



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga-SoC Lab 3**

## **SweRVolf 및 FuseSoC 소개**

표 1. RVfpga 용어

Name	Description
<b>Courses</b>	
<b>RVfpga</b>	RVfpgaNexys 및 RVfpgaSim, RISC-V SoC(System-on-Chip)를 사용하여 프로그램을 실행하고 주변 장치를 추가하여 시스템을 확장하고(RVfpga Labs 1-10), 시뮬레이션, 성능 측정, 명령 추가 및 메모리 시스템 수정(RVfpga Labs 11-20)을 실행하여 코어 및 메모리 시스템을 탐색하는 방법을 보여주는 과정입니다. 과정 전반에 걸쳐 사용자는 RISC-V 툴체인(컴파일러 및 디버거) 및 시뮬레이터, Verilator HDL 시뮬레이터 및 Western Digital 의 Whisper 명령어 세트 시뮬레이터(ISS)를 사용하는 방법도 보여줍니다.
<b>RVfpga-SoC</b>	SweRV 코어, 메모리 및 주변 장치와 같은 빌딩 블록을 사용하여 처음부터 SweRVolfX SoC 의 하위 집합을 구축하는 방법을 보여주는 과정입니다. 이 과정은 또한 Zephyr 실시간 운영 체제(RTOS)를 SweRVolf 에 로드하고 운영 체제 상단에서 Tensorflow Lite 의 hello world 예제를 포함한 프로그램을 실행하는 방법을 보여줍니다.
<b>코어 및 SoC</b>	
<b>SweRV EH1 Core</b>	Western Digital 에서 개발한 오픈 소스 상용 RISC-V 코어 ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRV EH1 Core Complex</b>	추가된 메모리(ICCM, DCCM 및 명령 캐시), 프로그래밍 가능한 인터럽트 컨트롤러(PIC), 버스 인터페이스 및 디버깅 장치가 있는 SweRV EH1 코어 ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRVolfX</b>	RVfpga 과정에서 사용하는 System on Chip 으로 SweRVolf 의 확장입니다.  <b>SweRVolf</b> ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ): SweRV EH1 Core Complex 를 기반으로 구축된 오픈 소스 SoC 입니다. boot ROM, UART 인터페이스, 시스템 컨트롤러, 상호 연결(AXI Interconnect, Wishbone Interconnect 및 AXI-to-Wishbone 브리지) 및 SPI 컨트롤러를 추가합니다.  <b>SweRVolfX</b> : SweRVolf 에 GPIO, PTC, 추가 SPI 및 8 자리 7-세그먼트 디스플레이 컨트롤러, 4 가지 새로운 주변 장치를 추가합니다.
<b>RVfpgaNexys</b>	Nexys A7 보드 및 주변 장치를 대상으로 하는 SweRVolfX SoC 입니다. DDR2 인터페이스, CDC(클럭 도메인 교차) 장치, BSCAN 로직(JTAG 인터페이스용) 및 클럭 생성기를 추가합니다.  RVfpgaNexys 는 SweRVolf Nexys( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> )와 동일하지만 SweRVolf Nexys 는 SweRVolf 를 기반으로 합니다.

<b>RVfpgaSim</b>	<p>시뮬레이션을 위한 테스트 벤치 래퍼(wrapper) 및 AXI 메모리가 있는 SweRVolfX SoC 입니다.</p> <p>RVfpgaSim 은 SweRVolf Sim(<a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a>)과 동일하지만 SweRVolf Sim 는 SweRVolf 를 기반으로 합니다.</p>
------------------	--

## 1. 소개

이 실습에서는 SweRVolf 를 자세히 분석합니다. 또한 FuseSoC 를 소개하고 이를 사용하여 SweRVolf 를 빌드하는 방법을 보여줍니다(<https://github.com/chipsalliance/Cores-SweRVolf/>).

실습 1(Lab 1)에서는 개별 모듈이 Verilog/SystemVerilog 로 작성되었습니다. 우리는 이 모듈을 연결하여 Vivado 의 Block Design Tool 을 사용하여 SweRVolfX 의 하위 집합을 생성했습니다. 일반 산업계의 디자이너는 모듈을 연결하는 시각적 방법을 사용하거나 정확하게 Verilog/SystemVerilog 코드를 사용하여 디자인 합니다.

다음 표는 블록 설계 접근 방식의 장점과 단점에 대한 간략한 개요를 제공합니다.

