



THE IMAGINATION UNIVERSITY PROGRAMME

# RVfpga

## 시작 가이드

## 감사드립니다

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

## Sponsors and Supporters



### 저자

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

### 고문

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

### 도움주신 분

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

### ASSOCIATES

- Prof. José Ignacio Gómez (<https://www.linkedin.com/in/jos%C3%A9-ignacio-gomez-182b981/>)
- Prof. Christian Tenllado (<https://www.linkedin.com/in/christian-tenllado-31578659/>)
- Prof. Daniel León (<https://www.linkedin.com/in/danileon-ufv/>)
- Prof. Katzalin Olcoz (<https://www.linkedin.com/in/katzalin-olcoz-herrero-5724b0200/>)
- Prof. Alberto del Barrio (<https://www.linkedin.com/in/alberto-antonio-del-barrio-garc%C3%ADa->)

[1a85586a/](#)

- Prof. Fernando Castro (<https://www.linkedin.com/in/fernando-castro-5993103a/>)
- Prof. Manuel Prieto (<https://www.linkedin.com/in/manuel-prieto-matias-02470b8b/>)
- Prof. Francisco Tirado (<https://www.linkedin.com/in/francisco-tirado-fern%C3%A1ndez-40a45570/>)
- Prof. Román Hermida (<https://www.linkedin.com/in/roman-hermida-correa-a4175645/>)
- Cathal McCabe (<https://www.linkedin.com/in/cathalmccabe/>)
- Dan Hugo (<https://www.linkedin.com/in/danhugo/>)
- Braden Harwood (<https://www.linkedin.com/in/braden-harwood/>)
- David Burnett (<https://www.linkedin.com/in/david-burnett-3b03778/>)
- Gage Elerding (<https://www.linkedin.com/in/gage-elerding-052b16106/>)
- Brian Cruickshank (<https://www.linkedin.com/in/bcruiksh/>)
- Deepen Parmar (<https://www.linkedin.com/in/deepen-parmar/>)
- Thong Doan (<https://www.linkedin.com/in/thong-doan/>)
- Oliver Rew (<https://www.linkedin.com/in/oliver-rew/>)
- Niko Nikolay (<https://www.linkedin.com/in/roy-kravitz-4725963/>)
- Guanyang He (<https://www.linkedin.com/in/guanyang-he-5775ba109/>)
- Prof. Ataur Patwary (<https://www.linkedin.com/in/ataurpatwary/>)
- 

## 목차

|                                 |           |
|---------------------------------|-----------|
| 감사의 말 .....                     | 错误!未定义书签。 |
| 1. 소개 .....                     | 错误!未定义书签。 |
| 2. 빠른 시작 가이드 .....              | 错误!未定义书签。 |
| 3. RISC-V 아키텍처 개요 .....         | 错误!未定义书签。 |
| 4. RVFPGA 시스템 개요 .....          | 错误!未定义书签。 |
| 5. 소프트웨어 도구 설치 .....            | 错误!未定义书签。 |
| 6. RVfpgaNexys 실행 및 프로그래밍 ..... | 错误!未定义书签。 |
| 7. VERILATOR의 시뮬레이션 .....       | 错误!未定义书签。 |
| 8. WHISPER 시뮬레이션 .....          | 错误!未定义书签。 |
| 9. 부록 .....                     | 错误!未定义书签。 |

---

### Update History:

- Version 1.0 (Released November 2020):
  - o Original release of the RVfpga course.

- Version 1.1 (Released June 2021):
    - Added description of Labs 11-20 in Lab 00.
    - Updated SweRVolf version to 0.7.3.
    - Updated Verilator version to 4.106.
    - Added Boot ROM initialization program.
    - Added new Figure 1 and Table 1 in the GSG describing the RVfpga System
    - Added a UART exercise to Lab 10.
    - Fixed some typos.
- 





## 1. 소개

RISC-V FPGA (RVfpga 로도 사용됨)는 상용 RISC-V 프로세서를 FPGA (Field Programmable Gate Array)와 시뮬레이터를 목표로 한 다음, 이를 사용 및 확장하여 컴퓨터 아키텍처, 디지털 디자인, 임베디드 시스템 및 프로그래밍을 학습하기 위한 안내와 도구 및 Labs(실습예제)를 포함하는 패키지입니다.

이 RVfpga 시작 안내서에는 아래에서 간략하게 설명된 대로 다음과 같은 주요 섹션이 있습니다:

- **빠른 시작 가이드 (섹션 2)**
- **배경 및 개요**
  - RISC-V 아키텍처 (섹션 3)
  - RVfpga 시스템(섹션 4)
- **하드웨어에서 RVfpga 시스템 사용**
  - 소프트웨어 도구 설치 (섹션 5)
  - RVfpga 시스템 실행 및 프로그래밍 (섹션 6)
- **RVfpga 시스템 시뮬레이션**
  - HDL 시뮬레이터 인 Verilator 사용 (섹션 7)
  - Western Digital 의 명령어 세트 시뮬레이터 인 Whisper 사용 (섹션 8)
- **부록**
  - native RISC-V 툴체인 및 OpenOCD 사용 (부록 A)
  - PlatformIO 를 사용하기 위해 Windows 에 드라이버 설치 (부록 B)
  - Windows 에 Verilator 및 GTKWave 설치 (부록 C)
  - macOS 에 Verilator 및 GTKWave 설치 (부록 D)
  - Vivado 를 사용하여 RVfpga 시스템을 FPGA 에 다운로드 (부록 E)
  - 예제: 산업용 IoT 애플리케이션에서 RVfpga 사용 (부록 F)

빠른 시작 가이드 (섹션 2)는 RVfpga 에 필요한 최소 소프트웨어 설치를 설명하고 RVfpga 에서 간단한 예제 프로그램을 다운로드하고 실행하는 방법을 보여줍니다. RVfpga 시스템을 더 완전히 이해하려면 섹션 2 를 건너 뛰고 섹션 3 에서 시작하는 전체 가이드로 시작하십시오.

섹션 3에서는 RISC-V 컴퓨터 아키텍처에 대해 간략하게 소개합니다. 섹션 4에서는 RVfpga 시스템 (섹션 4.A – 4.C) 및 시스템을 구성하는 Verilog 파일의 구성 (섹션 4.D)에 대해 설명합니다. RVfpga 시스템은 Western Digital (WD)의 오픈 소스 RISC-V SweRV EH1 Core (<https://github.com/chipsalliance/Cores-SweRV>)를 기반으로 하고 있습니다. 그림 1 과 표 1 은 SweRV EH1 Core 에서 RVfpgaNexys 및 RVfpgaSim 까지 RVfpga 시스템의 계층 적 구성을 보여줍니다.

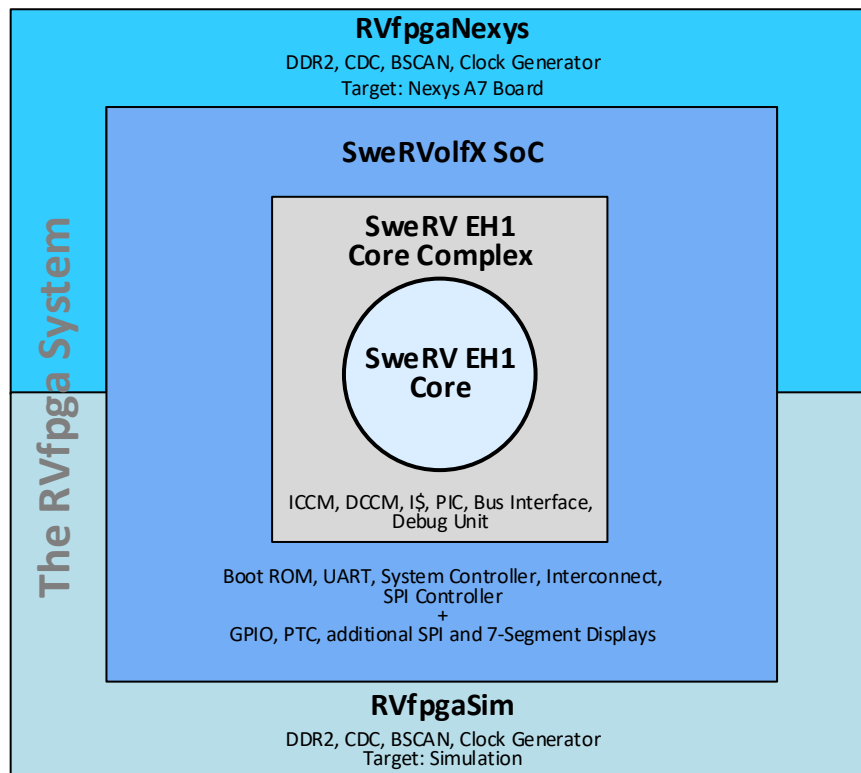


Figure 1. RVfpga System 계층 구조

Table 1. RVfpga System 계층 구조

| Name                                 | Description   |
|--------------------------------------|---|
| <b>SweRV EH1 Core</b>                | 오픈 소스 상용 RISC-V 코어는 Western Digital 로 개발되었습니다.<br>( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).  |
| <b>SweRV EH1 Core Complex</b>        | 추가 메모리 (ICCM, DCCM 및 명령 캐시), 프로그래밍 가능한 인터럽트 컨트롤러 (PIC), 버스 인터페이스 및 디버그 장치가 있는 SweRV EH1 코어.<br>( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).   |
| <b>SweRVolfX (Extended SweRVolf)</b> | RVfpga 과정에서 사용하는 System on Chip 으로 SweRVolf 의 확장입니다.<br><b>SweRVolf</b> ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ): SweRV EH1 Core Complex 를 중심으로 구축된 오픈 소스 SoC 입니다. 부트 ROM, UART 인터페이스, 시스템 컨트롤러, 상호 연결 (AXI Interconnect, Wishbone Interconnect 및 AXI-to-Wishbone 브리지) 및 SPI 컨트롤러를 추가합니다.<br>SweRVolfX: SweRVolf 에 GPIO, PTC, 추가 SPI 및 8 자리 7 세그먼트 디스플레이 용 컨트롤러 등 4 개의 새로운 주변 장치를 추가합니다. |
| <b>RVfpgaNexys</b>                   | SweRVolfX SoC 는 Nexys A7 보드 및 주변 장치를 대상으로합니다. DDR2 인터페이스, CDC (클럭 도메인 크로싱) 장치, BSCAN 로직 (JTAG 인터페이스 용) 및 클럭 생성기를 추가합니다.<br>RVfpgaNexys 는 SweRVolf 를 기반으로한다는 점을 제외하면 SweRVolf Nexys ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> )와 동일합니다.  |
| <b>RVfpgaSim</b>                     | 시뮬레이션을위한 테스트 벤치 래퍼 및 AXI 메모리가있는 SweRVolfX SoC.  |

|  |   |
|--|---|
|  | RVfpgaSim 은 SweRVolf 를 기반으로한다는 점을 제외하면 SweRVolf sim ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> )과 동일합니다. |
|--|---|

나머지 섹션에서는 하드웨어(RVfpgaNexys)와 시뮬레이션 (RVfpgaSim) 모두에서 RVfpga 시스템을 사용하는 방법을 보여줍니다. 섹션 5에서는 RVfpga 를 사용하는 데 필요한 소프트웨어 도구를 설치하는 방법을 보여줍니다. 섹션 6에서는 PlatformIO 를 사용하여 Nexys A7 FPGA 보드 (섹션 6.A)에 RVfpgaNexys 를 다운로드하고 여러 예제 프로그램을 다운로드하고 실행하는 방법 (섹션 6.B-6.H)을 보여줍니다. 섹션 7 과 8에서는 오픈 소스 HDL 시뮬레이터 인 Verilator (섹션 7)와 Western Digital 의 RISC-V ISS (Instruction Set Simulator) 인 Whisper (섹션 8)를 사용하여 RVfpgaSim 을 시뮬레이션하는 방법을 보여줍니다.

마지막으로 부록은 Linux 의 명령 프롬프트에서 RVfpga 를 사용하는 방법 (부록 A), Windows 및 macOS 컴퓨터에 필요한 드라이버와 소프트웨어를 설치하는 방법 (부록 BD), Vivado 를 사용하여 RVfpga 를 FPGA 에 다운로드하는 방법을 보여줍니다 (부록 E). 마지막 부록인 부록 F 는 산업용 IoT 애플리케이션 (부록 F)에서 RVfpga 를 사용하는 방법을 보여줍니다.

표 2 에는 RVfpga 에 필요한 소프트웨어 및 하드웨어가 나열되어 있습니다. 이 가이드는 Ubuntu 18.04 운영 체제 (OS)에서 이러한 도구 및 하드웨어를 설치하고 사용하는 방법을 보여줍니다. 다른 운영 체제는 (예: Windows 또는 macOS) 유사한 (정확히 동일하지는 않더라도) 단계를 따릅니다. 지침이 다른 경우, 강조 표시를 사용하여 **Windows** 및 **macOS** 에 대한 특정 지침을 추가합니다.

**Note:** Nexys A7 FPGA 보드에 액세스할 수 없는 경우에도 Western Digital 의 ISS (명령어 세트 시뮬레이터) 인 Whisper 와 오픈 소스 HDL 시뮬레이터 인 Verilator 를 사용하여 시뮬레이션에서 실험실을 완료할 수 있습니다. 이 경우 Vivado 를 설치할 필요가 없습니다 (섹션 5.A). VSCode/PlatformIO (섹션 2.A 에 설명 됨) 및 Verilator/GTKWave (섹션 5.C 에 설명됨) 만 설치하면 됩니다.

**표 2. RVfpga 에 필요한 소프트웨어 및 하드웨어**

| 소프트웨어   |   |      |
|---|---|------|
| Name  | Website   | Cost |
| Vivado 2019.2 WebPACK                           | <a href="https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html">https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2019-2.html</a> | free |
| VSCode  | <a href="https://code.visualstudio.com/Download">https://code.visualstudio.com/Download</a>   | free |
| PlatformIO                                      | <a href="https://platformio.org/">https://platformio.org/</a><br>Installed within VSCode  | free |
| Verilator (HDL 시뮬레이터) 및 GTKWave                 | <a href="https://github.com/verilator/verilator">https://github.com/verilator/verilator</a><br><a href="http://gtkwave.sourceforge.net/">http://gtkwave.sourceforge.net/</a>  | free |
| Whisper (Western Digital 의 RISC-V 명령어 세트 시뮬레이터) | <a href="https://github.com/chipsalliance/SweRV-ISS">https://github.com/chipsalliance/SweRV-ISS</a><br>Installed within PlatformIO  | free |
| RISC-V 툴체인 및 OpenOCD                            | <a href="https://github.com/riscv/riscv-gnu-toolchain">https://github.com/riscv/riscv-gnu-toolchain</a><br><a href="https://github.com/riscv/riscv-openocd">https://github.com/riscv/riscv-openocd</a><br>Installed within PlatformIO           | free |

| 하드웨어                               |   |                           |
|------------------------------------|---|---------------------------|
| Name                               | Website   | Cost                      |
| Nexys A7 FPGA 보드*                  | <a href="https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/">https://store.digilentinc.com/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/</a> | \$265<br>(학생가격:<br>\$199) |
| RISC-V 코어 및 SoC (System-on-Chip)** |   |                           |
| Name                               | Website   | Cost                      |
| Western Digital 의 SweRV EH1 Core   | <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a>   | free                      |
| SweRVolf                           | <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a>   | free                      |

\* 이 가이드에 설명된 모든 단계는 Digilent 의 Nexys4 DDR FPGA 보드에서도 작동합니다.

\*\* Imagination Technologies 의 RVfpga 다운로드와 함께 제공됩니다.

**주의사항:** 시작하기 전에 Imagination 의 University Program 에서 다운로드한 **RVfpga** 폴더를 Ubuntu/Windows/macOS 컴퓨터에 복사합니다. RVfpga 폴더를 [RVfpgaPath]로 배치하는 디렉토리의 절대 경로를 참조합니다. RVfpga 에는 5 개의 폴더가 있습니다. (1) **예제:** 이 가이드를 사용하는 동안 실행할 예제 프로그램, (2) **src:** RVfpga 시스템용 (Figure 1) 소스 코드 (Verilog 및 SystemVerilog) 포함, (3) **verilatorSIM:** 시뮬레이션 실행을 위한 스크립트 포함 Verilator 의 RVfpgaSim, (4) **driversLinux\_NexysA7:** Nexys A7 FPGA 보드용 Linux 드라이버가 포함되어 있고 (5) **Labs:** RVfpga Labs 1-10 에서 사용할 프로그램이 포함되어 있습니다.

## 예상되는 사전 지식:

이 RVfpga 시작 안내서 및 RVfpga 실습이 포함된 RVfpga 과정을 완료하기 전에 사용자는 최소한 다음 항목에 대한 기본적인 이해가 있어야 합니다:

- 디지털 로직 설계
- 고급 프로그래밍 (권장 C)
- 어셈블리 프로그래밍
- 명령어 세트 아키텍처
- 프로세서 마이크로 아키텍처
- 메모리 시스템

이 주제는 교과서에서 다룹니다. *Digital Design and Computer Architecture: RISC-V Edition*, Harris & Harris, © Morgan Kaufmann, 출판 예정일은 2021 년 여름입니다. 다음을 포함한 기타 교과서에서 *Computer Organization and Design RISC-V Edition*, Patterson & Hennessy, © Morgan Kaufmann 2017, 이러한 주제 중 일부를 다룹니다.

## 2. 빠른 시작 가이드

이 섹션에서는 RVfpgaNexys 를 사용하는 데 필요한 최소한의 도구를 설치하는 방법과 PlatformIO 를 사용하여 Nexys A7 FPGA 보드에 RVfpga 를 다운로드 한 다음 RVfpgaNexys 에서 프로그램을 실행하는 방법을 보여줍니다. FPGA 보드를 구입해야 합니다 (표 2 참조). 이 단계는 이전 버전 Nexys4-DDR FPGA 보드에서도 작동합니다.

**A. 최소 설치: VSCode, PlatformIO 및 Nexys A7 보드 드라이버.**

**B. FPGA 에 RVfpgaNexys 를 다운로드하고 프로그램을 실행합니다.**

아래 지침은 Ubuntu 18.04 시스템 용입니다. Windows 10 및 macOS 운영 체제에서도 작동합니다. 지침이 Ubuntu 와 다를 경우 **Windows** 및 **macOS** 에 대한 특정 지침이 있는 내용을 삽입합니다. Ubuntu 를 사용하는 경우 해당 내용을 무시할 수 있습니다. 경로는 슬래시 (/)를 사용하여 Linux 경로로 작성되지만 Windows 경로는 일반적으로 동일하지만 백 슬래시 (\)를 사용합니다.

### A. 최소 설치: VSCode, PlatformIO 및 Nexys A7 보드 드라이버

이 단계에서는 RVfpga 를 사용하는 데 필요한 최소 소프트웨어 및 드라이버를 설치합니다. 먼저 프로그래밍 환경을 설치 한 다음 Nexys A7 FPGA 보드 용 드라이버를 설치합니다.

**VSCode 및 PlatformIO 설치:** IDE (통합 개발 환경) 인 PlatformIO 를 사용하여 RVfpgaNexys 을 Nexys A7 보드에 다운로드하고 RVfpgaNexys 에서 프로그램을 다운로드하고 실행합니다. PlatformIO 는 Microsoft Visual Studio Code (VSCode)의 확장으로 구축되었습니다. PlatformIO 는 크로스 플랫폼이며 내장 디버거를 포함합니다.

다음 단계에 따라 VSCode 및 PlatformIO 를 모두 설치하십시오:

#### 1. VSCode 설치:

- 다음 링크에서 설치 파일을 다운로드하십시오: <https://code.visualstudio.com/Download>
- 터미널을 열고 VSCode 를 설치하고 실행합니다:
 

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

**Windows/macOS:** VSCode 패키지는 <https://code.visualstudio.com/Download> 에서 Windows (.exe 파일) 및 macOS (.zip 파일) 용으로도 제공됩니다. 이러한 운영 체제에서 응용 프로그램을 설치하고 실행하는 데 사용되는 일반적인 단계를 따르십시오.

#### 2. VSCode 위에 PlatformIO 를 설치합니다.

- 터미널에 다음을 입력하여 python3 유틸리티를 설치합니다.
 

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

**Windows/macOS:** Windows 에서는 이 단계 (2.a)가 필요하지 않습니다. macOS 의 경우 homebrew 를 사용하여 python3 을 설치할 수 있습니다: `brew install python3`


- b. 열리지 않은 경우 시작 버튼을 선택하고 검색 메뉴에 "VSCode"를 입력하여 VSCode 를 시작한 다음 VSCode 를 선택하거나 Ubuntu 터미널에 `code` 를 입력합니다.
- c. VSCode 에서 VSCode 의 왼쪽에있는 확장  아이콘을 클릭합니다 (그림 2 참조).



그림 2. VSCode 의 확장 아이콘

검색 상자에 *PlatformIO*를 입력하고 옆에있는 설치 버튼을 클릭하여 PlatformIO IDE를 설치합니다 (그림 3 참조).

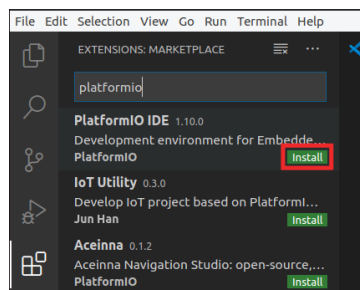


그림 3. PlatformIO IDE 확장

- d. 하단의 OUTPUT 창은 설치 과정을 알려줍니다. 완료되면 오른쪽 하단 창에서 "Reload Now"를 클릭하면 PlatformIO 가 VSCode 내부에 설치를 완료합니다 (그림 4 참조).

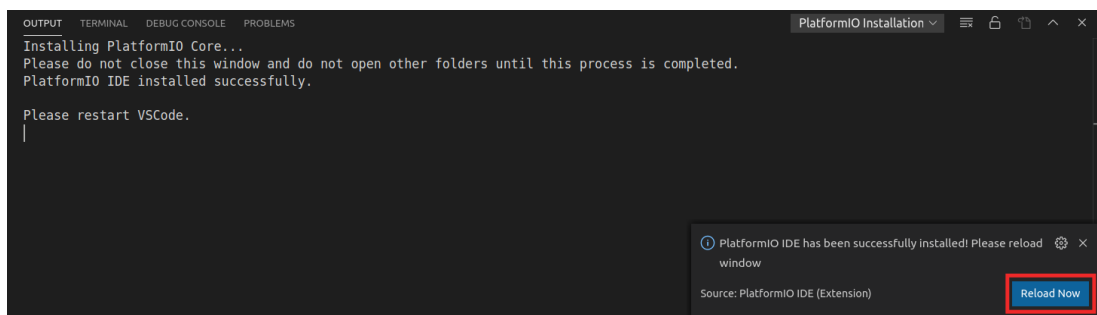


그림 4. PlatformIO 설치 후 지금 다시로드

**Nexys A7 케이블 드라이버 설치 :** Nexys A7 보드 용 드라이버를 수동으로 설치해야 합니다.

- 터미널을 엽니다.

- `[RVfpgaPath]/RVfpga/driversLinux_NexysA7` 디렉터리로 이동합니다. (간단하게 하려고 RVfpga 폴더에 이러한 드라이버를 제공합니다. 가이드의 섹션 5에서 Vivado를 설치할 때 해당 섹션에 설명된 대로 다운로드한 패키지에서 이러한 드라이버를 찾을 수도 있습니다.)
- 설치 스크립트 실행 :  

```
chmod 777 *
sudo ./install_drivers
```
- 컴퓨터에서 Nexys A7 보드를 분리하고 컴퓨터를 다시 시작하여 변경 사항을 적용하십시오.

**Windows:** Nexys A7 보드 용 드라이버를 설치하려면 부록 B에 제공된 지침을 따르십시오.

**macOS:** 추가 드라이버를 설치할 필요가 없습니다.

## B. FPGA에 RVfpgaNexys를 다운로드하고 RVfpgaNexys에서 프로그램 실행

이제 FPGA를 대상으로 하는 RISC-V 시스템인 RVfpgaNexys를 Nexys A7 FPGA 보드에 다운로드합니다. 시작 안내서에서 수정하지는 않겠지만 RVfpga 시스템용 Verilog은 `[RVfpgaPath]/RVfpga/src`에서 사용할 수 있습니다. GSG의 섹션 4에서 RVfpga 시스템의 소스 코드를 설명하고 RVfpga Labs 6-10에서 자세히 설명합니다. 또한 해당 실습의 일부 실습에서 RVfpga 시스템을 수정합니다.

다음 단계를 완료하여 Nexys A7 FPGA 보드에 RVfpgaNexys를 실행하십시오:

- 1 단계. Nexys A7 FPGA 보드를 컴퓨터에 연결하고 보드를 켭니다.
- 2 단계. PlatformIO 및 C 프로그램 열기
- 3 단계. Nexys A7 보드에 RVfpgaNexys 다운로드
- 4 단계. RVfpgaNexys에서 프로그램 다운로드 및 실행

### 1 단계. Nexys A7 FPGA 보드를 컴퓨터에 연결하고 보드를 켭니다.

제공된 USB 케이블을 사용하여 Nexys A7 보드를 컴퓨터에 연결합니다. 그림 5는 Nexys A7 FPGA 보드의 LED 및 스위치와 USB 커넥터, 온 스위치, 푸시 버튼 및 7 세그먼트 디스플레이의 물리적 위치를 보여줍니다. Nexys A7 보드의 USB 커넥터 포트 사이에 케이블을 연결하고 보드를 켭니다.

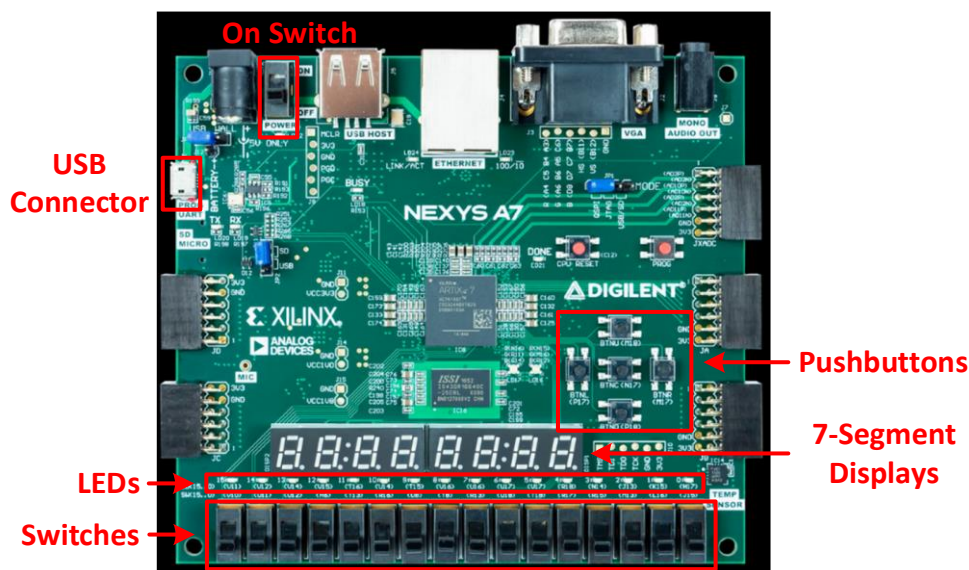


그림 5. Digilent의 Nexys A7 FPGA 보드의 I/O 인터페이스

(<https://reference.digilentinc.com/>의 보드 그림)



## 2 단계. PlatformIO 및 C 프로그램 열기

이제 시작 메뉴에 VSCode 를 입력하거나 (그림 6 참조) 터미널에 `code` 를 입력하여 Visual Studio Code (VSCode)를 엽니다.

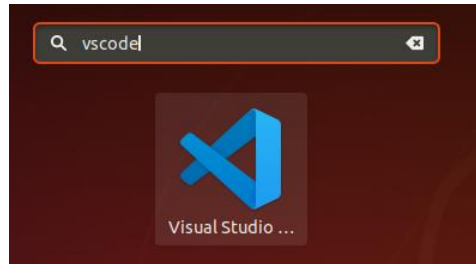



그림 6. VSCode 열기

PlatformIO 홈 (PIO 홈 Home) 창이 자동으로 열리지 않으면 왼쪽 리본 메뉴 에서 PlatformIO 아이콘을 클릭합니다. 그런 다음 PIO 홈을 확장하고 열기를 클릭합니다. 이제 PIO Home 이 환영 창으로 열립니다 (그림 7 참조).

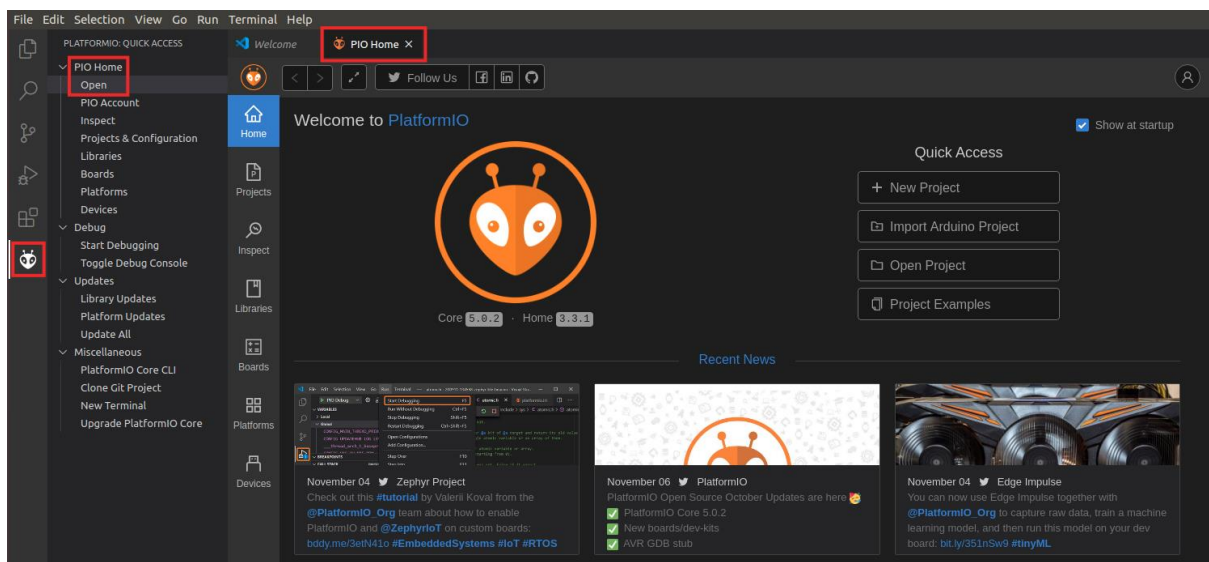


그림 7. PIO 홈 열기

상단 파일 메뉴에서 파일 → 폴더 열기를 클릭하고 다음을 선택하십시오:

*[RVfpgaPath]\RVfpga\examples\LedsSwitches\_C-Lang*

폴더를 선택하되 열지는 마십시오 (그림 8 참조). PlatformIO 는 이 프로그램 인 LedsSwitches\_C-Lang 을 열어 Nexys A7 보드의 스위치 값을 읽고 보드의 LED 에 값을 씁니다.



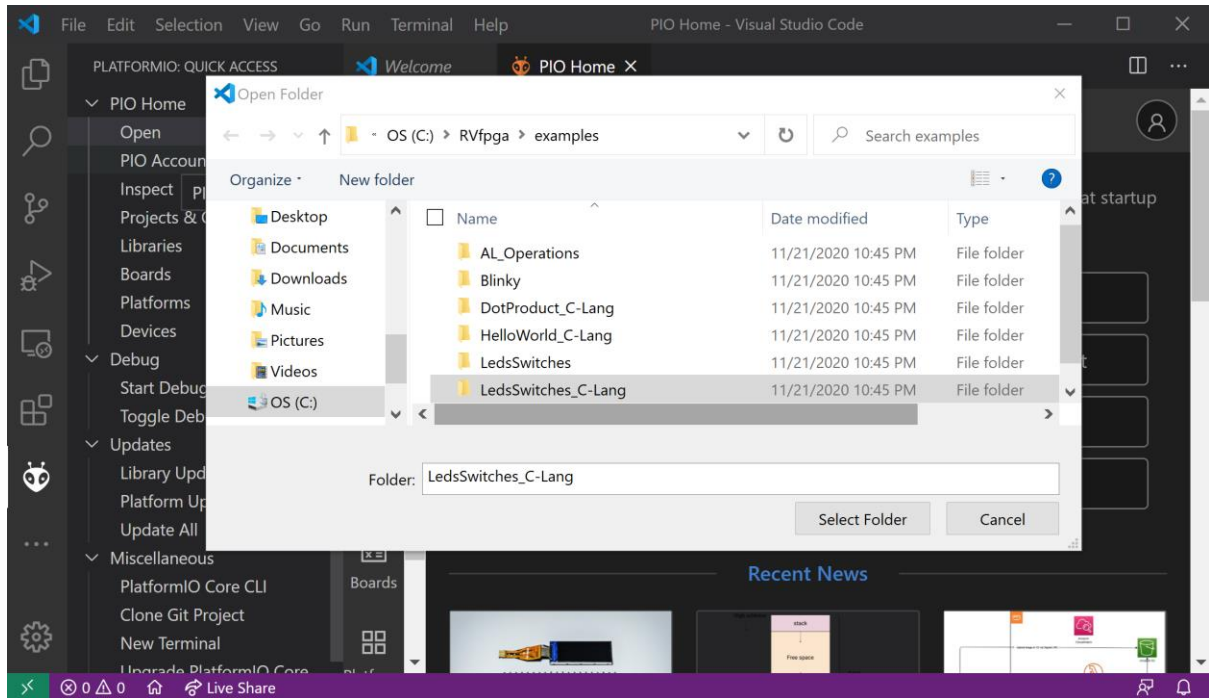


그림 8. Open LedsSwitches\_C-Lang 예제

src 폴더를 확장하고 LedsSwitches\_C-Lang.c 를 두 번 클릭하여 LedsSwitches\_C-Lang 프로그램을 볼 수 있습니다 (그림 9). 시작 안내서의 뒷부분에서 이 프로그램에 대해 자세히 설명합니다. 빠른 시작 가이드에서는 Nexys A7 보드에서 실행될 RVfpgaNexys 에 이 프로그램을 다운로드하면 됩니다.

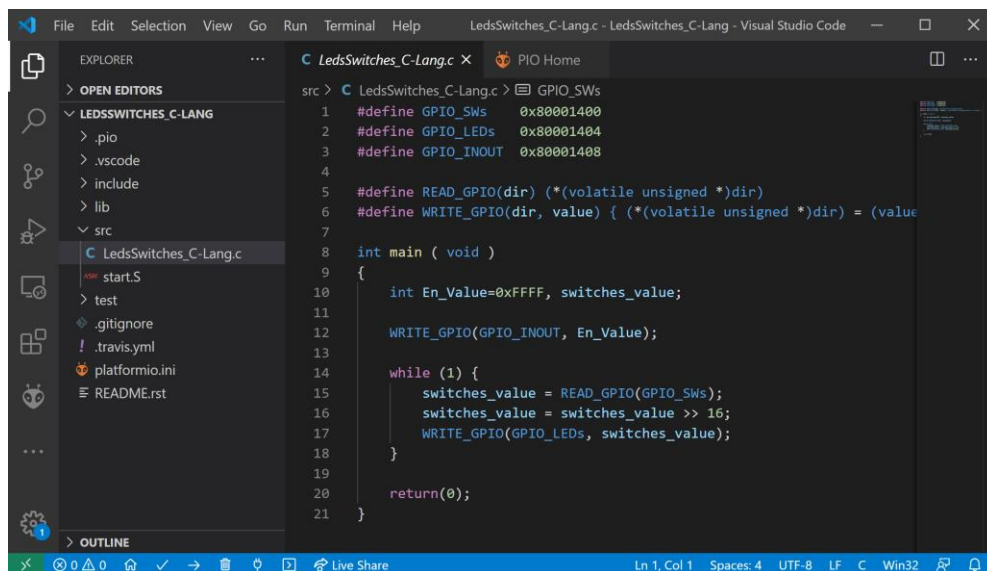


그림 9. Leds Switches C-Language.c 프로그램

PlatformIO 에서 RVfpga 예제를 처음 열면 Chips Alliance 플랫폼이 자동으로 설치됩니다 (그림 10 와 같이 PIO Home → Platforms 에서 볼 수 있음). 이 플랫폼에는 미리 빌드된 RISC-V 도구 모음, RISC-V 용 OpenOCD, RVfpgaNexys 비트 파일 및 RVfpgaSim, JavaScript 및 Python 스크립트, 여러 예제와 같이 나중에 사용할 여러 도구가 포함되어 있습니다. 어떤 이유로든 Chips Alliance 플랫폼이 자동으로 설치되지 않은 경우, 섹션 6.A 에 설명된 대로 수동으로 설치할 수 있습니다.

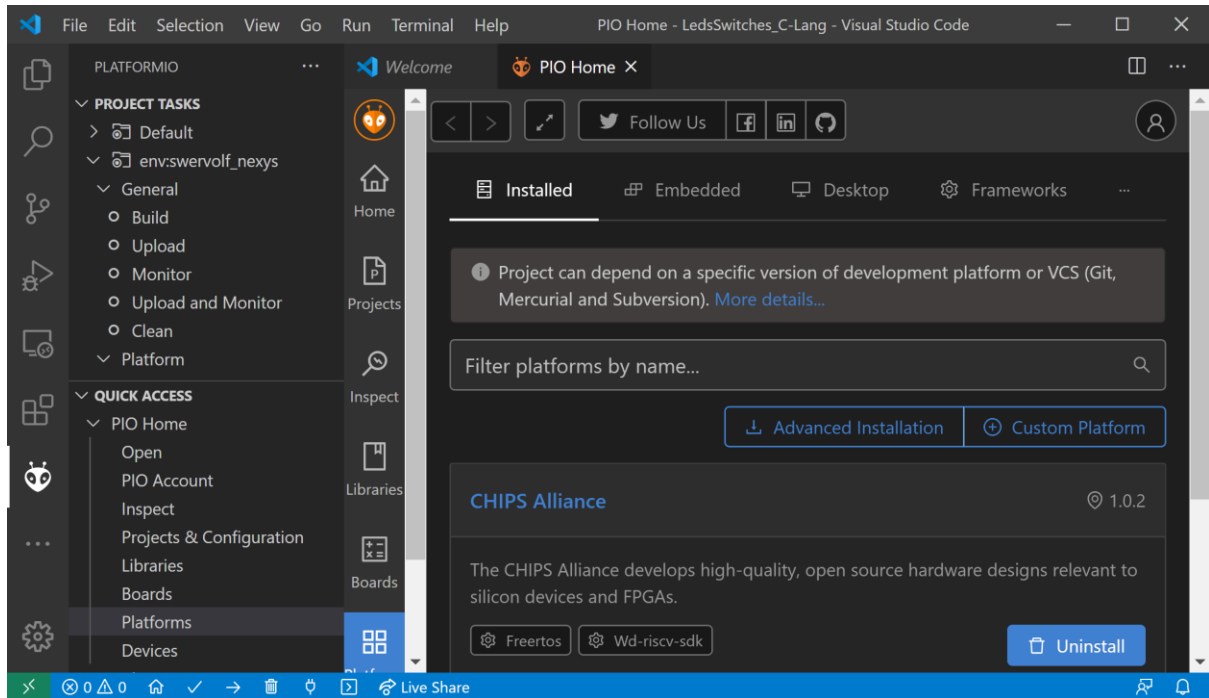



그림 10. PlatformIO 에 설치된 Chips Alliance 플랫폼

### 3 단계. Nexys A7 보드에 RVfpgaNexys 다운로드

이제 주변 장치를 지원하는 RISC-V 프로세서를 포함하는 RISC-V SoC 인 RVfpgaNexys 를 다운로드할 준비가 되었습니다. 그림 11 과 같이 EXPLORER 창에서 platformio.ini (PlatformIO 초기화 파일)를 두 번 클릭하여 엽니다. (탐색기가 아직 열려 있지 않은 경우 왼쪽 리본 메뉴에서  을 클릭하여 엽니다.) 이제, board\_build.bitstream\_file 경로를 자신의 경로로 대체하여 RVfpgaNexys 를 정의하는 비트 파일의 위치 경로를 추가합니다 (그림 10 참조).

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

Ctrl-s 를 눌러 platformio.ini 파일을 저장하십시오.

프로젝트 구성 파일 (platformio.ini)에 대한 많은 명령이 있습니다. 이러한 옵션에 대한 자세한 내용은 다음에서 확인할 수 있습니다. <https://docs.platformio.org/en/latest/projectconf/>.

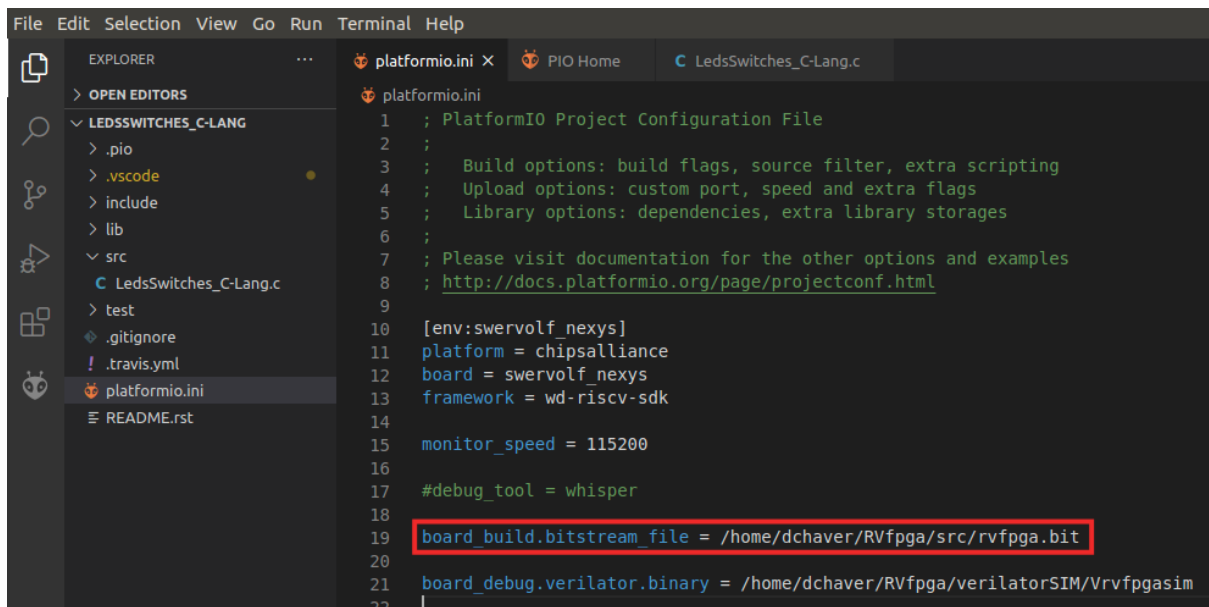


그림 11. RVfpgaNexys 비트 파일에 경로 추가

Nexys A7 보드에 RVfpgaNexys 를 (이 비트 파일에 정의 된대로) 다운로드합니다.



