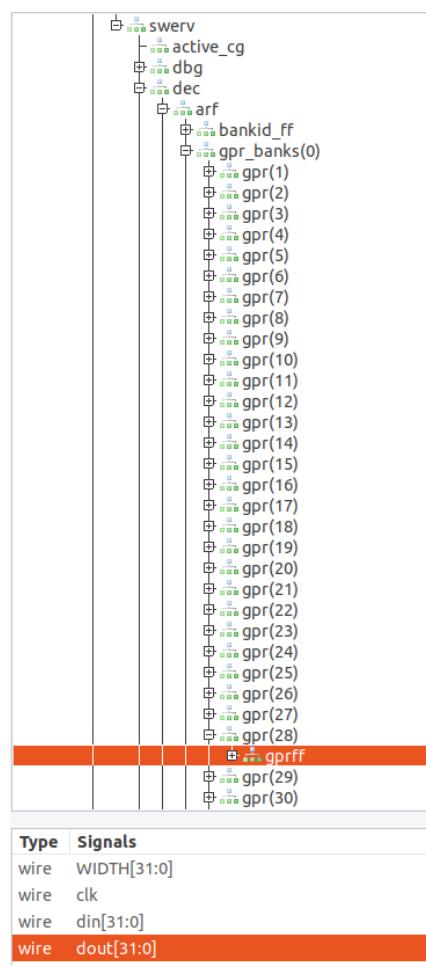


图87. 将信号**dout[31:0]**添加到图中

12. 在GTKWave中显示信号的另一种方法是使用.tcl文件。文件**test.tcl**位于[RVfpgaPath]/RVfpga/examples/AL_Operations中。打开该文件并对其进行分析。在每一行中，您将看到我们想要在图中显示的每个信号的路径和名称。

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

要在GTKWave上使用.tcl文件，只需单击“File – Read Tcl Script File”（文件 – 读取Tcl脚本文件）并选择[RVfpgaPath]/RVfpga/examples/AL_Operations/test.tcl文件即可。

13. 图88显示AL_Operations.S程序及其等效机器码。

```
# RISC-V assembly           # comment (t3 = x28)          # machine code
li    t3, 0x0               # t3 = 0                  # 0x000000E13
```

```

REPEAT:
    addi t3, t3, 6          # t3 = t3 + 6          # 0x006E0E13
    addi t3, t3, -1         # t3 = t3 - 1          # 0xFFFFE0E13
    andi t3, t3, 3          # t3 = t3 AND 3        # 0x003E7E13

    beq zero, zero, REPEAT  # Repeat the loop      # 0xFE000CE3
    nop                      # nop                  # 0x000000013

```

图88. AL_Operations.S及其等效机器码

现在查看信号随程序执行的变化情况。随着程序运行，预计指令和t3（寄存器x28）将变为图89所示的值：

```

li t3, 0x0                # t3 = 0          # 0x000000E13
REPEAT: addi t3, t3, 6      # t3 = 0 + 6 = 6  # 0x006E0E13
                    # t3 = 5          # 0xFFFFE0E13
                    # t3 = 1          # 0x003E7E13
                    # t3 = 6          # 0x006E0E13
                    # t3 = 7          # 0x006E0E13
                    # t3 = 6          # 0xFFFFE0E13
                    # t3 = 2          # 0x003E7E13
                    # t3 = 7          # 0x006E0E13
                    # t3 = 6          # 0xFFFFE0E13
                    # t3 = 2          # 0x003E7E13
                    # t3 = 7          # 0x006E0E13
                    # t3 = 6          # 0xFFFFE0E13
                    # t3 = 2          # 0x003E7E13
                    ...

```

图89. 指令流和AL_Operations执行期间寄存器t3 (x28) 的值

14. 在10100 ps附近放大，将在此时间内分析前两次循环迭代的三条算术逻辑指令的执行情况（图90）。前两条指令（li t3, 0x0 = 0x00000E13 和 addi t3, t3, 6 = 0x006E0E13）先取指，超标量RISC-V处理器的两路各一条（如信号ifu_i0_instr[31:0]和ifu_i1_instr[31:0]所示）。接下来的两条指令（addi t3, t3, -1 = 0xFFFFE0E13 和 andi t3, t3, 3 = 0x003E7E13）在下一个周期取指。最后两条指令（beq zero, zero, REPEAT = 0xFE000CE3 和 nop = 0x000000013）在接下来的下一个周期取指。

由于SweRV内核的9级流水线处理器和相关性，指令的结果在指令取指后的八个或更多周期才能展现出来。第一条和第二条指令取指完成8个周期后，由于第一条指令li t3, 0x0 (0x00000E13) 的原因，x28 (t3) 将变为0（原本就是0）。一个周期后，由于下一条指令addi t3, t3, 6 (0x006E0E13) 的原因，x28将更新为0x6。接下来，由于下一条指令addi t3, t3, -1 (0xFFFFE0E13) 的原因，x28将更新为5。最后，由于下一条指令andi t3, t3, 3 (0x003E7E13) 的原因，x28将更新为1。然后，取指以下两条指令：beq zero, zero, REPEAT (0xFE000CE3) 和nop (0x000000013)，将发生分支并重复循环。情况正如图89所预计的那样。使用类似的推理，可以分析第二次迭代，情况如图90和图89所示。

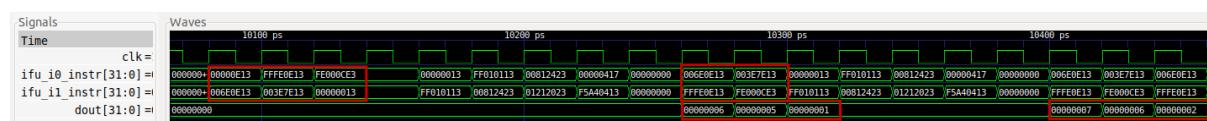


图90. 示例中三条算术逻辑指令的执行

8. 使用WHISPER进行仿真

Whisper (<https://github.com/chipsalliance/SweRV-ISS>) 是Western Digital开发的一款RISC-V指令集仿真器(ISS)，用于验证SweRV微处理器。借助这款仿真器，用户无需底层的RISC-V硬件即可运行RISC-V代码。利用Whisper，可以使用PlatformIO测试、运行和调试C或汇编程序，而无需Nexys A7 FPGA开发板。

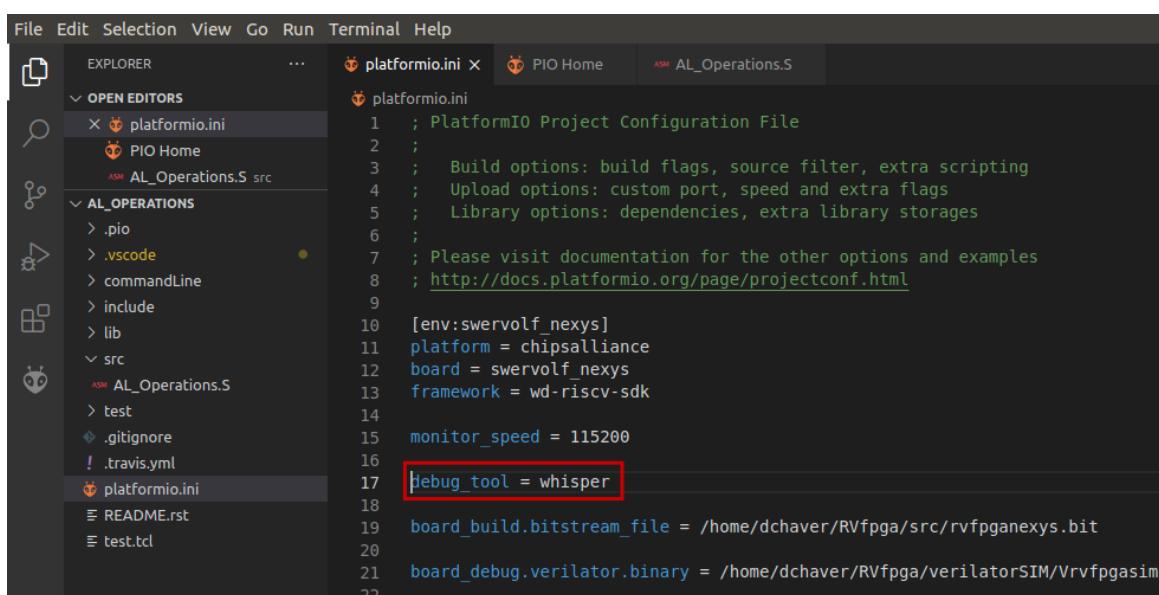
Windows: 本节中描述的所有说明都应适用于Windows（我们要感谢Jean-François Monestier，他率先将Whisper移植到Windows：<https://jean-francois.monestier.me/porting-western-digital-serv-iss-to-windows/>）。请注意，弹出窗口可能会要求用户允许Whisper通过Windows防火墙。

macOS: 本节中描述的所有说明也适用于macOS。

使用命令行和使用Eclipse或PlatformIO之类的IDE（集成开发环境）均可执行Whisper。在本部分中，我们演示一个示例以展示如何在PlatformIO中使用Whisper仿真程序。之后，可以使用此处所述的步骤来仿真其他程序。