- 왼쪽 메뉴 리본에서 PlatformIO 아이콘  을 클릭합니다 (그림 12 참조).



그림 12. PlatformIO 아이콘

- 프로젝트 작업 창이 비어 있는 경우 (그림 13)을 클릭  하여 먼저 프로젝트 작업을 새로 고쳐야 합니다. 몇 분 정도 걸릴 수 있습니다.

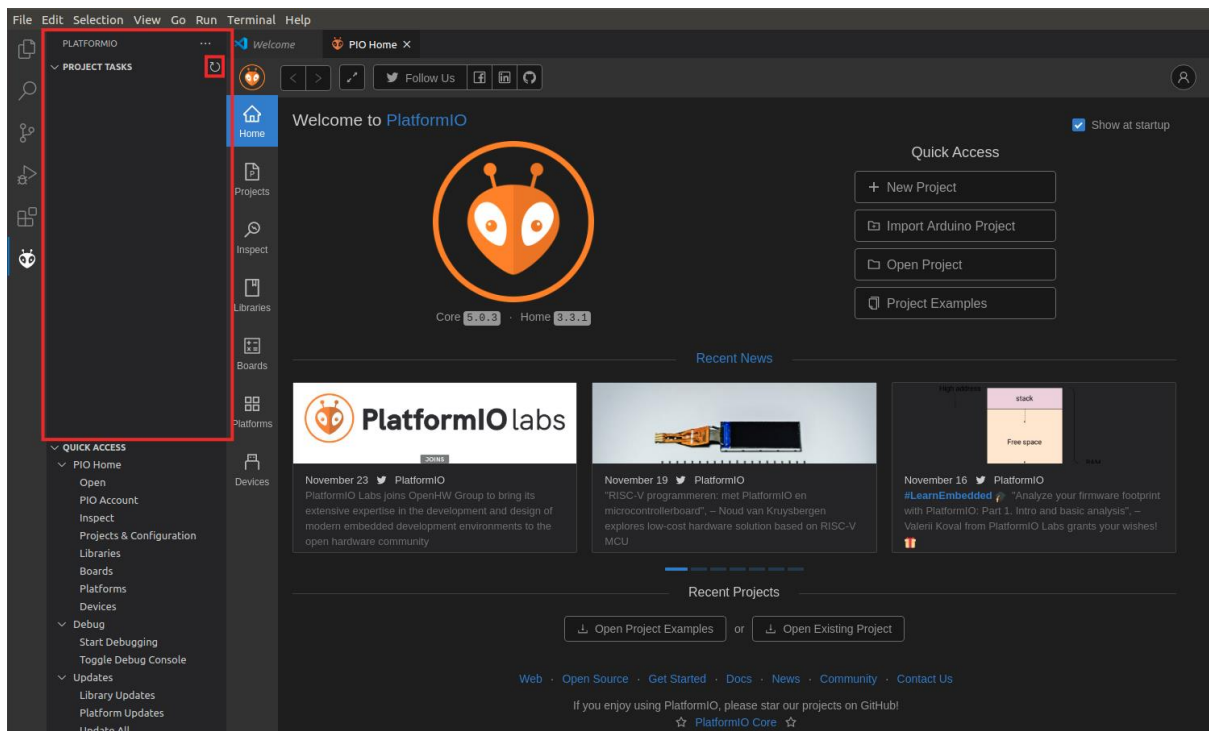


그림 13. 비어있는 PROJECT TASKS 창 – 새로 고침

- 그런 다음 Project Tasks → env: swervolf\_nexys → Platform 을 확장하고 Upload Bitstream 을 클릭합니다 (그림 14 참조). 1 ~ 2 초 후에 FPGA 는 RVfpgaNexys SoC 로 프로그래밍 됩니다.
- 기본적으로 프로세서는 주소 0x80000000 에서 명령 가져 오기를 시작합니다. 여기서 부팅 ROM 은 SoC 에 있습니다 (표 6 참조). Boot ROM 은 LED 와 7-Segment 디스플레이를 4 번 깜박인 다음 모든 LED 를 끄고 8 개의 7-Segment 디스플레이에 0 을 쓰고, 빈 루프를 유지하는 프로그램 (boot\_main.mem)으로 초기화 됩니다. 이 프로그램은 `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw` 폴더에서 찾을 수 있습니다. 변경하고 다시 컴파일하려면 부록 A – 섹션 III 에 설명 된 대로 수행하십시오 (boot\_main.mem 파일은 단순히 `boot_main.vh` 파일의 복사본 임). 실습 1 에서는 비트 스트림을 생성할 때 이 프로그램을 사용하여 Boot ROM 을 초기화하는 방법을 보여줍니다.

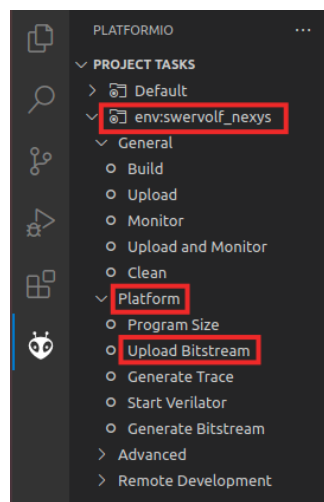




그림 14. 비트 스트림 업로드

#### 4 단계. RVfpgaNexys 에서 프로그램 다운로드 및 실행

이제 Nexys A7 보드에서 RVfpgaNexys 를 다운로드하여 실행 했으므로 RVfpgaNexys 의 메모리에 프로그램을 다운로드하고 프로그램을 실행/디버그합니다. 왼쪽 사이드 바에 있는 "실행 및 디버그" 버튼  을 클릭합니다. 재생 버튼  을 클릭하여 디버거를 시작합니다 ("PIO 디버그" 옵션이 선택되어 있는지 확인하십시오). 창 상단 근처에 이 버튼이 있습니다 (그림 15 참조). 프로그램이 먼저 컴파일되고 디버깅이 시작됩니다.

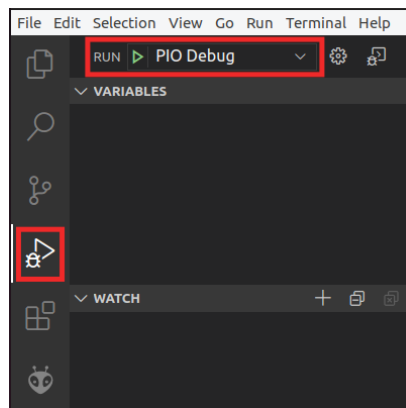



그림 15. 프로그램 컴파일 및 다운로드 및 디버거 시작

디버깅 세션을 제어하려면 편집기 상단 근처에 나타나는 디버깅 도구 모음을 사용할 수 있습니다 (그림 15 참조). 이 시작 안내서의 뒷부분에서 모든 옵션을 설명하고 테스트합니다.



그림 16. 디버깅 도구

PlatformIO 는 기본 기능의 시작 부분에 임시로 breakpoint 를 설정합니다. 따라서 계속 버튼  을 클릭하여 프로그램을 실행하십시오. 이제 Nexys A7 FPGA 보드의 스위치를 토글하고 해당 LED 가 켜지는 것을 확인합니다.

### 3. RISC-V 아키텍처 개요

RISC-V 는 캘리포니아 버클리 대학의 Par Lab 에서 2011 년에 만든 ISA (Instruction Set Architecture) 입니다. 목표는 RISC-V 가 작고 제한적이며 리소스가 적은 IoT 장치에서 슈퍼 컴퓨터에 이르기까지 모든 응용 프로그램에 사용되는 프로세서를 위한 "유니버설 ISA"가 되는 것이었습니다. RISC-V 설계자는이 목표를 달성하기 위해 아키텍처에 대한 5 가지 원칙을 수립했습니다:

- 광범위한 소프트웨어 패키지 및 프로그래밍 언어와 호환되어야 합니다.
- FPGA 에서 ASIC (application specific integrated circuits) 및 신흥 기술에 이르는 모든 기술 옵션에서 구현이 가능해야 합니다.
- 마이크로 코드 또는 유선 제어, 순차 또는 비 순차 파이프 라인, 다양한 유형의 병렬 처리를 구현하는 시나리오를 포함하여 다양한 마이크로 아키텍처 시나리오에서 효율적이어야 합니다.
- ISA 자체에서 발생하는 문제점없이 필요한 최대 성능을 달성하기 위해 특정 작업에 맞게 조정할 수 있어야 합니다.
- 기본 명령어 세트는 안정적이고 오래 지속되어야 하며 개발자에게 공통적이고 견고한 프레임 워크를 제공해야 합니다.

RISC-V 는 개방형 표준이며 실제로 사양은, 공개 도메인이며 2015 년부터 RISC-V 의 하드웨어 및 소프트웨어 개발을 촉진하는 비영리 조직인 **RISC-V International** 이라고하는 **RISC-V 재단**에서 관리하고 있습니다. 2018 년 RISC-V 재단은 Linux 재단과 지속적인 협력을 시작했으며 2020 년 3 월 RISC-V 재단은 스위스에 본사를 둔 RISC-V International 이 되었습니다. 이 전환은 향후 표준의 개방성에 대해 커뮤니티가 가질 수 있는 모든 우려를 없앴습니다. 2020 년 현재 RISC-V International 은 Microchip, NXP, Samsung, Qualcomm, Micron, Google, Alibaba, Hitachi, Nvidia, Huawei, Western Digital, ETH Zurich, KU Leuven, UNLV 및 UCM h 를 포함하여 연구, 학계 및 업계의 200 명 이상의 주요 업체에서 지원합니다.

RISC-V 는 incremental architectures 제품이 아닌 개방형 표준 및 모듈식이기 때문에 지난 10-20 년 동안 개발된 몇 안되는 세계적으로 유일한 ISA 중 하나입니다. 모듈식으로 인해 유연하고 매끄럽습니다. 프로세서는 기본 ISA 와 이를 사용한 확장 명령어로 만 구현합니다. 이 모듈식 접근 방식은 이전 ISA 가 확장되고 각각의 새 프로세서가 이전 소프트웨어 프로그램과의 호환성을 보장하기 위해 "구식"으로 태그가 지정된 명령어를 포함하여 모든 명령어를 구현해야하는 incremental architectures 인 x86 또는 ARM 과 같은 기존 ISA 와 다릅니다. 예를 들어, 80 개의 명령어로 시작된 x86 은 이제 기계어 코드에서 사용할 수 있는 모든 다른 opcode 를 고려할 경우 1300 개 또는 3600 개가 넘습니다. 이처럼 많은 수의 명령어와 이전 버전과의 호환성 요구 사항으로 인해 대부분의 짧은 opcode 또는 작은 명령어가 이미 사용 중이기 때문에 긴 명령어를 지원해야 하는 전력 소모가 큰 프로세서가 필요합니다.

RISC-V 에는 표 2 와 같이 32 비트 버전 2 개 (정수 및 임베디드 버전, RV32I 및 RV32E)와 64 비트 및 128 비트 버전 (RV64I 및 RV128I)의 네 가지 기본 ISA 옵션이 있습니다. Ratified 로 표시된 ISA 모듈은 지금 사용할 수 있습니다. Frozen 으로 표시된 모듈은 인증 받기 전에 크게 변경되지 않을 것으로 예상됩니다. Draft 로 표시된 모듈은 인증전에 변경 될 것으로 예상됩니다. 소형 프로세서를 구축할 수 있는 능력은 특히 비용, 공간 및 에너지 제약이있는 장치의 핵심 요구 사항입니다. 이러한 기본 ISA 위에 명령어 확장을 추가하여 부동 소수점 연산, 곱셈과 나눗셈, 벡터 연산과 같은 특정 작업을 수행할 수 있습니다. 이러한 특수 하드웨어 확장은 표준에 포함되어 있으며 컴파일러에서 알고 있으므로 컴파일러에서 원하는 옵션을 활성화하면 대상 바이너리 코드 생성이 가능합니다. 이러한 각 확장은 표 3 과 같은 하드웨어 기능을 나타 내기 위해 핵심 ISA 에 추가해야 하는 문자로 식별됩니다. 예를 들어 RVM 은 곱하기/나누기 확장이고 RVF 는 부동 소수점 확장, 등등 입니다.

표 3. RISC-V 기본 ISA

(표 출처: <https://riscv.org/technical/specifications/>)

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

표 4. RISC-V 표준 ISA 확장

(표 출처: <https://riscv.org/technical/specifications/>)

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

"General"을 나타내는 문자 G는 모든 MAFD 확장을 포함하는 표시에 사용됩니다. 회사 또는 개인은 본인만의 opcode를 사용하여 표준 모듈에서 사용되지 않는 확장 명령어를 개발할 수 있습니다. 이를 통해 타사 보다 빠른 시장 출시를 위하여 개발할 수 있습니다.

예를 들어, 4 개의 일반 ISA 확장과 비트 조작 및 사용자 레벨 인터럽트를 모두 포함하는 64 비트 RISC-V 구현을 RV64GBN ISA 라고합니다. 이러한 모든 모듈은 비 특권 또는 사용자 사양에 포함됩니다. RISC-V 재단은 범용 운영 체제를 실행하는 데 필요한 권한 있는 작업에 대한 일련의 요구 사항과 지침도 다룹니다.

## 4. RVFPGA 시스템 개요

이 섹션에서는 코어에서 FPGA 보드 인터페이스까지 전체 RVfpga 시스템을 설명합니다. 그림 17은 프로세서 코어에서 시작하여 코어 주변에 구축된 SoC, 마지막으로 시스템 및 보드 인터페이스로 시작하는 임베디드 시스템의 일반적인 계층 구조를 보여줍니다.

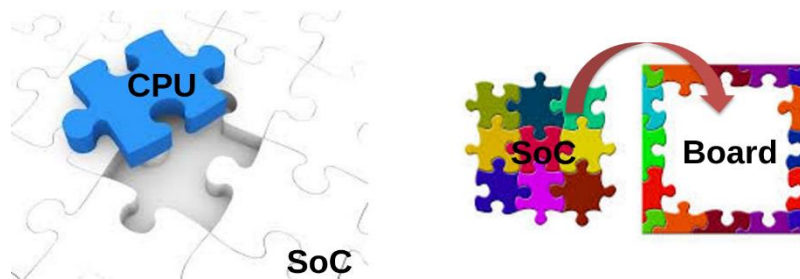


그림 17. 임베디드 시스템 구성

그림 1과 표 1은 SweRV EH1 Core에서 RVfpgaNexys 및 RVfpgaSim까지 시스템의 계층 구조를 보여줍니다. 다음 섹션에서는 RISC-V 명령을 실행하는 프로세서 코어 (Western Digital의 SweRV EH1 Core)에 대해 설명합니다. 그런 다음 섹션 B에서는 시스템의 하드웨어 구성 요소 (코어, 메모리 및 입력/출력)를 통합하는 SweRVofX SoC와 이를 RVfpga 내에서 사용하기 위해 수행된 확장에 대해 설명합니다. 섹션 C에서는 Nexys A7 FPGA 보드 (RVfpgaNexys)에 구현된 SweRVofX SoC를 설명하고 시뮬레이션에 사용되는 SweRVofX SoC (RVfpgaSim)에 대해서도 설명합니다. 마지막으로 섹션 D에서 전체 RVfpga 시스템의 파일 구조를 설명합니다.

### A. SweRV EH1 코어 및 SweRV EH1 코어 컴플렉스.

Western Digital은 지난 몇 년 동안 **SweRV EH1** (RVfpga에서 사용되는 코어), SweRV EH2 및 SweRV EL2 (RVfpga 시스템의 향후 버전에 이러한 코어가 포함될 수 있음)의 세 가지 RISC-V 코어를 개발했습니다. 각 코어에는 Apache 2.0 라이선스가 있습니다. SweRV EH1 코어는 32 비트, 2-웨이 슈퍼 스칼라, 9 단계 파이프라인 코어입니다. SweRV Core EH2는 EH1 Core를 구축하고 확장하여 추가 성능을 위한 이중 스레드 기능을 추가합니다. SweRV Core EL2는 적당한 성능을 가진 더 작은 코어입니다.

<https://www.westerndigital.com/company/innovations/risc-v>의 RISC-V 페이지에는 사용 가능한 세 가지 코어가 나와 있으며 주요 기능은 표 5에 나와 있습니다.

표 5. 3개의 WD RISC-V 코어의 주요 기능

(표 출처 <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           |

3 개의 코어 중 **SweRV EH1 코어** (RVfpga 패키지와 함께 제공되고

<https://github.com/chipsalliance/Cores-SweRV> 에서도 사용 가능)는 고성능/MHz 및 간단한 스레드 구조로 인해 선호합니다. 또한 오픈 소스 하드웨어 제공에 전념하는 그룹 인 Chips Alliance 는 SweRVolf (RVfpga 패키지와 함께 제공되며 <https://github.com/chipsalliance/Cores-SweRVolf> 에서도 사용 가능)라고 하는 완전하고 검증된 SoC 를 제공합니다. RVfpga 시스템은 Western Digital 의 **SweRV EH1 Core** 버전 **1.8** 을 사용하는 SweRVolf SoC 의 확장을 사용합니다.

**SweRV EH1 코어**는 RISC-V 의 정수 (I), 압축 명령 (C), 정수 곱셈 및 나눗셈 (M) 확장을 지원하는 머신 모드 (M 모드) 전용 32 비트 CPU 코어입니다. 프로그래머 참조 설명서

([https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V\\_SweRV\\_EH1\\_PRM.pdf](https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf))는 구조에서부터 타이밍 정보 및 메모리 맵에 이르기까지 코어의 모든 측면을 자세히 설명합니다.

SweRV EH1 은 2 개의 파이프 라인 I0 및 I1 에서 EX1 에서 EX4 로 레이블이 지정된 4 개의 산술 논리 장치 (ALU)를 지원하는 dual-issue 9 단계 파이프 라인 (그림 18 참조)이 있는 슈퍼 스칼라 코어입니다. 파이프 라인의 두 가지 방법 모두 ALU 작업을 지원합니다. 파이프 라인의 한 가지 방법은 로드/저장을 지원하고 다른 방법은 3-cycle laency multiplier 를 사용합니다. 또한 프로세서에는 파이프 라인 외부 34-cycle laency divider 가 1 개 있습니다. 파이프 라인에는 'Fetch 1', 'Align', 'Decode', 'Commit'의 네 가지 stall point 가 있습니다. 'Fetch 1'단계에는 Gshare branch predictor 가 포함됩니다. 'Fetch 1'단계에서는 3 개의 페치 버퍼에서 명령어를 검색합니다. 'Decode'단계에서는 4 개의 명령어 버퍼에서 최대 2 개의 명령어가 디코딩됩니다. 'Commit'단계에서는 싸이클 당 최대 2 개의 명령이 커밋됩니다. 마지막으로 'Writeback'단계에서 아키텍처 레지스터가 업데이트됩니다.

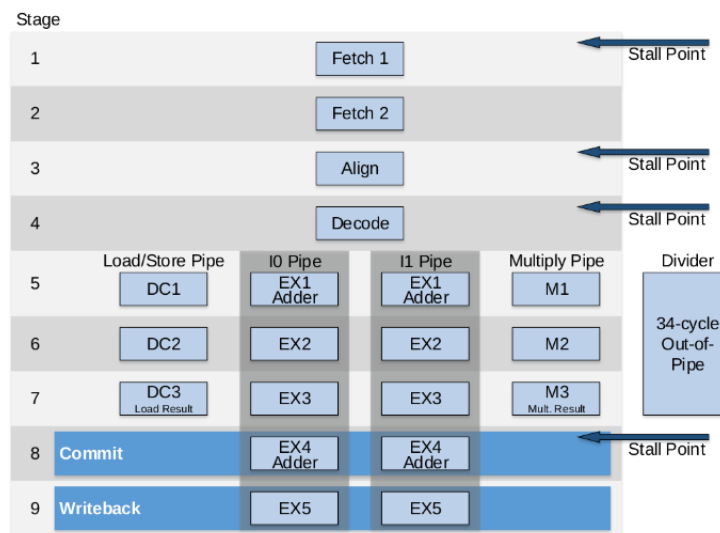


그림 18. SweRV EH1 코어 마이크로 아키텍처

(그림 출처: [https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V\\_SweRV\\_EH1\\_PRM.pdf](https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf))

그림 19 은 현재 다른 코어와 프로세서를 비교 한 것입니다. MHz 당 SweRV EH1 Core 성능은 4.9CM/MHz (MHz 당 CoreMark)로 매우 높습니다. ARM Cortex A8 보다 두 배 빠르며 성능은 ARM Cortex A15 성능을 능가합니다.

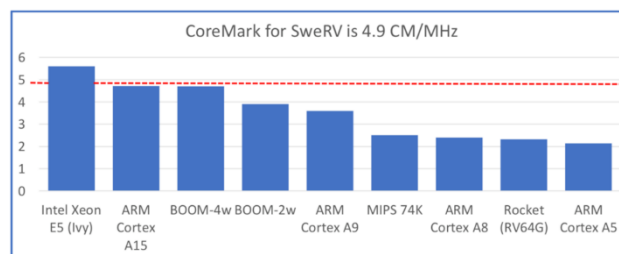


그림 19. 스레드 및 MHz 별 벤치 마크 비교

(그림 출처 : [https://content.riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD\\_SweRV\\_Cores\\_Roadmap\\_v4SCR.pdf](https://content.riscv.org/wp-content/uploads/2019/12/12.11-14.20a3-Bandic-WD_SweRV_Cores_Roadmap_v4SCR.pdf))

Western Digital 은 또한 **SweRV EH1 Core Complex** (그림 20 참조)라고하는 SweRV EH1 Core 에 대한 확장을 제공합니다. 이는 위에서 설명한 EH1 Core 에 다음 요소를 추가하고 그림에서 파란색으로 표시됩니다.

- 두 개의 전용 메모리, 하나는 명령어 (ICCM) 용이고 다른 하나는 데이터 용 (DCCM) 용으로 코어에 결합됩니다. 이러한 메모리는 지연 시간이 짧은 액세스 및 SECDED ECC (단일 오류 수정 및 이중 오류 감지 오류 수정 코드) 보호를 제공합니다. 각 메모리는 4, 8, 16, 32, 48, 64, 128, 256 또는 512KB 로 구성할 수 있습니다.
- 패리티 또는 ECC 보호 기능이있는 4 방향 세트 연관 명령어 캐시 (옵션).
- 최대 255 개의 외부 인터럽트를 지원하는 선택적 PIC (Programmable Interrupt Controller).
- 명령어 가져 오기 (IFU 버스 마스터), 데이터 액세스 (LSU 버스 마스터), 디버그 액세스

(디버그 버스 마스터) 및 밀접하게 연결된 메모리에 대한 외부 DMA 액세스 (DMA 슬레이브 포트)를 위한 4 개의 시스템 버스 인터페이스 (64 비트 AXI4 로 구성 가능) 또는 AHB-Lite 버스).

- RISC-V 디버그 사양을 준수하는 코어 디버그 장치.

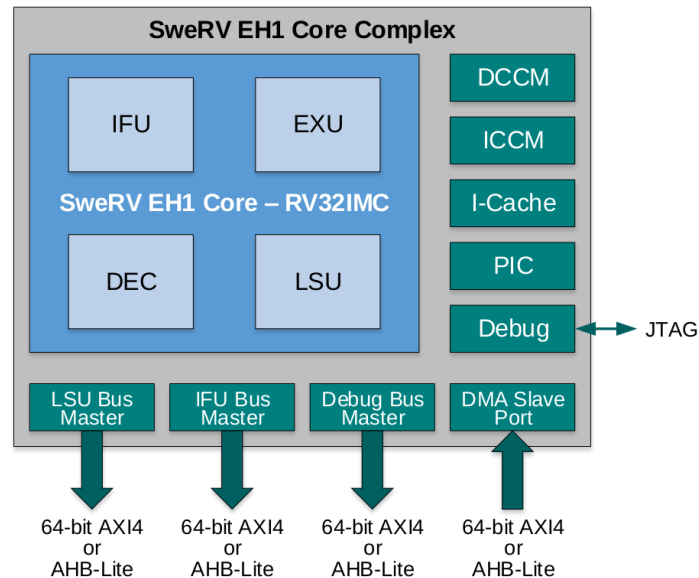


그림 20. SweRV EH1 코어 컴플렉스

(그림 출처 : [https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V\\_SweRV\\_EH1\\_PRM.pdf](https://github.com/chipsalliance/Cores-SweRV/blob/master/docs/RISC-V_SweRV_EH1_PRM.pdf))

## B. SweRVolfX SoC

이 RVfpga 패키지에 사용되는 SoC (System on Chip)는 SweRVolfX 라고 하며 그림 21 에 설명되어 있으며 SweRVolf 버전 0.7.3 (<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>), SweRV EH1 Core Complex 위에 구축되었습니다. SweRV EH1 코어 컴플렉스 (그림 20 참조) 외에도 SweRVolf SoC 에는 부트 ROM, UART, 시스템 컨트롤러 및 SPI 컨트롤러가 포함되어 있습니다 (그림 21 은 이러한 요소를 흰색으로 표시). SweRV EH1 Core 는 AXI 버스를 사용하고 주변 장치는 Wishbone 버스를 사용하므로 SoC 에는 AXI-Wishbone Bridge 도 있습니다.

RVfpga 에서 우리는 다른 SPI 컨트롤러 (SPI2), GPIO (범용 입출력) 컨트롤러, PTC (PWM/타이머/카운터) 모듈 및 8 자리 7 세그먼트 디스플레이 인터페이스의 컨트롤러와 함께 SweRVolf SoC 를 확장합니다. 그림 21 은 시스템 컨트롤러에 포함된 7 세그먼트 디스플레이 컨트롤러를 제외하고 이러한 새 주변 장치를 빨간색으로 표시합니다. 이 System on Chip SweRVolfX (X 는 eXtended 를 나타냄)라고 합니다.

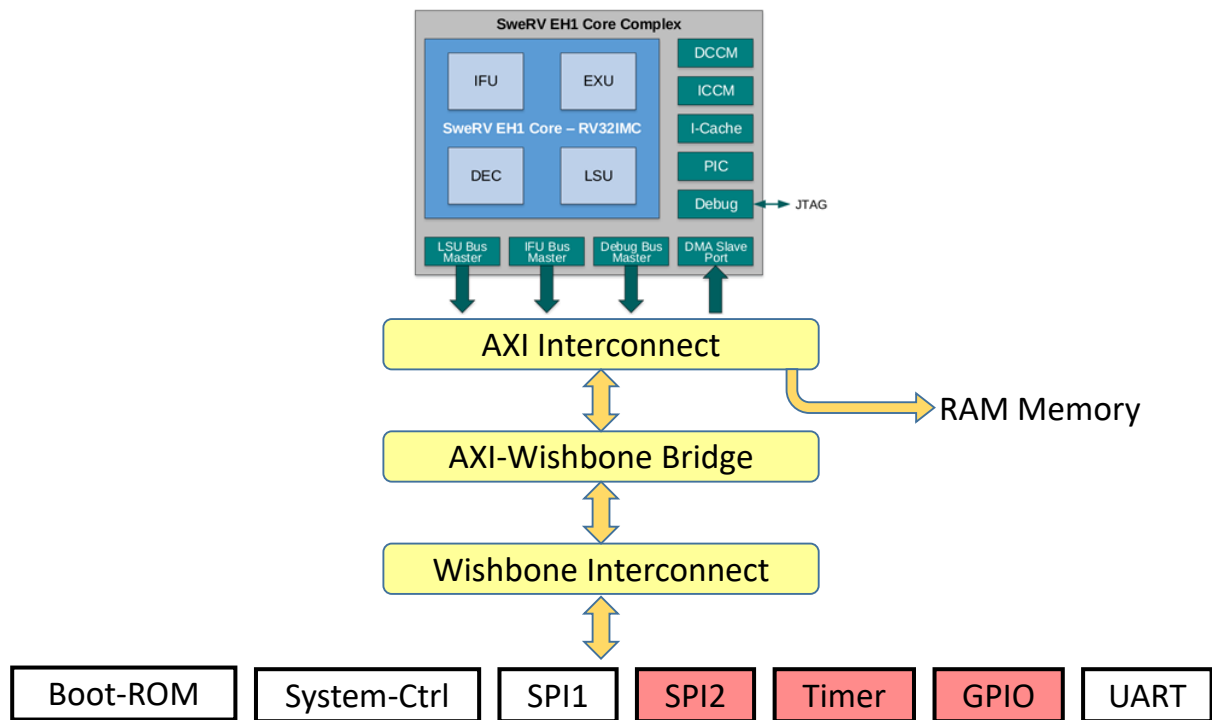


그림 21. SweRVVolFX (4 개의 새로운 주변장치와 같이 확장된 SweRVolf) SoC

표 6 는 Wishbone 상호 연결을 통해 코어에 연결된 주변 장치의 메모리 매핑 주소를 보여줍니다.

표 6. 확장 SweRVolfX SoC 주변 장치의 메모리 매핑 주소

| System            | Address                 |
|-------------------|-------------------------|
| Boot ROM          | 0x80000000 - 0x80000FFF |
| Syatem Controller | 0x80001000 - 0x8000103F |
| SPI1              | 0x80001040 - 0x8000107F |
| SPI2              | 0x80001100 - 0x8000113F |
| Timer             | 0x80001200 - 0x8000123F |
| GPIO              | 0x80001400 - 0x8000143F |
| UART              | 0x80002000 - 0x80002FFF |

### i. 입출력

SweRVolfX SoC 는 주변 장치와 통신하기 위해 두 가지 종류의 하드웨어 컨트롤러를 사용합니다: 무료 및 오픈 소스 협업 정신의 Verilog 로 작성된 맞춤형 컨트롤러와 OpenCores 의 오픈 소스 컨트롤러 [<https://opencores.org/>], 게이트웨어 IP (Intellectual Properties) 코어 개발을 위한 온라인 커뮤니티. 이 과정에서 사용하는 SweRVolfX SoC 의 확장 버전에는 아래 나열된 I/O 인터페이스가 포함되어 있습니다. 이 인터페이스는 RVfpga Labs 6-10 에서 사용하고 자세히 설명하며 확장할 것입니다.

- **시스템 컨트롤러:** 시스템 컨트롤러에는 SoC 버전 정보로 레지스터 유지, RAM 초기화 상태 및 RISC-V 기계 타이머와 같은 일반적인 시스템 기능이 포함되어 있습니다.  
<https://github.com/chipsalliance/Cores-SweRVolf> 에서 전체 메모리 맵을 찾을 수 있습니다.  
 이 모듈을 다음과 같이 수정했습니다.
  - o Nexys A7 보드에서 사용할 수 있는 8 숫자 7 세그먼트 디스플레이와 통신하기 위한 새 컨트롤러인 SevSegDisplays\_Controller 가 포함되었으며, 이 컨트롤러에 대해 주소 0x80001038 및 0x8000103C 에 매핑된 두 개의 새 레지스터가 포함되었습니다.
  - o 주소 0x80001018 에 매핑된 GPIO 및 PTC 의 인터럽트를 처리하기 위해 2 개의 1 비트 레지스터를 추가했습니다.
  - o SweRVolf 에서 제공하는 간단한 GPIO 레지스터를 제거하고 주소 0x80001010 ~ 0x80001017 에 매핑했습니다. 아래에 설명된 대로 보다 완전한 GPIO 컨트롤러를 추가했습니다.
- **SPI:** 두 개의 오픈 소스 SPI 컨트롤러가 ([https://opencores.org/projects/simple\\_spi](https://opencores.org/projects/simple_spi) 그리고 SPI1 과 SPI2) SweRVolfX 에서 구현됩니다. 노출된 레지스터 (SPI\_SPCR, SPI\_SPSR, SPI\_SPDR, SPI\_SPER, SPI\_SPSS)는 주소 0x80001040 과 0x8000107F (SPI1 의 경우) 사이와 주소 0x80001100 과 0x8000113F (SPI2 의 경우) 사이에 매핑됩니다.
- **PTC:** <https://opencores.org/projects/ptc>. <https://opencores.org/projects/ptc>.의 타이머 모듈을 사용합니다. 레지스터는 0x80001200 에서 0x800012FF 까지의 주소 범위에 매핑됩니다.
- **GPIO:** <https://opencores.org/projects/gpio> 의 GPIO 컨트롤러를 사용합니다. 여기에는 0x80001400 ~ 0x800014FF 주소 범위에 매핑된 32 개의 I/O 포트가 포함됩니다. 각 핀은 tristate (3-상태) 버퍼와 연결되어 있으므로 입력 또는 출력으로 구성할 수 있습니다.
- **UART:** 오픈 소스 UART 컨트롤러 (<https://opencores.org/projects/uart16550> 에서 얻음)는 SweRVolfX 에서 사용할 수 있습니다. 노출된 레지스터는 주소 0x80002000 과 0x80002FFF 사이에 매핑됩니다.

## ii. Memory

SweRVolfX SoC 에는 Boot ROM 메모리와 사용자가 RAM 및 SPI 플래시 메모리를 포함하는데 필요한 하드웨어가 포함되어 있습니다.

- **Boot ROM:** 부트 ROM에는 1 단계 부트 로더가 포함되어 있습니다. 시스템 재설정 후 SweRVofX SoC는 주소 0x80000000에서 0x80000FFF까지 차지하는 이 영역에서 초기 명령을 가져오기 시작합니다.
- **RAM:** SweRVofX SoC는 메모리 컨트롤러를 포함하지 않지만 메모리 맵 (0x00000000-0x07FFFFFF)의 첫 번째 128MiB를 예약하고 AXI 버스를 노출하므로 사용자가 메모리 컨트롤러를 사용하여 RAM 메모리에 액세스할 수 있습니다.
- **SPI 플래시:** 이전 섹션에서 설명한 SPI1 컨트롤러를 사용하여 SPI 플래시 메모리를 포함할 수도 있습니다 (주소 범위: 0x80001040-0x8000107F).

### iii. 상호 연결

SweRV EH1 Core는 AXI4 버스를 사용하여 코어와 메모리를 연결합니다. 버스는 AHB-Lite 버스로 구성할 수도 있지만 본 자료에서는 해당 옵션을 사용하지 않습니다. 모든 주변 장치 (I/O 장치)는 OpenCore CPU 및 주변 장치에서 많이 사용되는 오픈 소스 버스인 Wishbone 버스에 연결됩니다. 이 시스템에는 코어를 주변 장치에 연결하기 위한 AXI-Wishbone 브리지 (그림 21 참조)가 포함되어 있습니다.

이 섹션에서는 AXI4 버스와 Wishbone 버스의 작동에 대해 간략하게 설명합니다. 이러한 버스의 사양에 대한 지식을 확장하는 데 관심이 있는 경우 아래 제공된 참조를 사용할 수 있습니다.

#### AXI4 버스

SweRV EH1 Core Complex는 외부와 통신하기 위해 AXI4 Interconnect를 사용합니다 (그림 20 참조). AXI (Advanced eXtensible Interface)는 많은 프로세서에서 사용되는 공통 버스이며 ARM 고급 마이크로 컨트롤러 버스 아키텍처 온 칩 상호 연결 사양의 일부입니다.

다음 하위 섹션에서는 AXI4 상호 연결의 주요 측면 중 일부를 간략하게 설명합니다. 다음 문서에서 전체 AXI 사양을 찾을 수 있습니다.

[https://static.docs.arm.com/ih0022/e/IHI0022E\\_amba\\_axi\\_and\\_ace\\_protocol\\_spec.pdf](https://static.docs.arm.com/ih0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf)

- **AXI 버스 주요 기능**

AXI 버스 기술의 주요 기능은 다음과 같습니다.

- 고 대역폭 및 저 지연 (low-latency) 설계 모두에 적합합니다.
- 복잡한 bridge를 사용하지 않고 고주파수 운용 가능
- 다양한 구성 요소의 인터페이스 요구 사항을 충족할 수 있습니다.
- 초기 액세스 대기 시간이 긴 메모리 컨트롤러에 적합합니다.
- 상호 연결 아키텍처 구현에 유연성을 제공합니다.
- 기존 AHB 및 APB 인터페이스와 이전 버전 호환 가능
- 별도의 주소/제어 및 데이터 단계 제공
- 정렬되지 않은 데이터 전송 지원 (byte strobes 사용)

- 발급된 시작 주소만으로 burst-based transactions 을 허용합니다.
- 저비용 DMA 가 가능한 별도의 읽기 및 쓰기 데이터 채널을 제공합니다.
- 실제 데이터 전송에 앞서 주소 정보를 발행할 수 있습니다.
- 여러 개의 outstanding 주소 발행 및 비 순차 실행 완료를 지원합니다.
- 타이밍 클로저를 제공하기 위해 레지스터 단계를 쉽게 추가할 수 있습니다.

### • AXI 아키텍처

AXI 프로토콜은 다음과 같은 독립적인 트랜잭션(transaction) 채널을 정의합니다.

- 읽기 주소
- 읽기 데이터
- 쓰기 주소
- 데이터 쓰기
- 쓰기 응답

그림 22 은 읽기 트랜잭션이 읽기 주소를 사용하고 데이터 채널을 읽는 방법을 보여줍니다. 먼저 주소와 제어 비트가 마스터 장치에서 전송된 다음 슬레이브 장치가 읽기 데이터 채널의 데이터로 응답합니다.

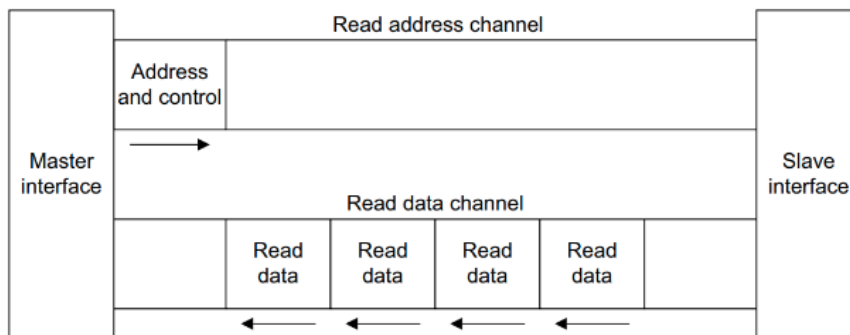


그림 22. 읽기의 채널 아키텍처

(그림 출처 : [https://static.docs.arm.com/ihi0022/e/IHI0022E\\_amba\\_axi\\_and\\_ace\\_protocol\\_spec.pdf](https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf))

그림 23 는 쓰기 트랜잭션이 쓰기 주소, 쓰기 데이터 및 쓰기 응답 채널을 사용하는 방법을 보여줍니다. 읽기와 유사하게 마스터 장치는 주소와 제어 비트를 보냅니다. 그런 다음 마스터 장치는 쓰기 데이터 채널로 데이터를 보내고 슬레이브 장치는 응답을 보냅니다.

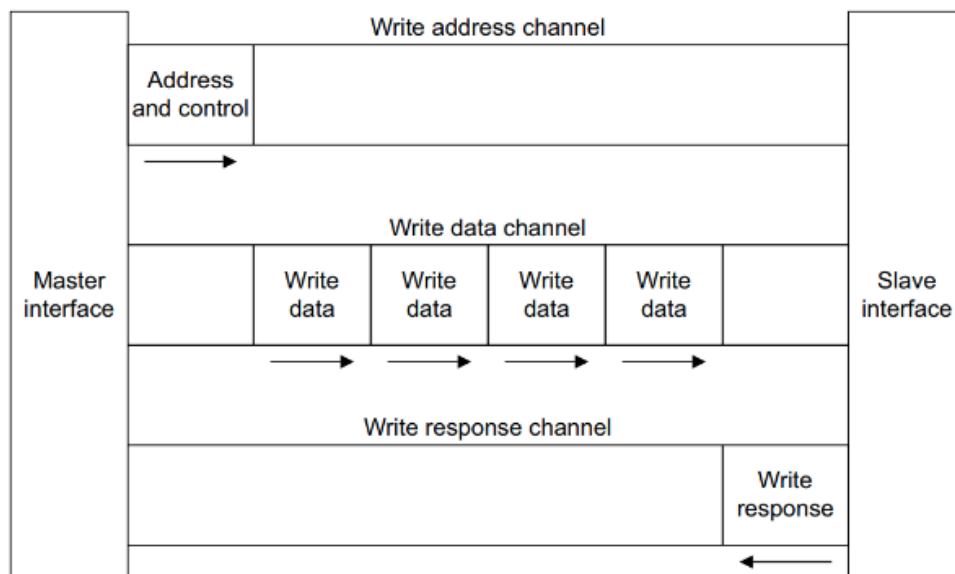


그림 23. 쓰기의 채널 아키텍처

(그림 출처 : [https://static.docs.arm.com/ihi0022/e/IHI0022E\\_amba\\_axi\\_and\\_ace\\_protocol\\_spec.pdf](https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf))

AXI 주소 채널은 전송할 데이터의 특성을 설명하는 주소 및 제어 정보를 전달합니다. 데이터는 다음 중 하나를 사용하여 마스터와 슬레이브간에 전송됩니다.

- 슬레이브에서 마스터로 데이터를 전송하기 위한 읽기 데이터 채널 (그림 22).
  - 마스터에서 슬레이브로 데이터를 전송하기 위한 쓰기 데이터 채널 (그림 23).
- 쓰기 트랜잭션에서 슬레이브는 쓰기 응답 채널을 사용하여 마스터로의 전송 완료를 알립니다 (그림 23).

#### • AXI 신호

표 7 은 AXI 버스에 사용되는 주요 신호와 각 신호에 대한 간략한 설명을 보여줍니다. 신호는 이전 섹션에서 설명한 5 개 채널에 해당하는 5 개 그룹으로 구성됩니다.

- 이름이 AW 로 시작하는 주소 채널 신호 쓰기
- 이름이 W 로 시작하는 데이터 채널 신호 쓰기
- 이름이 B 로 시작하는 응답 채널 신호를 작성합니다.
- 이름이 AR 로 시작하는 주소 채널 신호 읽기
- 이름이 R 로 시작하는 데이터 채널 신호 읽기



표 7. AXI 신호

(표 출처 : [https://static.docs.arm.com/ih0022/e/IHI0022E\\_amba\\_axi\\_and\\_ace\\_protocol\\_spec.pdf](https://static.docs.arm.com/ih0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf))

| Signal       | Source:<br>master/<br>slave | Input/<br>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 bus (위스본 버스)

SweRVofX 주변 장치는 포터블 IP 코어 용 Wishbone SoC (System-on-Chip) 상호 연결 아키텍처 (<https://opencores.org/howto/wishbone>)를 사용합니다. 이 버스의 주요 목적은 System-on-Chip 통합 문제를 완화하여 설계 재사용을 촉진하는 것입니다. 이전에는 IP 코어가 비표준 상호 연결 방식을 사용하여 통합을 어렵게 했습니다. 이러한 비표준 상호 연결에는 각 코어를 함께 연결하기 위해 사용자 지정 glue logic 을 만들어야 했습니다. Wishbone 버스와 같은 표준 상호 연결 체계를 채택함으로써 최종 사용자가 코어를 보다 빠르고 쉽게 통합할 수 있습니다.

#### • Wishbone 주요 기능

이 Wishbone 버스 기술의 주요 기능은 다음과 같습니다.