我们首先使用Whisper ISS来仿真AL_Operations，它是在第6部分中执行和调试的第一个简单汇编程序（参见图44）。按照下面的步骤在Whisper上运行和调试此代码：

1. 打开VSCode（和PlatformIO）。在顶部菜单栏上，单击“File”（文件）→“Open Folder”（打开文件夹），导航至目录[RVfpgaPath]/RVfpga/examples/，选择（但不打开）目录AL_Operations，然后单击“OK”（确定）。
2. 单击“File”（文件）→“Open File”（打开文件），双击[RVfpgaPath]/RVfpga/examples/AL_Operations/platformio.ini，然后通过取消第17行的注释，将whisper设置为调试工具（参见图91）。保存文件（按Ctrl-S）。



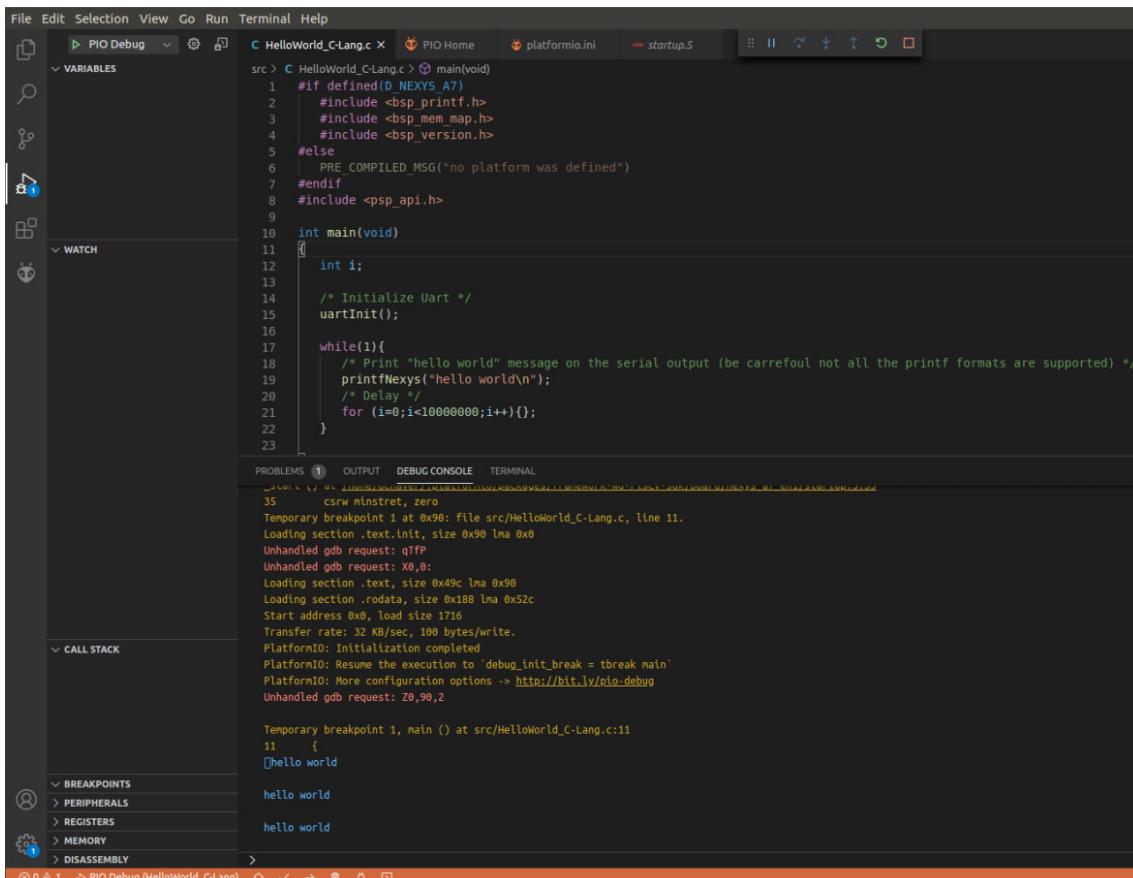
```

File Edit Selection View Go Run Terminal Help
EXPLORER ... platformio.ini × PIO Home ASK AL_Operations.S
OPEN EDITORS
platformio.ini
PIO Home
ASK AL_Operations.S src
AL_OPERATIONS
.pio
.vscode
commandLine
include
lib
src
ASK AL_Operations.S
test
.gitignore
.travis.yml
platformio.ini
README.rst
test.tcl
platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter, extra scripting
4 ; Upload options: custom port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ;
7 ; Please visit documentation for the other options and examples
8 ; http://docs.platformio.org/page/projectconf.html
9
10 [env:swervolf_nexys]
11 platform = chipsalliance
12 board = swervolf_nexys
13 framework = wd-riscv-sdk
14
15 monitor_speed = 115200
16
17 debug_tool = whisper
18
19 board_build.bitstream_file = /home/dchaver/RVfpga/src/rvfganexys.bit
20
21 board_debug.verilator.binary = /home/dchaver/RVfpga/verilatorSIM/Vrvfgasim
22

```

图91. 取消第17行注释

3. 照常通过依次单击  和  来启动调试器
4. 现在，可以像在第6.B部分中一样调试该程序，但这次该程序是在Whisper上而不是在Nexys A7 FPGA开发板上通过仿真运行。
5. 如果程序在Whisper中使用*printfNexys*函数，例如HelloWorld_C-Lang示例（第6.F部分），则不应打开PlatformIO串行监视器，因为消息将显示在调试控制台中（参见图92）。



The screenshot shows the PlatformIO IDE interface during a debug session on the Whisper board. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the project is "HelloWorld_C-Lang.c X" and the target is "PIO Debug". The main workspace shows the C code for the main() function, which initializes a UART and prints "hello world" to the serial output. The DEBUG CONSOLE tab at the bottom displays the output "hello world". The left sidebar contains tabs for VARIABLES, WATCH, CALL STACK, BREAKPOINTS, PERIPHERALS, REGISTERS, MEMORY, and DISASSEMBLY.

```

src > C HelloWorld_C-Lang.c > main(void)
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 <bsp_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 carrefoul not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<10000000;i++);
22     }
23

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

35 cswr_minstret, zero
Temporary breakpoint 1 at 0x00: file src>HelloWorld_C-Lang.c, line 11.
Loading section .text.init, size 0x90 lma 0x0
Unhandled gdb request: qTfP
Unhandled gdb request: X0,0:
Loading section .text, size 0x49c lma 0x90
Loading section .rodata, size 0x188 lma 0x52c
Start address 0x0, load size 1716
Transfer rate: 32 KB/sec, 100 bytes/write.
PlatformIO: Initialization completed
PlatformIO: Resume the execution to 'debug_init_break = tbreak main'
PlatformIO: More configuration options -> http://bit.ly/pio-debug
Unhandled gdb request: Z0,90,2

Temporary breakpoint 1, main () at src>HelloWorld_C-Lang.c:11
11 {
hello world

hello world

```

图92. 在Whisper中执行HelloWorld_C-Lang示例

9. 附录

以下附录显示了如何在Linux中使用开源RISC-V工具链和OpenOCD（而不是PlatformIO），如何在Windows中安装使用PlatformIO下载比特流的驱动程序，如何在Windows和Mac OS计算机上安装Verilator和GTKWave，以及如何使用Vivado对RVfpgaNexys进行编程。表10列出了本“RVfpga入门指南”中所有可用的附录。

表10. 附录列表

附录	说明	操作系统
A	在Ubuntu 18.04中将开源RISC-V工具链和OpenOCD用于RVfpga	Linux
B	在Windows中安装使用PlatformIO所需的驱动程序	Windows
C	在Windows中安装Verilator和GTKWave	Windows
D	在macOS中安装Verilator和GTKWave	macOS
E	使用Vivado将RVfpgaNexys下载到FPGA上	Windows 和Linux
F	在工业物联网应用中使用RVfpga	所有

附录A适用于想要使用开源gcc/gdb工具和OpenOCD在本机编译和运行/调试程序的用户。但是，建议**RVfpga**用户使用**PlatformIO**来代替，如本入门指南所述。

Windows用户必须遵循附录B和附录C中的说明。附录B中说明了如何下载驱动程序以便Windows系统可以使用PlatformIO来下载程序，以及如何将RVfpgaNexys下载到Nexys A7 FPGA开发板上。附录C展示了如何安装Verilator和GTKWave以便Windows用户可以仿真RVfpgaSim。

macOS用户必须遵循附录D中的说明以使用Verilator和GTKWave来仿真RVfpgaSim。

建议使用**PlatformIO**将**RVfpgaNexys**系统下载（由bit文件rvfganexys.bit定义）到Nexys A7 FPGA开发板上。该bit文件(rvfganexys.bit)可以由Vivado或PlatformIO生成。此外，还可以使用Vivado将RVfpgaNexys系统下载到Nexys A7 FPGA开发板上，如附录E所述。但是，不建议使用Vivado将RVfpgaNexys下载到开发板上 – 特别是**Windows**用户，因为此操作需要不断地交换驱动程序。