- 대규모 프로젝트 팀에서 사용하는 구조화된 설계 방법론을 지원합니다.
- 다음을 포함하여 널리 사용되는 데이터 전송 버스 프로토콜의 전체 세트를 포함합니다:
  - i. 읽기/쓰기 주기(cycle)

- ii. BLOCK 전송 주기(cycle)
  - iii. 읽기/수정/쓰기 주기(cycle)
- 모듈형 데이터 버스 폭과 최대 64 비트의 피연산자 크기를 제공합니다.  
BIG ENDIAN 및 LITTLE ENDIAN 데이터 순서를 모두 지원합니다.
- 지점간, 공유 버스, 크로스바 스위치 및 스위치 패브릭 상호 연결을 포함한 다양한 핵심 상호 연결 방법을 지원합니다.
- 여기에는 각 IP 코어가 데이터 전송 속도를 조절할 수 있는 핸드 셰이킹 프로토콜이 포함되어 있습니다.
- 단일 클럭 데이터 전송을 지원합니다.
- 정상 사이클 종료, 재시도 종료, 오류로 인한 종료를 지원합니다.
- 모듈 주소 widths 가 포함됩니다.
- 슬레이브에 대한 부분(partial) 주소 디코딩 방식을 제공합니다. 이는 고속 주소 디코딩을 용이하게하고 redundant logic 을 적게 사용하며 가변 주소 크기 조정 및 상호 연결 방법을 지원합니다.
- 사용자 정의 태그를 제공합니다. 이는 주소, 데이터 버스 또는 버스 사이클에 정보를 적용하는데 유용합니다. 사용자 정의 태그는 특히 다음과 같은 정보를 식별하기 위해 버스주기를 수정할 때 유용합니다.
  - i. 데이터 전송
  - ii. 패리티 또는 오류 수정 비트
  - iii. 인터럽트 벡터
  - iv. 캐시 (cache) 제어 작업
- 유연한 시스템 설계를위한 마스터/슬레이브 아키텍처를 포함합니다.
- 다중 처리 (multi-MASTER) 기능이 있습니다. 이를 통해 다양한 SoC 구성이 가능합니다.
- 최종 사용자가 정의할 수있는 arbitration methodology (priority arbiter, round-robin arbiter 등)가 포함됩니다.

#### • Wishbone 아키텍처 및 신호

그림 24 은 Wishbone 버스를 통한 마스터 (여기서는 SweRV EH1 Core)와 슬레이브 (여기서는 GPIO, SPI 와 같은 주변 장치) 간의 표준 연결을 보여줍니다. Wishbone 버스는 AXI4 버스보다 훨씬 간단하며 표 7 에 표시된 것처럼 신호를 더 적게 사용합니다.

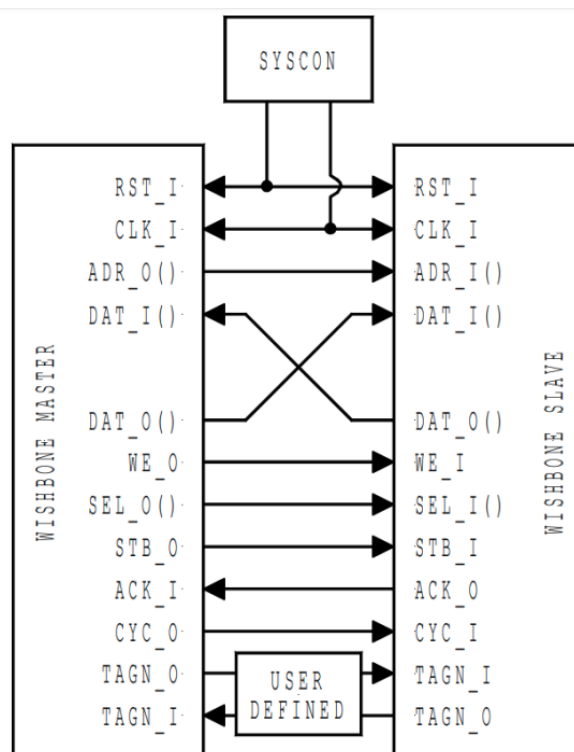


그림 24. Wishbone 아키텍처

(그림 출처 : <https://opencores.org/howto/wishbone>)

표 8. Wishbone 신호

(표 출처 : <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_I(63..0)]).       |
|             |   | RST_I()     | The reset input [RST_I] forces the WISHBONE interface to restart  |
|             |   | TGD_I()     | Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data input array [DAT_I()], and is qualified by signal [STB_I]. |
|             |   | TGD_O()     | Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with the data output array [DAT_O()], and is qualified by signal [STB_O] |

| Signal name | Description  | 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. Nexys A7 FPGA 보드 및 시뮬레이션에서 확장된 SweRVolf SoC

Extended SweRVolf SoC (그림 21)는 (1) 구성을 **RVfpgaNexys** 라고하는 Nexys A7 (또는 Nexys4 DDR) FPGA 보드에서 실행하거나 (2) 시뮬레이션 (**RVfpgaSim** 이라고 함)에서 실행할 수 있습니다.

#### i. RVfpgaNexys

RVfpgaNexys 는 Digilent Nexys A7 FPGA 보드를 대상으로 하는 SweRVolfX SoC 입니다 (그림 25). RVfpgaNexys 는 SweRVolf 를 기반으로 한다는 점을 제외하면 SweRVolf Nexys (<https://github.com/chipsalliance/Cores-SweRVolf>) 와 동일합니다. RVfpgaNexys 에서 사용하는 주요 요소는 그림 25 에 나와 있습니다.

- FPGA 에 프로그래밍된 하드웨어:
  - **SweRVolfX SoC (그림 21)**
  - **Lite DRAM 컨트롤러**
  - **클럭 생성기:** Nexys A7 보드에는 **Lite DRAM 컨트롤러**에서 사용하는 단일 **100MHz** 수정 발진기가 포함되어 있습니다. 이 클럭의 주파수는 **SweRVolfX SoC** 에서 사용하기 위해 **50MHz** 로 축소됩니다.
  - **Clock Domain Crossing 모듈:** 2 개의 클럭 도메인 연결: SweRVolfX SoC 및 Lite DRAM.
  - **JTAG 에 대한 BSCAN 로직:** 이 모듈에 대한 자세한 정보는 <https://github.com/chipsalliance/Cores-SweRVolf/issues/29> 에서 찾을 수 있습니다.
- Nexys A7 (또는 Nexys4 DDR) FPGA 보드의 RVfpga 에서 사용되는 메모리/주변 장치:
  - **DDR2 메모리 (위에서 언급 한 Lite DRAM 컨트롤러를 통해 액세스)**
  - **USB 연결**
  - **SPI 플래시 메모리**
  - **SPI 가속도계**
  - **16 개의 LED 및 16 개의 스위치**
  - **8 숫자의 7 세그먼트 디스플레이**

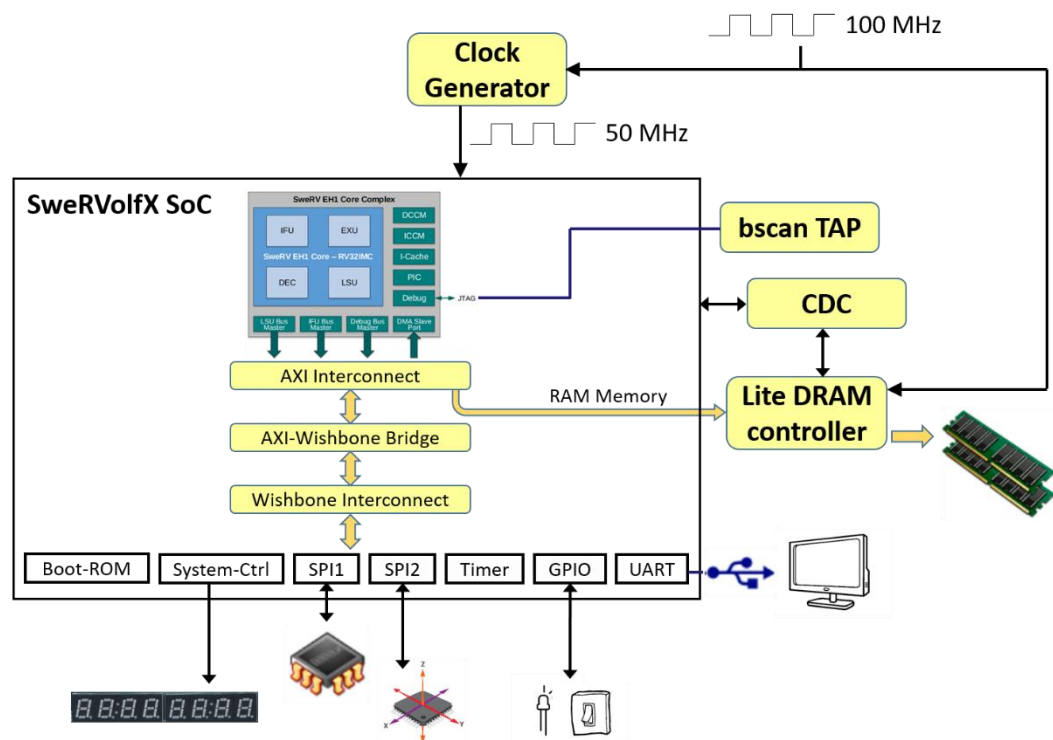


그림 25. RVfpgaNexys

Nexys A7 보드 (그림 26)는 전기 및 컴퓨터 엔지니어링 커리큘럼에 권장되는 강사용 보드입니다. 이 보드의 가격은 \$265 입니다. (또는 교육용 가격으로 \$198.75 할인된 가격 - .edu 이메일 주소로 Digilent 계정에 가입). Digilent 는

[https://reference.digilentinc.com/\\_media/reference/programmable-logic/nexys-a7/nexys-a7\\_rm.pdf](https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf) 에서 Nexys A7 보드의 광범위한 참조 매뉴얼을 제공합니다. 이 보드는 5V 변압기 (보드와 함께 제공되지 않음) 또는 보드의 microUSB 커넥터를 통해 PC 에서 전원을 공급받을 수 있습니다. Microchip PIC24 마이크로 컨트롤러는 FPGA 에 대한 로딩 프로세스를 관리하여 이 보드를 사용자 친화적인 옵션으로 만듭니다. 이 보드는 Xilinx 의 Vivado Design Suite 또는 OpenOCD 를 사용하여 프로그래밍할 수 있습니다. 원하는 구성은 FAT32 형식의 MicroSD 카드, FAT32 형식의 USB pendrive, 내부 플래시 메모리 또는 JTAG 인터페이스의 네 가지 소스 중 하나를 사용하여 FPGA 로 다운로드할 수 있습니다.



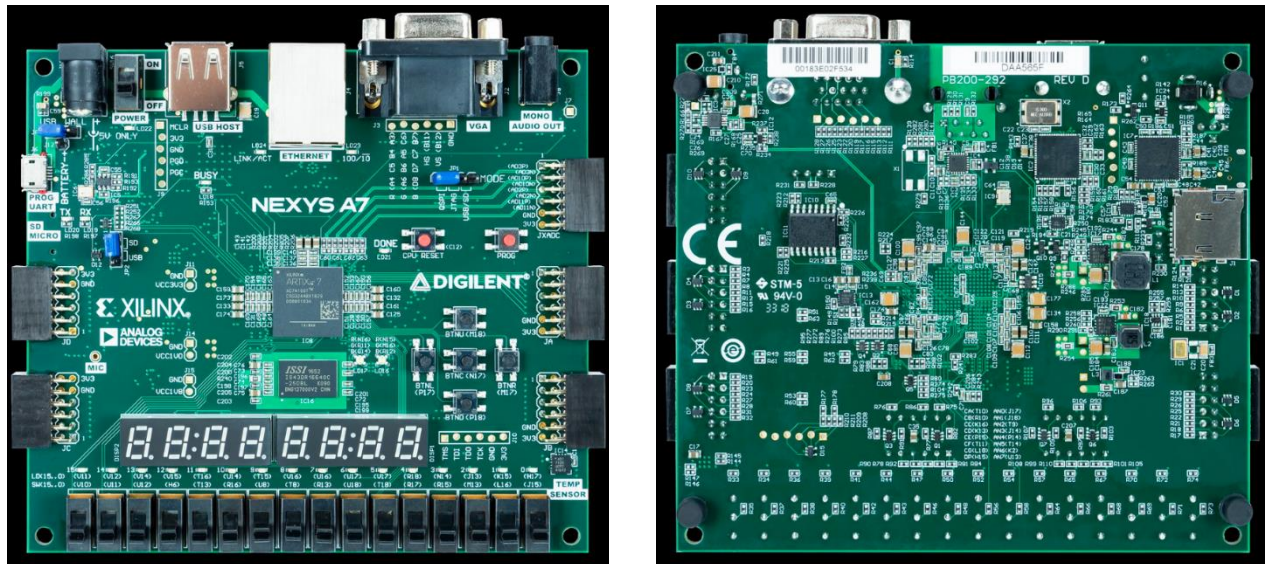


그림 26. Digilent의 Nexys A7 FPGA 보드

(그림 출처 <https://reference.digilentinc.com/>)

Nexys A7-100T FPGA 보드에는 다음 인터페이스 및 장치가 포함되어 있습니다.

- 128MiB DDR RAM
- 128 Mibit SPI 플래시 메모리
- 8 숫자의 7 세그먼트 디스플레이
- 16 개의 스위치
- 16 개의 LED
- 마이크, 오디오 잭, VGA 25 포트, USB 호스트 포트, RGB-LED, I2C 온도 센서, SPI 가속도계를 포함한 센서 및 커넥터.
- Xilinx Artix-7 FPGA: 다음과 같은 기능이 있습니다.
  - 4 개의 6 입력 LUT 와 8 개의 플립 플롭의 15.850 로직 슬라이스.
  - 4.860 Kibits 의 총 블록 RAM
  - 6 개의 시계 관리 타일 (CMT)
  - 170 I/O 핀
  - 450MHz 내부 클록 주파수

## ii. RVfpgaSim

SweRVolfX SoC (그림 21)는 시뮬레이션을 가능하게하는 Verilog 래퍼를 포함할 수도 있습니다.

RVfpgaSim 은 HDL 시뮬레이터에서 사용하기 위해 테스트 벤치에 래핑된 SweRVolfX SoC 입니다. RVfpgaSim 은 SweRVolf 를 기반으로한다는 점을 제외하면 SweRVolf sim (<https://github.com/chipsalliance/Cores-SweRVolf>) 과 동일합니다.

많은 오픈 소스 HDL 시뮬레이터가 존재하지만 Verilator

(<https://www.veripool.org/wiki/verilator>)를 사용합니다. 이 개방형 무료 HDL 시뮬레이터는 합성

가능한 Verilog 또는 SystemVerilog 를 허용하며 가장 빠른 Verilog/SystemVerilog 시뮬레이터라고 주장합니다. 그것은 산업 및 학계에서 널리 사용됩니다. ARM 및 RISC-V 공급 업체 IP 에서 기본 지원을 제공합니다. 그리고 그것은 Chips Alliance 와 Linux Foundation 에서 안내합니다.

## D. 파일 구조

이전 섹션에서 우리는 **SweRV EH1 Core Complex** (그림 20)에서 **SweRVolfX SoC** (그림 21), 마지막으로 **RVfpgaNexys** (그림 25) 및 **RVfpgaSim** 에 이르기까지, 이러한 자료에 사용하는 시스템의 상위 레벨 구성을 보여 주었습니다.

이 섹션에서는 전체 시스템의 파일 구조를 설명합니다. 이 설명을 읽으면서 파일을 열고 PC 에서 보십시오. 파일은 `[RVfpgaPath]/RVfpga/src` 에서 사용할 수 있습니다.

### i. SweRV EH1 코어 컴플렉스

그림 27 은 **SweRV EH1 Core Complex** 의 파일 구조를 보여줍니다 (그림 20). 코어는 세 가지 주요 블록으로 구성됩니다: SweRV EH1 코어 (녹색으로 강조 표시됨) 및 일부 기타 요소 (인터럽트 컨트롤러 또는 디버그 장치 등)를 포함하는 SweRV 래퍼 (회색으로 강조 표시됨) 및 데이터/명령어 메모리 및 명령 캐시 (빨간색으로 강조 표시됨).

## SweRV EH1 Core Complex (swerv\_wrapper\_dmi.sv)

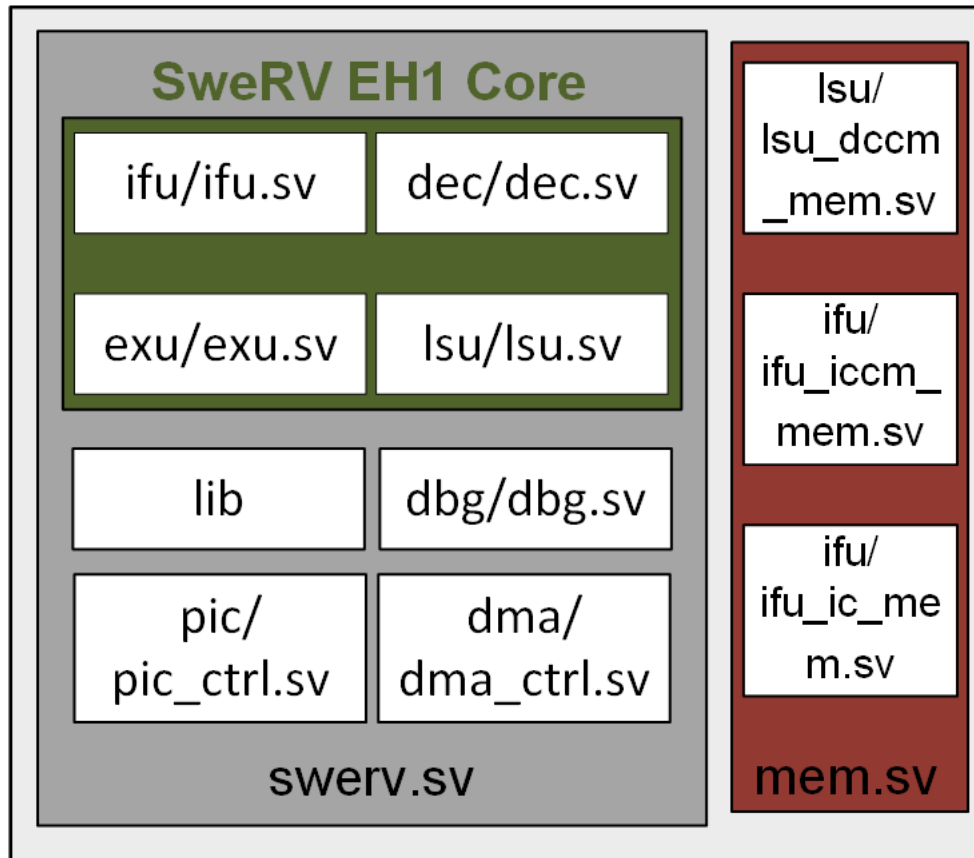


그림 27. SweRV EH1 코어 컴플렉스

SweRV EH1 Core Complex의 Verilog 파일은 다음 폴더에서 사용할 수 있습니다.

**[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex**

이 섹션에서 참조하는 파일을 보려면 PC에서 해당 디렉토리를 찾으십시오.

SweRV EH1 Core Complex의 최상위 파일은 다음 파일에 있습니다. swerv\_wrapper.sv; 최상위 모듈은 **swerv\_wrapper** 라고하며, 그림 27에서 회색과 빨간색으로 강조 표시된 두 블록에 해당하는 두 모듈을 예제를 이용하여 설명합니다.

- **mem** (mem.sv 내부에 구현 됨): 이 모듈은 DCCM (**lsu\_dccm\_mem**, *lsu/lsu\_dccm\_mem.sv* 파일에 구현 됨), ICCM (**ifu\_iccm\_mem**, *ifu/ifu\_iccm\_mem.sv* 파일에 구현 됨), 명령어 캐시 (**ifu\_ic\_mem**, *ifu/ifu\_ic\_mem.sv* 파일에 구현 됨). 구현을 위한 모듈을 예제를 이용하여 설명합니다.
- **swerv** (swerv.sv 내부에 구현 됨): 이 모듈은 코어를 구성하는 장치를 예제를 이용하여 설명합니다.



SweRV EH1 Core (그림 27 에서 녹색으로 강조 표시됨)는 다음 4 개의 장치로 구성됩니다.

- 폴더 **ifu** (Instruction Fetch Unit): 이 폴더에는 Icache (명령 캐시), Fetch, Branch Predictor 및 Aligner 에 대한 Verilog 파일이 (ifu.sv 내에서 사용 가능한 최상위 모듈) 포함됩니다.
- Folder **dec** (Decode Unit): 이 폴더에는 Instruction Decoding, Dependency Scoreboard 및 Register File 에 대한 Verilog 파일이 (dec.sv 내에서 사용 가능한 최상위 모듈) 포함됩니다.
- 폴더 **exu** (Execution Unit): 이 폴더에는 코어에서 사용할 수 있는 산술/논리 단위에 대한 Verilog 파일이 (exu.sv 내에서 사용 가능한 최상위 모듈) 포함됩니다.
- 폴더 **lsu** (저장 장치로드): 이 폴더에는 파이프 라인로드/저장 장치에 대한 Verilog 파일이 (lsu.sv 내에서 사용 가능한 최상위 모듈) 포함되어 있습니다.

이 모듈에 포함된 다른 유닛은 다음과 같습니다.

- 폴더 **dbg** (디버그 유닛): 이 폴더에는 디버그 유닛의 Verilog 파일 (dbg.sv 내에서 사용 가능한 최상위 모듈)이 포함되어 있습니다. 이 폴더는 나머지 코어를 대기 모드로 설정하고 명령/주소를 전송하고 쓰기를 전송합니다. 데이터를 읽고 데이터를 읽은 다음 코어를 다시 시작하여 정상 모드를 수행하십시오.
- 폴더 **lib**: 이 폴더에는 AXI 및 AHB-Lite 버스 용 Verilog 파일이 포함되어 있습니다.
- 모듈 **pic**: 모듈 **pic\_ctrl** - 이 모듈은 모듈 pic\_ctrl 에 있는 프로그래머블 인터럽트 컨트롤러를 구현합니다.
- 폴더 **dma**: 이 폴더에는 dma\_ctrl 모듈에서 직접 메모리 액세스를 구현하는 dma\_ctrl.sv 파일이 포함되어 있습니다.

## ii. SweRVofX SoC

그림 28 은 **SweRVofX SoC** 의 파일 구조를 보여줍니다 (그림 21). SoC 는 그림 21 에 표시된 블록에 해당하는 모듈로 구성됩니다.

## SweRVolfX SoC (swervolf\_core.v)

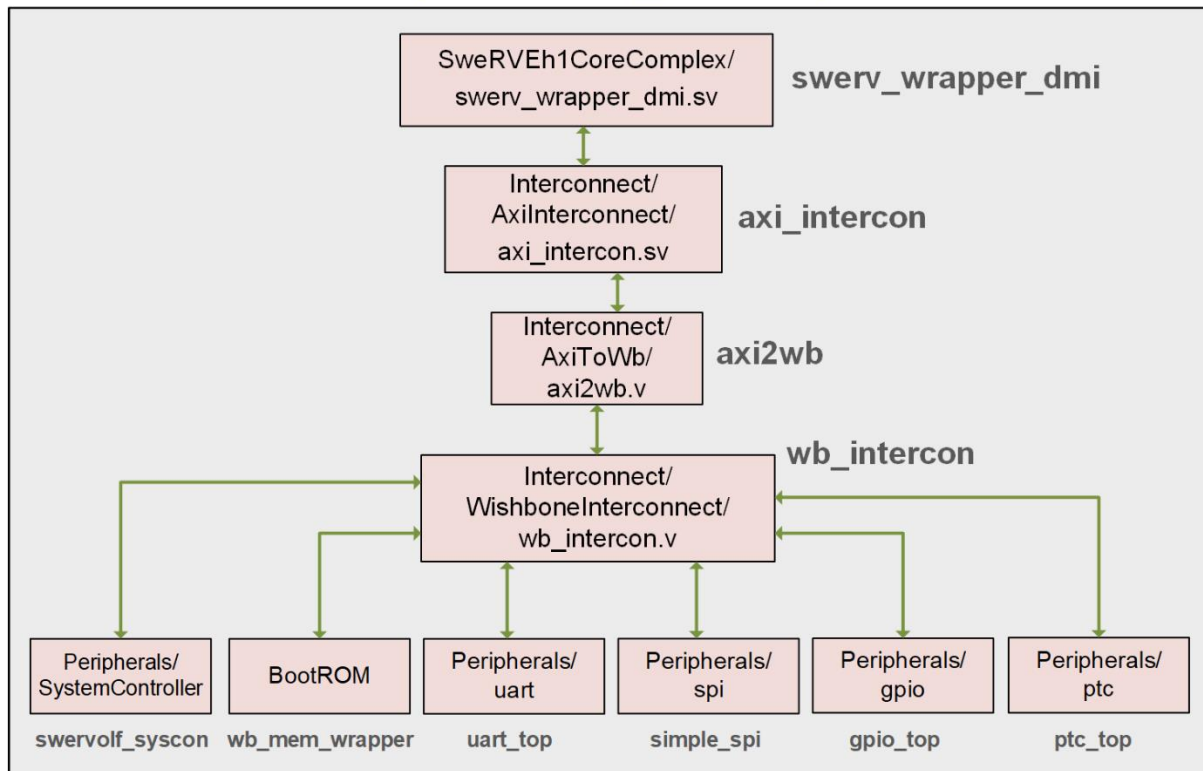


그림 28. SweRVolfX SoC

SweRVolfX SoC의 파일은 다음 위치에 있습니다:

**[RVfpgaPath]/RVfpga/src/SweRVolfSoC**

이 섹션에서 참조하는 파일을 보려면 PC에서 해당 디렉토리를 찾으십시오

SweRVolfX SoC의 최상위 모듈은

**[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf\_core.v**에서 사용할 수 있습니다. 해당 파일을 열고 SweRVolfX SoC (그림 21)에 포함된 모듈이 포함되어 있는지 확인합니다.

- **axi\_intercon** (*Interconnect/AxiInterconnect/axi\_intercon.v*에서 사용 가능): 이 모듈은 100행의 다른 파일을 통해 포함됩니다 (``include "axi_intercon.vh"`). SweRV EH1 Core Complex를 AXI-to-Wishbone Bridge와 연결합니다.
- **axi2wb** (*Interconnect/AxiToWb/axi2wb.v*에서 사용 가능): *swervolf\_core.v*의 라인 153에서 인스턴스화(예시)되는 이 모듈은 AXI 기반 EH1 Core와 Wishbone 기반 주변 장치 간의 통신을 허용하는 AXI-to-Wishbone 브리지입니다.
- **wb\_intercon** (*Interconnect/WishboneInterconnect/wb\_intercon.v*에서 사용 가능): 이 모듈은 145행의 다른 파일을 통해 포함됩니다 (``include "wb_intercon.vh"`). 나중에

분석하고 수정할 멀티플렉서를 통해 AXI-to-Wishbone Bridge 를 다른 주변기기와 연결합니다.

- **wb\_mem\_wrapper** (*BootROM/wb\_mem\_wrapper.v* 에서 사용 가능): 위에서 설명한 부트 메모리의 래퍼는 *swervolf\_core.v* 의 197 행에서 인스턴스화됩니다. 기본 RAM 모듈인 dpram64 모듈 (*Peripherals/BootROM/dpram64.v* 내에서 사용 가능)을 인스턴스화합니다.
- **swervolf\_syscon** (*Peripherals/SystemController/swervolf\_syscon.v* 내에서 사용 가능): *swervolf\_core.v* 의 215 행에서 인스턴스화되는 이 모듈은 시스템 컨트롤러를 정의합니다.
- **simple\_spi**: OpenCore 에서 얻은 SPI 컨트롤러는 *Peripherals/spi/simple\_spi\_top.v*에서 사용할 수 있습니다. *swervolf\_core.v* 의 246 (spi1) 및 387 (spi2) 행에서 인스턴스화됩니다.
- **uart\_top**: OpenCore 에서 얻은 UART 컨트롤러는 *Peripherals/uart/uart\_top.v*에서 사용할 수 있습니다. *swervolf\_core.v* 의 272 행에서 인스턴스화됩니다.
- **gpio\_top**: OpenCore 에서 얻은 GPIO 컨트롤러는 *Peripherals/gpio/gpio\_top.v*에서 사용할 수 있습니다. *swervolf\_core.v* 의 338 행에서 인스턴스화됩니다.
- **ptc\_top**: OpenCore 에서 얻은 PTC 컨트롤러이며 *Peripherals/ptc/ptc\_top.v*에서 사용할 수 있습니다. *swervolf\_core.v* 의 361 행에서 인스턴스화됩니다.
- **swerv\_wrapper\_dmi** (*SweRVeh1CoreComplex/swerv\_wrapper\_dmi.v*에서 사용 가능): 이전 섹션에서 (그림 27) 설명한 Western Digital 의 SweRV EH1 Core Complex 인스턴스화 (*swervolf\_core.v*의 407 행).

### iii. 온보드 실행 및 시뮬레이션을 위한 래퍼

#### 시뮬레이션:

**RVfpgaSim** 은 HDL 시뮬레이터에서 사용하는 테스트 벤치에서 **SweRVolfX SoC** 를 (그림 21) 래핑하는 시뮬레이션 대상입니다. *[RVfpgaPath]/RVfpga/src/rvfpgasim.v*에서 사용할 수 있습니다.

#### 온보드 실행:

**RVfpgaNexys** (*[RVfpgaPath]/RVfpga/src/rvfpganexys.v*에서 사용 가능) **SweRVolfX SoC** 를(그림 21) Nexys A7 FPGA 보드 및 주변 장치를 대상으로하는 래퍼에 연결합니다 (그림 25). 이 모듈은 다른 모듈 (클럭 생성기 모듈, **clk\_gen\_nexys**, 클럭 도메인 교차 모듈, **axi\_cdc\_intf** 또는 JTAG 포트 용 BSCAN 모듈, **bscan\_tap**)과 함께 두 가지 주요 SoC 구조를 인스턴스화합니다:

- **swervolf\_core**: 이전 하위 섹션에서 설명한 **SweRVolfX SoC** 의 인스턴스화입니다 (그림 28). 또한, SoC 와 보드 간의 연결을 정의하는 *rvfpganexys.xdc* (*[RVfpgaPath]/RVfpga/src/*에서 사용 가능)라는 제약 조건 파일이 필요합니다.
- **litedram\_top**: SweRVolfX SoC 와 DDR2 메모리를 연결하고 *[RVfpgaPath]/RVfpga/src/LiteDRAM/litedram\_top.v* 파일에 구현되는 LiteDRAM DDR2 컨트롤러 용 래퍼입니다. 또한 메모리 컨트롤러와 온보드 DDR2 메모리 간의 연결을

정의하는 *litedram.xdc* (*[RVfpgaPath]/RVfpga/src/LiteDRAM* 내에서 사용 가능)라는 제약 조건 파일이 필요합니다.

요약하면, 그림 29 은 Nexys A7 FPGA 보드에서 전체 RVfpgaNexys 구현의 계층 구조를 보여줍니다.

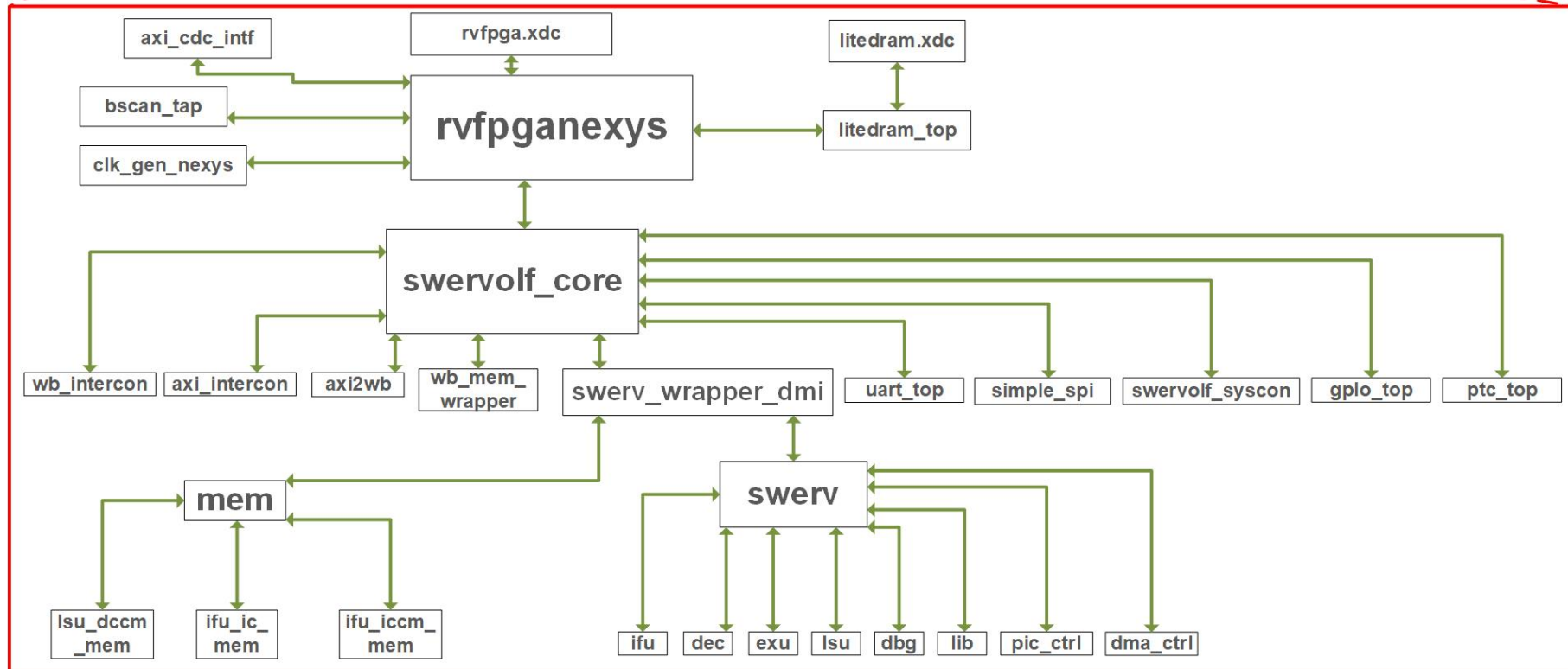
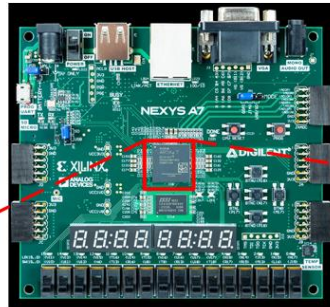


그림 29. Nexys A7 FPGA 보드 구현을위한 모듈 계층

## 5. 소프트웨어 도구 설치

아래 지침은 **Ubuntu 18.04 OS 용이지만** 다른 Linux 운영 체제와 Windows 또는 macOS 는 유사한 (정확히 동일하지는 않은 경우) 단계를 따릅니다. 경우에 따라 다른 OS 에 대한 특정 지침이 있는 상자(유닛)를 삽입합니다. Ubuntu 를 사용하는 경우 해당 상자(유닛)를 무시할 수 있습니다.

지침은 다음 도구를 설치하는 방법을 보여줍니다.

- A. Vivado:** System on Chip 을 재 합성하는 데 필요합니다. 이것은 기본 SoC 에 다른 기능이 포함되는 Labs 6-10 에서 주로 수행할 작업입니다.
- B. VSCode (비주얼 스튜디오 Code) and PlatformIO:** 이들은 GSG 와 실험실에서 사용되는 주요 도구입니다. FPGA 를 프로그래밍하고 프로그램을 실행/디버깅하는 데 사용됩니다.
- C. Verilator and GTKWave:** SoC 를 시뮬레이션하고 다양한 신호를 분석하는 데 필요합니다. 다시 말하지만, 실습 6-10 에서 주로 이러한 도구를 사용합니다.

이 GSG 와 랩에서 수행할 대부분의 작업에 대해 VSCode 및 PlatformIO 를 설치하면 충분합니다. 그러나 나중에 더 이상 설치할 필요가 없도록 다른 도구 (Vivado, Verilator 및 GTKWave)도 설치하는 것이 좋습니다.

이 프로세스는 몇 시간 (또는 다운로드 속도에 따라 그 이상)이 걸릴 수 있지만 대부분의 시간은 프로그램이 다운로드되고 설치되는 동안 대기하는 데 소요됩니다.

### A. Vivado 설치

Vivado 는 RISC-V FPGA 용 Verilog 코드를보고, 수정하고, 합성하기위한 Xilinx 도구입니다. 나중에 실습에서 광범위하게 사용할 것입니다. 설치 지침은

<https://reference.digilentinc.com/vivado/installing-vivado/start> 에서 확인할 수 있으며 아래에 요약되어 있습니다.

**Windows:** 위에서 언급 한 웹 페이지 (<https://reference.digilentinc.com/vivado/installing-vivado/start>)에는 Windows 에 Vivado 를 설치하는 자세한 지침도 포함되어 있습니다. Windows 에 특정 지침이 필요한 경우 아래에 상자를 삽입합니다.

**macOS:** Vivado 는 macOS 에서 지원되지 않습니다. 따라서 이 OS 에서 Vivado 를 실행하려면 Linux/Windows 가상 머신이 필요합니다.

1. <https://reference.digilentinc.com/vivado/installing-vivado/start> 로 이동합니다.
2. Xilinx 다운로드 페이지로 이동합니다: <https://www.xilinx.com/support/download.html>

3. “자체 추출 웹 설치 프로그램”을 설치하는 것이 좋습니다. 이 문서를 작성할 당시 다운로드 페이지의 다음 링크에 있습니다. [Xilinx Unified Installer 2019.2: Linux Self Extracting Web Installer](#)

**WINDOWS:** 이 문서를 작성할 당시 Windows 용 “자체 추출 웹 설치 프로그램”은 다운로드 페이지의 다음 링크에 있습니다: [Xilinx Unified Installer 2019.2: Windows Self Extracting Web Installer](#)

4. 설치 프로그램을 다운로드하기 전에 Xilinx 계정에 로그인하라는 메시지가 표시됩니다. 아직 계정이 없는 경우 계정을 만들어야 합니다.

5. 바이너리 파일을 실행합니다. 터미널을 열고 루트로 만듭니다 (“sudo su”입력). 그런 다음 바이너리 파일 (Xilinx\_Unified\_2019.2\_1106\_2127\_Lin64.bin)을 터미널로 드래그합니다. 파일을 실행 가능하게 만들고 실행하라는 메시지가 나타나면 확인을 선택합니다.

- **문제 해결:** 터미널에 권한이 거부되었다고 표시되면 터미널에 다음을 입력합니다 (바이너리 파일과 동일한 디렉토리에 있음):  

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

**WINDOWS:** Windows에서는 3 단계와 4 단계에서 다운로드한 .exe 파일을 두 번 클릭하여 간단히 실행할 수 있습니다.

6. Vivado 설치 프로그램이 설치 과정을 안내합니다. 중요 사항:

- 설치할 제품으로 **Vivado** (Vitis 아님)를 선택합니다.
- Vivado HL **Webpack** (Vivado HL System Edition 아님)을 선택합니다. Webpack 은 무료입니다.
- 그렇지 않으면 기본값을 선택해야 합니다.

**Hint:** Vivado 의 설치 디렉토리를 변경한 경우 다음 단계에서 경로를 적절하게 수정해야 합니다.

**WINDOWS:** Windows에서는 7 단계와 8 단계가 필요하지 않습니다. 이 두 단계를 무시하고 9 단계로 바로 이동할 수 있습니다.

7. Vivado 를 설치한 후 환경을 설정해야 합니다. 터미널을 열고 다음을 입력하십시오.  

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

다음 줄을 (source /tools/Xilinx/Vivado2019.2/settings64.sh), 사용자의 ~/.bashrc 파일에 추가하여 터미널을 시작할 때마다 실행되도록 합니다.

8. 터미널에 다음을 입력하여 Vivado 를 테스트합니다.

```
vivado
```

#### 문제 해결:

- 시스템에서 해당 실행 파일을 찾을 수 없는 경우 경로에 다음을 추가해야 합니다:

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

- "응용 프로그램 별 초기화 실패..."와 같은 오류가 발생하면 터미널에 다음을 입력합니다.

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

9. Nexys A7 FPGA 보드 용 케이블 드라이버를 수동으로 설치해야 합니다. 터미널 창에 다음을 입력하십시오:

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

**WINDOWS:** Windows 에 Vivado 를 설치하면 PlatformIO 와 호환되지 않는 Nexys A7 보드용 드라이버가 자동으로 설치됩니다. 따라서 Windows 를 사용하는 경우 부록 B 에 설명된 대로 드라이버를 업데이트해야 합니다. Vivado 설치시 드라이버를 덮어 썼으므로 빠른 시작 가이드 섹션에서 이미 업데이트 했더라도 작업을 수행해야 합니다.

10. 또한 Digilent 보드 파일을 수동으로 설치해야 합니다.

- Github 저장소에서 vivado 보드의 [archive](#) 를 다운로드하고 압축을 풉니다.
- 아카이브에서 추출한 폴더를 열고 *new/board\_files* 디렉토리로 이동합니다. 이 디렉토리 내의 모든 폴더를 선택하고 복사하십시오.
- Vivado 가 설치된 폴더를 엽니다 (기본적으로 */tools/Xilinx/Vivado*). 이 폴더 아래에서 *<version>/data/boards/board\_files* 디렉토리로 이동한 다음 보드 파일을 이 디렉토리에 붙여 넣습니다.
- *new/board\_files* 디렉토리로 이동하여 다음을 입력하여 터미널을 사용할 수 있습니다.

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

**WINDOWS:** 10 단계에 설명된 대로 다운로드 한 폴더를 복사/붙여 넣기합니다. Windows 에서는 *C:\Xilinx\Vivado\2019.2\data\boards\board\_files* 에서 Vivado 의 board\_files 폴더를 찾을 수 있습니다.



## B. VSCode 및 PlatformIO 설치

이제 VSCode 및 PlatformIO 를 설치합니다. 빠른 시작 가이드 – 섹션 1 에서 이미 이 작업을 수행한 경우 여기에서 프로세스를 다시 반복할 필요가 없으며 섹션 C 로 바로 이동할 수 있습니다.

PlatformIO 는 Microsoft 의 Visual Studio (VS) 코드를 기반으로 구축된 임베디드 시스템용 통합 개발 환경 (IDE)입니다. 이를 통해 C 또는 어셈블리를 사용하여 RISC-V 프로세서 (FPGA 에 있는)를 프로그래밍할 수 있습니다. PlatformIO 는 크로스 플랫폼이며 내장 디버거를 포함합니다.

VSCode 와 PlatformIO 를 모두 설치하려면 다음 단계를 따르십시오:

**LINUX command-line:** VSCode + PlatformIO 를 사용하는 것이 권장되는 방법이지만 부록 A 는 Linux 에서 기본 RISC-V 도구 체인 및 OpenOCD 를 설치 및 사용하고 PlatformIO 대신 사용하려는 모든 사용자를 위한 지침을 제공합니다. PlatformIO 를 사용하려면 부록 A 를 무시하십시오.

### 1. VSCode 설치:

다음 단계에 따라 VSCode 를 설치하십시오:

- a. 다음 링크에서 .deb 파일을 다운로드하십시오:

<https://code.visualstudio.com/Download>

- b. 터미널을 열고 터미널에 다음을 입력하여 VSCode 를 설치하고 실행합니다.

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

**Windows/macOS:** VSCode 패키지는 <https://code.visualstudio.com/Download> 에서 Windows (.exe 파일) 및 macOS (.zip 파일) 용으로도 제공됩니다. 이러한 운영 체제에서 응용 프로그램을 설치하고 실행하는 데 사용되는 일반적인 단계를 따르십시오.

### 2. VSCode 위에 PlatformIO 설치:

PlatformIO 를 설치하려면 다음 단계를 따르십시오.

- a. 터미널에 다음을 입력하여 python3 유틸리티를 설치합니다.

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

**Windows/macOS:** Windows 에서이 단계 (2.a)가 필요하지 않습니다. macOS 의 경우 homebrew 를 사용하여 python3 을 설치할 수 있습니다. `brew install python3`

- b. 아직 열리지 않은 경우, 시작 버튼을 선택하고 검색 메뉴에 "VSCode"를 입력하여 VSCode 를 시작한 다음 VSCode 를 선택하거나, 터미널에 코드를 입력합니다.


- c. VSCode 에서 VSCode 의 왼쪽에 있는 확장 아이콘  을 클릭합니다 (그림 30 참조).



그림 30. VSCode 의 확장 아이콘

- d. 검색 상자에 *PlatformIO* 를 입력하고 옆에 있는 설치 버튼을 클릭하여 PlatformIO IDE 를 설치합니다 (그림 31 참조).

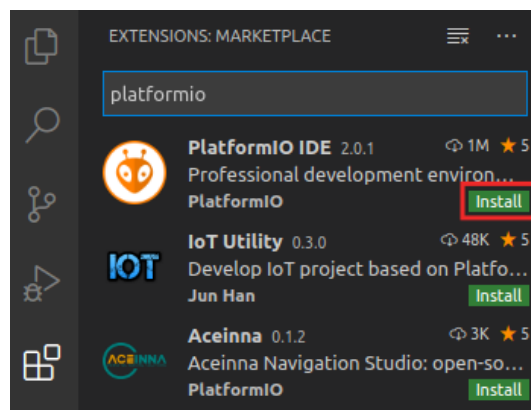
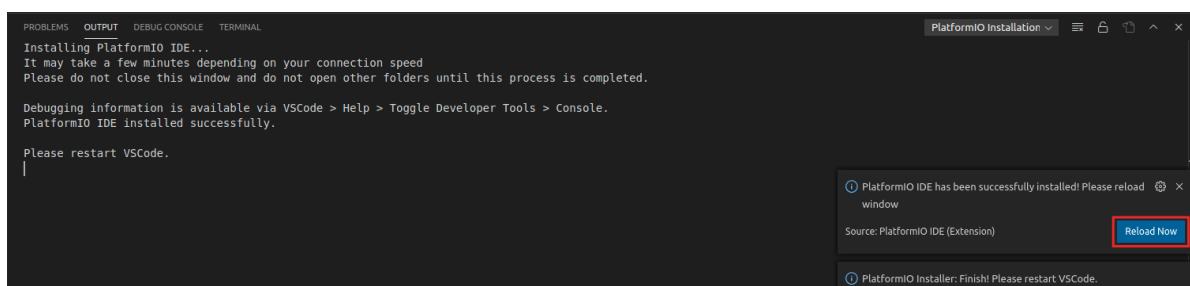


그림 31. PlatformIO IDE 확장

- e. 하단의 OUTPUT 창은 설치 과정을 알려줍니다. 완료되면 오른쪽 하단 창에서 "Reload Now"를 클릭하면 PlatformIO 가 VSCode 내에 설치됩니다 (그림 32 참조).



## 그림 32. PlatformIO 설치 후 지금 다시로드

### C. Ubuntu 18.04 에 Verilator 및 GTKWave 설치

이 섹션의 지침은 Linux 시스템에만 유효합니다.

**Windows:** 이 섹션에 제공된 지침 대신 부록 C 를 사용하십시오.

**macOS:** 이 섹션에 제공된 지침 대신 부록 D 를 사용하십시오.

다음 단계에 따라 Ubuntu 18.04 Linux 시스템에 Verilator (지침은 <https://www.veripool.org/projects/verilator/wiki/Installing> 에서 확인할 수 있지만 아래에 요약되어 있음) 및 GTKWave 를 설치합니다. 이 프로세스는 시간이 오래 걸립니다:

- `sudo apt-get install git make autoconf g++ flex bison libfl2 libfl-dev`
- `sudo apt-get install -y gtkwave`
- `git clone https://git.veripool.org/git/verilator`
- `cd verilator`
- `git pull`
- `git checkout v4.106`
- `autoconf`
- `./configure`
- `make` (또는 `make -j$(nproc)` 를 사용하여 더 빠르게 만들 수 있습니다.)
- `sudo make install`
- `export PATH=$PATH:/usr/local/bin` (시스템의 경로 변경)

`/usr/local/bin` 을 경로에 영구적으로 추가하려면 `~/.bashrc` 파일에 마지막 행을 추가하십시오.

## 6. RVfpgaNexys 실행 및 프로그래밍

이 섹션에서는 RVfpgaNexys 에서 7 개의 간단한 프로그램을 실행하는 방법을 보여줍니다 (그림 25 참조).

**LINUX / Windows / macOS:** 이 섹션에 설명된 모든 지침은 섹션 5 에 설명된 대로 모든 필수 도구와 드라이버가 올바르게 설치되었다고 가정할 때 세 가지 운영 체제에서 작동해야 합니다. 경우에 따라 Linux 에서 사용되는 슬래시 또는 Windows 에서 사용되는 백 슬래시와 같은 사소한 세부 정보를 수정해야 할 수도 있습니다.

표 9 에 나열된 7 개의 예제 프로그램을 실행하는 방법을 보여줌으로써 RVfpgaNexys 를 사용하는 방법을 보여줍니다. 처음 세 프로그램은 RISC-V 어셈블리 언어로 작성되고 마지막 4 개 프로그램은 C 로 작성됩니다. 각 프로그램을 실행하는 방법 RVfpgaNexys 는 아래에 설명되어 있습니다.

**표 9. RVfpgaNexys 예제 프로그램**

| Program Name                | Description                              | Language        |
|-----------------------------|--|-----------------|
| <b>AL_Operations</b>        | 산술 및 논리 연산 연습                            | RISC-V assembly |
| <b>Blinky</b>               | Nexys A7 보드의 LED 를 깜박입니다.                | RISC-V assembly |
| <b>LedsSwitches</b>         | Nexys A7 보드에서 스위치 값을 읽고 해당 값을 LED 에 씁니다. | RISC-V assembly |
| <b>LedsSwitches_C-Lang</b>  | Nexys A7 보드에서 스위치 값을 읽고 해당 값을 LED 에 씁니다. | C               |
| <b>HelloWorld_C-Lang</b>    | 직렬 포트를 통해 shell 에 짧은 메시지를 인쇄합니다.         | C               |
| <b>VectorSorting_C-Lang</b> | 벡터를 가장 큰 것에서 가장 작은 것까지 정렬합니다.            | C               |
| <b>DotProduct_C-Lang</b>    | 두 벡터의 내적(스칼라 곱)을 계산합니다.                  | C               |

이 7 가지 예제 중 하나를 실행하기 전에 다음 섹션에서 설명하는 것처럼 **RVfpgaNexys** 로 FPGA 를 프로그래밍해야 합니다.

## A. RVfpgaNexys 로 FPGA 프로그래밍

이 섹션에서는 PlatformIO 를 사용하는 RVfpgaNexys 로 FPGA 를 프로그래밍하는 데 권장되는 방법을 설명합니다. RVfpgaNexys 로 FPGA 를 프로그래밍하려면 다음 단계를 따르십시오.

(FPGA 프로그래밍에 Vivado 를 사용하는 데 관심이 있는 경우 아래 지침 대신이 가이드의 부록 E 에 제공된 지침을 따를 수 있습니다. 그러나 여기에 설명된 방법은 Linux 및 Windows 시스템 (macOS 아님)에서만 가능합니다.) – 그리고 전반적으로 Vivado 를 사용하여 RVfpgaNexys 를 FPGA 에 다운로드하는 방법은 권장되지 않습니다. 대신에 아래 지침을 따르고 부록 E 를 무시하는 것이 좋습니다.)

- a. Nexys A7 보드를 컴퓨터에 연결합니다.

- b. 왼쪽 상단의 스위치를 사용하여 Nexys A7 보드를 컵니다.
- c. 아직 열려 있지 않은 경우 VSCode 및 PlatformIO 를 엽니 다.
- d. 상단 메뉴 표시 줄에서 File → Open Folder (그림 33 참조)를 클릭하고 `[RVfpgaPath]/RVfpga/examples/` 디렉토리로 이동합니다.

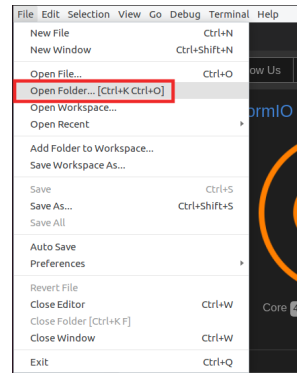


그림 33. 폴더 열기

- e. 사용할 PlatformIO 프로젝트를 선택하십시오. 이 섹션에서는 예를 들어 표 98 에 언급된 첫 번째 예인 AL\_Operations 를 사용합니다. 이 예는 다음 섹션에서 디버깅할 것입니다. 그러나 다른 예에서도 동일한 단계를 수행할 수 있습니다. 따라서 `AL_Operations` 디렉토리를 선택하고 (열지 말고 선택하기만 하면됩니다. 그림 34 참조) 창 상단에서 OK 를 클릭합니다. PlatformIO 는 이제 예제를 엽니다.

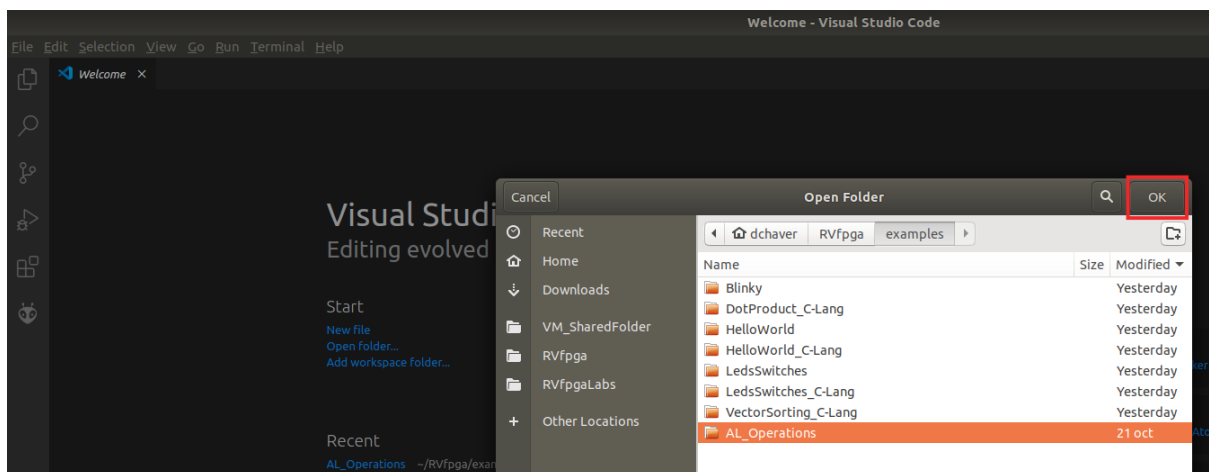


그림 34. AL\_Operations 폴더 열기

- f. 왼쪽 사이드 바에서 `platformio.ini` 를 클릭하여 `platformio.ini` 파일을 엽니 다 (그림 35 참조). 다음 줄을 편집하여 시스템에서 RVfpgaNexys 비트 스트림 경로를 설정합니다 (그림 35 참조). RVfpgaNexys 의 사전 합성된 비트 스트림은 `[RVfpgaPath]/RVfpga/src/rvfpganexys.bit` 의 RVfpga 폴더에 제공됩니다.

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

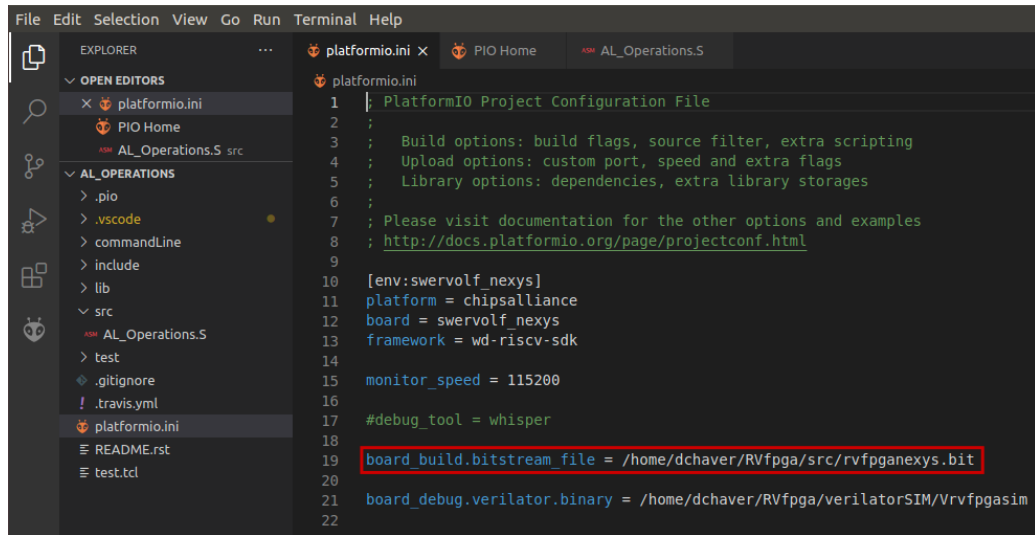


그림 35. Platformio 초기화 파일: platformio.ini

프로젝트 구성 파일 (*platformio.ini*)에서 사용할 수 있는 다양한 명령이 있으며 다음 위치에서 정보를 찾을 수 있습니다: <https://docs.platformio.org/en/latest/projectconf/>.



- g. 왼쪽 메뉴 리본에서 PlatformIO 아이콘  을 클릭합니다 (그림 36 참조).



그림 36. PlatformIO 아이콘

프로젝트 작업창이 비어 있는 경우 (그림 37) 먼저을 클릭  하여 프로젝트 작업을 새로 고쳐야 합니다. 몇 분 정도 걸릴 수 있습니다.

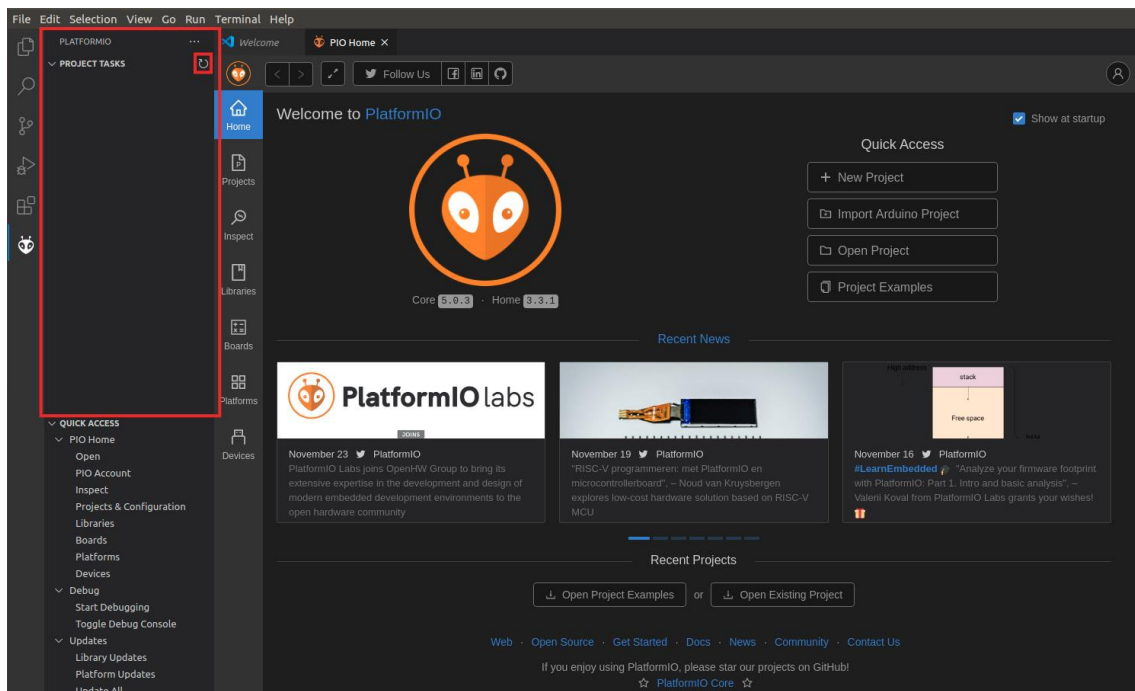


그림 37. 비어있는 PROJECT TASKS 창 – 새로 고침

그런 다음 Project Tasks → env : swervolf\_nexys → Platform 을 확장하고 Upload Bitstream 을 클릭합니다 (그림 37 참조). 1 ~ 2 초 후에 FPGA 는 RVfpgaNexys 로 프로그래밍 됩니다

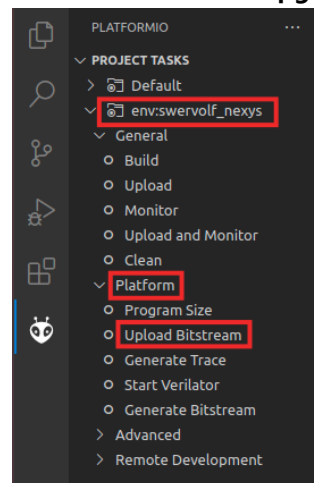



그림 38. Upload Bitstream

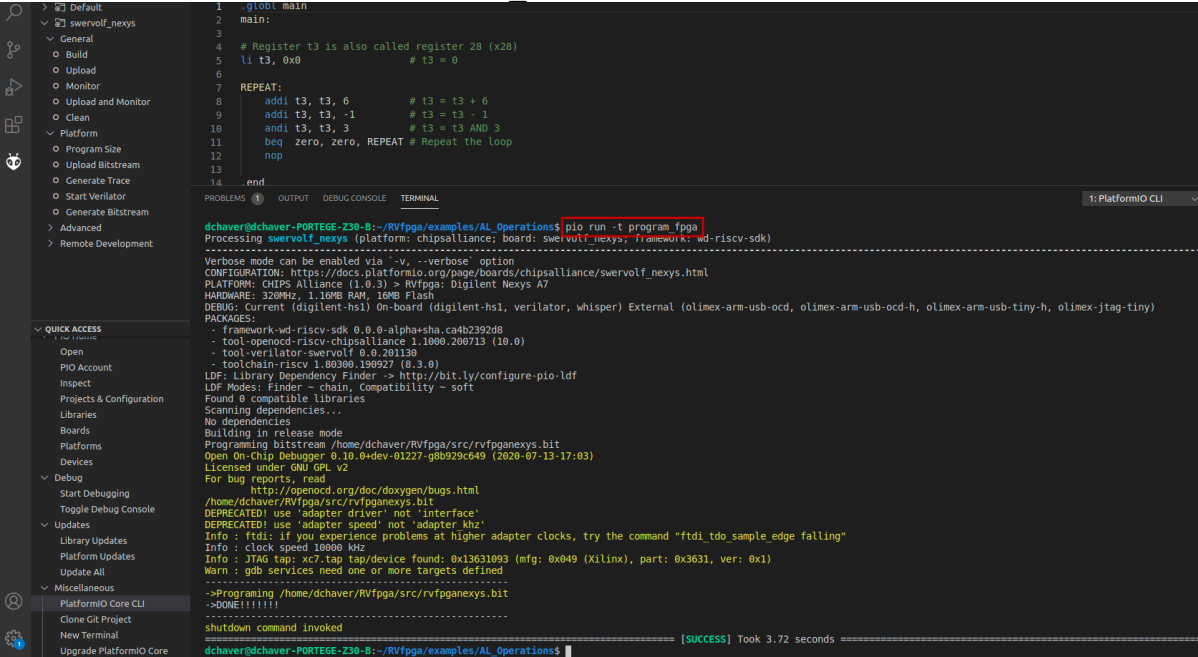
기본적으로 프로세서는 주소 0x80000000 에서 명령 가져 오기를 시작합니다. 여기서 부팅 ROM 은 SoC 에 있습니다 (표 6 참조). Boot ROM 은 LED 와 7-Segment 디스플레이를 4 번 깜박인 다음 모든 LED 를 끄고 8 개의 7-Segment 디스플레이에 0 을 쓰고 빈 루프를 유지하는 프로그램 (boot\_main.mem)으로 초기화됩니다. 이 프로그램은 [RVfpgaPath]/RVfpga/src/SweRVolfSoC/BootROM/sw 폴더에서 찾을 수 있습니다.

Nexys A7 보드 (그림 26)의 CPU 재설정 버튼을 누르면 이 프로그램이 다시 실행됩니다.

이 프로그램을 변경하고 다시 컴파일하려면 부록 A – 섹션 III 에 설명된 대로 수행하십시오 (boot\_main.mem 파일은 단순히 boot\_main.vh 파일의 복사본입니다). 실습 1 에서는 비트 스트림을 생성할 때 이 프로그램을 사용하여 Boot ROM 을 초기화하는 방법을 보여줍니다.

- h. 이전 단계 (g 단계)의 대안으로 그림 39 과 같이 PlatformIO 터미널 창에서 RVfpgaNexys 를 다운로드할 수 있습니다. 새 터미널을 열려면 PlatformIO 창 하단에있는  버튼 (PlatformIO: 새 터미널 버튼)을 클릭합니다. 창을 클릭한 다음 PlatformIO 터미널에 다음 명령을 입력 (또는 복사)합니다:

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



```

> Default:
> swervolf_nexys
  General
  Build
  Upload
  Monitor
  Upload and Monitor
  Clean
  Platform
  Program Size
  Upload Bitstream
  Generate Trace
  Start Verilog
  Generate Bitstream
  Advanced
  Remote Development

  QUICK ACCESS
  Open
  PIO Account
  Inspect
  Projects & Configuration
  Libraries
  Boards
  Platforms
  Devices
  Debug
  Start Debugging
  Toggle Debug Console
  Updates
  Library Updates
  Platform Updates
  Update All
  Miscellaneous
  PlatformIO Core CLI
  Clone Git Project
  New Terminal
  Upgrade PlatformIO Core

1  .glbl 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

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
1: PlatformIO CLI

dchaver@dchaver-PORTege-Z30-B:~/RVfpga/examples/AL_Operations$ pio run -t program fpga
Processing swervolf_nexys (platform: chipsalliance; board: swervolf_nexys; framework: wd-riscv-sdk)
Verbose mode can be enabled via '-v, --verbose' option
CONFIGURATION: https://docs.platformio.org/page/boards/chipsalliance/swervolf_nexys.html
PLATFORM: CHIPS Alliance (1.0.2) > RVfpga: Diligent Nexys A7
HARDWARE: 32MHz, 1.16MB RAM, 10MB Flash
DEBUG: Current (digilent-hsl) On-board (digilent-hsl, verilator, whisper) External (olimex-arm-usb-ocd, olimex-arm-usb-ocd-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-swervolf 0.0.201130
  - toolchain-riscv 1.80300.190927 (8.3.0)
LDF: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF Modes: Finder - chain, Compatibility - soft
Found 0 compatible Libraries
Scanning dependencies...
No dependencies
Building in release mode
Programming bitstream /home/dchaver/RVfpga/src/rvfpganexys.bit
Open On-Chip Debugger 0.10.0+dev-01227-g8b929c649 (2020-07-13-17:03)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
/home/dchaver/RVfpga/src/rvfpganexys.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter kHz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_udo_sample_edge falling"
Info : clock speed 10000 KHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
-----
->Programming /home/dchaver/RVfpga/src/rvfpganexys.bit
->DONE!!!!!!
shutdown command invoked
===== [SUCCESS] Took 3.72 seconds =====
dchaver@dchaver-PORTege-Z30-B:~/RVfpga/examples/AL_Operations$

```

**그림 39. PlatformIO 를 사용하여 Nexys A7 FPGA 보드에 RVfpgaNexys 업로드**

예제가 PlatformIO 에서 처음 열리면 Chips Alliance 플랫폼이 자동으로 설치됩니다 (그림 40 와 같이 PIO 홈에서 볼 수 있음). 이 플랫폼에는 미리 빌드 된 RISC-V 도구 모음, RISC-V 용 OpenOCD, RVfpgaNexys 비트 파일 및 RVfpgaSim, JavaScript 및 Python 스크립트, 여러 예제와 같이 나중에 사용할 여러 도구가 포함되어 있습니다.



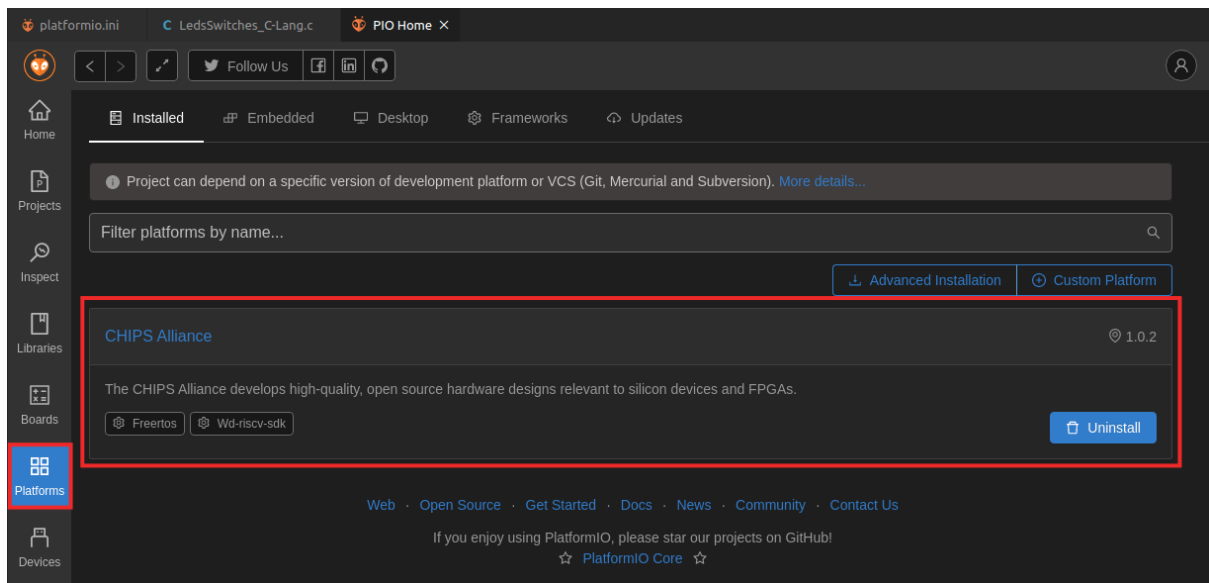






그림 40. PlatformIO 에 설치된 Chips Alliance 플랫폼

어떤 이유로든 Chips Alliance 플랫폼이 자동으로 설치되지 않은 경우 다음 단계에 따라 수동으로 설치할 수 있습니다 (일반적으로이 절차를 건너 뛰고 섹션 B 를 계속할 수 있습니다).

- 왼쪽 사이드바에 있는 버튼  을 클릭하여 빠른 액세스 메뉴를 봅니다 (그림 41 참조). 그런 다음 PIO 홈에서  버튼을 클릭 한 다음 탭  Embedded 을 클릭합니다 (그림 41). Chipsalliance (RVfpga 에서 사용하는 플랫폼)를 찾아서 버튼  을 클릭하여 엽니다 (그림 41).

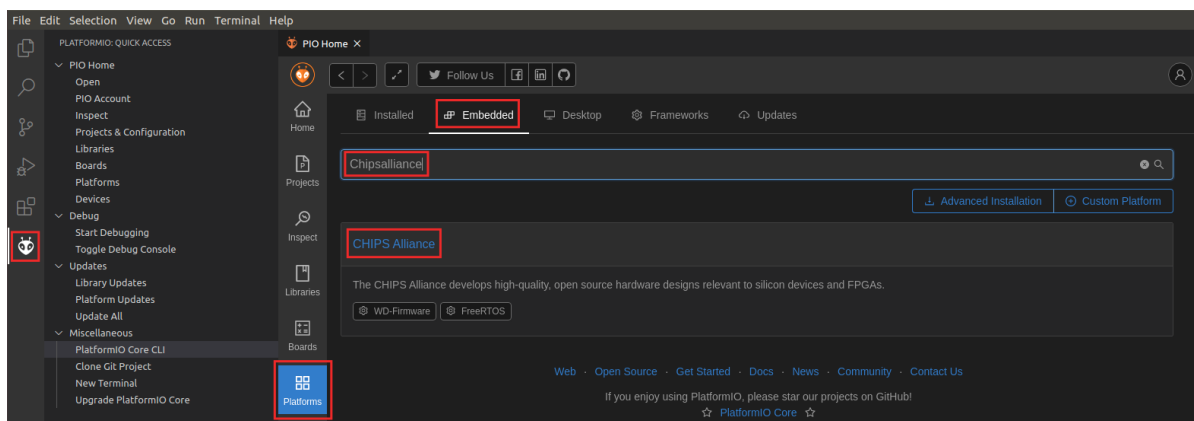




그림 41. CHIPS Alliance 플랫폼 선택

-  버튼을 클릭하면 Chips Alliance 플랫폼의 세부 정보가 표시됩니다 (그림 42 참조).  버튼을 클릭하여 설치하십시오 (그림 42).

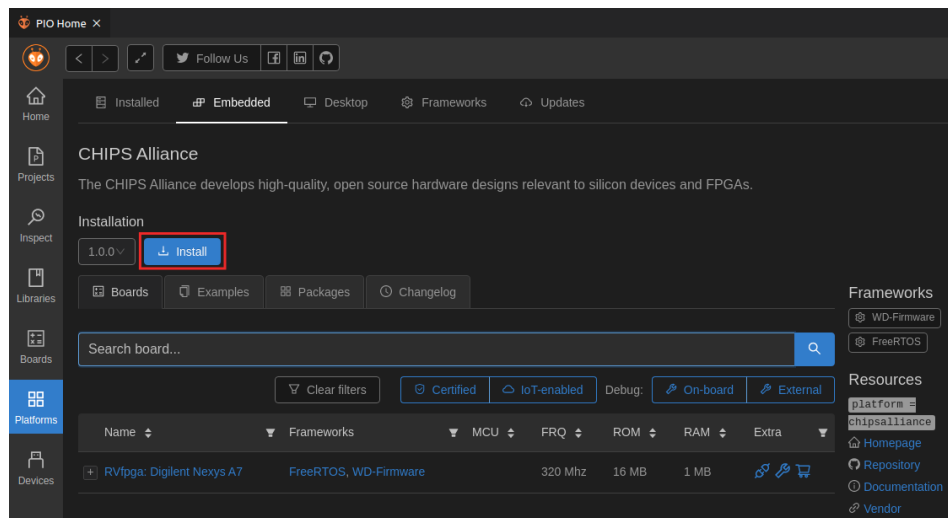



그림 42. CHIPS Alliance 플랫폼 설치

- 설치가 완료되면 그림 43 와 같이 설치된 도구의 요약이 표시됩니다. 해당 창을 닫으려면 클릭  하십시오.

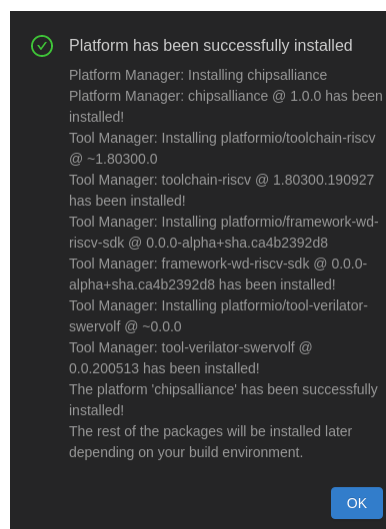


그림 43. CHIPS Alliance Platform의 성공적인 설치

## B. AL\_Operations 프로그램

첫 번째 예제 프로그램 인 AL\_Operations.s (그림 44 참조)는 무한 루프 내에서 동일한 레지스터 t3 (x28 이라고도 함)에 대해 세 개의 산술 논리 명령어 (더하기, 빼기 및 논리 and)를 수행하는 어셈블리 프로그램입니다.

```
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

```

그림 44. AL\_Operations 프로그램: AL\_Operations.S

PlatformIO 를 사용하여 Nexys A7 FPGA 보드에서 이 코드를 실행하고 디버깅하려면 다음 단계를 따르십시오.

1. 이전 섹션에서 설명한대로 FPGA 를 프로그래밍 합니다. PlatformIO 에서 이미 AL\_Operations 프로젝트가 열려 있습니다.

2. 왼쪽 메뉴 리본의 탐색기 아이콘  을 클릭하고 왼쪽 사이드바의 *AL\_OPERATIONS* 아래에 있는 *src* 를 확장한 다음 AL\_Operations.S 를 클릭하여 어셈블리 프로그램 AL\_Operations.S 를 엽니다 (그림 45 참조).

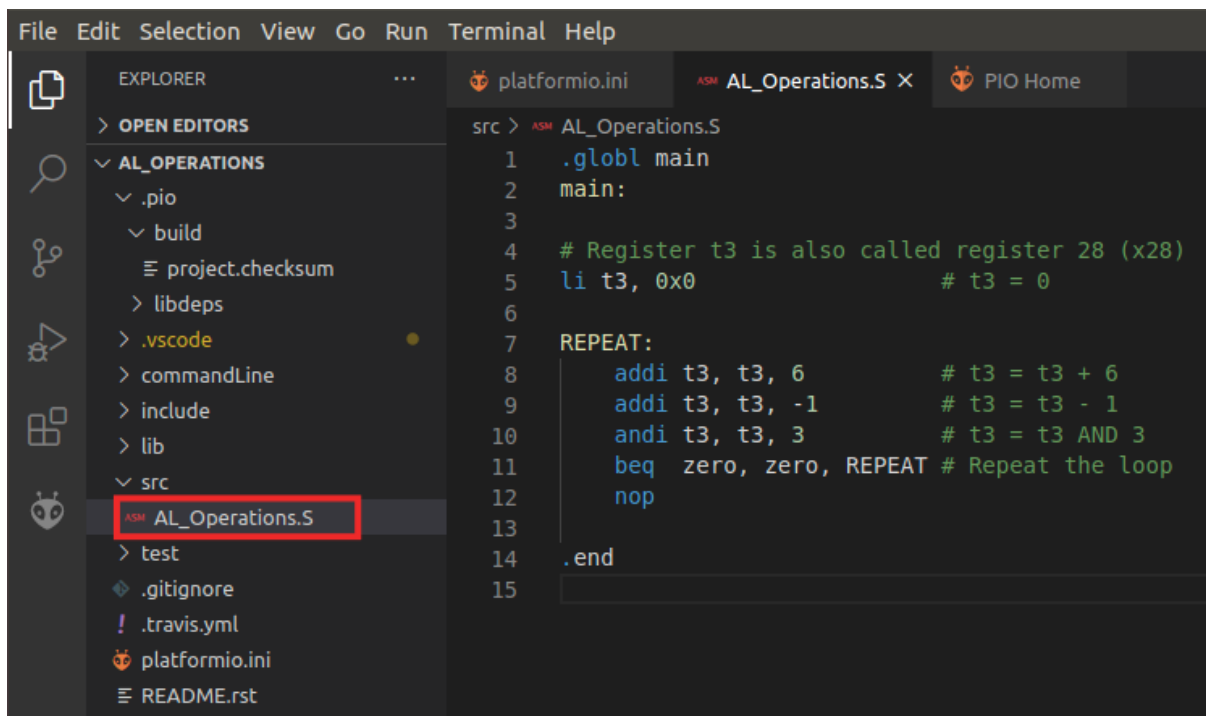








그림 45. 어셈블리 파일 AL\_Operations.S 보기

3. VSCode 및 PlatformIO 는 프로그램을 컴파일, 정리 및 디버깅하는 다양한 방법을 제공합니다.

VSCode 하단에는 유용한 기능을 제공하는 몇 가지 버튼  이 있습니다. 예를  들어  는 프로젝트를 빌드하는 데 사용하거나 정리하는 데 사용할 수

있습니다. 왼쪽 사이드바 (그림 30 참조)에서 "실행"  버튼을 사용하여 프로그램을 컴파일한 다음 디버거를 열 수 있습니다.

4. 실행"  버튼을 클릭합니다. 재생 버튼  을 클릭하여 디버거를 시작합니다 ("PIO 디버그" 옵션이 선택되어 있는지 확인하십시오). 창 상단 근처에 이 버튼이 있습니다 (그림 46 참조). 프로그램이 먼저 컴파일되고 디버깅이 시작됩니다. PlatformIO 는 주 함수의 시작 부분에 임시 breakpoint 를 설정하므로 실행이 중지됩니다.

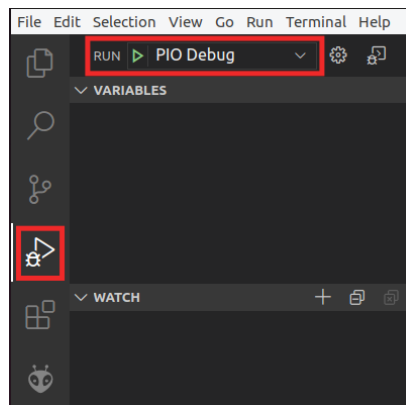
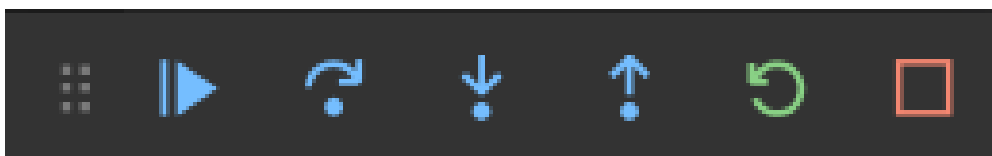


그림 46. 디버거 시작


5. 디버깅 세션을 제어하기 위해 편집기 상단 근처에 나타나는 디버깅 도구 모음을 사용할 수 있습니다 (그림 47 참조). 다음은 옵션입니다.
  - **Continue** 다음 중단점까지(breakpoint) 프로그램을 실행합니다.
  - **Breakpoints** 편집기에서 줄 번호 왼쪽을 클릭하여 추가할 수 있습니다.
  - **Step Over** 현재 행을 실행한 다음 중지합니다.
  - **Step Into** 현재 줄을 실행하고 현재 줄에 함수 호출이 포함되어 있으면 해당 함수로 점프하여 중지합니다.
  - **Step Out** 현재있는 함수의 모든 코드를 실행한 다음 해당 함수가 반환되면 중지합니다.
  - **Restart** 프로그램 시작부터 디버깅 세션을 다시 시작합니다.
  - **Stop** 디버깅 세션을 중지하고 일반 편집 모드로 돌아갑니다.
  - **Pause** 실행을 일시 중지합니다. 프로그램이 실행 중이면 계속 버튼이 일시 중지 버튼으로 바뀝니다.



CONTINUE STEP-OVER STEP-INTO STEP-OUT RESTART STOP

## 그림 47. 디버깅 도구

6. 왼쪽 사이드 바에서 디버거 옵션을 볼 수 있습니다. 다음 옵션을 사용할 수 있습니다.
- **Variables:** 프로그램에있는 로컬, 글로벌 및 정적 변수를 값과 함께 나열합니다.
  - **Call Stack:** 현재 실행중인 함수, 호출 함수 (있는 경우) 및 메모리에서 현재 명령어의 위치를 보여줍니다.
  - **Breakpoints:** 설정된 중단점을 표시하고 해당 라인 번호를 강조 표시합니다. 이 섹션에서 중단점을 관리할 수 있습니다. 체크 박스를 토글하여 중단점을 제거하지 않고 일시적으로 비활성화할 수도 있습니다.
  - **Peripherals:** 장치의 메모리 매핑된 주변 장치의 레지스터 상태를 표시합니다 (자세한 내용은 RVfpga Labs 에서 다룹니다).
  - **Registers:** 프로세서의 각 레지스터에있는 현재 값을 나열합니다.
  - **Memory:** 특정 메모리 주소의 내용을 표시합니다.
  - **Disassembly:** 특정 함수에 대한 어셈블리 코드를 표시합니다. C 와 같은 상위 수준 코드의 경우, 이를 통해 명령을 하나씩 디버깅하기 위한 어셈블리를 볼 수 있습니다.

7. 디버거 사이드 바에서 레지스터 옵션  을 확장하고 단계별 실행을 계속합니다. 레지스터 x28 (REGISTERS 섹션에 표시된 대로 t3 이라고도 함)이 *더하기, 빼기 및 논리 AND*의 세 가지 산술 논리 연산의 결과를 저장하는 것을 볼 수 있습니다. 그림 48 을 참조하십시오.