附录A：在Ubuntu 18.04中使用开源RISC-V工具链和OpenOCD

尽管我们建议使用PlatformIO，但在本部分中，我们将说明如何安装、运行和使用开源RISC-V工具链，如何使用OpenOCD将RVfpgaNexys下载到Nexys A7 FPGA开发板上，以及如何使用gdb在RVfpgaNexys上运行和调试程序。该工具链由gnu编译器、调试器、汇编器等组成。我们展示了如何在Ubuntu 18.04操作系统（OS）上安装RISC-V工具链和OpenOCD，但是此过程也应适用于其他Linux发行版本。这些说明假定使用全新的Ubuntu系统。

如果使用的是PlatformIO，则无需执行以下步骤，如本指南前面所述。建议使用PlatformIO、Vivado和Verilator或Whisper来运行、调试和仿真RISC-V程序，但是，对于有兴趣使用开源RISC-V工具链和OpenOCD来替代PlatformIO和Vivado硬件管理器的任何用户，提供了以下说明。

I. 在Linux Ubuntu OS上进行安装

在本部分中，我们介绍了如何在Ubuntu 18.04计算机中安装RISC-V工具链、OpenOCD和Whisper。这些工具仅替代PlatformIO；如本GSG第5部分所述，仍然需要安装Vivado和Verilator。

RISC-V工具链

在此我们展示如何在您的计算机上安装完整的RISC-V工具链，即gnu编译器、调试器等。

RISC-V International在以下位置提供了安装说明：<https://github.com/riscv/riscv-gnu-toolchain>。这些说明总结如下。

注：安装RISC-V工具链和OpenOCD可能需要几个小时，大部分时间花在了等待工具链下载、编译和安装上

在终端输入以下内容（该过程可能需要一个多小时，但大部分时间都花在了等待程序下载和安装上）：

- 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，请使用sudo make -j\$(nproc)，此指令可以显著缩短编译时间)
- export PATH=\$PATH:/opt/riscv/bin (在系统中更改路径)

OpenOCD

OpenOCD是一个开放的片上调试器，允许用户对嵌入式目标设备进行编程和调试。按照下面的步骤将RISC-V OpenOCD安装到计算机上：

- 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

按照下面的步骤将**Whisper**安装到计算机上（有关说明，请参见：

<https://github.com/chipsalliance/SweRV-ISS>, 下面也对此进行了总结）：

```
➤ 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 (根据需要替换
[RVfpgaPath] )。
```

II. 通过**OpenOCD**在**RVfpgaNexys**上执行程序

步骤A. 将RVfpgaNexys**下载（图25）到**Nexys A7****

1. 转到包含**RVfpgaNexys** bit文件的项目目录：

```
cd [RVfpgaPath]/RVfpga/src
```

2. 使用**OpenOCD**将**RVfpgaNexys**下载到开发板中：

```
riscv-openocd -c "set BITFILE rvfpganexys.bit" -f
OtherSources/ConfigFiles/swervolf_nexys_program.cfg
```

步骤B. 执行LedsSwitches**, 该程序读取开关并将其状态呈现在**LED**上**

3. 转到**LedsSwitches/commandLine**目录：

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

在该目录中，可找到用于编译源代码的**Makefile**、链接脚本、**python**脚本以及**LedsSwitches.S**程序。

4. 编译**.elf**文件：

```
make clean
make LedsSwitches.elf
```

5. 将**OpenOCD**连接到**SoC**：

```
riscv-openocd -
f ../../src/OtherSources/ConfigFiles/swervolf_nexys_debug.cfg
```

OpenOCD开始运行后，将出现几条消息，其中一条显示：

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

6. 打开一个新终端，转到程序目录

(cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/CommandLine)，并运行以下命令：

```
telnet localhost 4444
```

然后，在telnet连接中，输入：

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

这三条命令的作用如下：(1)将LedsSwitches.elf程序加载到RVfpgaNexys上，(2)将程序计数器（Program Counter，PC）设置为0（程序第一条指令的地址存储单元）(3)恢复执行。

该程序将开始在RVfpgaNexys上运行，RVfpgaNexys是已在步骤2中下载到Nexys A7 FPGA开发板上的RISC-V SweRVolfX SoC。该程序使LED显示开关的状态。拨动开关时，LED会立即改变以反映开关的值。

步骤C. 对执行简单算术逻辑运算的AL_Operations_CommandLine程序进行调试

现在，我们展示如何使用OpenOCD和gdb调试另一个程序（AL_Operations_CommandLine）。

7. 保持OpenOCD连接断开状态（参见第5步）。

8. 在运行telnet的其他终端中（从第6步开始），通过输入以下内容断开telnet连接：

```
exit
```

9. 转到包含AL_Operations/commandLine的项目目录：

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

在该目录中，可找到用于编译源代码的Makefile、链接脚本、python脚本以及AL_Operations.S程序。

10. 编译.elf文件：

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

11. 然后，在此终端中，通过输入以下内容来启动gdb：

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

12. 在gdb控制台中，输入：

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

这将连接到OpenOCD，并将AL_Operations.elf程序装载到存储器中。

13. 现在即可调试程序。输入以下序列并分析输出：

i. disas 0,20

该序列显示从地址0到20（不包括地址20）的汇编代码。请参见图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.
```

图93. 查看汇编程序

ii. i r t3

该序列显示寄存器t3的内容。或者，可以输入更长的版本：info reg t3。请参见图94。

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

图94. 打印寄存器t3中包含的值

iii. i r pc

该序列显示程序计数器（pc）的内容。请参见图95。

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

图95. 打印指向第一条指令的寄存器PC中包含的值

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

stepi使程序执行一条指令。i r t3随后显示寄存器t3的内容。请参见图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
```

图96. 逐一执行几条指令并查看t3寄存器

使用gdb完成对程序和寄存器的调试和探究后，通过在gdb终端中输入quit来退出gdb，在OpenOCD终端中输入^C来退出OpenOCD。

III. 使用Verilator在RVfpgaSim上仿真程序

1. 在Ubuntu中打开终端
2. 在终端窗口中，通过执行以下命令来生成仿真器二进制文件：

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

文件*Vrvfpgasim*（RVfpgaSim仿真二进制文件）应在目录
[RVfpgaPath]/RVfpga/verilatorSIM内生成。

3. 转到包含示例程序的文件夹：

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

4. 创建用于仿真的十六进制程序。

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

5. 执行仿真器。

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

几秒钟后，通过在终端中输入^C停止仿真。文件*trace.vcd*应已生成，可通过GTKWave将其打开。

```
gtkwave trace.vcd
```

6. 按照第7部分中步骤8至12中提供的说明，将信号添加到图形中并进行分析。

IV. 在Whisper上仿真程序

1. 在Ubuntu中打开终端

2. 转到包含示例程序的文件夹:

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

3. 创建反汇编程序。

```
make AL_Operations.dis
```

4. 在编辑器中打开*AL_Operations.dis*。将看到如下内容:

```
<_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. 在交互模式下执行仿真器。

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

6. 调试程序。

```
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*完成对程序和寄存器的调试和探究后，通过在终端中输入*quit*来退出。