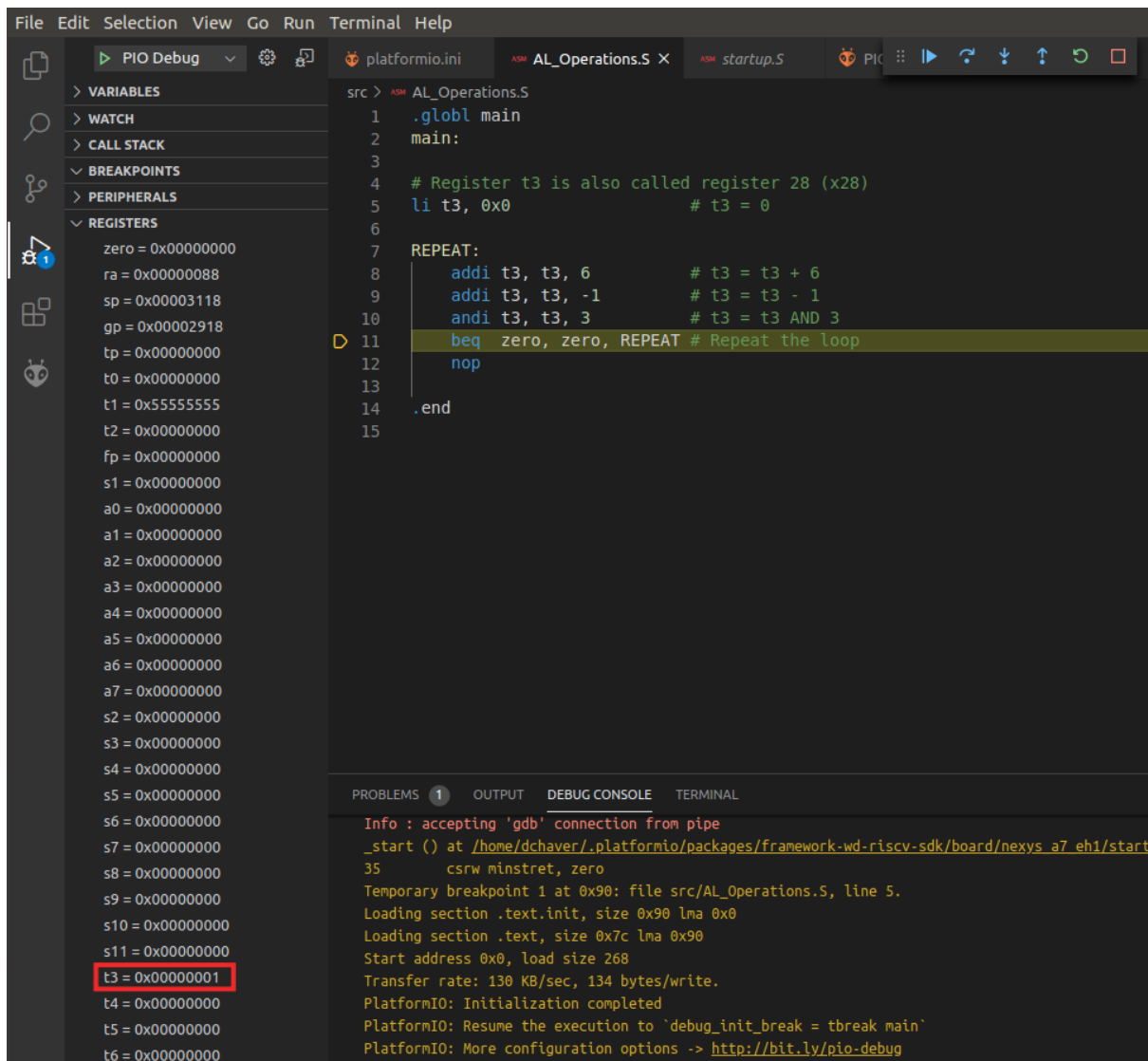


그림 48. 레지스터 내용보기



- main 함수를 호출하기 전에 Western Digital 에서 `~/platformio/packages/framework-wd-riscv-sdk/board/nexys_a7_eh1/startup.S`에서 제공하는 시작 파일이 실행됩니다. 이 파일은 명령어 캐시 설정, 초기화 등록 (예: `sp` 또는 `gp`) 등의 코어를 구성합니다. 디버깅이 시작되면 이 파일이 기본 창에서 열리고 (그림 48 참조) 여기서 검사할 수 있습니다.

**Windows:** `.platformio` 폴더는 사용자 폴더 (`C:\Users\<USER>`)에 있습니다. 숨겨진 파일/폴더를 보려면 시스템을 활성화해야 할 수도 있습니다.

**macOS:** Linux 와 마찬가지로 `.platformio` 폴더는 홈 폴더 (`~/platformio`) 안에 있습니다.

- 또한 동일한 디렉토리 (`~/platformio/packages/framework-wd-riscv-sdk/board`)에 제공되어

모든 프로젝트에서 사용할 링커 스크립트(*link.lds*)를 구성한다는 점을 강조해야 합니다. 이 파일은 메모리에서 어셈블리 섹션 (*텍스트, 데이터, bss...*)의 배치를 결정합니다.

10. 마지막으로 디버깅을 중지하고 , 맨 왼쪽 사이드 바의 상단에 있는 를 클릭하여 탐색기 창으로 돌아갑니다. 상단 메뉴 표시 줄에서 *파일* → *폴더 닫기*를 클릭합니다.

### C. 깜박이는 프로그램

두 번째 예제 프로그램인 *blinky.S* 는 Nexys A7 보드의 가장 오른쪽에있는 LED 를 깜박이게 하는 조립 프로그램 입니다 (그림 49 참조). 프로그램은 각 반전 사이에 지연을두고 가장 오른쪽 LED 에 연결된 값을 반복적으로 반전시킵니다.

```

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

```

그림 49. *blinky.S*

다음 단계에 따라 FPGA 보드에 로드된 RISC-V SoC 인 RVfpgaNexys 에서 이 코드를 실행하고 디버깅합니다.

1. 첫 번째 예제 (*AL\_Operations*)를 실행한 경우 RVfpgaNexys 는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys 를 보드에 다시 프로그래밍해야 하는 경우 *AL\_Operations* 예제 대신 *Blinky* 예제를 사용하여 섹션 A 에 설명된 대로 수행하십시오.

- 상단 표시 줄에서 **파일** → **폴더 열기**를 클릭하고 `[RVfpgaPath]/RVfpga/examples/` 디렉토리로 이동합니다.

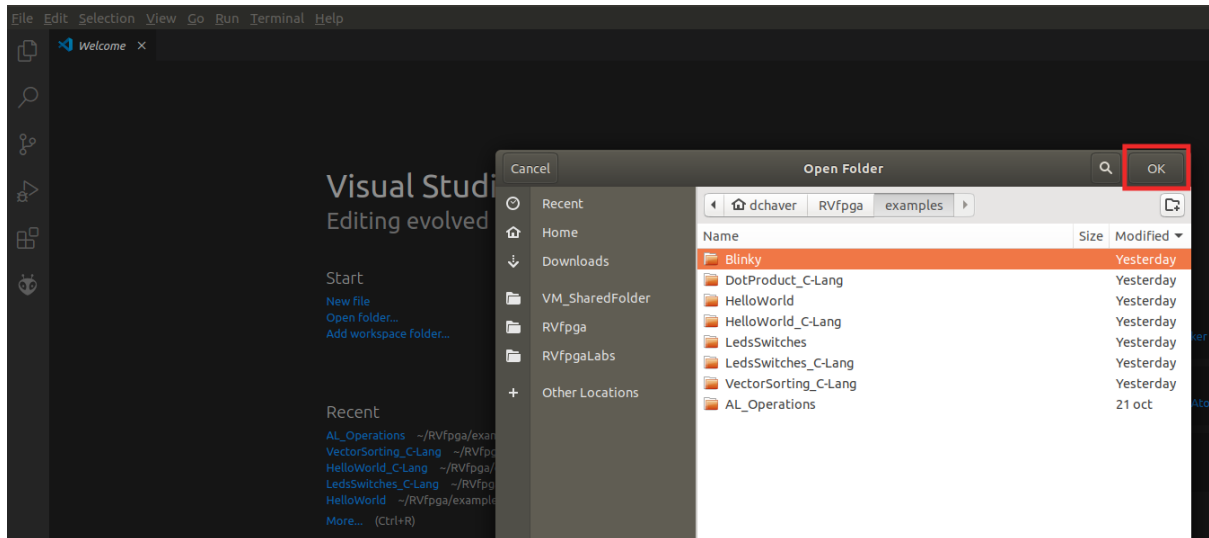


그림 50. 깜박이는 프로그램 폴더

- `Blinky` 디렉토리를 선택하고 확인을 클릭합니다.(그림 50)
- 편집기에서 예제의 어셈블리 코드인 `blinky.S` 파일을 클릭하여 엽니다 (그림 51).

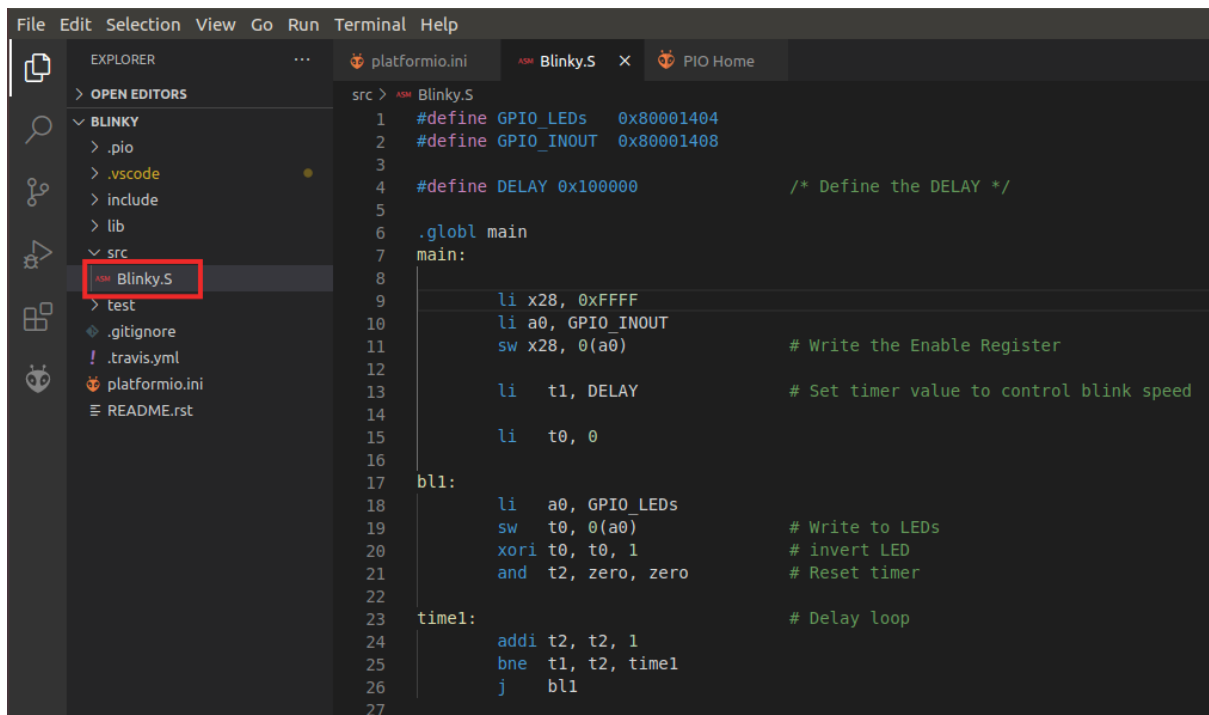

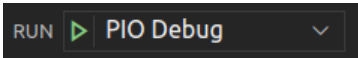




그림 51. PlatformIO 의 `blinky.S`



5. 프로그램을 실행하고 디버그하려면  를 클릭하십시오. 그런 다음 재생 버튼  을 클릭하여 디버깅을 시작합니다. PlatformIO 는 기본 기능의 시작 부분에 임시 중단점을 설정합니다. 따라서 계속 버튼  을 클릭하여 프로그램을 실행하십시오.
6. 보드에서 가장 오른쪽에있는 LED 가 깜박이기 시작하는 것을 볼 수 있습니다.
7. 일시 중지 버튼  을 클릭하여 실행을 일시 중지합니다. 실행은 무한 루프 내부 (아마도 `time1` 지연 루프 내부)에서 중지됩니다.
8. 줄 번호 18 의 왼쪽을 클릭하여 중단점을 설정합니다. 빨간색 점이 나타나고 중단점이 BREAKPOINTS 탭에 추가됩니다 (그림 52 참조).

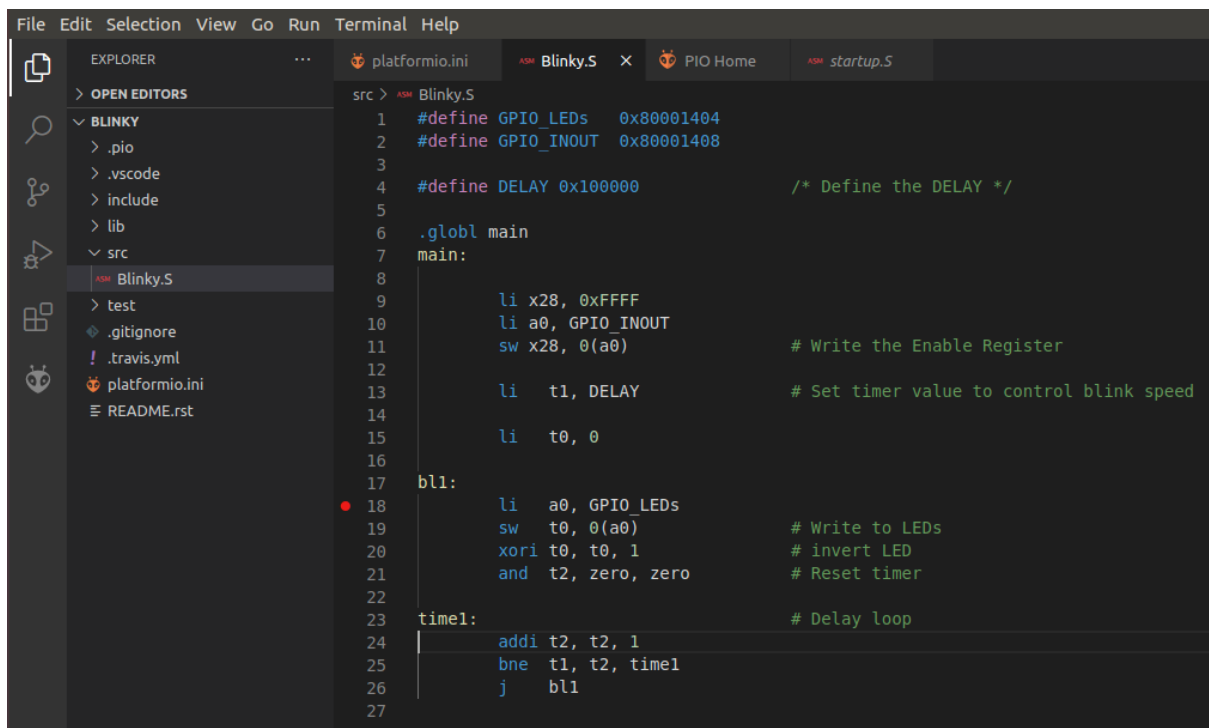





그림 52. blinky.S 에서 중단점 설정

9. 그런 다음 계속 버튼  을 클릭하여 실행을 계속합니다. 실행은 계속되고 가장 오른쪽 LED 에 1 (또는 0)을 쓰는 저장 바이트 (sb) 명령 이후에 중지됩니다.
10. 실행을 여러 번 계속하십시오. 맨 오른쪽 LED 로 구동되는 값이 매번 변경되는 것을 볼 수 있습니다.

11. 를 클릭하여 디버깅을 중지하고 탐색기 창으로 돌아갑니다.

파일 → 폴더 열기 를 선택하여 프로그램을 닫습니다.

## D. LedsSwitches 프로그램

세 번째 어셈블리 예제는 보드에서 사용 가능한 LED 및 스위치와 통신합니다 (그림 53 참조).

```

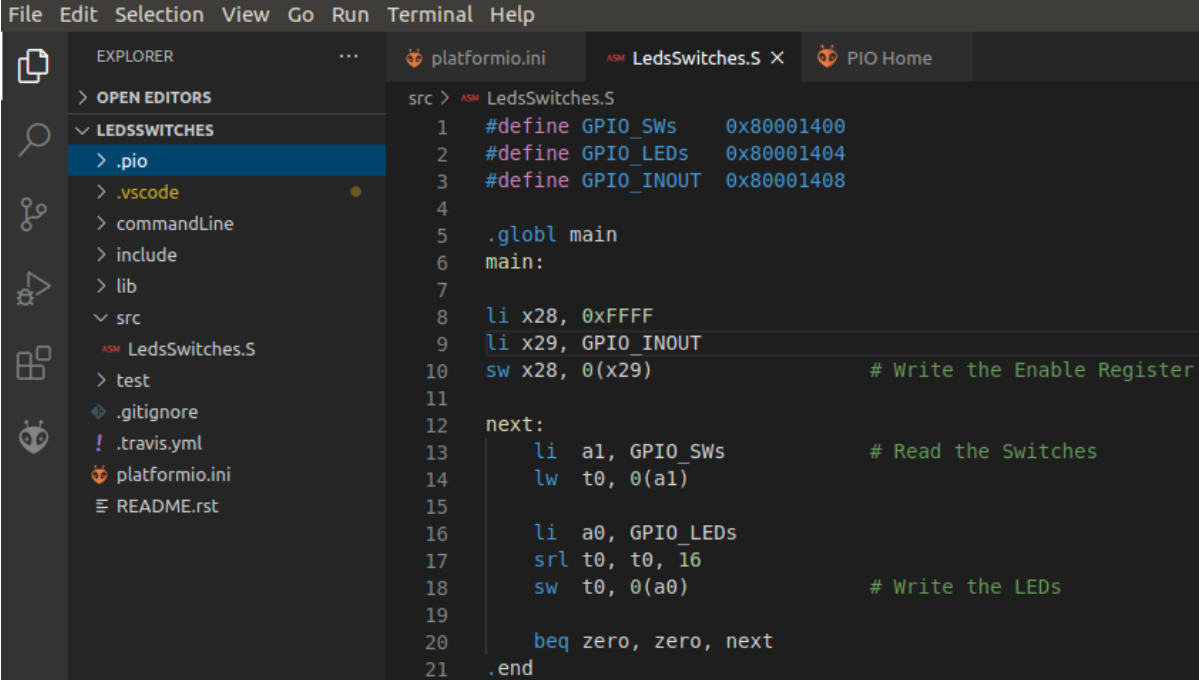
1  #define GPIO_SWs      0x80001400
2  #define GPIO_LEDs     0x80001404
3  #define GPIO_INOUT    0x80001408
4
5  .globl main
6  main:
7
8  li x28, 0xFFFF
9  li x29, GPIO_INOUT
10 sw x28, 0(x29)           # Write the Enable Register
11
12 next:
13  li a1, GPIO_SWs         # Read the Switches
14  lw t0,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

```

그림 53. LedsSwitches.S

FPGA 보드에서 이 코드를 실행하고 디버깅하려면 다음 단계를 따르십시오.


- 이전 예제를 실행한 경우 RVfpgaNexys 는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys 를 보드에 다시 프로그래밍해야 하는 경우 AL\_Operations 예제 대신 LedsSwitches 예제를 사용하여 섹션 A 에 설명된 대로수행하십시오.
- 상단 표시 줄에서 파일 → 폴더 열기를 클릭하고 *[RVfpgaPath]/RVfpga/examples/* 디렉토리를 찾습니다. *LedsSwitches* 디렉토리를 선택하고 확인을 클릭하십시오.
- LedsSwitches.S* 프로그램에는 스위치를 읽고 LED 에 상태를 표시하는 무한 루프가 있습니다. (그림 54)



```

File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
LEDSSWITCHES
> .pio
> .vscode
> commandLine
> include
> lib
src
  ASM LedsSwitches.S
  test
  .gitignore
  .travis.yml
  platformio.ini
  README.rst
src > ASM LedsSwitches.S
1  #define GPIO_SWs    0x80001400
2  #define GPIO_LEDs   0x80001404
3  #define GPIO_INOUT  0x80001408
4
5  .globl main
6  main:
7
8  li x28, 0xFFFF
9  li x29, GPIO_INOUT
10 sw x28, 0(x29)           # Write the Enable Register
11
12 next:
13   li a1, GPIO_SWs        # Read the Switches
14   lw t0, 0(a1)
15
16   li a0, GPIO_LEDs
17   srl t0, t0, 16
18   sw t0, 0(a0)           # Write the LEDs
19
20   beq zero, zero, next
21 .end
  
```

그림 54. PlatformIO 의 LedsSwitches.S

4. 이전 프로그램에서 설명한 대로 디버거를 시작한 후 프로그램이 실행되기 시작합니다.  
PlatformIO 는 기본 기능의 시작 부분에 임시 중단점을 설정합니다. 따라서 계속 버튼  을 클릭하여 프로그램을 실행하십시오.
5. Nexys A7 보드 하단에 있는 스위치를 전환합니다. LED 가 스위치의 새 값을 표시하는 것을 보드에서 즉시 볼 수 있습니다. 위에서 설명한 대로 실행을 일시 중지하고 단계별로 실행하고 레지스터를 검사할 수 있습니다. 완료되면 *파일* → *폴더 닫기*를 클릭하여 프로젝트를 닫습니다.
6. 때로는 메모리에 저장된 값을 검사하는 것이 매우 유용할 수 있습니다. 이를 위해 PlatformIO 는 메모리 디스플레이를 제공합니다.
  - a. 다음 루프가 시작될 때까지 실행 및 단계를 일시 중지합니다. 창 왼쪽에 있는 메모리 디스플레이 (그림 55 참조)를 확장하고 주소 *입력...*을 클릭합니다.

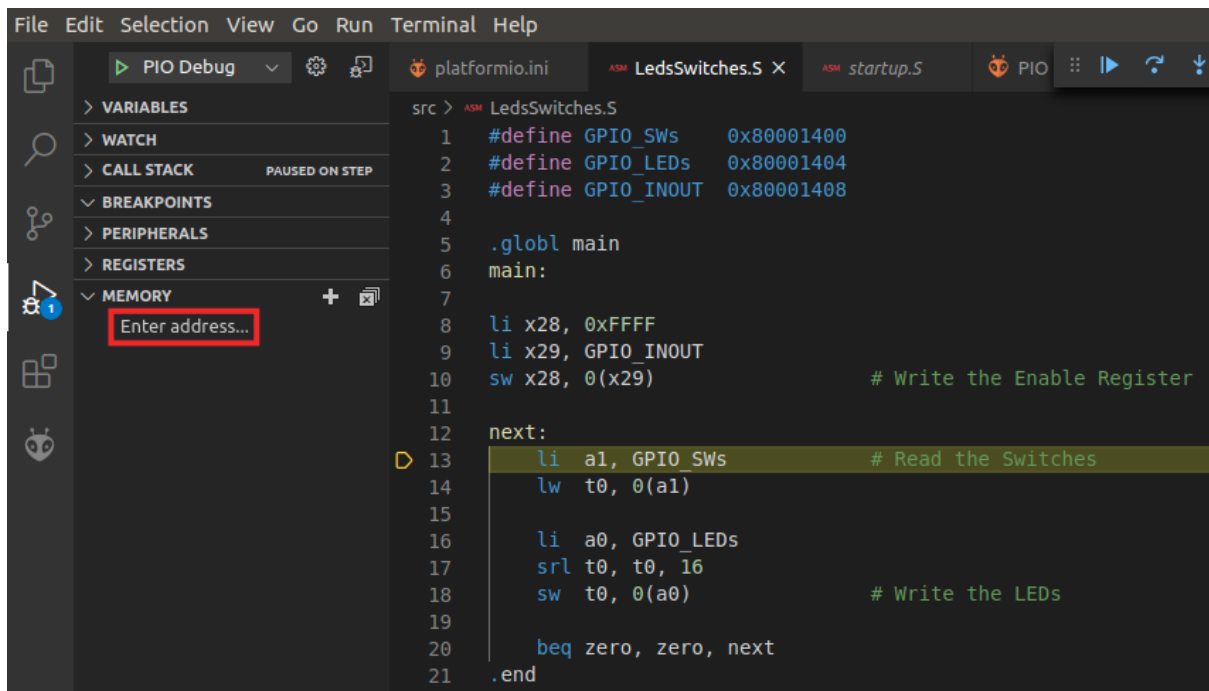


그림 55. 메모리 디스플레이

- b. 초기 메모리 주소가 요청됩니다 (그림 56 참조). 스위치가 매핑된 초기 주소 (이 경우 0x80001400)를 삽입합니다.

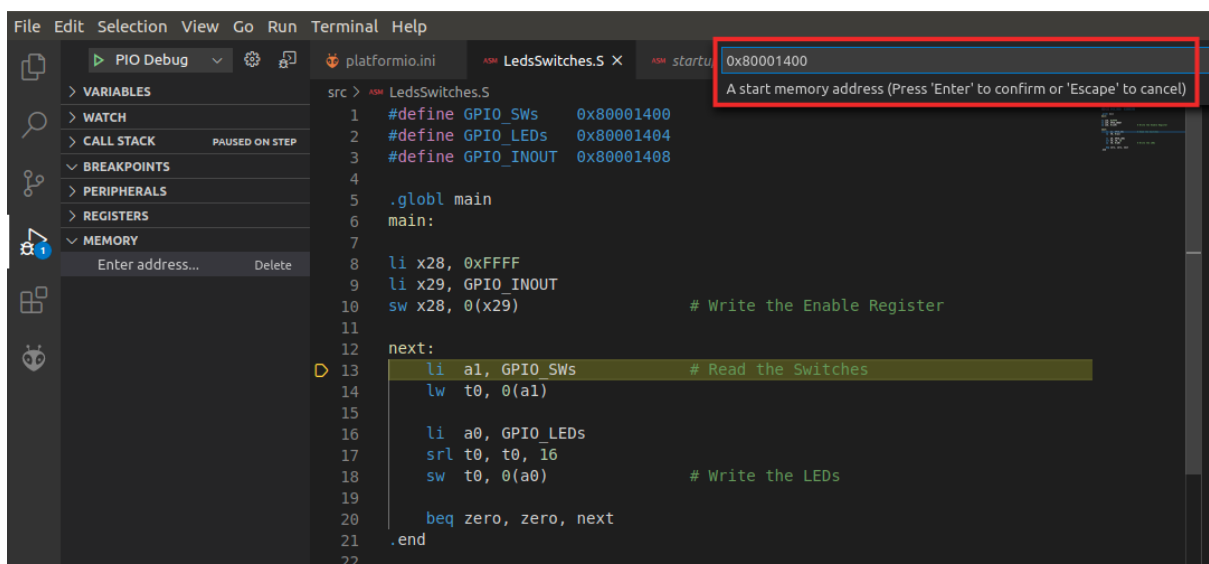


그림 56. 표시할 초기 메모리 주소

- c. 그런 다음 검사할 바이트 수를 요청합니다 (그림 57 참조). 0xc 값을 삽입합니다 (3개의 4 바이트 I/O 레지스터를 검사하려고 함으로 12 바이트가 필요함).

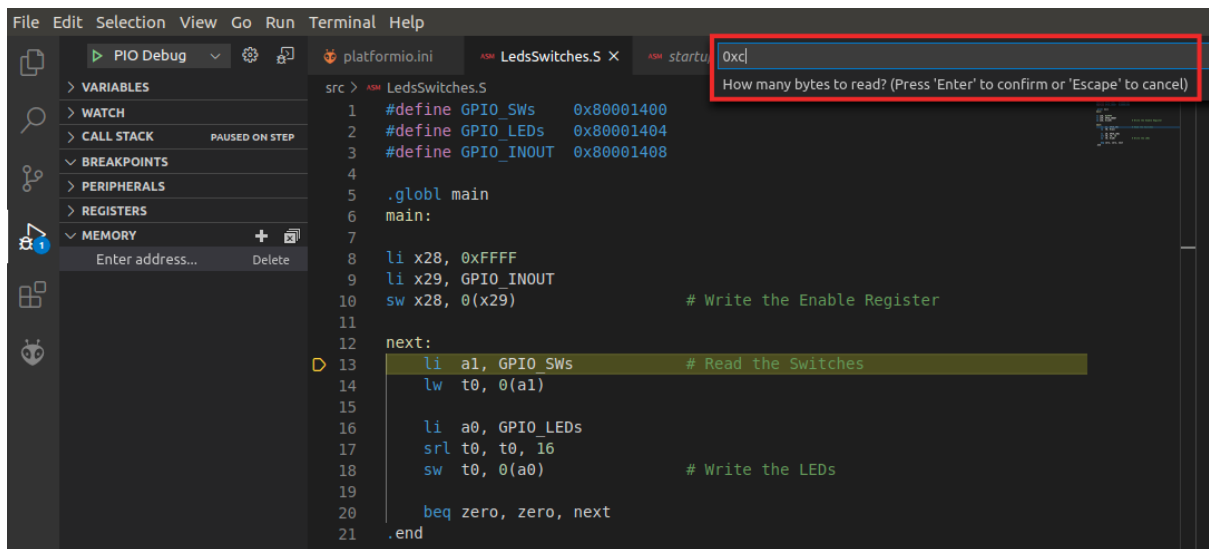


그림 57. 표시할 바이트 수

- d. 메모리 디스플레이가 오른쪽에 열리고 요청한 12 바이트가 표시됩니다 (그림 58 참조). 16 개의 스위치에 있는 값은 0x123C 입니다 (주소 0x80001402 및 0x80001403 의 바이트 참조). RISC-V 아키텍처가 little endian 이라는 점을 고려하면 그림에 표시된 값은 그와 일관됩니다. 16 개의 LED (주소 0x80001404 및 0x80001405 에 저장 됨)는 동일한 값을 표시합니다.

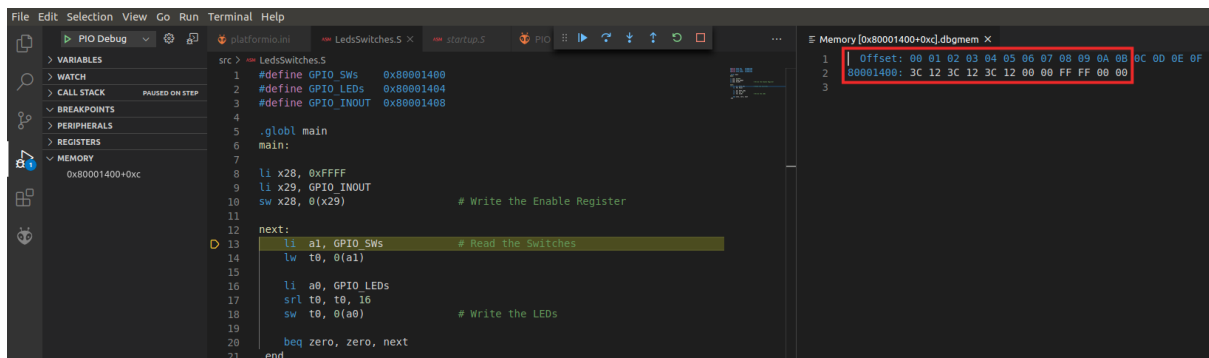
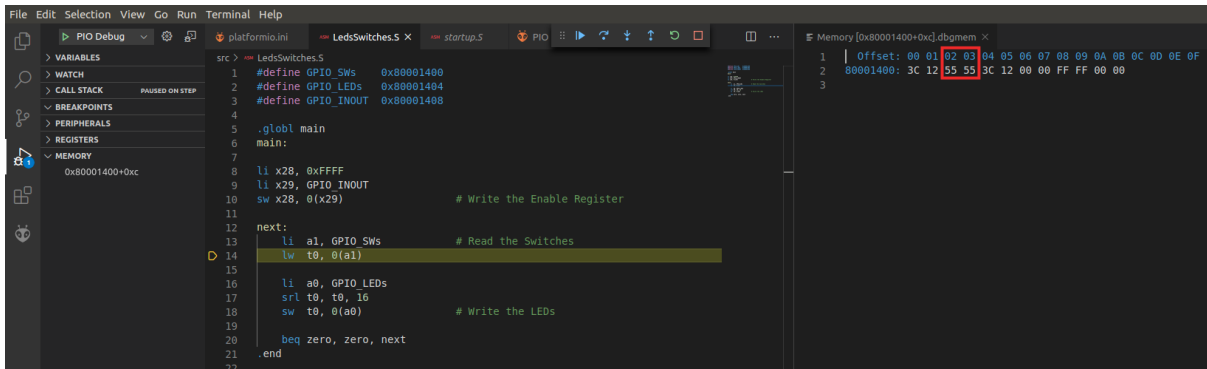


그림 58. 메모리 주소 0x80001400-0x8000140B

- e. 예를 들어, 보드의 스위치 값을 0x5555 로 변경하고 루프를 단계별로 한 번 더 반복합니다. 메모리의 스위치 값은 첫 번째 명령 (그림 59, 맨 위)을 실행한 직후에 변경되어야 하며 LED 값은 sw 명령을 실행한 후에 그에 따라 변경되어야 합니다 (그림 59, 맨 아래).



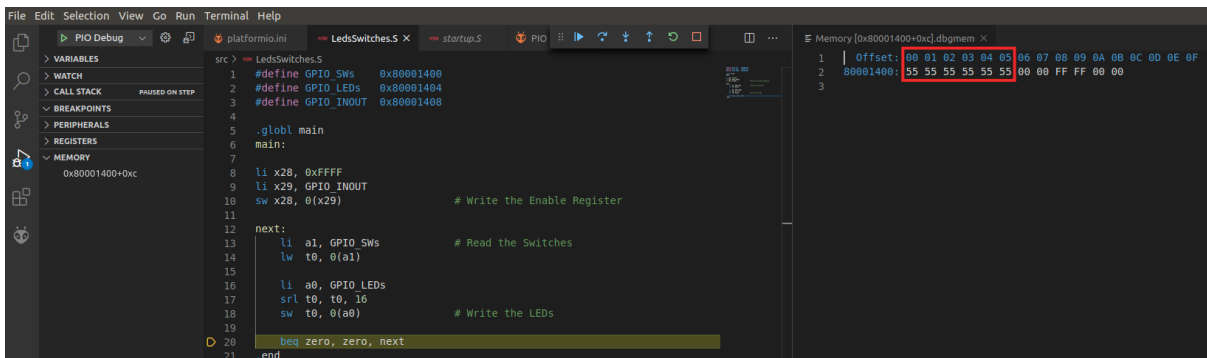
```

src > LedsSwitches.S
1 #define GPIO_SWS 0x80001400
2 #define GPIO_LEDS 0x80001404
3 #define GPIO_INOUT 0x80001408
4
5 .globl main
6 main:
7
8 li x28, 0xFFFF
9 li x29, GPIO_INOUT
10 sw x28, 0(x29)          # Write the Enable Register
11
12 next:
13 li a1, GPIO_SWS        # Read the Switches
14 lw t0, 0(a1)
15
16 li a0, GPIO_LEDS
17 srl t0, t0, 16
18 sw t0, 0(a0)          # Write the LEDs
19
20 beq zero, zero, next
21 .end

```

Memory [0x80001400+0xc].dbgmem X

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



```

src > LedsSwitches.S
1 #define GPIO_SWS 0x80001400
2 #define GPIO_LEDS 0x80001404
3 #define GPIO_INOUT 0x80001408
4
5 .globl main
6 main:
7
8 li x28, 0xFFFF
9 li x29, GPIO_INOUT
10 sw x28, 0(x29)          # Write the Enable Register
11
12 next:
13 li a1, GPIO_SWS        # Read the Switches
14 lw t0, 0(a1)
15
16 li a0, GPIO_LEDS
17 srl t0, t0, 16
18 sw t0, 0(a0)          # Write the LEDs
19
20 beq zero, zero, next
21 .end

```

Memory [0x80001400+0xc].dbgmem X

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

그림 59. 스위치 및 LED 변경

- f. 프로그램의 기계 명령어를 저장하는 RAM 주소와 같은 다른 메모리 위치도 볼 수 있습니다. 0x0 (RAM 메모리에 할당된 초기 주소)에서 시작하고 0x100 바이트를 차지하는 다른 메모리 범위를 엽니다 (그림 60). 시작 프로그램 (Startup.S) 직후 주소 범위 0x90-0xC4 에 저장된 LedsSwitches 프로그램의 지침을 볼 수 있습니다.

Memory [0x0+0x100].dbgmem X

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

그림 60. 메모리 주소 0x0 ~ 0x100

- g. `[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/firmware.di`  
s에서 사용할 수 있는 프로그램의 디스어셈블리를 열어 프로그램의 지침에 대한  
기계어 코드를 볼 수 있습니다 (그림 61 참조). 두 그림을 비교하고 프로그램의  
지침을 확인하십시오.

```

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

```

그림 61. LedsSwitches 프로그램의 디스어셈블리 버전

## E. Led 스위치 C-Language 프로그램

프로그램 LedsSwitches\_C-Lang.c (그림 62)는 이전에 표시된 LedsSwitches.s 프로그램 (그림 53)과 동일하지만 어셈블리 대신 C로 작성됩니다.

```

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 }

```

**그림 62. LedsSwitches\_C-Lang.c**

FPGA 보드에서 이 프로그램을 실행하고 디버깅하려면 다음 단계를 따르십시오:

1. 이전 예제를 실행한 경우 RVfpgaNexys 는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys 를 보드에 다시 프로그래밍해야하는 경우 AL\_Operations 예제 대신 LedsSwitches\_C-Lang 예제를 사용하여 섹션 A 에 설명된 대로 수행하십시오.
2. 상단 메뉴 표시 줄에서 파일 → 폴더 열기를 클릭하고 *[RVfpgaPath]/RVfpga/examples/* 디렉토리로 이동합니다. LedsSwitches\_C-Lang 디렉토리를 선택하고 확인을 클릭하십시오.
3. 디버거를 호출하기 전에 C 코드의 15 행에 중단점을 설정합니다.
4. 그런 다음 디버깅을 시작합니다. 프로그램이 실행을 시작하고 중단점에서 중지합니다 (그림 63).



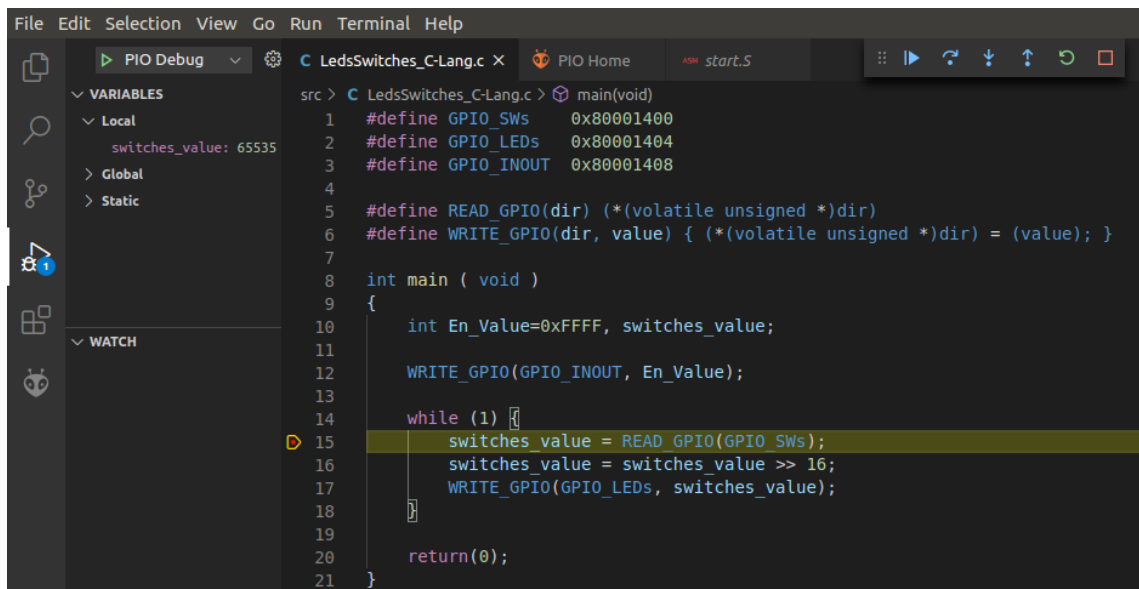



그림 63. 중단점에서 중지된 실행

5. 프로그램을 여러 번 계속 실행하되  클릭할 때마다 스위치를 변경합니다. LED 는 스위치의 값을 표시해야 합니다.
6. 위와 같이 C 에서 프로그램 실행을 보거나 그림 64 에서 강조 표시된 어셈블리로 전환을 클릭하여 컴파일러에 의해 생성된 어셈블리 프로그램의 실행을 볼 수 있습니다.

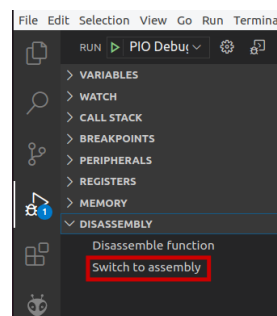


그림 64. 어셈블리로 전환

7. 어셈블리의 프로그램 (그림 65)은 먼저 로드 명령 `lw a5, 1024(a4)` 을 사용하여 스위치의 값을 읽은 다음 저장 명령 (`sw a5, 1028(a4)`)을 사용하여 LED 에 씁니다. 단계별로 실행 및 스위치를 변경하고 LED 가 새 스위치 값을 반영하도록 변경되는지 확인합니다.

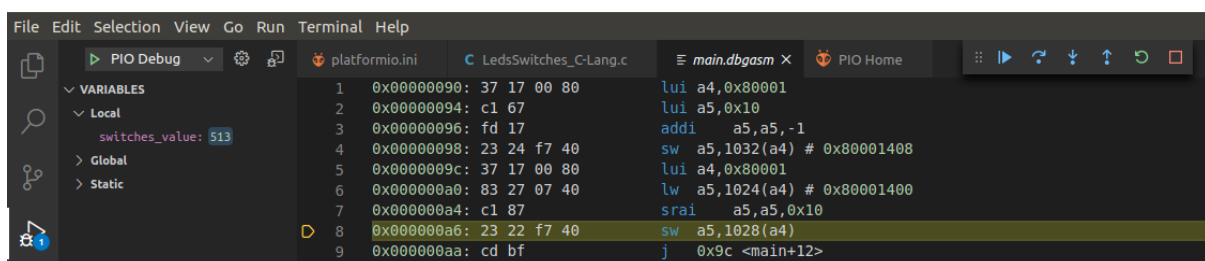


그림 65. 어셈블리 프로그램

## F. HelloWorld C-Language 프로그램

두 번째 C 예제는 직렬 포트를 통해 shell 에 짧은 메시지를 인쇄합니다. 이 메시지를 보려면 *gtkterm*, *minicom* 등과 같은 터미널 에뮬레이터를 사용할 수 있습니다. 그러나 PlatformIO 는 자체 직렬 모니터를 제공하므로 여기에서는 이 모니터를 사용하는 방법을 보여줍니다.

PlatformIO 직렬 모니터를 구성하려면 일부 매개 변수를 구성해야 합니다. 특히 직렬 데이터 전송을 위한 데이터 속도 (초당 비트 수 또는 전송 속도)를 설정해야 합니다. 이는 *platformio.ini* 파일의 *monitor\_speed* 매개 변수를 사용하여 수행할 수 있습니다 (이 파일은 PlatformIO 프로젝트의 일부입니다). 그림 66 를 참조하십시오.

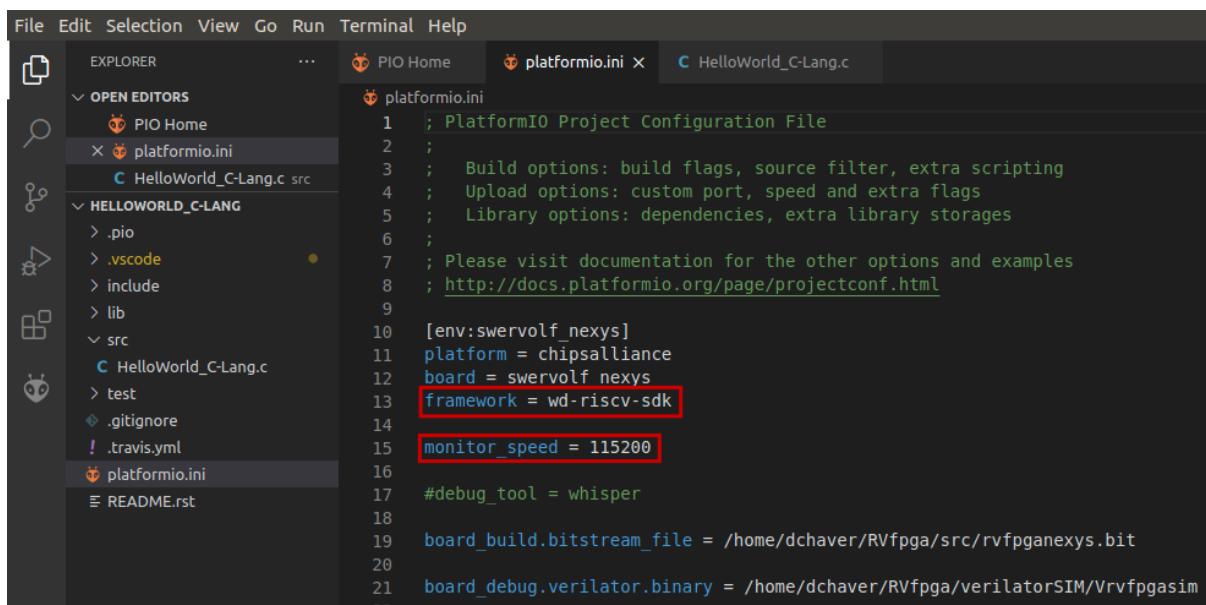


그림 66. 직렬 모니터 구성

또한 터미널에 다음 명령을 입력하여 dialout, tty 및 uucp 그룹에 자신을 추가해야 합니다.

```

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

```

세 명령 후에 컴퓨터를 다시 시작하면 그룹의 변경 사항이 적용됩니다.