附录B：在Windows中安装使用PlatformIO所需的驱动程序

要下载Zadig可执行文件，请浏览至以下网站（参见图97）：

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

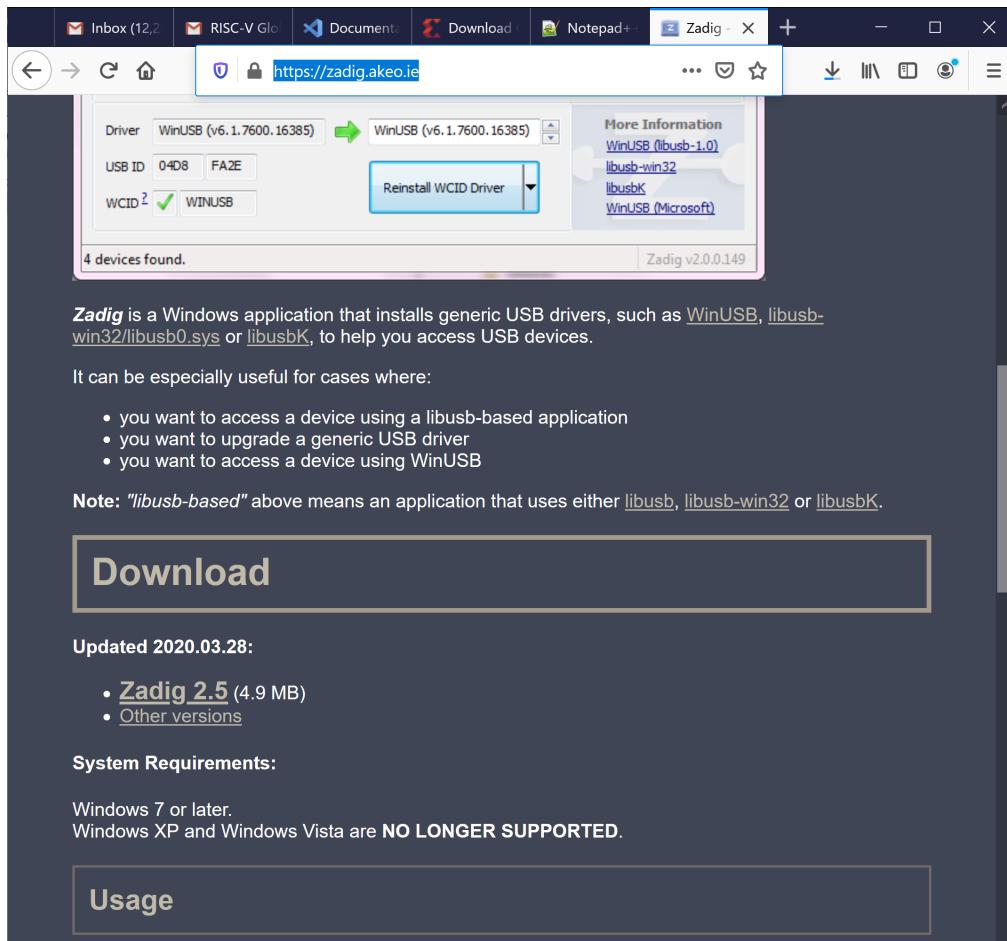


图97. 安装PlatformIO使用的Nexys A7开发板驱动程序

单击Zadig 2.5并保存可执行文件。然后运行该文件（zadig-2.5.exe），该文件位于其下载位置。此外，也可以在“Start”（开始）菜单中输入zadig来找到该文件。系统可能会询问您是否要允许Zadig对计算机进行更改以及是否允许其检查更新。两次都单击“Yes”（是）。

将Nexys A7开发板连接到计算机并接通。在Zadig中，单击“Options”（选项）→“List All Devices”（列出所有设备）（参见图98）。

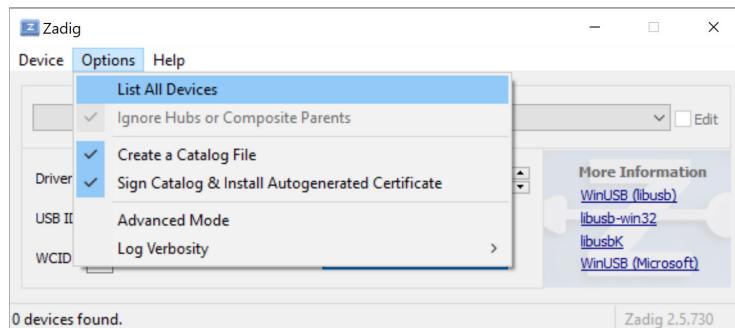


图98. Zadig中的“List All Devices”（列出所有设备）

如果单击下拉菜单，将看到列出的“Digilent USB Device (Interface 0)”（Digilent USB设备（接口0））和“Digilent USB Device (Interface 1)”（Digilent USB设备（接口1））。将仅为“Digilent USB Device (Interface 0)”（Digilent USB设备（接口0））安装新的驱动程序（参见图99）。

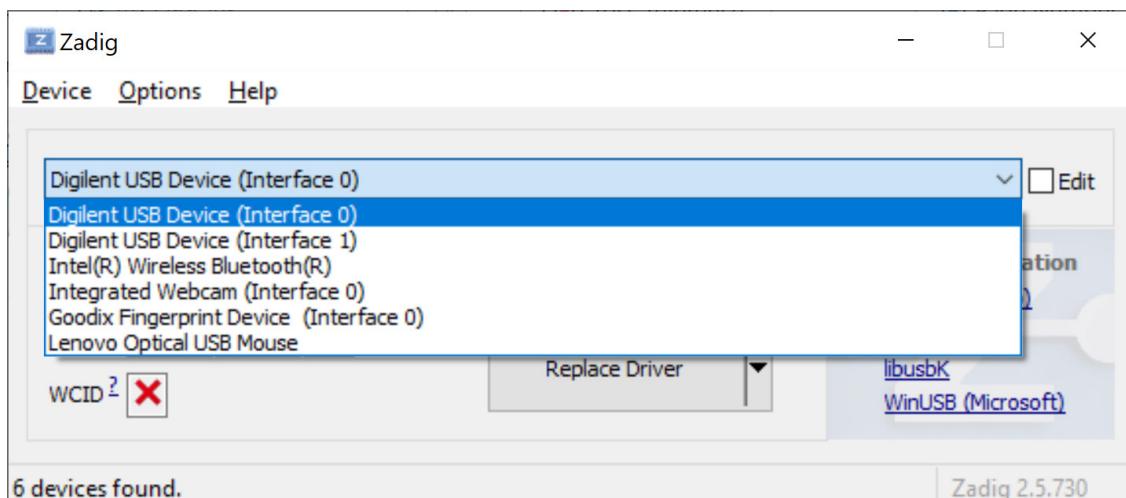


图99. 为Digilent USB设备（接口0）安装WinUSB驱动程序

现在，将使用WinUSB驱动程序替换FTDI驱动程序，如图100所示。单击“Digilent USB Device (Interface 0)”（Digilent USB设备（接口0））对应的“Replace Driver”（替换驱动程序）（或“Install Driver”（安装驱动程序））。将安装Nexys A7开发板的驱动程序，或者，如果先前已安装Vivado，则会用PlatformIO使用的WinUSB驱动程序替换Vivado使用的FTDI驱动程序。

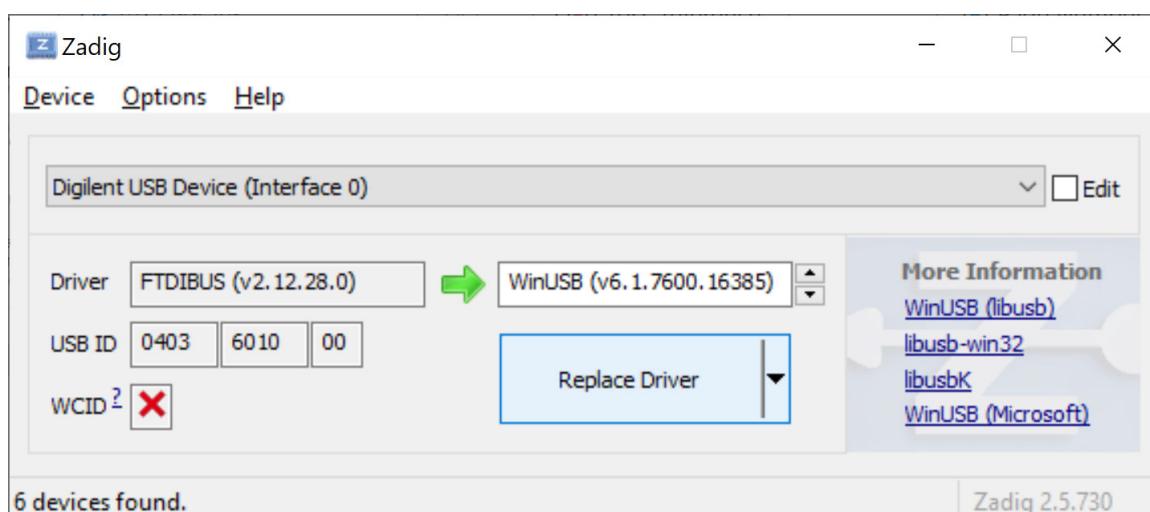


图100. 更换Nexys A7开发板的驱动程序

一段时间（通常为几分钟）后，Zadig将指示驱动程序已正确安装。单击“Close”（关闭），然后关闭Zadig窗口。

下次使用PlatformIO时，无需重新安装该驱动程序。但请注意，在**Windows**中，该驱动程序与**Vivado**不兼容。因此，不能再使用Vivado将bit文件下载到FPGA开发板上。如果要使用Vivado下载bit文件（不建议），则需要将驱动程序还原为随Vivado一起安装的原始驱动程序，如附录E中所述。

附录C：在Windows中安装Verilator和GTKWave

在本部分中，我们说明如何在Windows 10中安装Verilator和GTKWave。在Windows中，必须使用Cygwin来安装Verilator，因此我们首先说明如何安装此编程/运行环境。

Cygwin安装：

如其网页 (<https://www.cygwin.com>) 所述，Cygwin由GNU和开源工具组成，它们在Windows上提供类似于Linux发行版的功能。按照下面的步骤在Windows 10上安装Cygwin。

1. 导航至安装网页 (<https://cygwin.com/install.html>)，然后下载名为setup-x86_64.exe的安装文件（图101）。

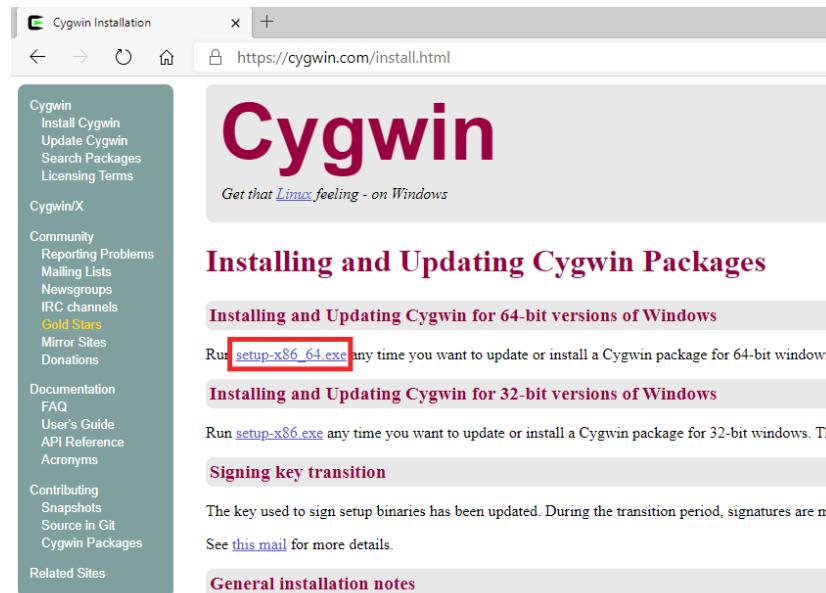


图101. Cygwin安装网页

2. 双击安装文件以在计算机上执行此文件（图102）。单击“Next”（下一步）几次，保留默认选项。安装程序会要求您“Choose a Download Site”（选择下载站点）（图103），可以选择其中任一选项。