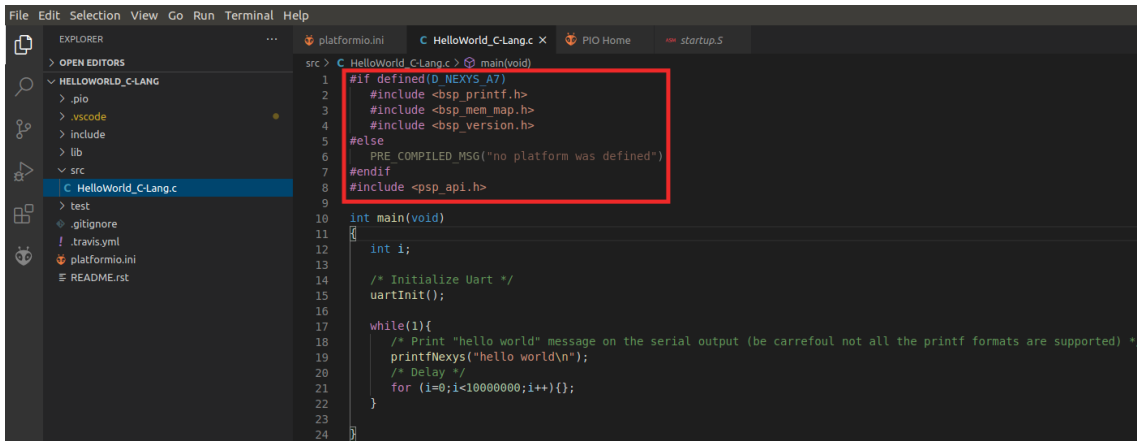
**Windows / macOS:** Windows 및 macOS 사용자는 위 단계를 완료할 필요가 없습니다.

또한 이 프로그램은 펌웨어 패키지 (<https://github.com/westerndigitalcorporation/riscv-fw-infrastructure>) 내에서 WD 가 제공하는 프로세서 지원 패키지 (PSP) 및 보드 지원 패키지 (BSP)를

사용합니다. 이러한 라이브러리는 그림 66 에 표시된 대로 platformio.ini 의 특정 명령 (framework = wd-riscv-sdk), 을 사용하고 그림 67 에 표시된 대로 C 프로그램 시작 부분에 적절한 파일을 포함하여 프로젝트에 포함됩니다. 다음 경로에서 시스템의 전체 라이브러리를 찾을 수 있습니다:

- PSP: ~/.platformio/packages/framework-wd-riscv-sdk/psp/
- BSP: ~/.platformio/packages/framework-wd-riscv-sdk/board/nexys\_a7\_eh1/bsp/

이러한 라이브러리는 인터럽트 사용, 문자열 인쇄, 개별 레지스터 읽기/쓰기와 같은 많은 작업을 수행할 수 있는 많은 함수와 매크로를 제공합니다. 이 예에서는 직렬 모니터에 메시지를 인쇄하기 위해 printfNexys 함수를 사용합니다. 다음 예제와 실습에서는 다른 목적으로 다른 함수와 매크로를 사용하는 방법을 보여줍니다.



```

1  #if defined(D_NEXYS_A7)
2  #include <bsp_printf.h>
3  #include <bsp_mem_map.h>
4  #include <bsp_version.h>
5  #else
6  PRE_COMPILED_MSG("no platform was defined")
7  #endif
8  #include <psp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be carrefoul not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<100000000;i++){
22             ;
23         }
24     }
25 }

```

그림 67. *HelloWorld\_C-Lang.c*에 .h 파일 포함

FPGA 보드에서 이 코드를 실행하고 디버깅하려면 다음 단계를 따르십시오.

1. 이전 예제를 실행한 경우 RVfpgaNexys 는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys 를 보드에 다시 프로그래밍해야 하는 경우 AL\_Operations 예제 대신 HelloWorld\_C-Lang 예제를 사용하여 섹션 A 에 설명된 대로 수행하십시오.
2. VSCode 를 엽니다. PlatformIO 는 VSCode 를 열 때 VSCode 내에서 자동으로 열립니다. 상단 표시 줄에서 파일 → 폴더 열기를 클릭하고 [RVfpgaPath]/RVfpga/examples/ 디렉토리를 찾습니다. HelloWorld\_C-Lang 폴더를 선택하고 확인을 클릭하십시오.
3. HelloWorld\_C-Lang.C 프로그램 (그림 68)은 UART (function **uartInit**)를 초기화한 다음 **printfNexys** 함수를 사용하여 직렬 포트를 통해 문자열을 전송합니다 (이러한 함수의 구현은 파일에서 찾을 수 있습니다. ~/.platformio/packages/framework-wd-riscv-

*sdk/board/nexys\_a7\_eh1/bsp/bsp\_printf.c*). 그런 다음 루프의 시작으로 돌아가기 전에 약간의 시간이 지연됩니다.

```

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

```

그림 68. HelloWorld\_C-Lang.C 주요 기능

- PlatformIO 에서 디버거를 시작합니다. 프로그램이 실행되기 시작하면 VS Code 하단에 있는 *plug* 버튼을 클릭하여 직렬 모니터를 엽니다 (그림 69).

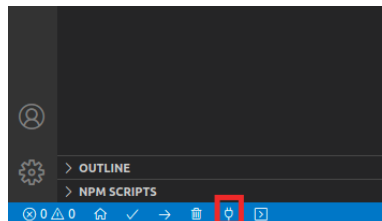
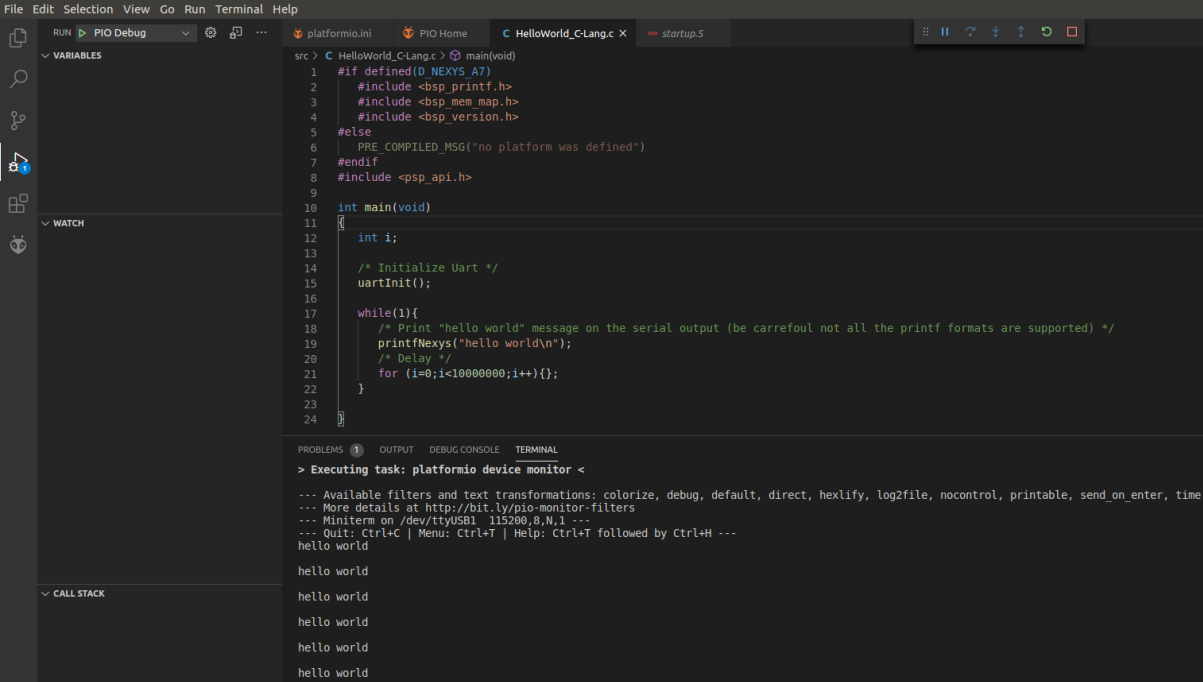


그림 69. 직렬 터미널 열기

- 직렬 모니터는 그림 70 와 같이 "HELLO WORLD !!!"메시지를 반복해서 인쇄합니다.



```

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  #error "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 careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0;i<100000000;i++){};
22     }
23 }
24
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor <
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at http://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+F | Help: Ctrl+T followed by Ctrl+H ---
hello world
hello world
hello world
hello world
hello world
hello world

```

그림 70. 프로그램 실행

## G. VectorSorting\_C-Lang 프로그램

마지막으로 벡터 A의 요소를 가장 큰 요소에서 가장 작은 요소로 정렬하고 정렬된 값을 두 번째 벡터 B에 배치하는 또 다른 C 프로그램을 보여줍니다. 벡터 A 값은 0으로 대체됩니다. 그림 70은 프로그램을 보여줍니다.

```

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 }

```

그림 71. VectorSorting\_C-Lang.c

FPGA 보드에서 이 프로그램을 실행하고 디버깅하려면 다음 단계를 따르십시오:

1. 이전 예제를 실행한 경우 RVfpgaNexys 는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys 를 보드에 다시 프로그래밍해야 하는 경우 AL\_Operations 예제 대신 VectorSorting\_C-Lang 예제를 사용하여 섹션 A 에 설명된 대로 수행하십시오..
2. 상단 메뉴 표시 줄에서 *파일* → *폴더 열기*를 클릭하고 *[RVfpgaPath]/RVfpga/examples/* 디렉토리로 이동합니다. *VectorSorting\_C-Lang* 폴더를 선택하고 확인을 클릭합니다.
3. 10 행에 중단점을 놓고 디버깅을 시작합니다. 실행은 `for` 루프의 시작에서 중지됩니다 (그림 72). 디버거 사이드 바에서 VARIABLES 섹션을 확장하고 A 및 B 배열의 값을 분석합니다 (그림 72 에서 빨간색으로 강조 표시됨).

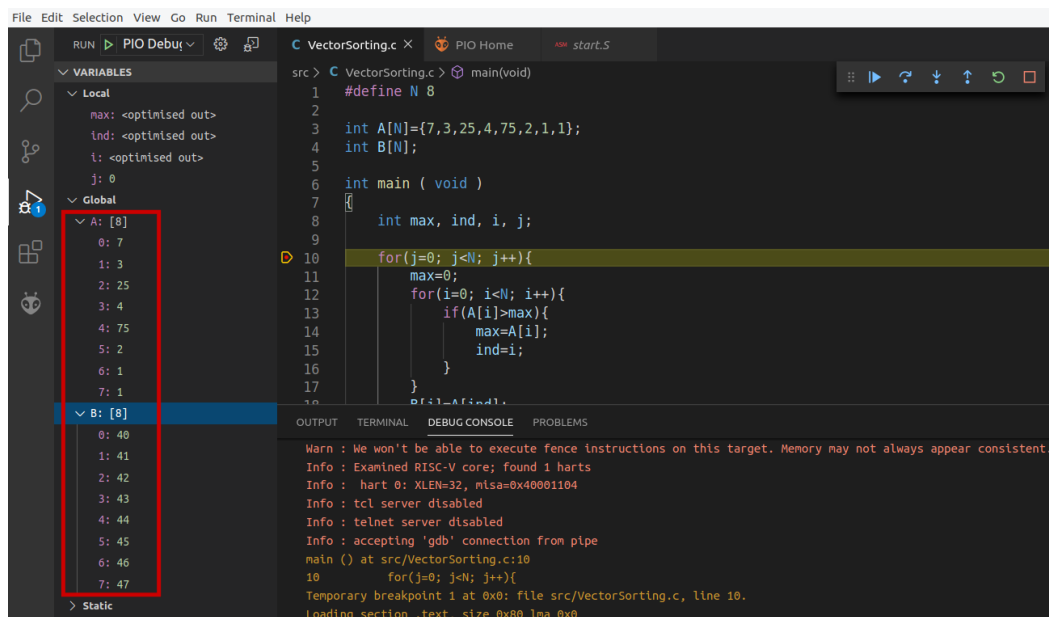



그림 72. 프로그램 시작시 실행 중지됨

4. 이제 18 행에 또 다른 중단점을 놓고를 클릭  하여 실행을 계속합니다 (그림 73 참조). 메모리 디스플레이를 열고 (그림 55 의 LedsSwitches 프로그램에 대해 설명) 벡터 A 가 이 프로그램의 메모리에 저장되는 주소인 0x2148 (그림 73 참조)에서 시작하는 0x50 바이트를 표시합니다. 벡터 A (0x2148-0x2167 범위) 및 B (0x2178-0x2197 범위)의 초기 값을 볼 수 있습니다.

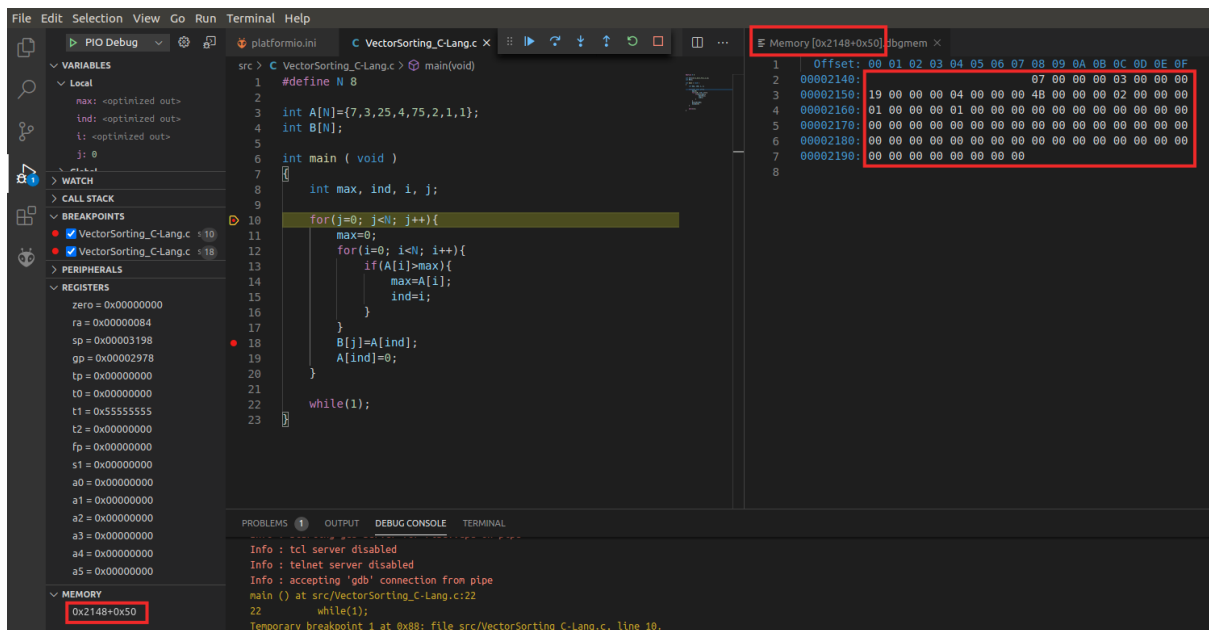


그림 73. 어레이 A 및 B 의 메모리 디스플레이 – 초기 상태.

그림 64 에 설명된 대로 어셈블리로 전환하고 이러한 벡터에 액세스하는 명령을 분석하여 벡터 A 와 B 가 메모리에 저장된 주소를 쉽게 찾을 수 있습니다 (그림 74). 그림에서 볼 수 있듯이 대부분의 경우 주석이 이 정보를 제공합니다. 그러나 이러한 명령으로 이동하여 레지스터에 저장된 값을 볼 수도 있습니다.

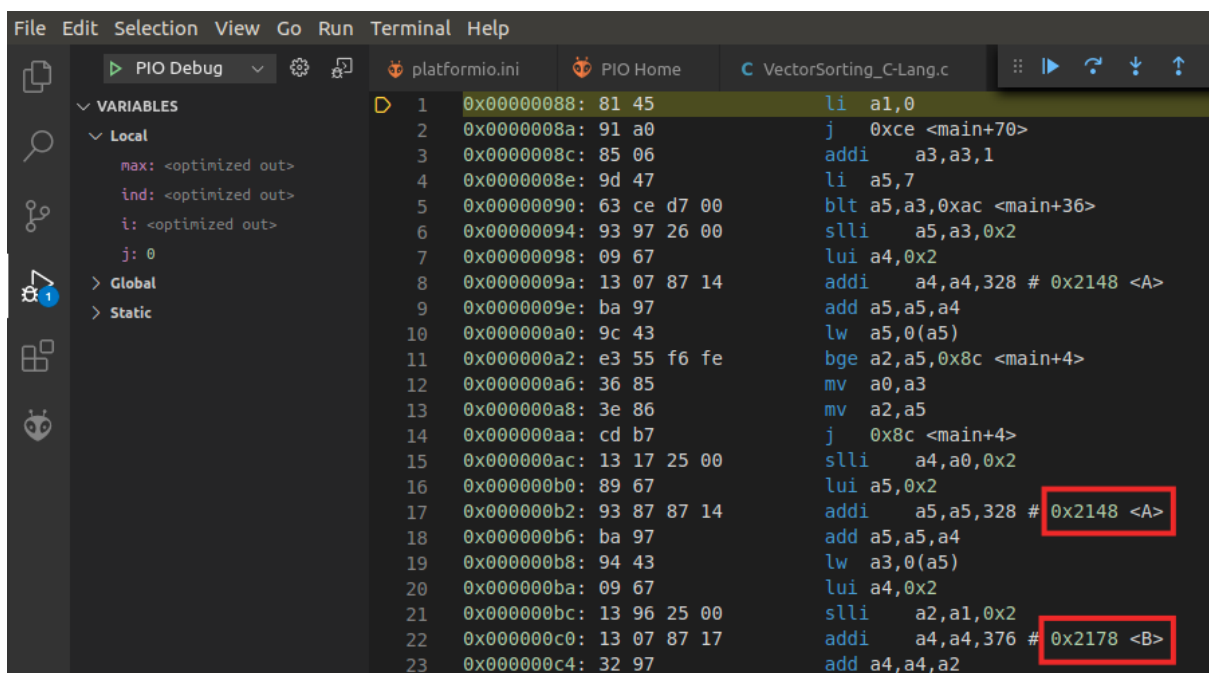



그림 74. A 와 B 가 메모리에 저장되는 주소.

Step Over 버튼 (  ) 을 두 번 클릭하면 메모리에 저장된 B 의 첫 번째 구성 요소와 0 으로 설정된 A 의 해당 값이 표시됩니다 (그림 75 참조).

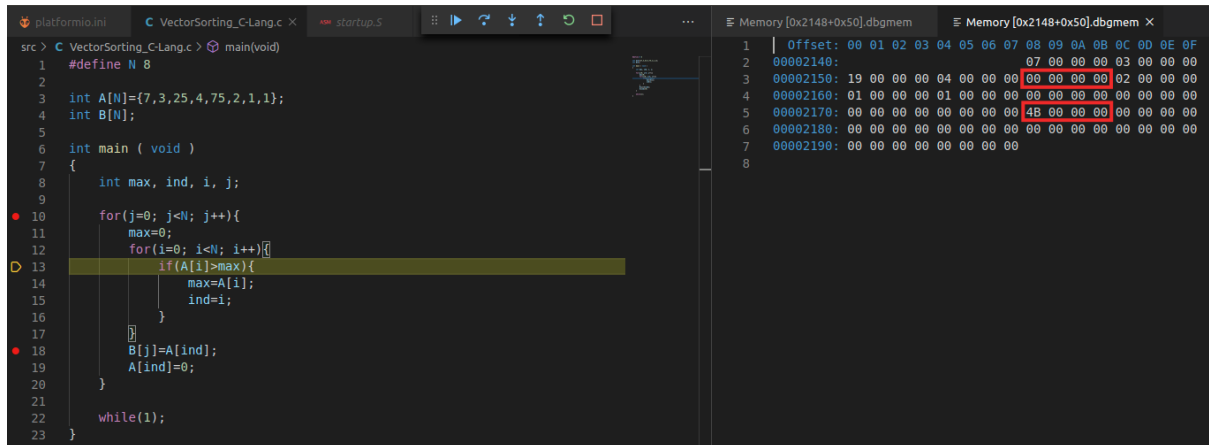


그림 75. 어레이 A 및 B의 메모리 디스플레이 – B의 첫 번째 구성 요소를 저장하고 A에 해당 구성 요소를 재설정합니다.

- 모든 중단점을 제거하고 실행을 계속한 다음 몇 초 후에 일시 중지합니다. 이 시점에서 프로그램 실행이 완료됩니다. 다시 A 및 B 배열에 저장된 값을 분석하십시오. 그림 76에서 볼 수 있듯이 벡터 B는 가장 큰 것에서 가장 작은 것까지 정렬된 원래 벡터 A의 값을 보유하고 벡터 A는 모두 0을 보유합니다 (왼쪽의 변수 목록과 오른쪽의 메모리 콘솔에서 모두 볼 수 있음).

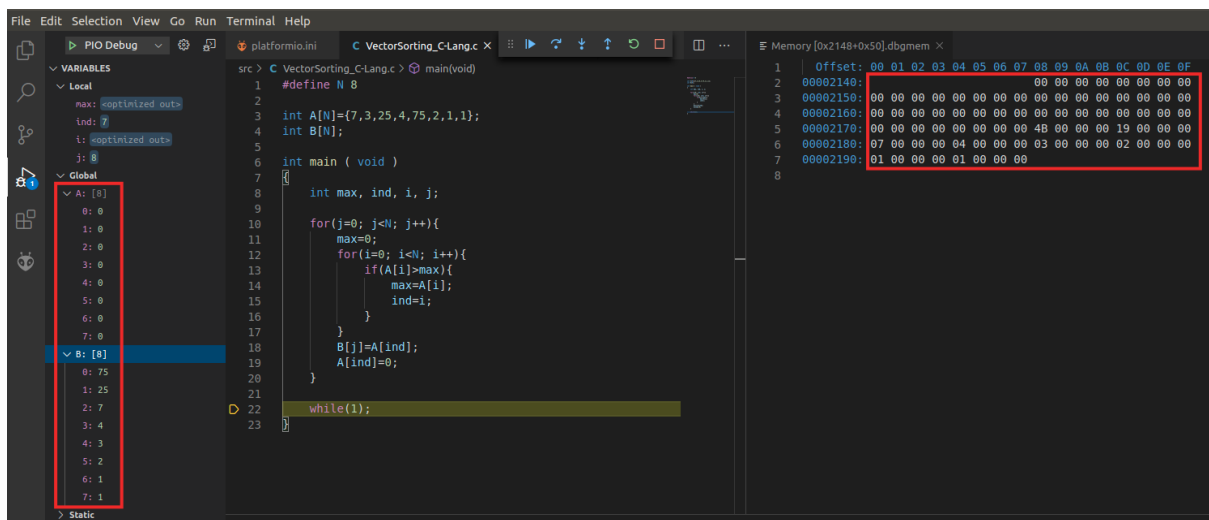


그림 76. 프로그램 끝에서 실행이 중지됨

## H. DotProduct\_C-Lang Program

마지막 예제 프로그램인 DotProduct\_C-Lang.c는 (그림 77) 두 벡터의 내적을 계산합니다. 이 프로그램에는 main 및 dotproduct (내적) 두 가지 기능이 있습니다. 첫 번째 함수는 벡터 크기와 두



벡터의 초기 주소라는 세 개의 입력 인수를 사용하여 두 번째 함수를 호출합니다. 그런 다음 내적 함수는 두 벡터의 내적을 계산하고 결과를 반환합니다.

```

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 }

```

**그림 77 DotProduct\_C-Lang.c**

이 예에서 우리는 실수로 연산합니다 (변수 *x*, *y* 및 *dot*의 데이터 유형은 *double*입니다). 그러나 SweRV EH1 프로세서는 부동 소수점 지원을 포함하지 않습니다. 따라서 예제에서는 *gcc* (<https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html>)에서 제공하는 소프트웨어 부동 소수점 라이브러리를 통해 부동 소수점 에뮬레이션을 사용합니다. 이 라이브러리는 부동 소수점 명령어 생성을 비활성화하기 위해 *-msoft-float*가 포함될 때마다 사용됩니다.

FPGA 보드에서이 코드를 실행하고 디버깅하려면 다음 단계를 따르십시오.

1. 이전 예제를 실행한 경우 RVfpgaNexys는 이미 FPGA 보드에 프로그래밍되어 있으므로 다시 프로그래밍할 필요가 없습니다. 그러나 RVfpgaNexys를 보드에 다시 프로그래밍해야 하는 경우 AL\_Operations 예제 대신 DotProduct\_C-Lang 예제를 사용하여 섹션 A에 설명된 대로 수행하십시오.
2. 상단 메뉴 표시 줄에서 *파일* → *폴더 열기*를 클릭하고 *[RVfpgaPath]/RVfpga/examples/* 디렉토리로 이동합니다. DotProduct\_C-Lang 디렉토리를 선택하고 확인을 클릭합니다.
3. 디버거를 호출하기 전에 10 번 줄에 중단점을 설정하고 19 번 줄에 다른 중단점을 설정합니다 (그림 78 참조).
4. 그런 다음 디버깅을 시작합니다. 프로그램이 실행되기 시작합니다. 첫 번째 중단점에서

중지합니다 (그림 78 참조).

- 디버거 사이드 바에서 변수 섹션을 확장합니다 (그림 78 참조). 두 벡터에는 main 에 할당된 초기 값이 포함됩니다. 도트 변수는 0 으로 초기화됩니다.

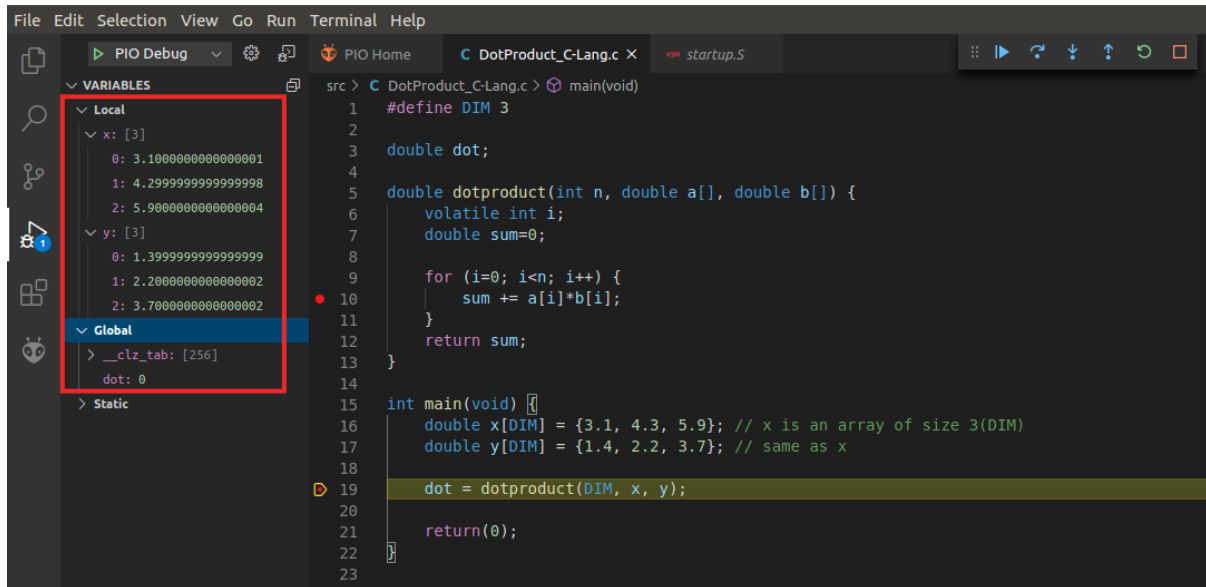



그림 78. DotProduct\_C-Lang 프로그램: 첫 번째 중단점의 변수 값

- 프로그램이 계속 실행되도록 합니다 . 프로그램은 두 번째 중단점 (10 행)에서 중지됩니다.
- 어셈블리로 전환합니다 (그림 64 에서와 같이). 부동 소수점 에뮬레이션 루틴을 보고 단계별로 자세히 분석할 수 있습니다 (그림 79 참조).

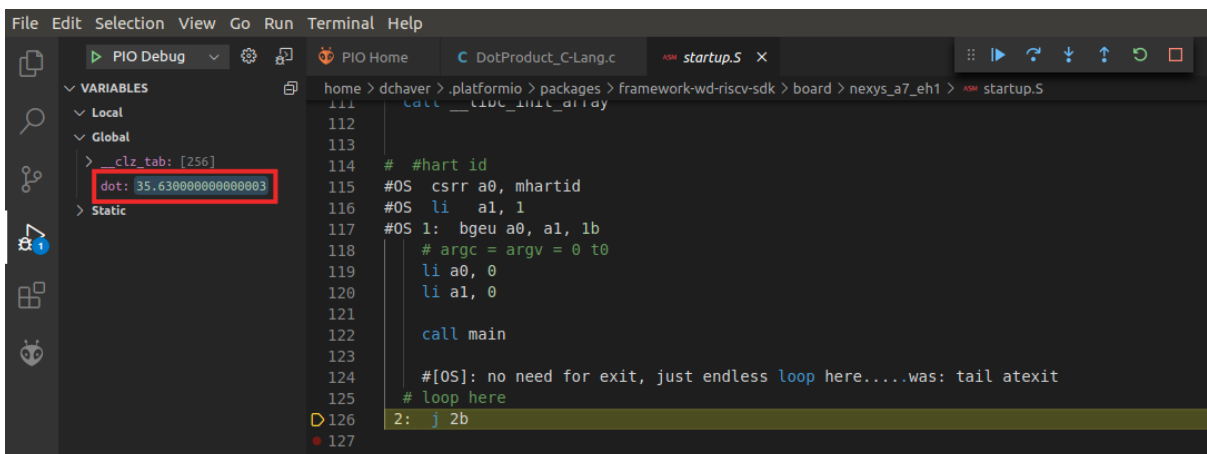
```

C DotProduct_C-Lang.c  dotproduct.dbgasm  PIO Home  platformio.ini
6  0x00000092: 4e ce      sw  s4,28(sp)
7  0x00000094: 52 cc      sw  s4,24(sp)
8  0x00000096: 2a 89      mv  s2,a0
9  0x00000098: 2e 8a      mv  s4,a1
10 0x0000009a: b2 89      mv  s3,a2
11 0x0000009c: 02 c6      sw  zero,12(sp)
12 0x0000009e: 01 44      li  s0,0
13 0x000000a0: 81 44      li  s1,0
14 0x000000a2: b2 47      lw  a5,12(sp)
15 0x000000a4: 63 d8 27 03    bge a5,s2,0xd4 <dotproduct+76>
16 0x000000a8: b2 47      lw  a5,12(sp)
17 0x000000aa: 8e 07      slli a5,a5,0x3
18 0x000000ac: d2 97      add a5,a5,s4
19 0x000000ae: 32 47      lw  a4,12(sp)
20 0x000000b0: 0e 07      slli a4,a4,0x3
21 0x000000b2: 4e 97      add a4,a4,s3
22 0x000000b4: 10 43      lw  a2,0(a4)
23 0x000000b6: 54 43      lw  a3,4(a4)
24 0x000000b8: 88 43      lw  a0,0(a5)
25 0x000000ba: cc 43      lw  a1,4(a5)
26 0x000000bc: e9 2d      jal 0x796 <_muldf3>
27 0x000000be: 2a 86      mv  a2,a0
28 0x000000c0: ae 86      mv  a3,a1
29 0x000000c2: 22 85      mv  a0,s0
30 0x000000c4: a6 85      mv  a1,s1
31 0x000000c6: cd 20      jal 0x1a8 <_adddf3>
32 0x000000c8: 2a 84      mv  s0,a0
33 0x000000ca: ae 84      mv  s1,a1
34 0x000000cc: b2 47      lw  a5,12(sp)
35 0x000000ce: 85 07      addi a5,a5,1
36 0x000000d0: 3e c6      sw  a5,12(sp)
37 0x000000d2: c1 bf      j 0xa2 <dotproduct+26>
38 0x000000d4: 22 85      mv  a0,s0
39 0x000000d6: a6 85      mv  a1,s1
40 0x000000d8: b2 50      lw  ra,44(sp)
41 0x000000da: 22 54      lw  s0,40(sp)
42 0x000000dc: 92 54      lw  s1,36(sp)
43 0x000000de: 02 59      lw  s2,32(sp)
44 0x000000e0: f2 49      lw  s3,28(sp)
45 0x000000e2: 62 4a      lw  s4,24(sp)
46 0x000000e4: 45 61      addi sp,sp,48
47 0x000000e6: 82 80      ret

```

그림 79. DotProduct\_C-Lang 프로그램: 두 번째 중단점의 어셈블리 코드

8. C 로 다시 전환하고 두 개의 중단점을 삭제합니다. 실행을 계속하고 일시 중지하십시오. 가변 점의 값이 두 벡터의 내적 (그림 80)으로 변경되는 것을 볼 수 있습니다.



```

File Edit Selection View Go Run Terminal Help
PIO Debug  PIO Home  C DotProduct_C-Lang.c  startup.S
VARIABLES
  Local
  Global
  > _clz_tab: [256]
  dot: 35.630000000000003
  Static
home > dchaver > .platformio > packages > framework-wd-riscv-sdk > board > nexys_a7_eh1 > startup.S
111  call __libc_init_array
112
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

```

그림 80. DotProduct\_C-Lang 프로그램: 내적 결과

9. 이 프로그램 탐색을 마치면 파일 → 폴더 닫기를 클릭하여 프로젝트를 닫습니다.

## 7. Verilator 에서 시뮬레이션

이 섹션에서는 Verilator 를 사용하여 RVfpgaSim 에서 이전 섹션 (*AL\_Operations*)에서 사용된 첫 번째 프로그램을 실행합니다. Verilator 는 Soc 를 정의하는 Verilog 를 시뮬레이션하는 HDL (하드웨어 설명 언어) 시뮬레이터입니다 (*[RVfpgaPath]/RVfpga/src*에서 사용 가능). SoC 를 실행하는이 방법을 사용하면 시스템의 내부 신호를 분석할 수 있으며, 이는 내부 작업 또는 새 하드웨어를 SoC 에 추가하는 미래의 실험실 및 연습에 특히 유용합니다.

여기에서는 Verilator 를 사용하여 주기 별 명령어를 보고 섹션 6 에서 실행하고 디버깅한 첫 번째 간단한 어셈블리 프로그램인 *AL\_Operations*의 값을 등록하는 방법을 보여줍니다 (그림 44). PlatformIO 를 사용하여 시뮬레이션 트레이스를 생성한 다음 클럭, 수퍼 스칼라 프로세서의 양방향 명령어를 추가하고 x28 (즉, t3 레지스터) 신호를 시뮬레이션 파형에 등록하고 프로그램이 실행됨에 따라 GTKWave 명령어 및 레지스터 신호가 변경되는 사항을 확인합니다.

### 시뮬레이션 바이너리 *Vrvfpgasim* 생성 :

*[RVfpgaPath]/RVfpga/verilatorSIM* 디렉토리에는 RVfpgaSim 용 시뮬레이터 바이너리를 생성하기 위한 *Makefile* 및 스크립트 (*swervolf\_0.7.vc*)가 포함되어 있습니다. 스크립트에는 Verilator 가 SoC 의 소스가 어느 곳에 있는지 알 수 있는 정보가, 우리 경우에는 *[RVfpgaPath]/RVfpga/src*, 포함되어 있습니다. 다음으로 RVfpgaSim 용 바이너리를 생성하는 방법을 보여줍니다. 나중에 RVfpgaSim 에서 실행되는 프로그램 *AL-Operations*의 시뮬레이션 추적을 만드는 데 사용됩니다.

1. 터미널 창에서 다음 명령을 실행하여 시뮬레이터 바이너리를 생성합니다.

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

파일 ***Vrvfpgasim*** (RVfpgaSim 시뮬레이션 바이너리)은 *[RVfpgaPath]/RVfpga/verilatorSIM* 디렉토리 내에 생성되어야 합니다.

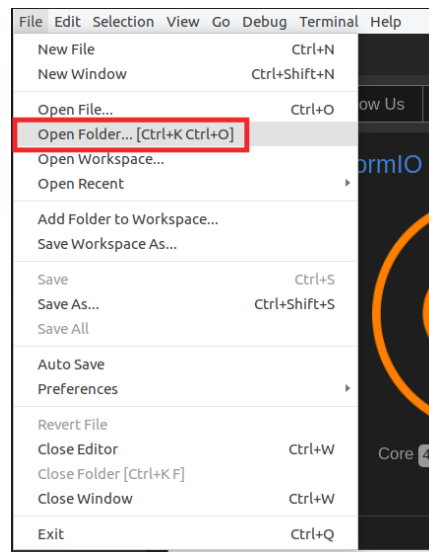
**Windows:** Windows 를 사용하는 경우 Cygwin 터미널 내에서 이와 동일한 단계를 수행해야 합니다 (자세한 지침은 부록 C 참조). C: Windows 폴더는 Cygwin (*/cygdrive/c*)에서 찾을 수 있습니다. 이 섹션의 다른 모든 지침은 Linux 에 대해 설명된 지침과 동일합니다.

**macOS:** 자세한 지침은 부록 D 를 참조하십시오.

### ***Vrvfpgasim* 을 사용하여 PLATFORMIO 에서 시뮬레이션 추적 생성:**

시뮬레이터 바이너리 (*Vrvfpgasim*)가 생성되면 프로그램 *AL\_Operations* 의 시뮬레이션 추적 (*trace.vcd*)을 생성하기 위해 PlatformIO 내에서 이를 사용합니다.

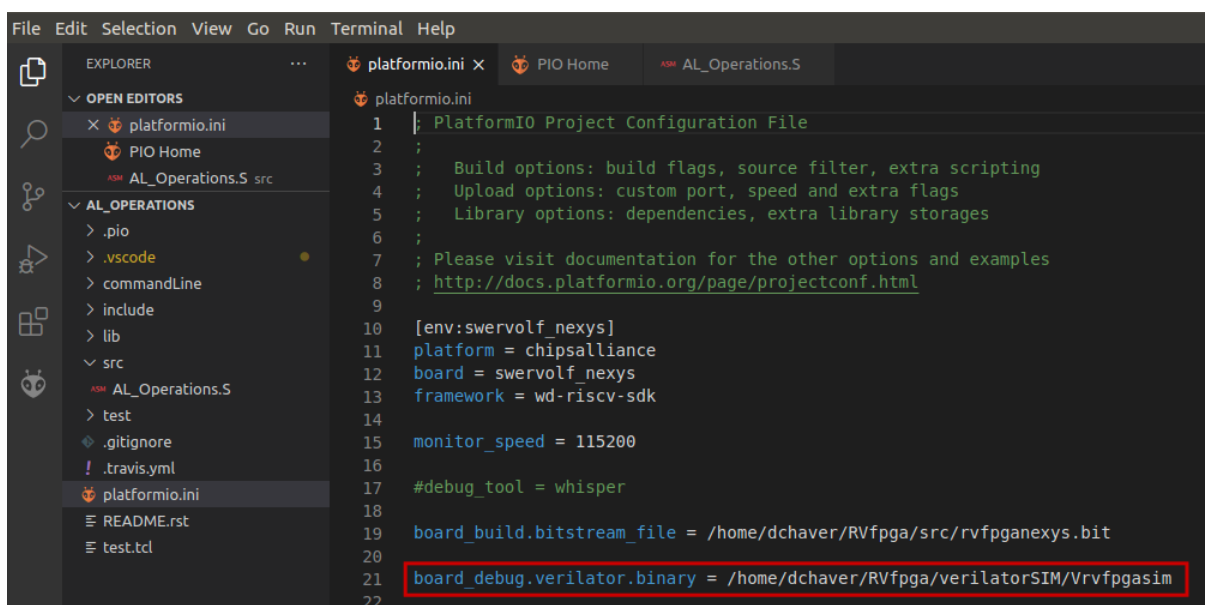
2. VSCode 를 연 다음 컴퓨터에서 PlatformIO 를 엽니다.
3. 상단 표시 줄에서 *File* → *Open Folder ...* (그림 81)를 클릭하고 *[RVfpgaPath]/RVfpga/examples/* 디렉터리로 이동합니다.



**Figure 11. Open the AL\_Operations.S example**

4. *AL\_Operations* 디렉토리를 선택하고 (열지 말고 선택만 하십시오) 확인을 클릭하십시오. 이 예제는 PlatformIO 에서 열립니다.
5. *platformio.ini* 파일을 엽니다. 다음 행을 편집하여 첫 번째 단계 (*Vrvfpgasim*)에서 생성된 RVfpgaSim 시뮬레이션 바이너리의 경로를 설정합니다 (그림 82 참조).


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



## 그림 82. PlatformIO 초기화 파일: platformio.ini

**Windows:** Windows 에서 RVfpgaSim 시뮬레이션 실행 파일은 vrvfpgasim.exe 입니다. 그러므로:

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

6. 왼쪽 메뉴 리본  에서 PlatformIO 아이콘을 클릭하여 시뮬레이션을 실행한 다음 Project Tasks  
→ env: swervolf\_nexys → Platform 을 확장하고 Generate Trace 를 클릭합니다 (그림 83 참조).