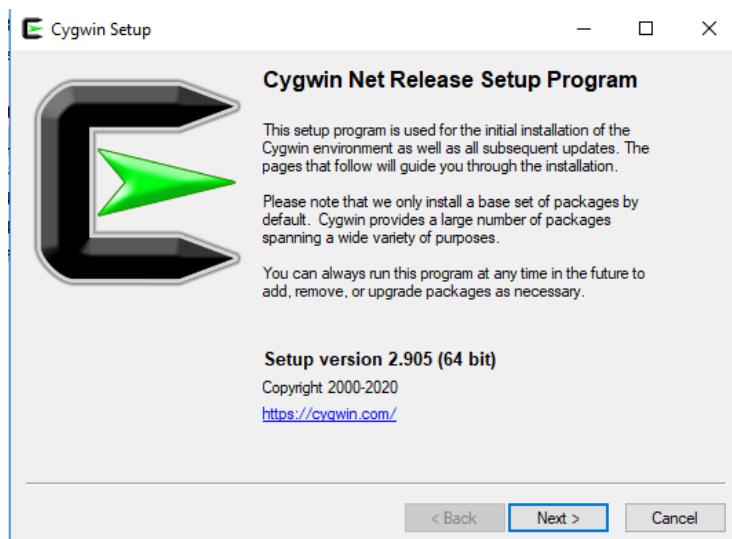


图102. Cygwin安装窗口

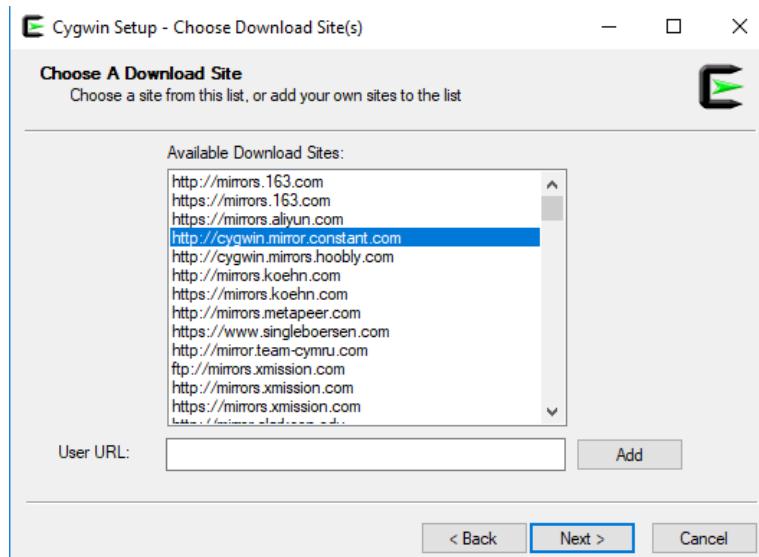


图103. 选择下载站点

3. 几个步骤后，将进入“**Select Packages**”（选择软件包）窗口（图104）。选择“**Full**”（完整）视图，如图104所示。

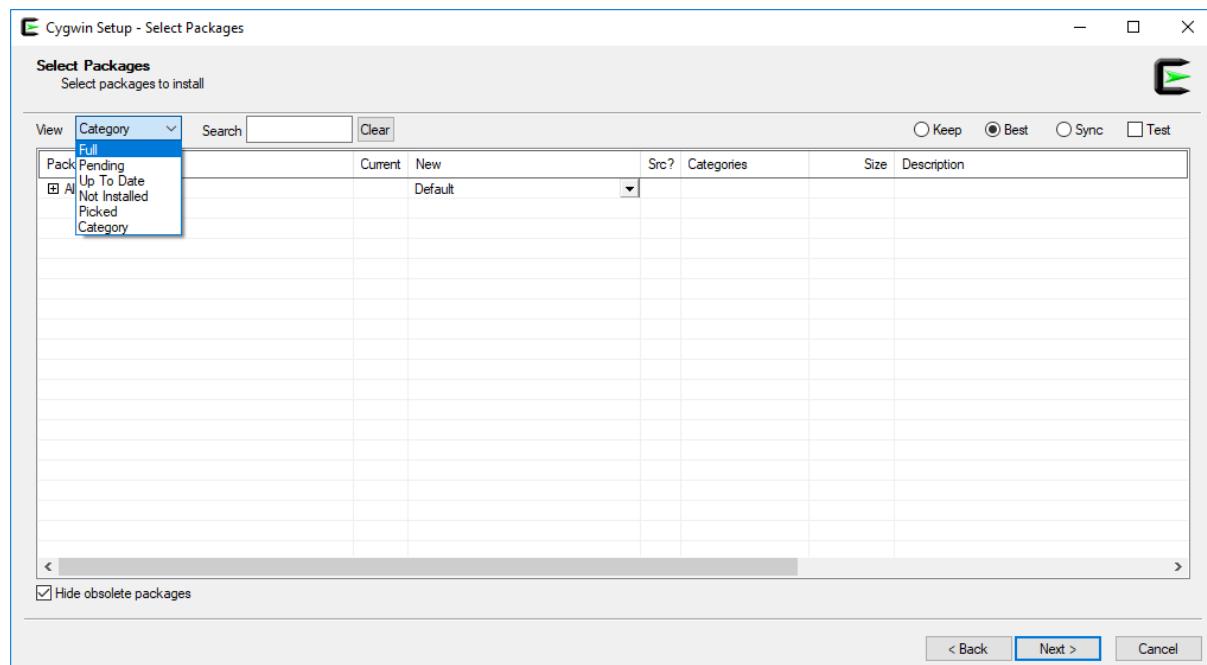


图104. “Select Packages”（选择软件包）窗口

4. 将显示可以安装的软件包的完整列表（图105）。在“Search”（搜索）框中，选择要安装的特定软件包。

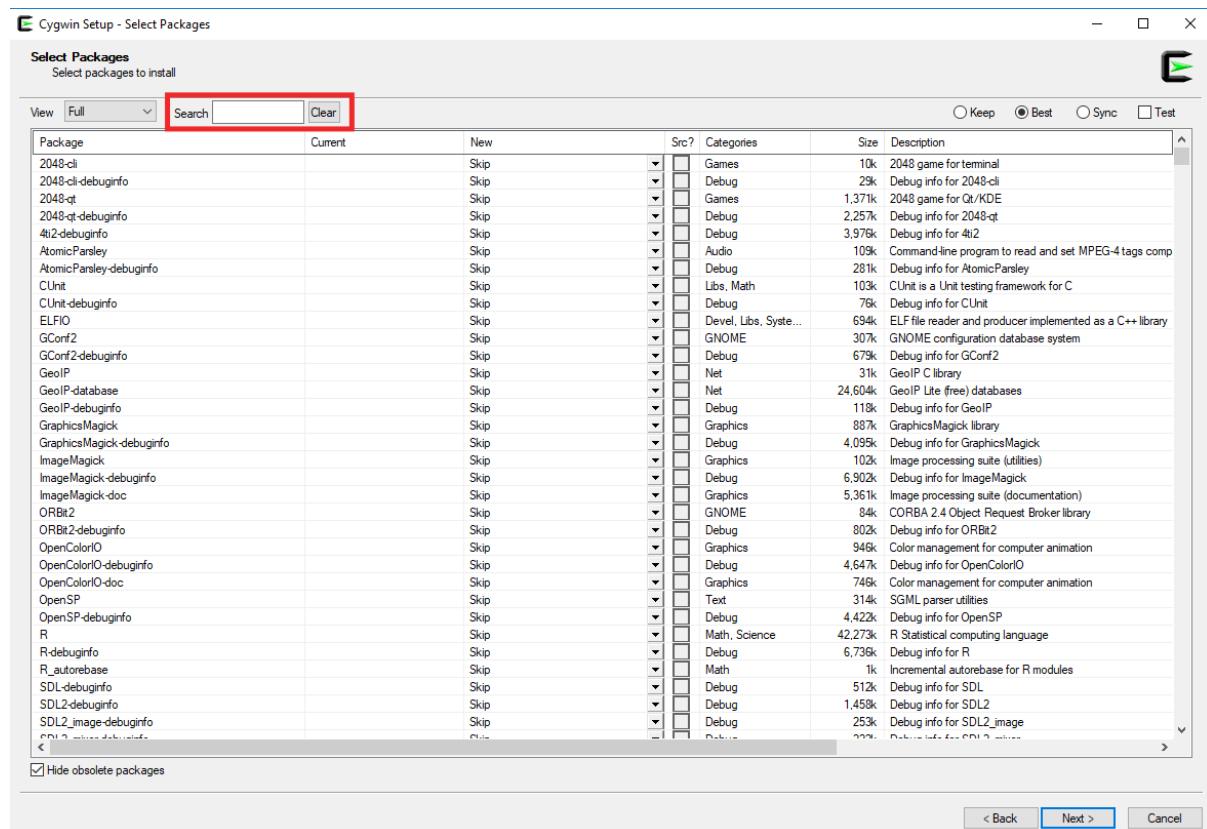


图105. “Select Packages”（选择软件包）窗口 – “Full”（完整）视图

为了能够编译Verilator并生成新的仿真器二进制文件，需要安装以下软件包：

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

在Cygwin安装中至少包括这些软件包。按照以下步骤逐一选择这些软件包（我们只显示列表中第一个软件包git的详细步骤；其他软件包的过程与此相同）：

- 在“**Search**”（搜索）框中查找git软件包（图106）。

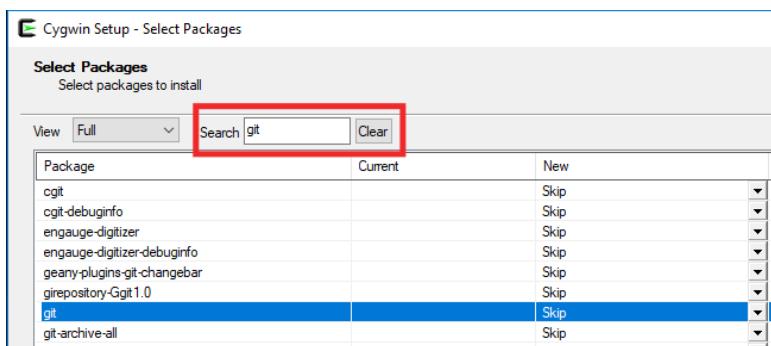


图106. 查找git软件包

- 在下拉菜单中选择最新版本并勾选对应的框（图107）。



图107. 选择最新版本并勾选对应的框

- 对以上列表中的其余软件包执行相同的操作。在大多数情况下，可以使用最新版本，但对于软件包gcc-core和gcc-g++，应当使用版本10.2.0，因为最新版本（在撰写本文档时为11.2.0）与Verilator存在一些冲突。
5. 选择九个软件包后，在随后的窗口中单击“**Next**”（下一步）以在Cygwin安装（对于安装过程，请参见图108，该过程可能需要几分钟）中包括这些软件包，然后单击“**Finish**”（完成）完成安装（图109）。

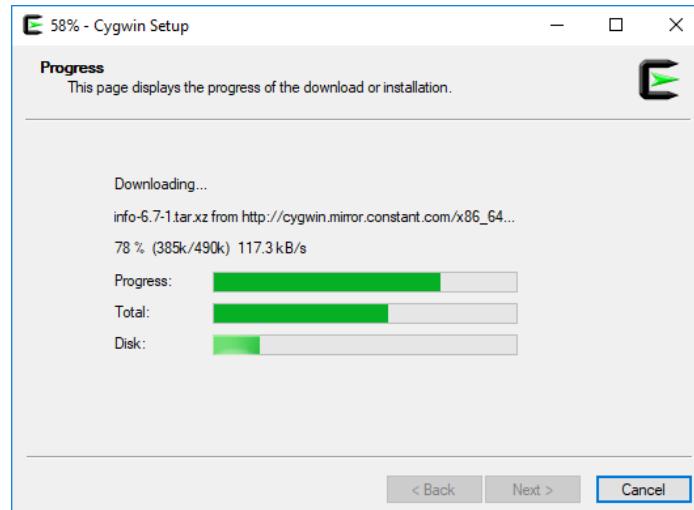


图108. Cygwin安装

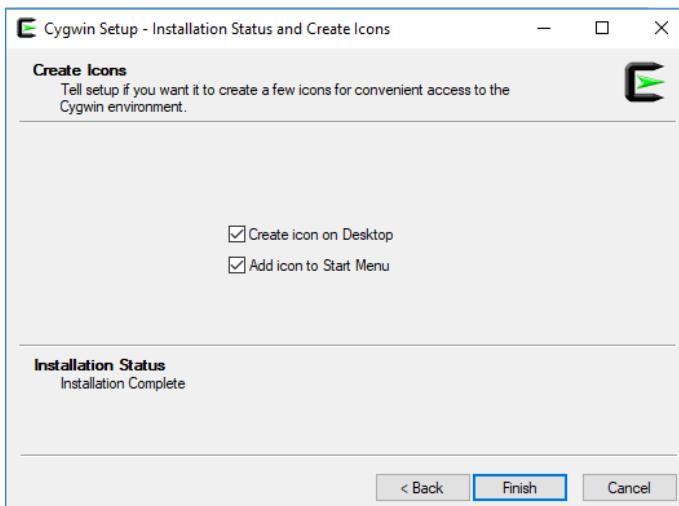


图109. 完成安装

6. 如果需要将某个软件包添加到Cygwin安装中，请对该软件包重复步骤2-5。

Verilator安装:

按照下面的步骤在Windows 10上安装Verilator。

1. 打开Cygwin终端（图110），可在Windows桌面或“Start”（开始）菜单中找到该终端。



图110. Cygwin终端

2. 按照以下步骤编译并安装Verilator。这可能需要一些时间（甚至几个小时），具体取决于计算机的速度：

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

GTKWave安装：

GTKWave可以作为预编译软件包从<https://sourceforge.net/projects/gtkwave/files/>下载。查找最新的Windows软件包（在编写本文档时，它称为**gtkwave-3.3.100-bin-win64**），然后下载该软件包并将其解压缩。可在**bin**文件夹内找到一个名为**gtkwave**的可执行文件，可以在Windows计算机中执行和使用该文件。

附录D：在macOS中安装Verilator和GTKWave

在本部分中，我们说明如何在macOS中安装Verilator和GTKWave。这些说明使用macOS Catalina 10.15.6进行了测试，但预计可以在其他版本的OS中使用。Homebrew (<https://brew.sh/>) 软件包管理器用于安装。对于macOS中另一个广泛使用的软件包管理器MacPorts，可以找到类似的步骤 (<https://www.macports.org/>)。

gcc安装：

为了使用Verilator编译新的仿真器，需要在系统中安装编译器工具链。有多种方法可用于安装有效的编译器工具链。下面介绍其中两种方法：

1. 安装XCode命令行工具。请注意，这将安装LLVM，但在安装后，`gcc`命令将始终可用。为此，请在“Terminal”（终端）窗口中输入以下命令：
 - `xcode-select -install`
2. 使用Homebrew安装`gcc`。输入如下命令：
 - `brew install gcc@9`

Verilator安装：

使用Homebrew安装Verilator就像在打开的终端中输入以下命令一样简单：

```
➤ brew install verilator
```

gtkwave安装：

再次，我们将使用Homebrew安装`gtkwave`。但这次是一个GUI macOS应用程序，因此我们需要使用`cask`。在打开的终端中输入以下命令：

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

安装完成后，“Application”（应用程序）文件夹中应出现`gtkwave.app`图标。为了通过命令行使用该应用程序，可能需要安装Perl的开关模块：

```
➤ cpan install Switch
```

附录E：使用Vivado将RVfpgaNexys下载到FPGA上

按照以下步骤使用RVfpgaNexys对FPGA编程（通过Vivado）：

WINDOWS: 在Windows中，在执行下面的步骤之前，需要按照本附录（附录E）末尾所述，将驱动程序还原为Vivado使用的驱动程序。

- 将Nexys A7开发板连接到计算机。
- 打开左上方的开关接通Nexys A7开发板。
- 打开Vivado 2019.2。
- 在Vivado中打开可用的硬件管理器，如图111中的突出显示部分所示。