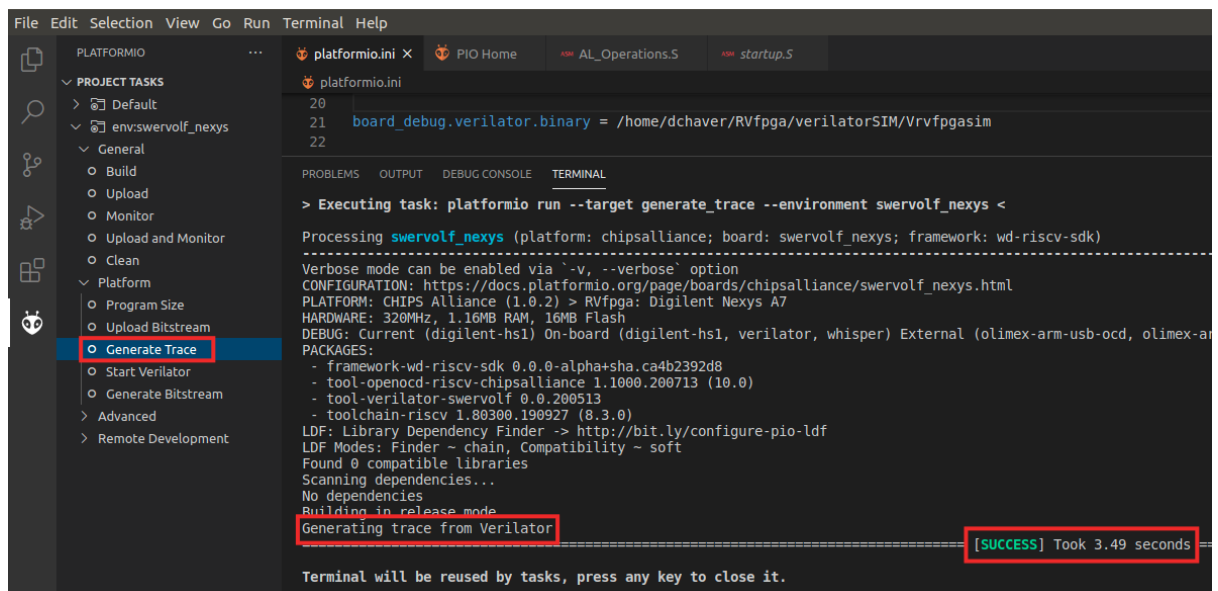



그림 83. Verilator 에서 추적 생성

또는 PlatformIO 터미널 창에서 추적을 생성할 수 있습니다. 이를 위해 PlatformIO 창 하단에 있는  버튼 (PlatformIO: New Terminal 버튼)을 클릭하여 새 터미널 창을 열고 다음 명령을 PlatformIO 터미널에 입력 (또는 복사)합니다: `pio run --target generate_trace`

7. 이전 단계 후 몇 초 후에 `trace.vcd` 파일이

`[RVfpgaPath]/RVfpga/examples/AL_Operations/.pio/build/swervolf_nexys` 에 생성되어야 하며  
GTKWave 로 열 수 있습니다.

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

**WINDOWS:** 다운로드한 `gtkwave64` 폴더에는 `bin` 폴더 안에 `gtkwave.exe` 라는 애플리케이션이 포함되어 있습니다. 해당 응용 프로그램을 두 번 클릭하여 GTKWave 를 시작합니다. 애플리케이션 상단에서 **File – Open New Tab** 을 클릭하고 아래 폴더에 생성된 `trace.vcd` 파일을 엽니다.

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

## GTKWAVE 에서 시뮬레이션 추적 분석 :

8. 이제 클럭, 명령 및 레지스터 신호를 추가합니다. *GTKWave*의 왼쪽 상단 창에서 그래프에 신호를 추가할 수 있도록 SoC의 계층을 확장합니다. 계층을 **TOP** → **rvfpgasim** → **swervolf** → **swerv\_eh1** → **swerv**로 확장하고 모듈 **ifu** (그림 84에 표시된 대로 강조 표시됨)를 클릭하고 신호 **clk** (코어에 사용되는 클럭)를 선택하고 흰색 신호 창 또는 오른쪽의 검은 색 웨이브 창으로 드래그합니다.

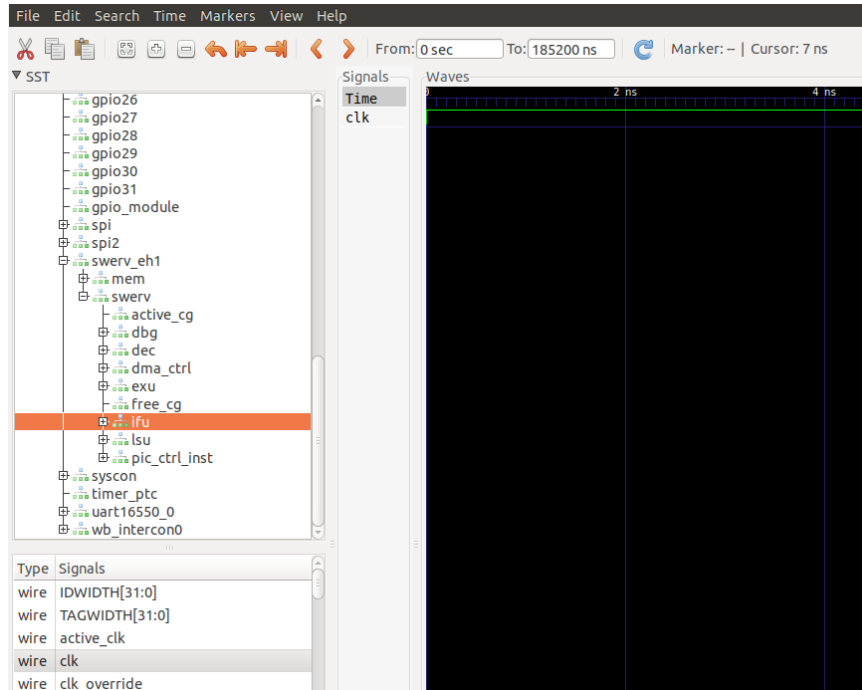


그림 84. 그래프에 신호 clk 추가

9. 시계 신호 변화를 볼 수 있도록 여러 번 확대합니다 (그림 85).

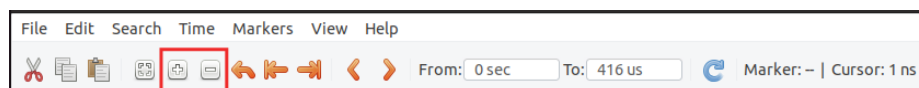


Figure 25. Zoom in

10. 이제 양방향 슈퍼 스칼라 RISC-V 코어의 각 방식으로 실행되는 명령어를 보여주는 신호를 추가합니다. 동일한 모듈 (**ifu**)에서 신호 **ifu\_i0\_instr [31:0]** 및 **ifu\_i1\_instr [31:0]** (그림 86)을 찾아 검은 색 Waves 창으로 드래그합니다. 접두사 **ifu**는 명령어 패치 단위를 나타내고, **i0**은 슈퍼 스칼라 방식 0을, **i1**은 슈퍼 스칼라 방식 1을 나타냅니다. **instr [31:0]**은 32 비트 명령어를 나타냅니다.

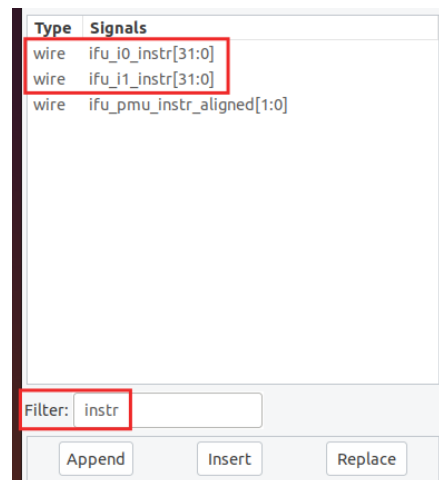


그림 86. 타이밍 파형에 신호 ifu\_i0\_instr [31:0] 및 ifu\_i1\_instr [31:0] 추가

11. 이제 레지스터 t3 (즉, 레지스터 번호 28, x28)의 값을 보유하는 신호를 추가합니다. **swerv** 아래의 계층 구조를 **dec** → **arf** → **gpr\_banks (0)** → **gpr (28)**로 확장하고 **gprff** 모듈 (다음 그림과 같이 강조 표시됨)을 클릭하고 신호 **dout [31:0]**를 선택합니다. *AL\_Operations.S* 예제에서 사용되는 레지스터 x28)을 검은 색 Waves 창으로 드래그합니다 (그림 87).

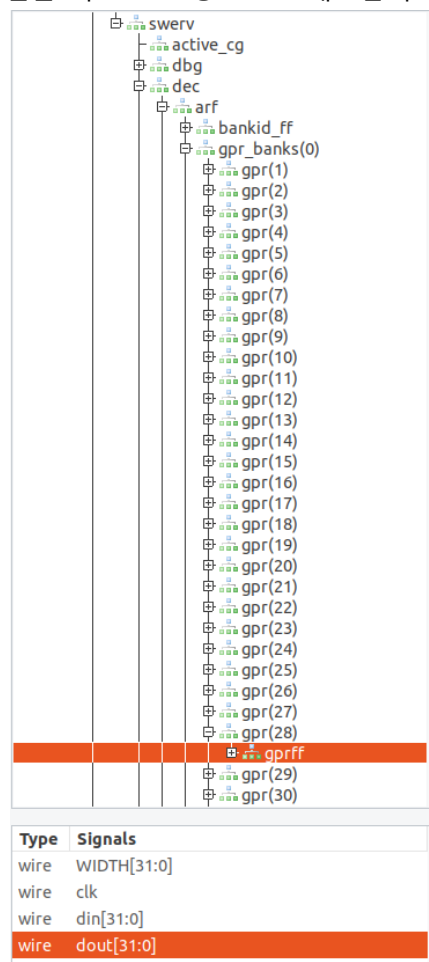


그림 87. 그래프에 신호 dout [31:0] 추가



12. GTKWave 에서 신호를 표시하는 또 다른 방법은 .tcl 파일을 사용하는 것입니다. test.tcl 파일은 *[RVfpgaPath]/RVfpga/examples/AL\_Operations* 에서 제공됩니다. 해당 파일을 열고 분석하십시오. 각 선에는 그래프에 표시하려는 각 신호의 경로와 이름이 표시됩니다.

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

GTKWave 에서 .tcl 파일을 사용하려면 File – Read Tcl Script File 을 클릭하고 *[RVfpgaPath]/RVfpga/examples/AL\_Operations/test.tcl* 파일을 선택하면 됩니다.

13. 그림 88 은 AL\_Operations.S 프로그램과 이에 상응하는 기계 명령어를 보여줍니다.

| # RISC-V assembly      | # comment (t3 = x28) | # machine code |
|------------------------|----------------------|----------------|
| li t3, 0x0             | # t3 = 0             | # 0x00000E13   |
| <b>REPEAT:</b>         |                      |                |
| 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                | # 0x00000013   |

**그림 88. 동등한 기계 코드가있는 AL\_Operations.S**

이제 프로그램이 실행됨에 따라 신호 변경을 확인하십시오. 프로그램이 실행될 때 명령어와 t3 (레지스터 x28)이 그림 89 에 표시된 값이 될 것으로 예상합니다:

|         |                        |                   |               |
|---------|------------------------|-------------------|---------------|
|         | li t3, 0x0             | # t3 = 0          | # 0x00000E13  |
| REPEAT: | addi t3, t3, 6         | # t3 = 0 + 6 = 6  | # 0x006E0E13  |
|         | addi t3, t3, -1        | # t3 = 5          | # 0xFFFFE0E13 |
|         | andi t3, t3, 3         | # t3 = 5 & 3 = 1  | # 0x003E7E13  |
|         | beq zero, zero, REPEAT | # Repeat the loop | # 0xFE000CE3  |
|         | nop                    | # nop             | # 0x00000013  |
| REPEAT: | addi t3, t3, 6         | # t3 = 1 + 6 = 7  | # 0x006E0E13  |
|         | addi t3, t3, -1        | # t3 = 7 - 1 = 6  | # 0xFFFFE0E13 |
|         | andi t3, t3, 3         | # t3 = 6 & 3 = 2  | # 0x003E7E13  |
|         | beq zero, zero, REPEAT | # Repeat the loop | # 0xFE000CE3  |
|         | ...                    |                   |               |

**그림 89. AL\_Operations 실행 중 레지스터 t3 (x28)의 명령 흐름 및 값**

14. 약 10100ps 를 확대하면 루프의 첫 번째 및 두 번째 반복에 대한 세 개의 산술 논리 명령어 실행을 분석할 수 있습니다 (그림 90). 처음 두 개의 명령어 (li t3, 0x0 = 0x00000E13 및 addi t3, t3, 6 = 0x006E0E13)를 먼저 가져옵니다. 신호 *ifu\_i0\_instr[31:0]* 및 *ifu\_i1\_instr[31:0]*에 표시된 것처럼 슈퍼 스칼라 RISC-V 프로세서의 각 방식으로 하나씩 가져 옵니다. 다음 두 명령 (addi t3, t3, -1 = 0xFFFFE0E13 및 andi t3, t3, 3 = 0x003E7E13)은 다음 사이클에서 가져옵니다. 마지막 두 명령어는 (beq zero, zero, REPEAT = 0xFE000CE3 및 nop = 0x00000013) 다음주기에서 가져 옵니다.

SweRV 코어의 9 단계 파이프 라인 프로세서 및 종속성으로 인해 명령을 가져온 후 8 또는 그이상의 사이클에서 명령의 효과가 나타납니다. 첫 번째 명령과 두 번째 명령을 가져온 후 8 주기,  $x28(t3)$  은 첫 번째 명령인 `li t3, 0x0 (0x00000E13)` 때문에 0 (이미 그대로)이 됩니다. 한 사이클 후에  $x28$  은 다음 명령어인 `addi t3, t3, 6 (0x006E0E13)` 때문에 0x6 으로 업데이트됩니다. 다음으로  $x28$  은 다음 명령어인 `addi t3, t3, -1 (0xFFFFE0E13)` 때문에 5 로 업데이트됩니다. 마지막으로  $x28$  은 다음 명령어인 `andi t3, t3, 3 (0x003E7E13)` 때문에 1 로 업데이트됩니다. 그런 다음 다음 두 명령어를 가져 옵니다: `beq zero, zero, REPEAT (0xFE000CE3)` 및 `nop (0x00000013)`, 분기가 취해지며 루프가 반복됩니다. 이것은 그림 89 에서 예측한 것과 같습니다. 유사한 추론을 사용하여 두 번째 반복을 분석할 수 있습니다. 두 번째 반복은 그림 90 에서도 강조 표시되고 그림 89 에서 예측됩니다.

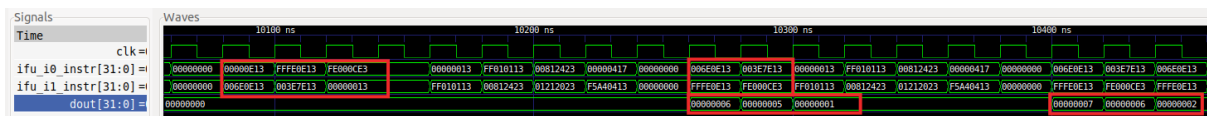


그림 90. 예제에서 세 개의 산술 논리 명령어 실행

## 8. Whisper 시뮬레이션

Whisper (<https://github.com/chipsalliance/SweRV-ISS>) 는 SweRV 마이크로 컨트롤러의 검증을 위해 Western Digital 에서 개발한 RISC-V ISS (명령어 집합 시뮬레이터)입니다. 이를 통해 사용자는 기본 RISC-V 하드웨어 없이도 RISC-V 코드를 실행할 수 있습니다. Whisper 를 사용하면 Nexys A7 FPGA 보드 없이도 PlatformIO 를 사용하여 C 또는 어셈블리 프로그램을 테스트, 실행 및 디버깅할 수 있습니다.

**Windows:** 이 섹션에 설명된 모든 지침은 Windows 에서 작동해야 합니다 (Whisper 를 Windows 로 처음 이식한 Jean-François Monestier 에게 감사드립니다: <https://jean-francois.monestier.me/porting-western-digital-swerv-iss-to-windows/>). Windows 방화벽을 통해 Whisper 를 허용할지 묻는 팝업 창이 나타날 수 있습니다.

**macOS:** 이 섹션에 설명된 모든 지침은 macOS 에서도 작동합니다.

Whisper 는 명령줄을 사용하거나 Eclipse 또는 PlatformIO 와 같은 IDE (통합 개발 환경)를 사용하여 실행할 수 있습니다. 이 섹션에서는 PlatformIO 에서 Whisper 로 프로그램을 시뮬레이션하는 방법을 보여주는 한 가지 예를 보여줍니다. 그런 다음 여기에 설명된 단계와 동일한 단계를 사용하여 다른 프로그램을 시뮬레이션할 수 있습니다.

Whisper ISS 를 사용하여 섹션 6 에서 실행하고 디버깅한 첫 번째 간단한 어셈블리 프로그램인 *AL\_Operations* 를 시뮬레이션합니다 (그림 44 참조). Whisper 에서 이 코드를 실행하고 디버깅하려면

다음 단계를 따르세요.

1. VSCode (및 PlatformIO)를 엽니다. 상단 메뉴 표시 줄에서 파일 → 폴더 열기를 클릭하고 `[RVfpgaPath]/RVfpga/examples/` 디렉토리로 이동하고 `AL_Operations` 디렉토리를 선택한 (열지 않음) 다음 확인을 클릭합니다.
2. 파일 → 파일 열기를 클릭하고 `[RVfpgaPath]/RVfpga/examples/AL_Operations/platformio.ini` 를 두 번 클릭하고 17 행의 주석 처리를 제거하여 **whisper** 를 디버그 도구로 설정합니다 (그림 91 참조). 파일을 저장합니다 (Ctrl-S).

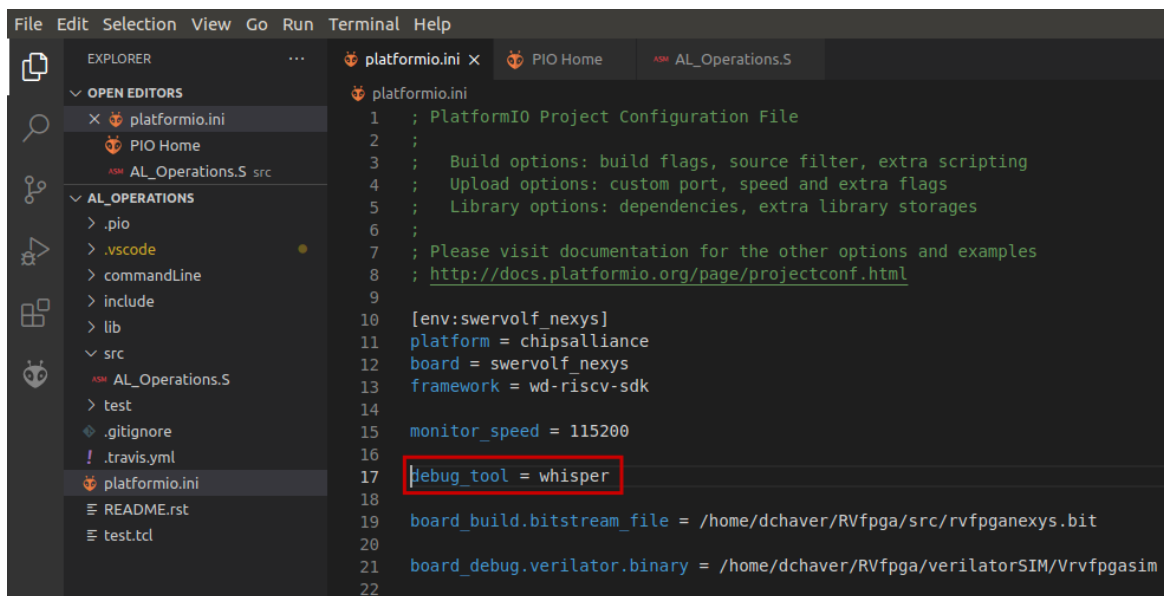


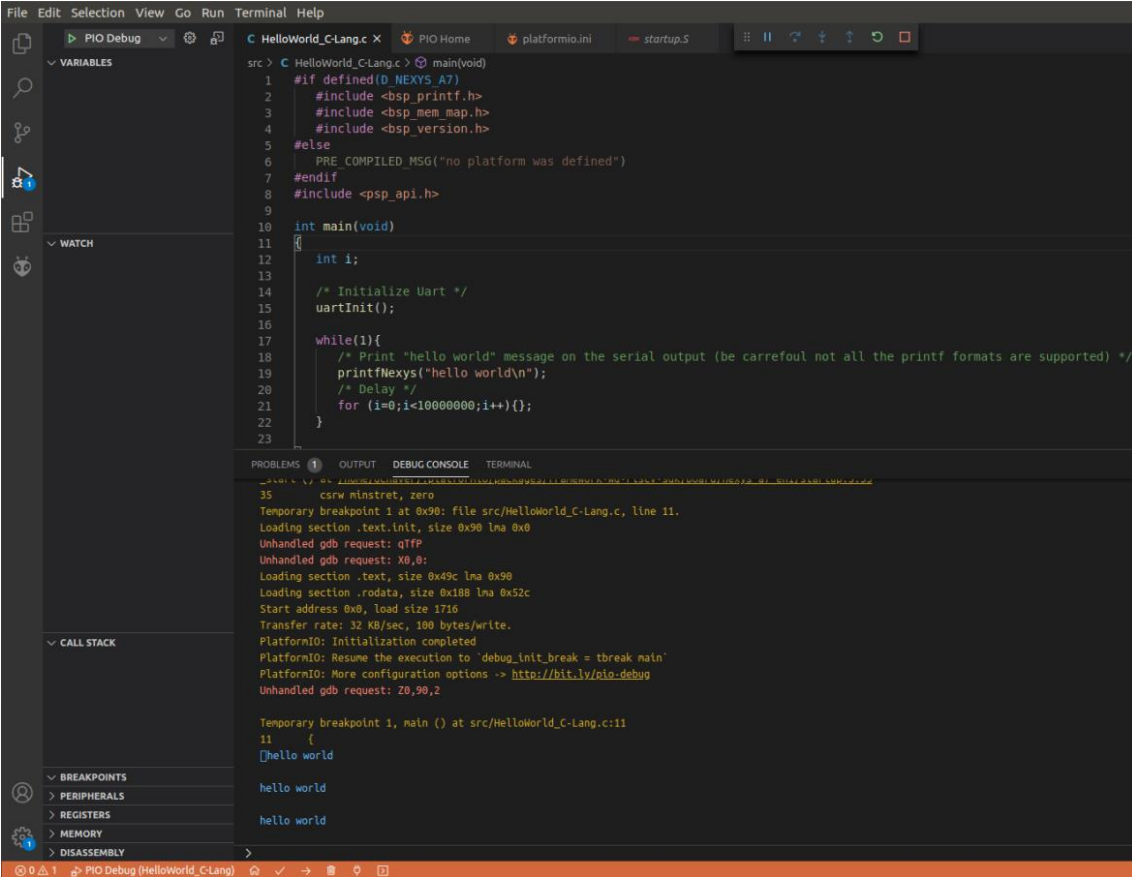


그림 91. 주석 제거 17 행.

3. 평소처럼 디버거를 시작합니다. (  클릭후 `RUN`  `PIO Debug` 클릭 )
4. 이제 섹션 6.B 에서 했던 것과 똑같이 프로그램을 디버깅할 수 있지만 이번에는 프로그램이 Nexys A7 FPGA 보드 대신 Whisper 에서 시뮬레이션으로 실행됩니다.
5. 프로그램이 HelloWorld\_C-Lang 예제 (섹션 6.F)와 같이 Whisper 의 `printfNexys` 함수를 사용하는 경우 메시지가 대신 DEBUG 콘솔에 표시되므로 PlatformIO 직렬 모니터를 열면 안됩니다 (그림 92 참조).



```

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 <psp_api.h>
9
10 int main(void)
11 {
12     int i;
13
14     /* Initialize Uart */
15     uartInit();
16
17     while(1){
18         /* Print "hello world" message on the serial output (be careful not all the printf formats are supported) */
19         printfNexys("hello world\n");
20         /* Delay */
21         for (i=0; i<100000000; i++){};
22     }
23
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
35  csrw minstret, zero
Temporary breakpoint 1 at 0x90: file src/HelloWorld_C-Lang.c, line 11.
Loading section .text.init, size 0x90 lma 0x0
Unhandled gdb request: qTFF
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  {
    [h]ello world

hello world

hello world

```

그림 92. Whisper 에서 HelloWorld C-Language 예제 실행

## 9. 부록

다음 부록은 Linux 에서 기본 RISC-V 도구 모음 및 OpenOCD (PlatformIO 대신)를 사용하는 방법, PlatformIO 를 사용하여 비트 스트림을 다운로드하기 위해 Windows 에 드라이버를 설치하는 방법, Windows 및 Mac OS 시스템에 Verilator 및 GTKWave 를 설치하는 방법, Vivado 를 사용하여 RVfpgaNexys 를 프로그래밍하는 방법, 표 10 에는 이 RVfpga 시작 안내서에서 사용할 수 있는 모든 부록이 나열되어 있습니다.

**표 10. 부록 목록**

| Appendix | Description   | Operating System                  |
|----------|---|-----------------------------------|
| A        | Ubuntu 18.04 에서 RVfpga 용 Native RISC-V 도구 체인 및 OpenOCD 사용 | Linux                             |
| B        | PlatformIO 를 사용하기 위해 Windows 에 드라이버 설치                    | <a href="#">Windows</a>           |
| C        | Windows 에 Verilator 및 GTKWave 설치                          | <a href="#">Windows</a>           |
| D        | macOS 에 Verilator 및 GTKWave 설치                            | macOS                             |
| E        | Vivado 를 사용하여 FPGA 에 RVfpgaNexys 다운로드                     | <a href="#">Windows</a> and Linux |
| F        | 산업용 IoT 애플리케이션에서 RVfpga 사용                                | All                               |

부록 A 는 기본 gcc/gdb 도구 및 OpenOCD 를 사용하여 기본적으로 프로그램을 컴파일하고 실행/디버그하려는 사용자가 사용해야 합니다. 그러나 **RVfpga 사용자는 이 시작 안내서에 설명된 대로 PlatformIO 를 대신 사용하는 것이 좋습니다.**

[Windows](#) 사용자는 **부록 B 및 C 의 지침**을 따라야 합니다. 부록 B 의 지침은 Windows 시스템이 PlatformIO 를 사용하여 프로그램을 다운로드하고 Nexys A7 FPGA 보드에 RVfpgaNexys 를 다운로드할 수 있도록 드라이버를 다운로드하는 방법을 보여줍니다. 부록 C 는 Windows 사용자가 RVfpgaSim 을 시뮬레이션할 수 있도록 Verilator 및 GTKWave 를 설치하는 방법을 보여줍니다.

macOS 사용자는 Verilator 및 GTKWave 를 사용하여 RVfpgaSim 을 시뮬레이션하려면 **부록 D 의 지침**을 따라야 합니다.

**PlatformIO 를 사용하여 RVfpgaNexys 시스템** (비트 파일, rvfpganexys.bit 에 정의된 대로)을 **Nexys A7 FPGA 보드에 다운로드하는 것이 좋습니다.** 이 비트 파일 (rvfpganexys.bit)은 Vivado 또는 PlatformIO 에서 생성할 수 있습니다. 부록 E 에 설명된 대로 Vivado 를 사용하여 Nexys A7 FPGA 보드에 RVfpgaNexys 시스템을 다운로드할 수도 있습니다. 그러나 Vivado 를 사용하여 보드에 RVfpgaNexys 를 다운로드하는 것은 **권장되지 않습니다.** 특히 [Windows](#) 사용자에게는 지속적으로 드라이버를 교체해야 하기 때문입니다.

## 부록 A : Ubuntu 18.04 에서 Native RISC-V 도구 체인 및 OpenOCD 사용

PlatformIO 사용을 권장하지만 이 섹션에서는 기본 RISC-V 도구 체인을 설치, 실행 및 사용하는 방법과 OpenOCD 를 사용하여 Nexys A7 FPGA 보드에 RVfpgaNexys 를 다운로드하고 gdb 를 사용하여 RVfpgaNexys 에서 프로그램을 실행하고 디버깅하는 방법을 보여줍니다. 툴체인은 gnu 컴파일러, 디버거, 어셈블러 등으로 구성됩니다. Ubuntu 18.04 운영 체제 (OS)에 RISC-V 툴체인과 OpenOCD 를 설치하는 방법을 보여 주지만 이 프로세스는 다른 Linux 배포에서도 작동합니다. 이 지침은 새로운 Ubuntu 시스템을 가정합니다.

이 가이드의 앞부분에서 설명한 대로 PlatformIO 를 사용하는 경우 다음 단계가 필요하지 않습니다. PlatformIO, Vivado 및 Verilator 또는 Whisper 를 사용하는 것이 RISC-V 프로그램을 실행, 디버깅 및 시뮬레이션하는 데 권장되는 방법이지만 PlatformIO 와 Vivado 하드웨어 매니저 대신 기본 RISC-V 도구 체인 및 OpenOCD 를 사용하려는 모든 사용자를 위해 다음 지침이 제공됩니다.

## I. Linux Ubuntu OS 에 기본 설치

이 섹션에서는 Ubuntu 18.04 컴퓨터에 RISC-V 도구 모음, OpenOCD 및 Whisper 를 기본적으로 설치하는 방법을 설명합니다. 이러한 도구는 PlatformIO 만 대체합니다. 이 GSG 의 섹션 5 에 설명된 대로 Vivado 및 Verilator 를 설치해야 합니다.

### RISC-V 툴체인

여기에서는 전체 RISC-V 툴체인 (예: gnu 컴파일러, 디버거 등)을 컴퓨터에 설치하는 방법을 보여줍니다. 설치 지침은 RISC-V International (<https://github.com/riscv/riscv-gnu-toolchain>)에서 제공합니다. 이러한 지침은 아래에 요약되어 있습니다.

NOTE: RISC-V 도구 모음과 OpenOCD 를 설치하는 데 몇 시간이 걸릴 수 있습니다. 대부분 도구 모음이 다운로드, 컴파일 및 설치되는 동안 대기합니다.

터미널에서 다음을 입력합니다 (프로세스는 한 시간 이상 걸릴 수 있지만 대부분의 시간은 프로그램이 다운로드되고 설치되는 동안 대기하는 데 소요됩니다).

- `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` (가능한 경우 컴파일 시간을 크게 줄이므로 `sudo make -j$(nproc)` 사용)
- `export PATH=$PATH:/opt/riscv/bin` (시스템에서 경로 변경)

### OpenOCD

OpenOCD 는 사용자가 임베디드 대상 장치를 프로그래밍하고 디버깅할 수 있는 개방형 온 칩 디버거입니다. 컴퓨터에 RISC-V OpenOCD 를 설치하려면 다음 단계를 따르십시오.

- `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

컴퓨터에 Whisper 를 설치하려면 다음 단계를 따르십시오. (지침은

<https://github.com/chipsalliance/SweRV-ISS> 에서 확인할 수 있지만 아래에 요약되어 있습니다).

- `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` (필요에 따라 [RVfpgaPath] 를 바꿉니다).

## II. OpenOCD 를 사용하여 Nexys A7 FPGA 보드를 사용하여 RVfpgaNexys 에서 프로그램 실행

### 단계 A. Nexys A7 에 RVfpgaNexys (그림 25) 다운로드

1. RVfpgaNexys 용 비트 파일이 포함 된 프로젝트 디렉토리로 이동합니다.  
`cd [RVfpgaPath]/RVfpga/src`
2. OpenOCD 를 사용하여 RVfpgaNexys 를 보드에 다운로드합니다.  
`riscv-openocd -c "set BITFILE rvfpganexys.bit" -f OtherSources/ConfigFiles/swervolf_nexys_program.cfg`

### 단계 B. 스위치를 읽고 LED 에 상태를 인쇄하는 프로그램 인 LedsSwitches 를 실행 합니다.

3. Leds 스위치/명령줄 디렉토리로 이동합니다.  
`cd [RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine`

이 디렉토리에서 소스, 링크 스크립트, Python 스크립트 및 *LedsSwitches.S* 프로그램을 컴파일하기 위한 Makefile 을 찾을 수 있습니다.

4. .elf 파일 빌드:  
`make clean`  
`make LedsSwitches.elf`
5. OpenOCD 를 SoC 에 연결합니다.  
`riscv-openocd -f ../../../../src/SweRVolfSoC/OtherSources/swervolf_nexys_debug.cfg`

OpenOCD 가 실행되기 시작하면 다음 메시지를 포함한 여러 메시지가 표시됩니다.

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

#### 6. 새 터미널을 열고 프로그램 디렉토리 (cd

[RVfpgaPath]/RVfpga/examples/LedsSwitches/commandLine) 로 이동하여 다음 명령을 실행합니다.

```
telnet localhost 4444
```

그런 다음 텔넷 연결 내부에 다음을 입력합니다.

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

이 세 명령은 (1) LedsSwitches.elf 프로그램을 RVfpgaNexys 에 로드하고, (2) 프로그램 카운터 (PC)를 0 (프로그램의 첫 번째 명령어 주소 위치)으로 설정하고 (3) 실행을 다시 시작합니다.

이 프로그램은 2 단계에서 Nexys A7 FPGA 보드에 이미 다운로드된 RISC-V SweRVofX SoC 인 RVfpgaNexys 에서 실행되기 시작합니다. 이 프로그램은 LED 가 스위치의 상태를 표시하도록 합니다. 스위치를 전환하면 LED 가 스위치 값을 반영하도록 즉시 변경되어야 합니다.

### 단계 C. 간단한 산술 논리 연산을 실행하는 AL\_Operations\_CommandLine 프로그램 디버그

이제 OpenOCD 및 gdb 를 사용하여 다른 프로그램 (AL\_Operations\_CommandLine)을 디버깅하는 방법을 보여줍니다.

#### 7. OpenOCD 연결을 열어 둡니다 (5 단계 참조).

#### 8. 텔넷이 실행중인 다른 터미널 (6 단계에서)에서 다음을 입력하여 텔넷 연결을 종료합니다.

```
exit
```

#### 9. 작업/명령 줄이 포함 된 프로젝트 디렉토리로 변경합니다.

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

이 디렉토리에서 소스, 링크 스크립트, 파이썬 스크립트 및 AL\_Operations.S 프로그램을 컴파일하기위한 Makefile 을 찾을 수 있습니다.

#### 10. .elf 파일을 빌드합니다.

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

#### 11. 그런 다음이 터미널에서 다음을 입력하여 gdb 를 시작합니다.

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



12. gdb 콘솔에서 다음을 입력합니다.

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

이것은 OpenOCD 에 연결되고 AL\_Operations.elf 프로그램을 메모리에 로드합니다.

13. 이제 프로그램을 디버그할 수 있습니다. 다음 순서를 입력하고 출력을 분석하십시오.

i. `disas 0,20`

이것은 주소 0 에서 20 까지의 어셈블리 코드를 보여줍니다 (주소 20 제외). (그림 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.
```

그림 93. 어셈블리 프로그램보기

ii. `i r t3`

이것은 레지스터 t3 의 내용을 표시합니다. 또는 더 긴 버전을 입력할 수 있습니다:

`info reg t3.` (그림 94 참조)

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

그림 94. 레지스터 t3 에 포함 된 값 인쇄

iii. `i r pc`

프로그램 카운터 (pc)의 내용을 표시합니다. (그림 95 참조)

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

그림 95. 첫 번째 명령어를 가리키는 레지스터 PC 에 포함된 값 인쇄

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

`stepi` 는 프로그램이 하나의 명령을 실행하도록 합니다. 그런 다음 `i r t3` 은 레지스터 `t3` 의 내용을 표시합니다. (그림 96 참조)

```
(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
```

그림 96. 여러 명령어를 하나씩 실행하고 `t3` 레지스터보기

디버깅 및 프로그램 탐색을 마치고 `gdb` 를 사용하여 등록하고 `gdb` 터미널에 `quit` 를 입력하여 `gdb` 를 종료하고 OpenOCD 터미널에 `^C` 를 입력하여 OpenOCD 를 종료합니다.

### III. Verilator 를 사용하여 RVfpgaSim 에서 프로그램 시뮬레이션

1. Ubuntu 에서 터미널을 엽니다.
2. 터미널 창에서 다음 명령을 실행하여 시뮬레이터 바이너리를 생성합니다.

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

파일 *Vrvfpgasim* (RVfpgaSim 시뮬레이션 바이너리)은 `[RVfpgaPath]/RVfpga/verilatorSIM` 디렉토리 내에 생성되어야 합니다.

3. 예제 프로그램이 포함된 폴더로 이동합니다.

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

4. 시뮬레이션을 위한 16 진수 프로그램을 만듭니다.

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

5. 시뮬레이터를 실행합니다.

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

몇 초 후 터미널에 `^C` 를 입력하여 시뮬레이션을 중지합니다. *trace.vcd* 파일이 생성되며

GTKWave 로 열 수 있습니다.  
`gtkwave trace.vcd`

6. 그래프에 신호를 추가하고 분석하려면 섹션 7 의 8-12 단계에 제공된 지침을 따르십시오.

#### IV. Whisper 에서 프로그램 시뮬레이션

1. Ubuntu 에서 터미널을 엽니다.
2. 예제 프로그램이 포함된 폴더로 이동합니다:  
`cd [RVfpgaPath]/RVfpga/examples/AL_Operations/commandLine`
3. Disassembly 프로그램을 만듭니다.  
`make AL_Operations.dis`
4. 편집기에서 *AL\_Operations.dis* 를 엽니다. 이것은 당신이 봐야할 것 입니다:

```
<_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. 대화 형 모드에서 시뮬레이터를 실행합니다.  
`whisper --interactive AL_Operations.elf`
6. 프로그램을 디버그합니다.

```
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
```

프로그램 디버깅 및 탐색을 마치고 whisper 를 사용하여 등록한 후 터미널에 quit 를 입력하여 종료합니다.

## 부록 B : PlatformIO 를 사용하기 위해 Windows 에 드라이버 설치

Zadig 실행 파일을 다운로드하려면 다음 웹 사이트로 이동하십시오 (그림 97 참조).

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

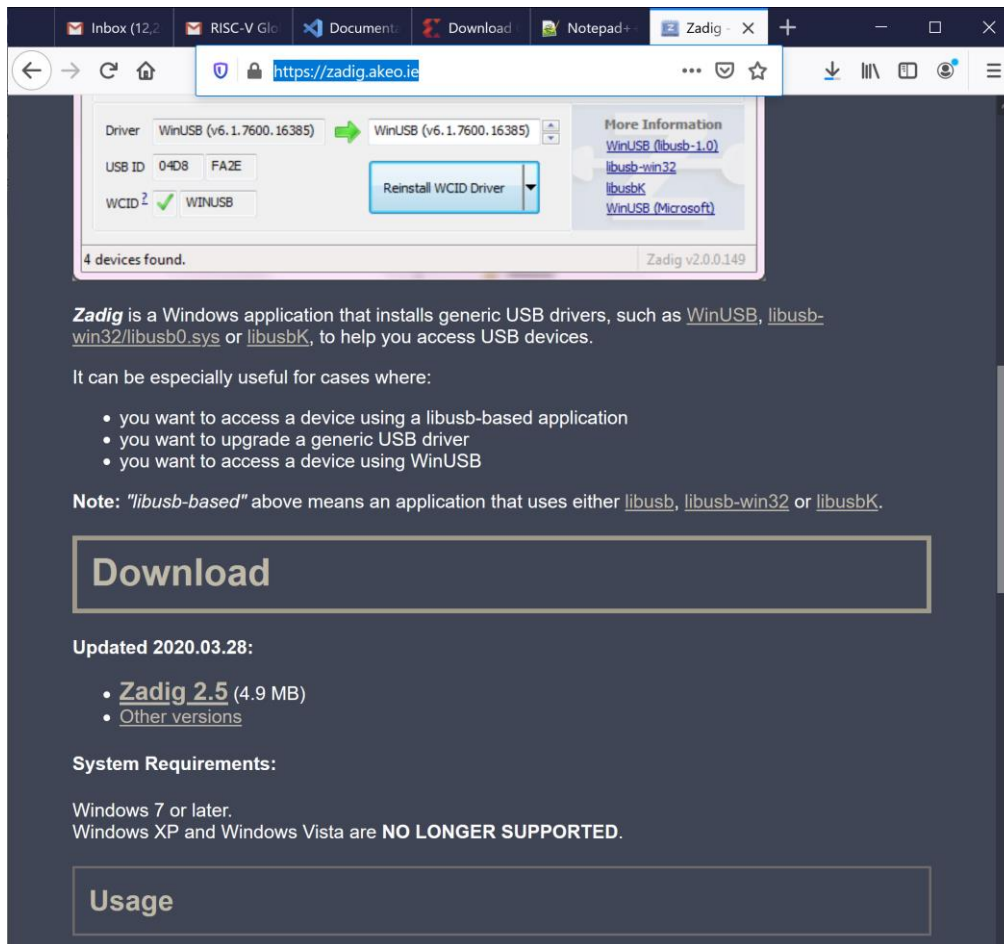


그림 97. PlatformIO 에서 사용하는 Nexys A7 보드 드라이버 설치

Zadig 2.5 를 클릭하고 실행 파일을 저장하십시오. 그런 다음 다운로드한 위치에 있는 파일 (zadig-2.5.exe)을 실행합니다. 시작 메뉴에 zadig 를 입력하여 찾을 수도 있습니다. Zadig 가 컴퓨터를 변경하도록 허용할지 여부와 업데이트 확인을 허용할지 묻는 메시지가 표시될 것입니다. 예를 두 번 클릭하십시오.

Nexys A7 보드를 컴퓨터에 연결하고 전원을 켭니다. Zadig 에서 Options → List All Devices 를 클릭합니다 (그림 98 참조).

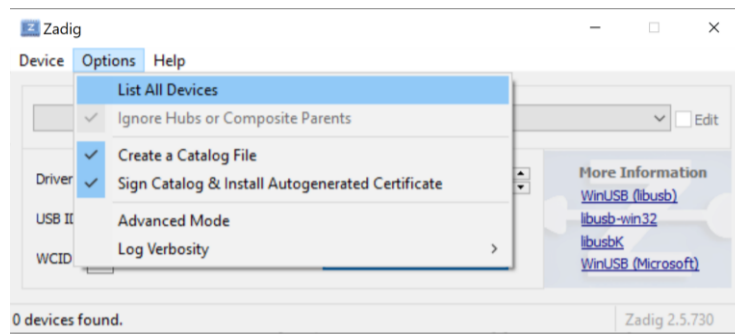


그림 98. Zadig 의 모든 장치 나열

드롭 다운 메뉴를 클릭하면 Digilent USB 장치 (인터페이스 0) 및 Digilent USB 장치 (인터페이스 1)가 나열됩니다. Digilent USB 장치 (인터페이스 0) 전용 새 드라이버를 설치합니다 (그림 99 참조).