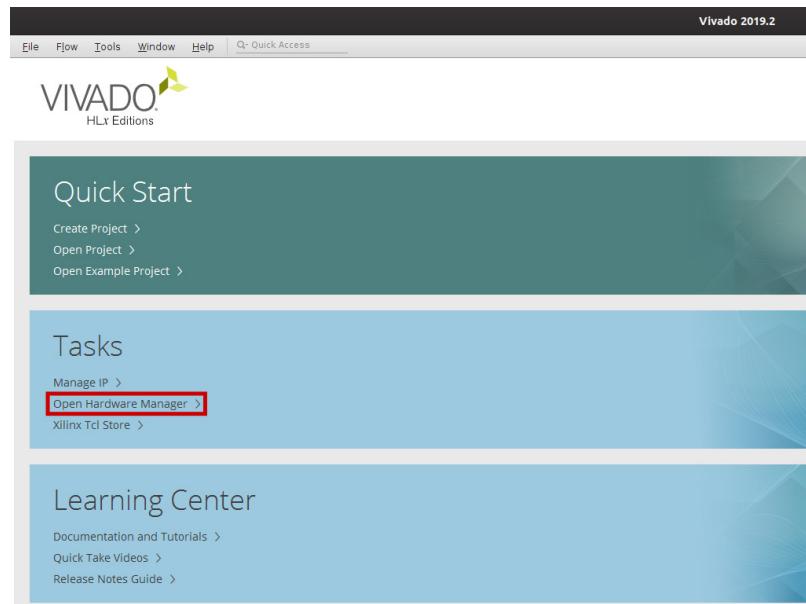


图111. 打开硬件管理器

- “Hardware Manager”（硬件管理器）将打开，并通知用户没有打开任何硬件目标。通过单击“Open target”（打开目标）–“Auto connect”（自动连接）打开目标（图112）。

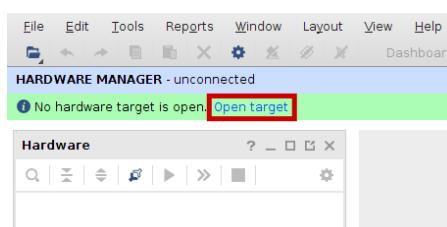


图112. 打开目标

- 选择“Program device”（编程设备），如图113所示。现在，需要将RVfpgaNexys加载到FPGA上。在新窗口中，从[RVfpgaPath]/RVfpga/src/rvfpganexys.bit中选择“Bitstream file”（比特流文件）。单击“Program”（编程）。

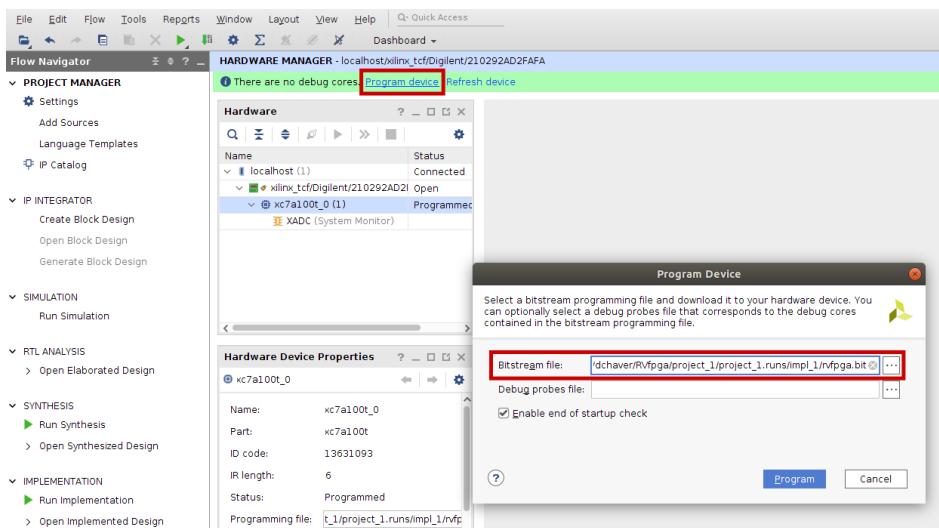


图113. 编程设备

- g. 几秒钟后，将使用RVfpgaNexys（以FPGA为目标的SweRVolfX SoC，参见图25）对FPGA进行编程。
- h. 最后，在Vivado中单击“Hardware Manager”（硬件管理器）窗格右上方的X按钮关闭“Hardware Manager”（硬件管理器）（图114），以便Vivado释放开发板。

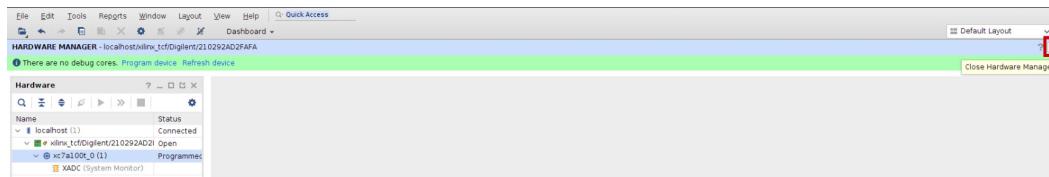


图114. 关闭硬件管理器

如何将驱动程序还原为Windows中Vivado使用的驱动程序

遗憾的是，在Windows中，Nexys A7 FPGA开发板的驱动程序在Vivado和PlatformIO上有所不同。强烈建议您使用PlatformIO对FPGA进行编程，如本GSG第5.A部分中所述。但是，如果要使用Vivado下载bit文件，必须将在附录B中安装的驱动程序还原为Nexys A7 FPGA开发板的Vivado（FTDI）驱动程序。为此，通过以下方式打开“Device Manager”（设备管理器）：单击“Start”（开始）菜单，在搜索框中输入“device manager”，然后单击“Device Manager”（设备管理器）（参见图115）。

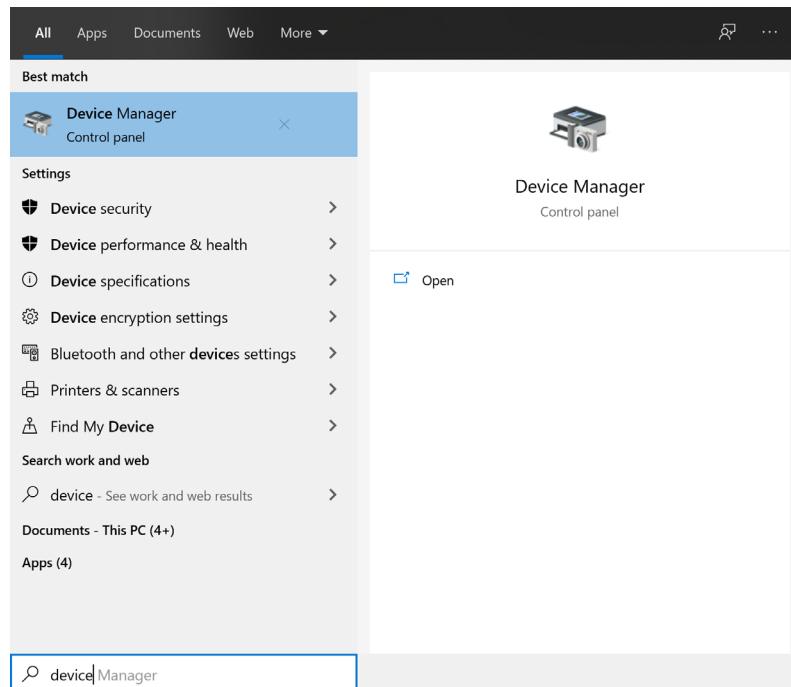


图115. 打开“Device Manager”（设备管理器）

接下来，展开“Universal Serial Bus Devices”（通用串行总线设备），右键单击“Digilent USB Device”（Digilent USB设备），然后选择“Properties”（属性）（参见图116）。

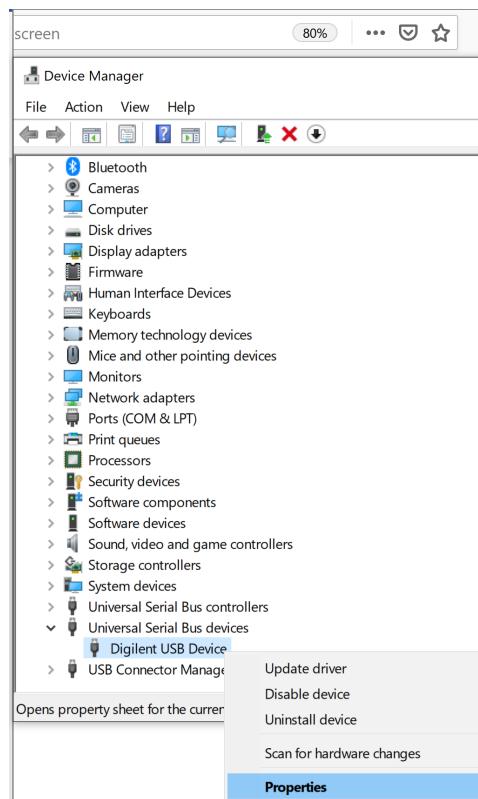


图116. 打开Digilent的Nexys A7 FPGA开发板的驱动程序属性

在“Properties”（属性）窗口中，单击“Driver”（驱动程序）选项卡，然后选择“Roll Back Driver”（恢复驱动程序）（参见图117）。

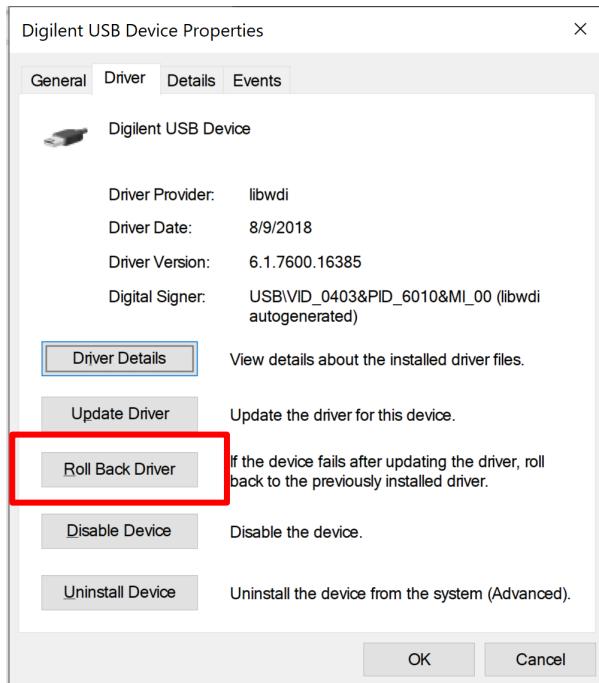


图117. 恢复驱动程序

将弹出一个窗口，询问恢复驱动程序的原因。选择一个原因，然后单击“**Yes**”（是）（参见图118）。

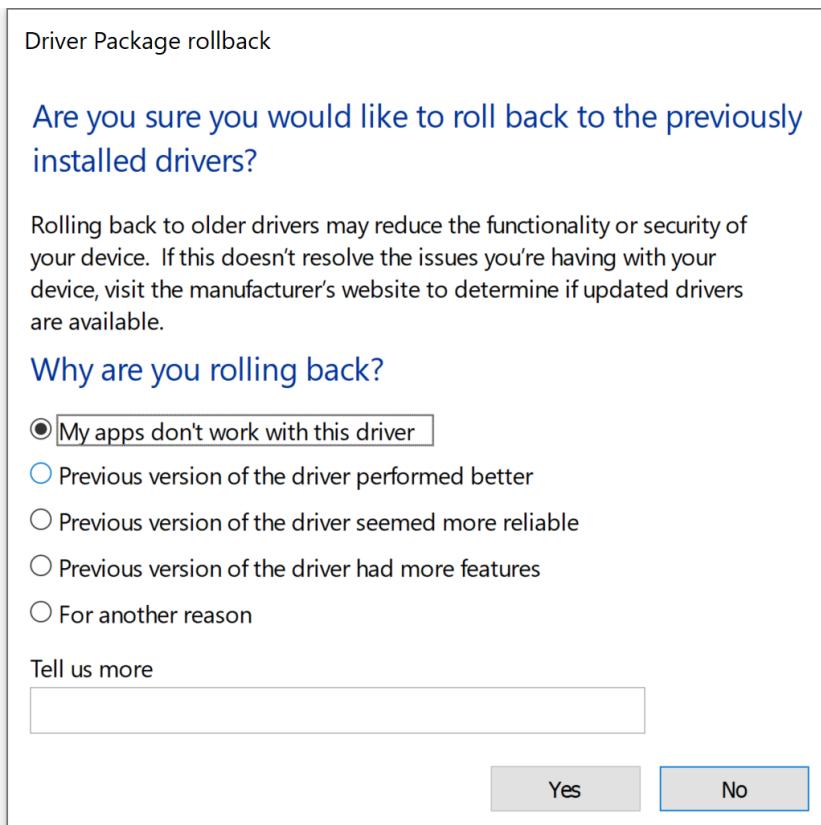


图118. 确认恢复

驱动程序恢复为先前的驱动程序后，“Driver Provider”（驱动程序提供方）应显示为FTDI（参见图119）。

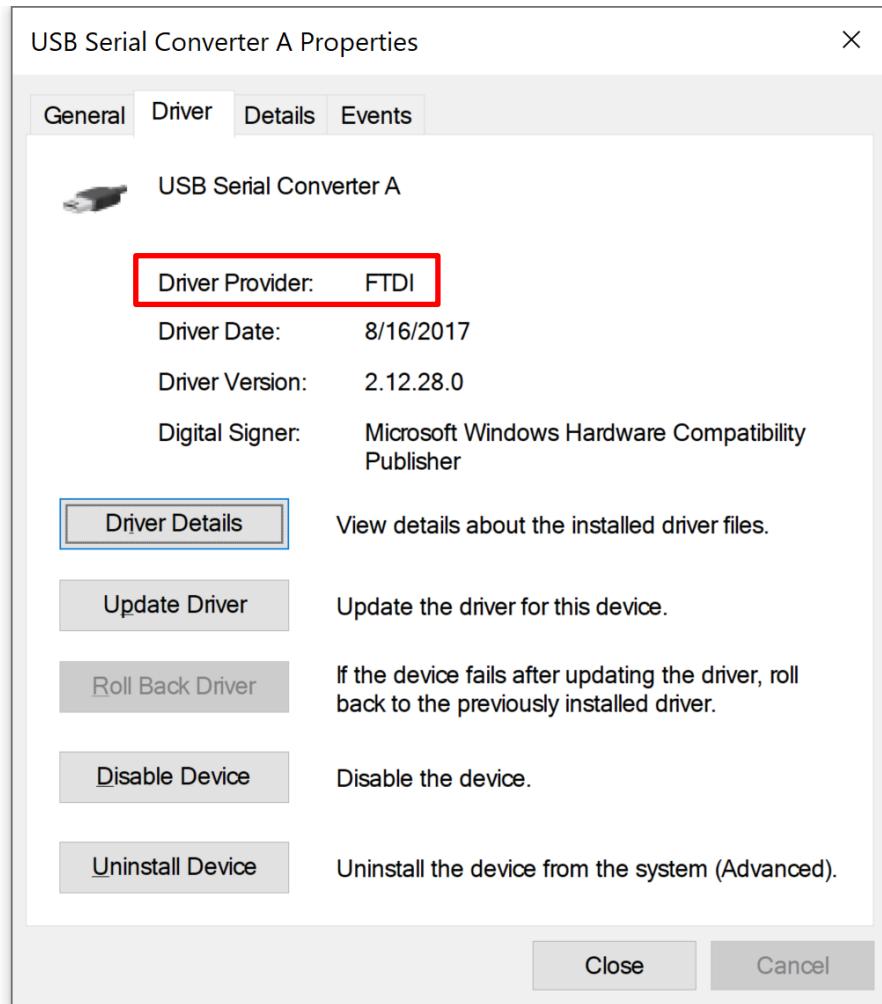


图119. FTDI驱动程序显示为提供的驱动程序

现在，可以使用Vivado将比特流文件加载到FPGA开发板上。但是，仍需要使用Zadig替换Nexys A7开发板的驱动程序，以便PlatformIO可以将程序下载到RVfpgaNexys上。因此，同样建议使用PlatformIO下载bit文件（而不使用Vivado） – 这样便不必一直交换驱动程序。

附录F：在工业物联网应用中使用RVfpga

2020年7月，马德里康普顿斯大学硕士研究生Daniel León González完成了他的硕士学位论文《适合工业物联网的专用RISC-V片上系统的FPGA实现》。这篇论文展示了RVfpga在实际工业物联网应用中的使用。我们在下面提供了项目摘要，可通过以下链接获得完整论文：
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。

适合工业物联网的专用RISC-V片上系统的FPGA实现

摘要：物联网的节点设备需要具备能源效率和成本效益，但在很多情况下它们并不需要很高的计算能力。但是在工业物联网环境中，这种情况明显不同，大量传感器的采用和事件的快速处理需要更高的处理能力。为了平衡这些要求并提供最佳解决方案，可以使用一种采用高效处理器以及高性能的全功能操作系统的定制开发节点。该项目利用Artix-7 FPGA解决了基于RISC-V处理器架构的原型物联网节点的硬件实现问题，并展示了为支持部署在定制边界路由器周围的星型网络中的概念验证应用而实现的定制SoC以及所需的Zephyr OS驱动程序的开发过程。在节点与ThingSpeak云平台之间可以发送和接收端到端消息。本文包括对现有RISC-V处理器实现的分析、所需元件的描述以及环境配置和项目设计的详细指南。