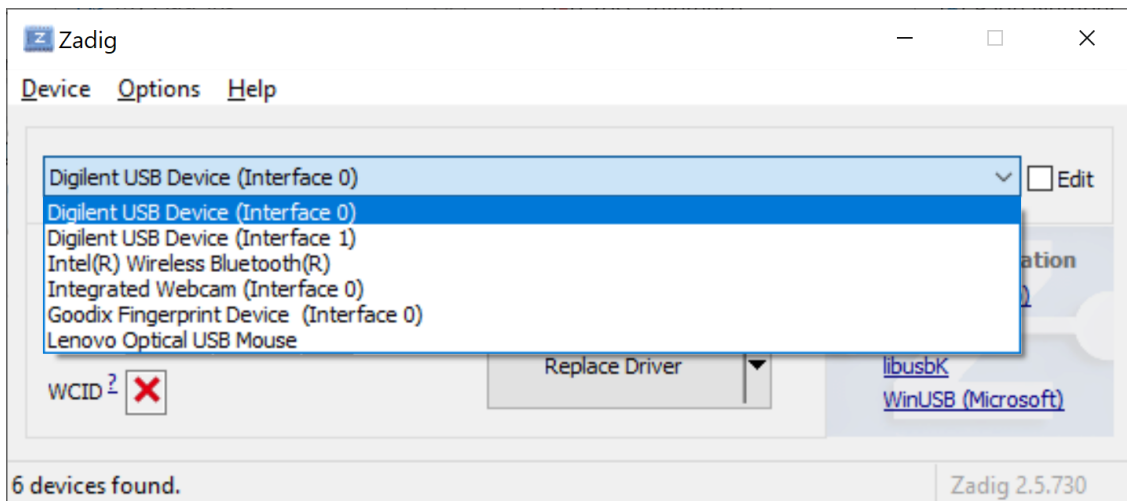


그림 99. Digilent USB 장치 (인터페이스 0) 용 WinUSB 드라이버 설치

이제 그림 100 와 같이 FTDI 드라이버를 WinUSB 드라이버로 교체합니다. Digilent USB 장치 (인터페이스 0)에 대한 드라이버 교체 (또는 드라이버 설치)를 클릭합니다. Nexys A7 보드 용 드라이버를 설치하거나 이전에 Vivado 를 설치한 경우 Vivado 에서 사용하는 FTDI 드라이버를 PlatformIO 에서 사용하는 WinUSB 드라이버로 교체합니다.

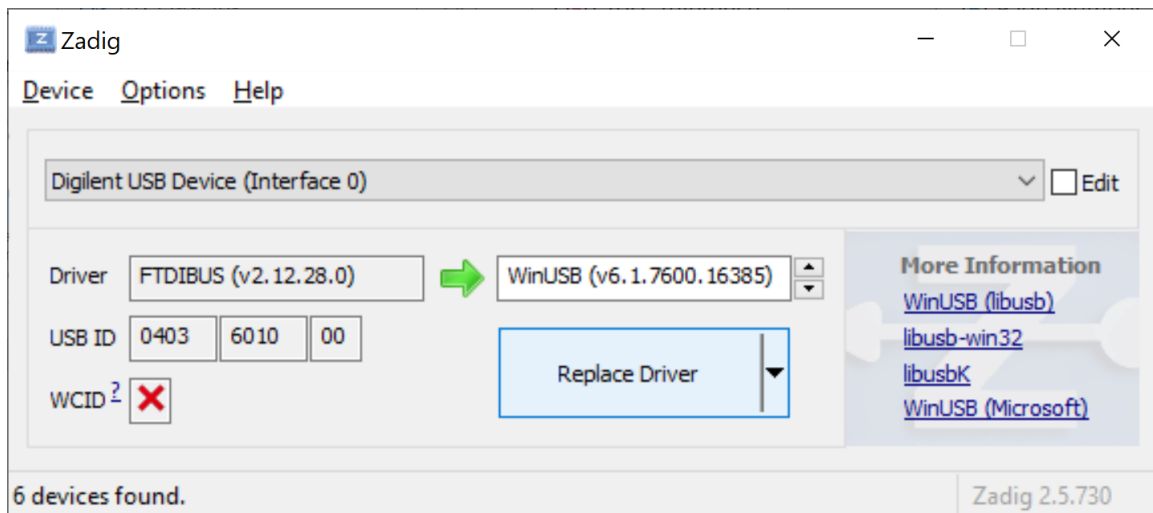


그림 100. Nexys A7 보드 용 드라이버 교체

일반적으로 몇 분 후 Zadig 는 드라이버가 올바르게 설치되었음을 표시합니다. 닫기를 클릭한 다음 Zadig 창을 닫습니다.

다음에 PlatformIO 를 사용할 때는 드라이버를 다시 설치할 필요가 없습니다. 그러나 **이 드라이버는 Windows 의 Vivado 와 호환되지 않습니다.** 따라서 더 이상 Vivado 를 사용하여 FPGA 보드에 비트 파일을 다운로드할 수 없습니다. Vivado 를 사용하여 비트 파일을 다운로드하려면 (권장하지 않음) 부록 E 에 설명된 대로 드라이버를 Vivado 와 함께 설치된 원래 드라이버로 되돌려야 합니다.

## 부록 C : Windows 에 Verilator 및 GTKWave 설치

이 섹션에서는 Windows 10 에 Verilator 및 GTKWave 를 설치하는 방법을 설명합니다.

Windows 에서는 Cygwin 을 사용하여 Verilator 를 설치해야 함으로 먼저 이 프로그래밍/런타임 환경을 설치하는 방법을 설명합니다.

### Cygwin 설치:

웹 페이지 (<https://www.cygwin.com>)에 설명된 대로 Cygwin 은 Linux 배포판과 유사한 Windows 에서 기능을 제공하는 GNU 및 오픈 소스 도구로 구성됩니다. Windows 10 에 Cygwin 을 설치하려면 다음 단계를 따르십시오.

1. 설치 웹 페이지 (<https://cygwin.com/install.html>)로 이동하여 `setup-x86_64.exe` 라는 설치 파일을 다운로드합니다 (그림 101).

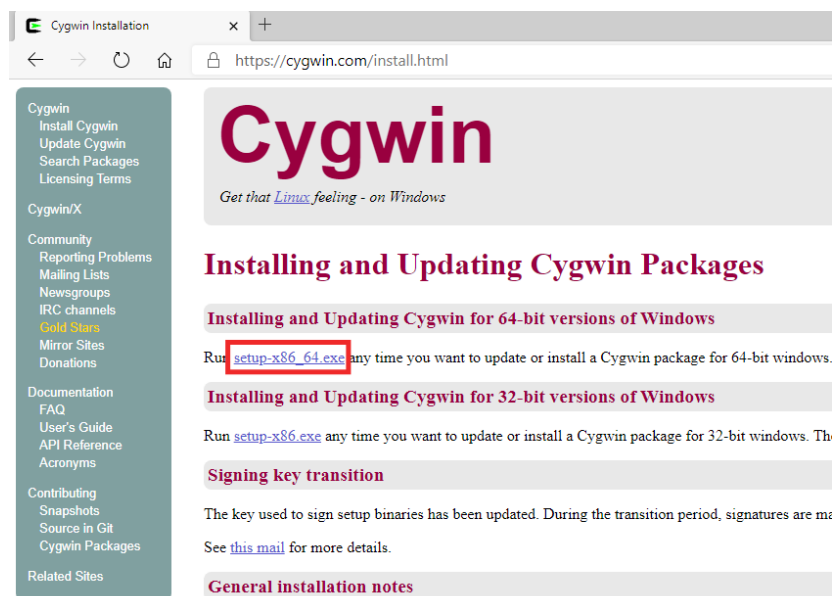


그림 101. Cygwin 설치 웹 페이지

2. 컴퓨터에서 설치 파일을 두 번 클릭하여 실행합니다 (그림 102). 기본 옵션을 유지하면서 다음을 여러 번 클릭합니다. 설치 프로그램은 **다운로드 사이트를 선택하라는** 메시지를 표시합니다 (그림 103). 그중 하나를 선택할 수 있습니다.



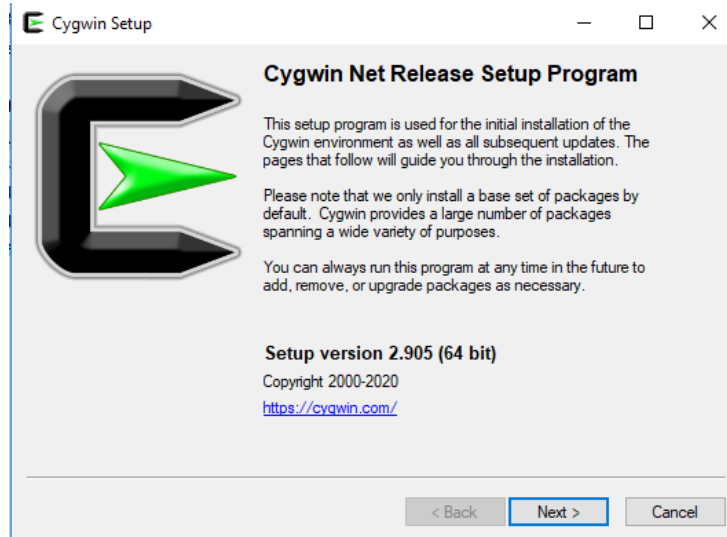


그림 102. Cygwin 설치 창

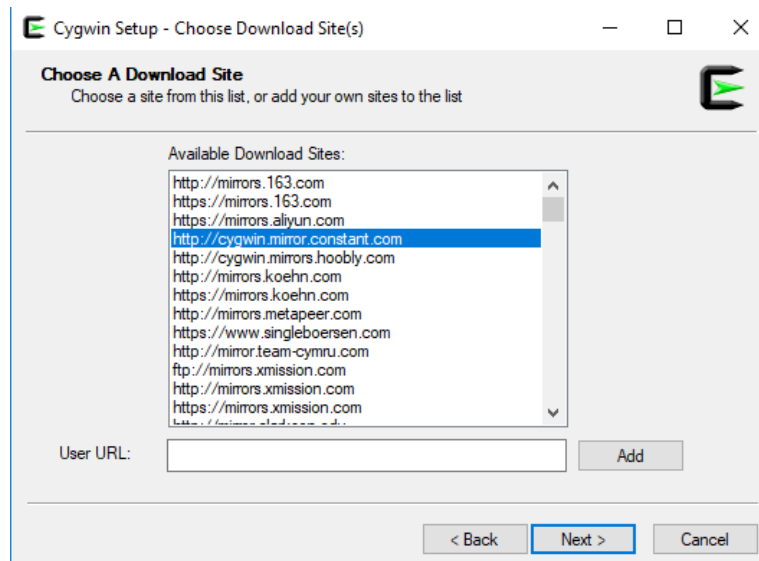


그림 103. 다운로드 사이트 선택

3. 몇 단계를 거쳐 패키지 **선택 창**에 도달합니다 (그림 104). 그림 104 에 표시된 대로 **전체보기**를 선택하십시오.

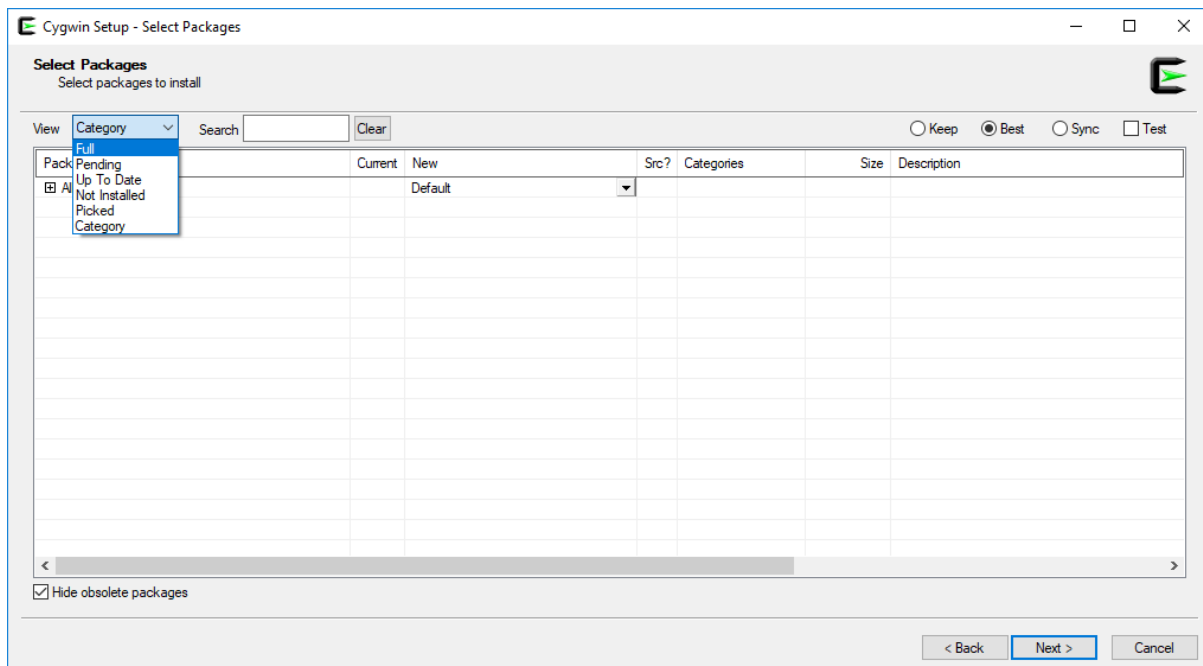


그림 104. 패키지 선택 창

4. 설치할 수 있는 전체 패키지 목록이 나타납니다 (그림 105). 검색 상자에서 설치할 특정 패키지를 선택합니다.

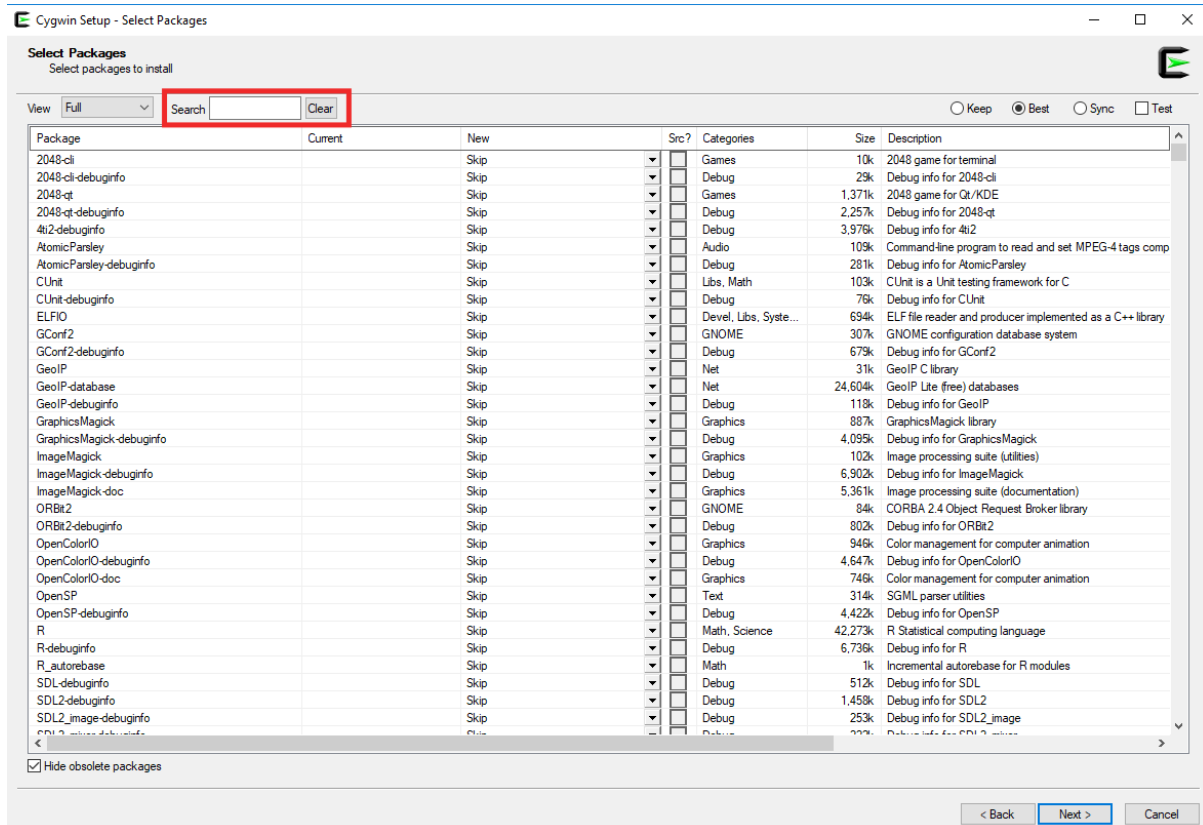


그림 105. 패키지 선택 창 – 전체보기

Verilator 를 컴파일하고 새 시뮬레이터 바이너리를 생성하려면 다음 패키지를 설치해야 합니다.

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

Cygwin 설치에 최소한 이러한 패키지를 포함하십시오. 아래 단계에 따라 하나씩 선택합니다 (목록의 첫 번째 패키지인 git 에 대한 세부 단계만 표시합니다. 프로세스는 다른 패키지에 대해서도 동일합니다).

- 검색 상자에서 git 패키지를 찾습니다 (그림 106).

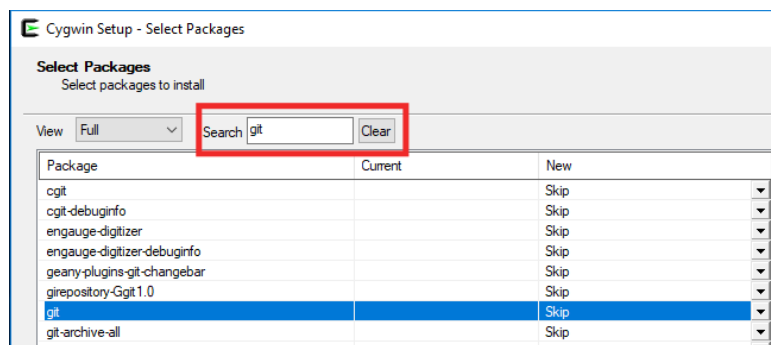


그림 106. git 패키지 찾기

- 드롭 다운 메뉴에서 최신 버전을 선택하고 상자를 선택합니다 (그림 107).



그림 107. 최신 버전을 선택하고 상자를 선택합니다.

- 위 목록의 나머지 패키지에 대해 동일한 작업을 수행합니다.
5. 9 개의 패키지를 선택했다면 다음 창에서 다음을 클릭하여 Cygwin 설치에 이러한 패키지를 포함하고 (설치 프로세스, 그림 108 참조, 몇 분 정도 걸릴 수 있음) Finish 를 클릭하여 설치를 완료합니다 (그림 109).

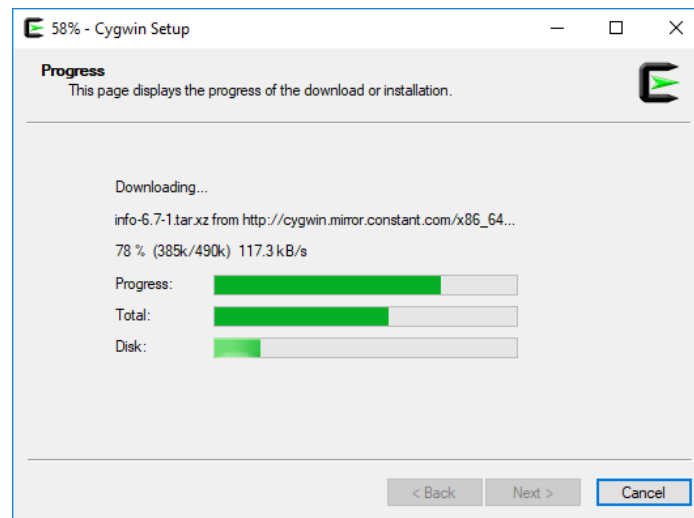


그림 108. Cygwin 설정

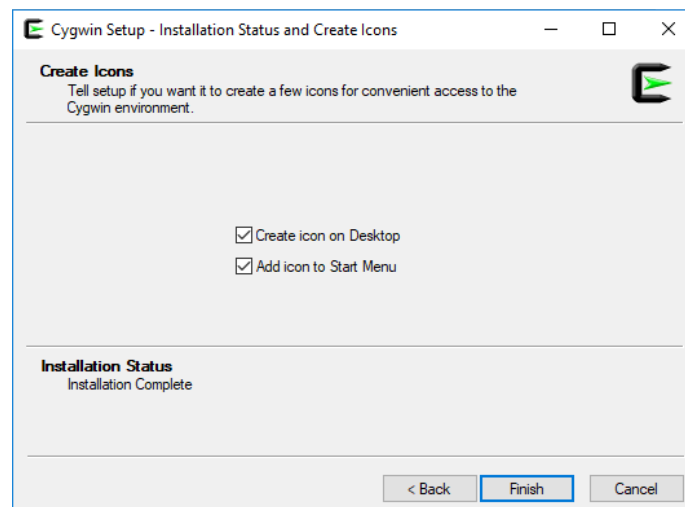


그림 109. 설치 완료

6. Cygwin 설치에 패키지를 추가해야 하는 경우 해당 패키지에 대해 2-5 단계를 반복하십시오.

## Verilator 설치 :

Windows 10 에 Verilator 를 설치하려면 다음 단계를 따르십시오.

1. Windows 바탕 화면이나 시작 메뉴에서 사용할 수 있는 Cygwin 터미널 (그림 110)을 엽니다.



그림 110. Cygwin 터미널

2. 다음 단계에 따라 Verilator 를 빌드하고 설치합니다. 컴퓨터 속도에 따라 몇 시간이 걸릴 수 있습니다.

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

## GTKWave 설치 :

GTKWave 는 <https://sourceforge.net/projects/gtkwave/files/>에서 미리 컴파일된 패키지로 다운로드할 수 있습니다. 최신 Windows 패키지 (이 문서 작성 당시에는 **gtkwave-3.3.100-bin-win64** 라고 함)를 찾아서 다운로드하고 압축을 풀니 다 (압축 해제). 폴더 bin 안에 gtkwave 라는 실행 파일을 찾을 수 있으며, Windows 시스템에서 실행하고 사용할 수 있습니다.

## 부록 D : macOS 에 Verilator 및 GTKWave 설치

이 섹션에서는 macOS 에 Verilator 및 GTKWave 를 설치하는 방법을 설명합니다. 지침은 macOS Catalina 10.15.6 에서 테스트 되었지만 다른 버전의 OS 에서도 작동할 것으로 예상됩니다.

Homebrew (<https://brew.sh/>) 패키지 관리자가 설치에 사용됩니다. macOS 에서 널리 사용되는 다른 패키지 관리자인 MacPorts (<https://www.macports.org/>)에서도 유사한 단계를 찾을 수 있습니다.

### gcc 설치 :

Verilator 를 사용하여 새 시뮬레이터를 빌드하려면 시스템에 컴파일러 도구 체인을 설치해야 합니다. 유효한 컴파일러 도구 모음을 설치하는 방법에는 여러 가지가 있습니다. 아래에서 두 가지를 인용합니다.

1. XCode 명령줄 도구를 설치합니다. 이렇게하면 LLVM 이 설치되지만, 설치 후에 gcc 명령을 사용할 수 있습니다. 이렇게하려면 터미널 창에 다음 명령을 입력하십시오.

- `xcode-select -install`

2. Homebrew 를 사용하여 gcc 를 설치합니다. 다음 명령어를 사용하십시오.

- `brew install gcc@9`

### Verilator 설치 :

Homebrew 와 함께 Verilator 를 설치하는 것은 열린 터미널에 다음 명령을 입력하는 것만큼 간단합니다.

- `brew install verilator`

### gtkwave 설치 :

다시 한 번 Homebrew 를 사용하여 gtkwave 를 설치합니다. 하지만 이번에는 GUI macOS 애플리케이션이기 때문에 cask 를 사용해야 합니다. 열린 터미널에 다음 명령을 입력하십시오.

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

설치 후 *gtkwave.app* 아이콘이 Application 폴더에 나타납니다. 명령 줄에서 사용하려면 Perl 의 스위치 모듈을 설치해야 할 수 있습니다.

- `cpan install Switch`

## 부록 E : Vivado 를 사용하여 FPGA 에 RVfpgaNexys 다운로드

Vivado 를 사용하여 RVfpgaNexys 로 FPGA 를 프로그래밍하려면 다음 단계를 따르십시오.

**WINDOWS:** 다음 단계를 수행하기 전에 Windows 에서 이 부록의 끝에 설명된 대로 드라이버를 Vivado 가 사용하는 드라이버로 되돌려야 합니다 (부록 E).

- a. Nexys A7 보드를 컴퓨터에 연결합니다.
- b. 왼쪽 상단의 스위치를 사용하여 Nexys A7 보드를 켭니다.
- c. Vivado 2019.2 를 엽니다.
- d. Vivado 에서 사용할 수 있고 그림 111 에서 강조 표시된 하드웨어 관리자를 엽니다.

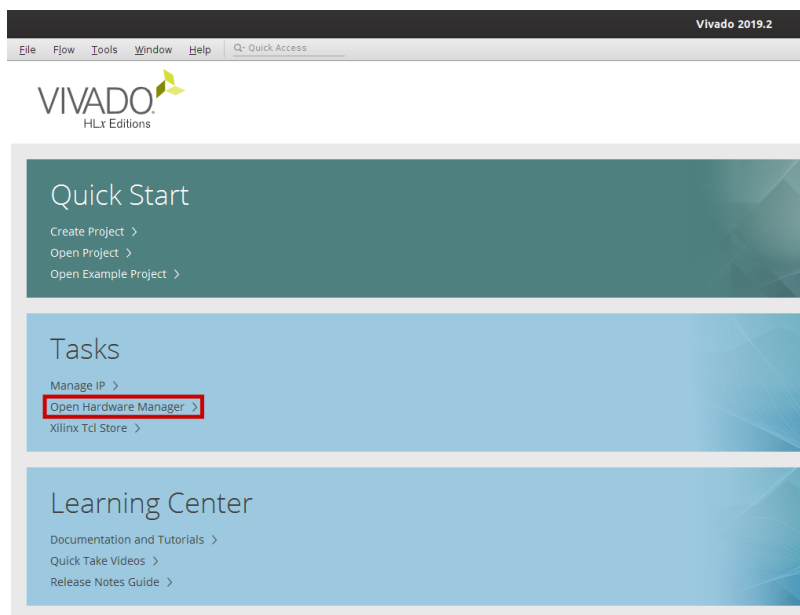


그림 111. 하드웨어 관리자 열기

- e. 하드웨어 관리자가 열리고 열려있는 하드웨어 대상이 없음을 알려줍니다. *대상 열기 (Open target) – Auto connect (자동 연결)*을 클릭하여 대상을 엽니다 (그림 112).

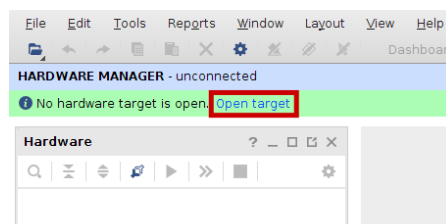


그림 112. Open target

- f. 그림 113 과 같이 Program device 를 선택합니다. 이제 RVfpgaNexys 를 FPGA 에 로드합니다. 새 창에서 `[RVfpgaPath]/RVfpga/src/rvfpganexys.bit` 에서 Bitstream 파일을

선택합니다. 프로그램을 클릭하십시오.

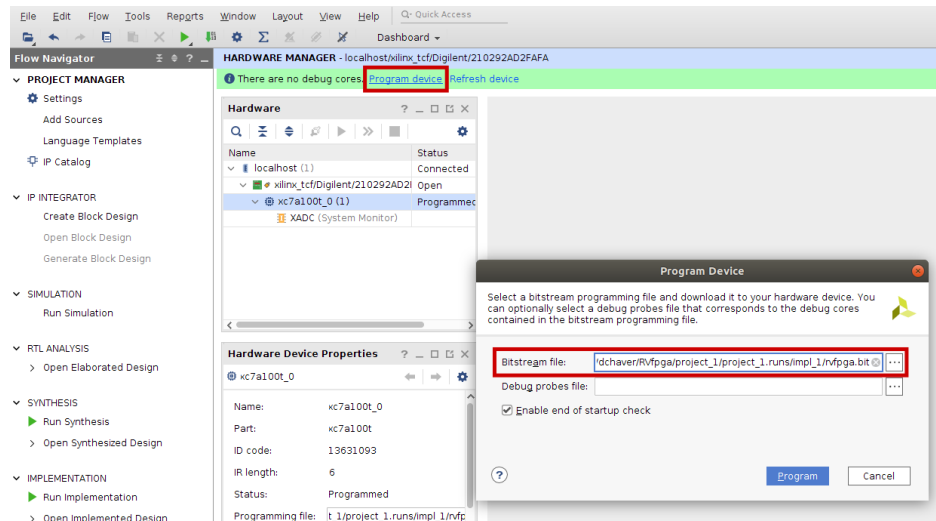


그림 113. 프로그램 장치

- g. 몇 초 후 **FPGA** 는 **FPGA** 를 대상으로하는 **SweRVolfX SoC** 인 **RVfpgaNexys** 로 프로그래밍 됩니다 (그림 25 참조).
- h. 마지막으로 Vivado 의 하드웨어 관리자 창 오른쪽 상단에있는 X 버튼을 클릭하여 하드웨어 관리자를 닫습니다 (그림 114). 그러면 Vivado 가 보드를 해제합니다.

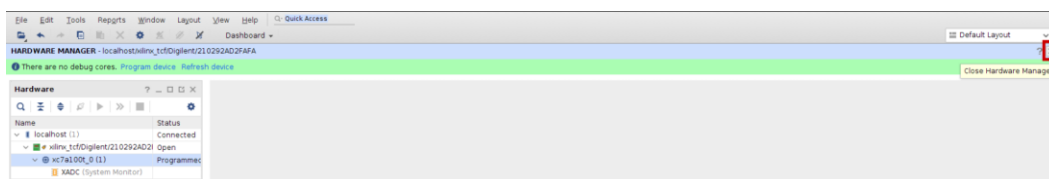
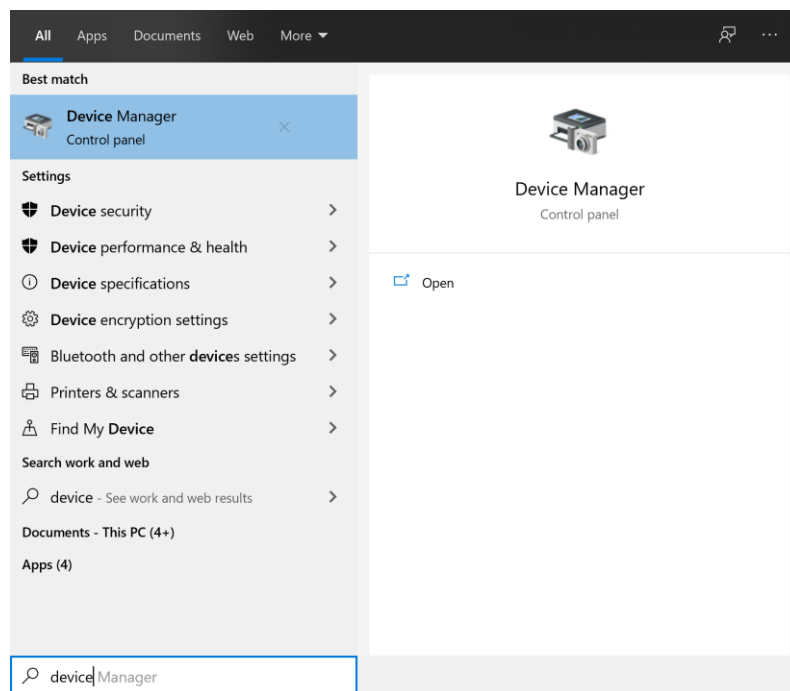


그림 114. 하드웨어 관리자 닫기

## Windows 에서 Vivado 가 사용하는 드라이버로 되돌리는 방법

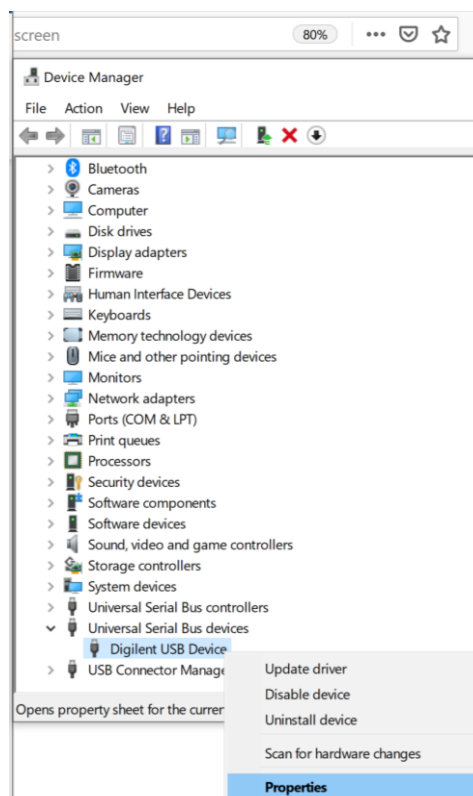
불행히도 Windows 에서 Nexys A7 FPGA 보드 용 드라이버는 Vivado 와 PlatformIO 에서 다릅니다. 이 **GSG** 의 섹션 5.A 에 설명된 대로 **FPGA** 를 프로그래밍하기 위해 **PlatformIO** 를 사용하는 것이 좋습니다. 그러나 Vivado 를 사용하여 비트 파일을 다운로드하려면 부록 B 에서 설치한 드라이버를 Nexys A7 FPGA 보드 용 Vivado (FTDI) 드라이버로 되돌려야 합니다. 이렇게 하려면 시작 메뉴를 클릭하고 검색 상자에 장치 관리자를 입력한 다음 장치 관리자를 클릭하여 장치 관리자를 엽니다 (그림 115 참조).





**그림 115. 장치 관리자 열기**

다음으로 범용 직렬 버스 장치를 확장하고 Digilent USB 장치를 마우스 오른쪽 단추로 클릭한 다음 속성을 선택합니다 (그림 116 참조).



**그림 116. Digilent의 Nexys A7 FPGA 보드에 대한 오픈 드라이버 속성**

속성 창에서 드라이버 탭을 클릭하고 드라이버 롤백을 선택합니다 (그림 117 참조).

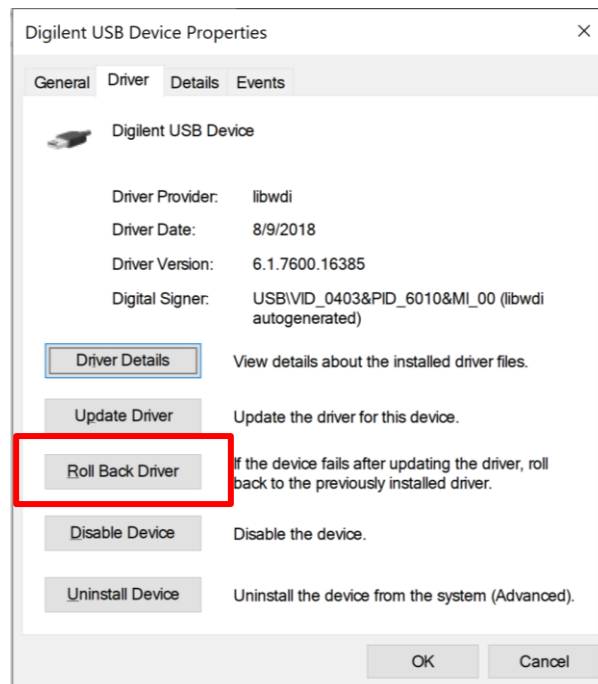


그림 117. 드라이버 롤백

드라이버를 롤백하는 이유를 묻는 창이 나타납니다. 이유를 선택하고 예를 클릭합니다 (그림 118 참조).

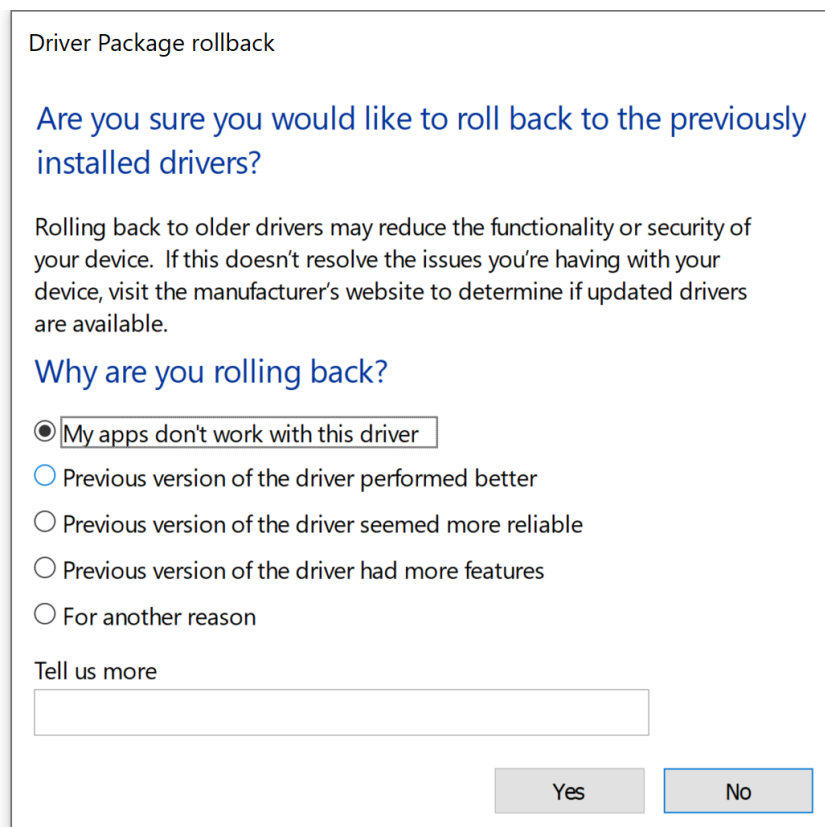
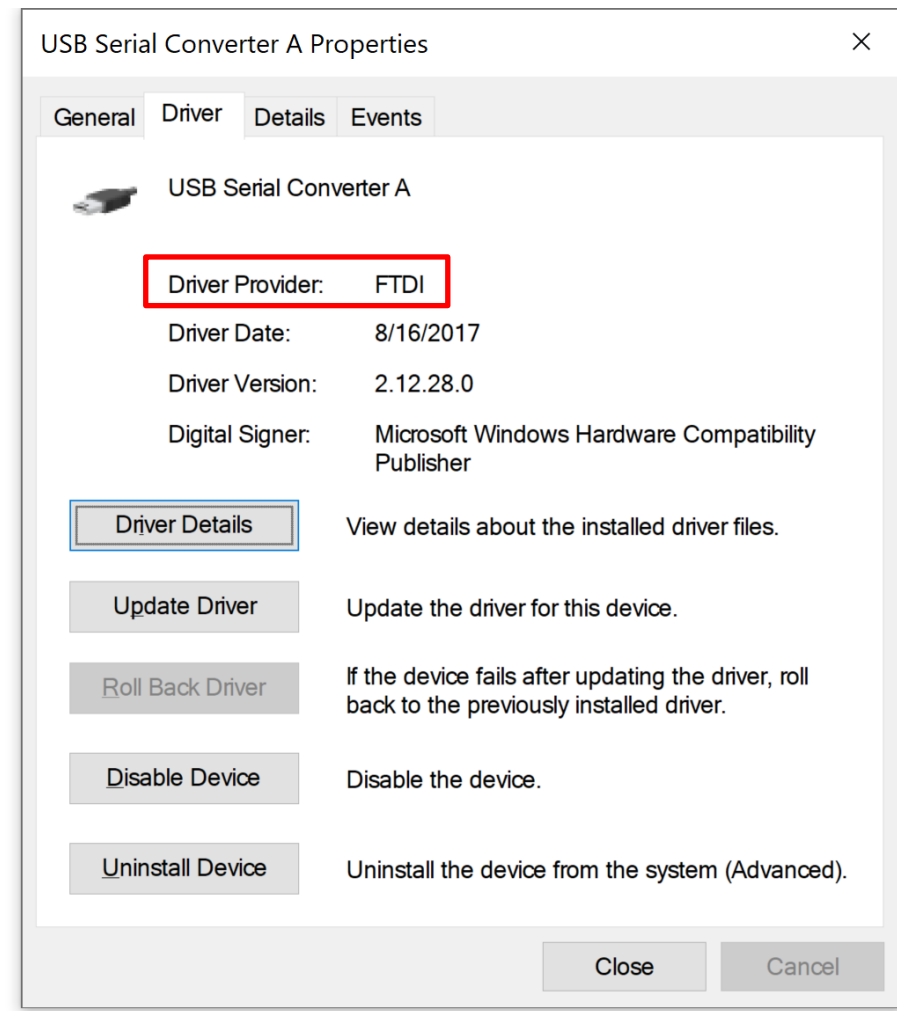


그림 118. 롤백 확인

드라이버가 이전 드라이버로 되돌아 가면 드라이버 공급자가 FTDI 로 나열되어야 합니다 (그림 119 참조).



**그림 119. 제공된 드라이버로 표시된 FTDI 드라이버**

이제 Vivado 를 사용하여 FPGA 보드에 비트 파일을 로드할 수 있습니다. 그러나 여전히 Zadig 를 사용하여 Nexys A7 보드의 드라이버를 교체해야 PlatformIO 가 프로그램을 RVfpgaNexys 에 다운로드할 수 있습니다. 따라서 Vivado 를 사용하는 대신 PlatformIO 를 사용하여 비트 파일을 다운로드하는 것이 좋습니다. 이렇게하면 계속해서 드라이버를 교체할 필요가 없습니다.

## 부록 F : 산업용 IoT 애플리케이션에서 RVfpga 사용

2020 년 7 월 University Complutense of Madrid 의 석사 과정 학생인 Daniel León González 는 "산업용 IoT 를 위한 ad-hoc RISC-V 시스템 온 칩의 FPGA 구현"이라는 제목의 석사 논문을 완료했습니다. 이 작업은 실제 산업용 IoT 애플리케이션에서 RVfpga 를 사용하는 방법을 보여줍니다. 우리는 아래에 프로젝트 초록을 제공하며 전체 논문은 다음에서 볼 수 있습니다.

[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](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).

### 산업용 IoT 를위한 ad-hoc RISC-V 시스템 온 칩의 FPGA 구현

**요약:** IoT 용 노드 장치는 에너지가 효율적이고 비용도 효율적이어야 하지만 많은 시나리오에서 높은 컴퓨팅 성능이 필요하지 않습니다. 이는 대규모 센서 활용과 빠른 속도의 이벤트가 더 많은 처리 능력을 필요로 하는 산업용 IoT 환경에서 크게 변화합니다. 효율적인 프로세서와 고성능 및 기능이 가득한 운영 체제를 사용하는 맞춤형 개발 노드는 이러한 요구 사항의 균형을 맞추고 최적의 솔루션을 제공할 수 있습니다. 이 프로젝트는 RISC-V 프로세서 아키텍처를 기반으로 한 프로토 타입 IoT 노드의 Artix-7 FPGA 를 사용하여 하드웨어 구현을 다룹니다. 이 프로젝트는 구현된 맞춤형 SoC 와 custom border router 주변의 star network 에 배포되는 개념 증명 애플리케이션을 지원하는 데 필요한 Zephyr OS 드라이버의 개발을 보여줍니다.

노드와 ThingSpeak 클라우드 플랫폼 사이에서 종단간 메시지를 보내고 받을 수 있습니다. 이 논문에는 기존 RISC-V 프로세서 구현 분석, 필수 요소 설명, 환경 구성 및 프로젝트 설계에 대한 자세한 가이드가 포함됩니다.