**표 2. 블록 디자인의 장단점**

블록 디자인의 장점	블록 설계의 단점
이해하기 쉬움	팀워크가 어려울 수 있음
Learning curve (학습 곡선)	Ecosystem 이 요구. (Vivado Block Design)
복잡한 모듈을 직관적인 방식으로 결합할 수 있습니다.	도구를 업그레이드하면 시스템이 손상될 수 있습니다.
	FPGA 를 넘어 실제 SoC 칩으로 이동하는 데 어려움을 겪을 수 있음
	기계 생성 코드는 읽고, 이해하기 어려울 수 있습니다.
	제조를 위해 SoC 를 미세 조정하거나 더 많은 성능을 위하여 Verilog/SystemVerilog 수준에서 수행해야 합니다.

	IP 핵심 제공업체는 Verilog 소스 코드를 제공하지만 반드시 특정 Vivado 블록 디자인 도구와 함께 작동하는 형식은 아닙니다.
--	---

다음 표는 수동 조정 방식의 장점과 단점에 대한 간략한 개요를 제공합니다.

표 3. 수동 조정 코드의 장단점

손으로 조정한(Hand-tuned) 코드의 장점	손으로 조정한(Hand-tuned) 코드의 단점
FPGA 또는 SoC 제조 프로세스에 매우 구체적으로 수동 조정할 수 있습니다.	Steep learning curve
버전 관리 시스템을 사용할 수 있습니다.	Verilog/SystemVerilog 에 대한 지식이 필요합니다.
팀으로 쉽게 작업하고 협업할 수 있음(다양한 사람들이 서로 다른 모듈에서 작업)	모든 파일/폴더/모듈이 함께 구성되는 방식에 익숙해지는 데 시간이 걸립니다.

SweRVolf 의 모듈(상위 모듈, swervolf\_core, SweRV)은 손으로 작성되었습니다. SweRV CPU 코어 자체는 Western Digital 에서 만들었습니다. 상호 연결 및 기타 모듈은 오픈 소스 라이선스로 출시한 다양한 공급업체의 IP 코어입니다. 상위 모듈은 Olof Kindgrenqns 분이 작성했습니다.

모든 다른 모듈을 연결하고 Verilog 를 손으로 작성하는 것이 가능합니다. 그러나 시간이 지남에 따라 이것은 지루해질 수 있습니다. 특히 더 큰 팀이 함께 작업할 때 코드의 일부를 공유합니다.

SoC 설계 과정에서 여러 대규모 팀이 동시에 함께 작업합니다. 다른 팀은 다른 영역에 초점을 맞추지만 다양한 구성 요소와 IP 블록도 공유합니다.

예를 들어 시뮬레이션 팀은 동일한 IP 코드(SoC+interconnect+CPU+Peripheral)에 관심을 가지지만 이를 명령어 세트 시뮬레이터 또는 Verilator 로 타겟팅하기 위한 래퍼를 사용합니다.

한편 FPGA 팀은 동일한 IP 코드(SoC+Interconnect+CPU+Peripheral)를 사용하지만 주변기기가 다른 다양한 개발 보드에서 테스트를 실행하려고 합니다.

빌드 시스템은 서로 다른 팀이 동일한 IP 코드(SoC+Interconnect+CPU+Peripheral)로 작업하는 것을 용이하게 하지만 서로의 영역을 침범하지는 않습니다.

빌드 시스템은 다양한 팀과 다양한 제약 조건/요구 사항에 적합할 수 있을 만큼 유연하고 동적이어야 합니다. SweRVolf 는 FuseSoC("fuse sock"으로 발음)라는 빌드 시스템을 사용하여 만들어졌습니다.

FuseSoC 빌드 시스템은 개별 빌딩 블록의 하드웨어 설계를 함께 구성합니다.

이 실습에서는 FuseSoC 내에서 SweRVolf의 빌드 버전을 살펴보겠습니다. FuseSoC의 "SweRVolf" 라이브러리를 추가한 다음 시뮬레이션과 보드 구현 모두를 대상으로 하는 단계별 절차를 보여줍니다. 그런 다음 SweRVolf에서 예제 프로그램을 실행합니다.

## 2. 요구사항

이 실습을 완료하려면 다음 도구를 설치해야 합니다.

- Vivado 2019.2 Web Pack (설치가이드(04 페이지) 참조)
- Verilator (V4.106) (설치가이드(08 페이지) 참조)
- GTKWave (설치가이드(08 페이지) 참조)
- FuseSoC (설치가이드(09 페이지) 참조)
- OpenOCD (RISC-V-specific version) (설치가이드(09 페이지) 참조)

**중요:** RVfpga-SoC 실습들을 시작하기 전에 RVfpga-SoC 설치 가이드를 완료하는 것이 좋습니다.

예를 들어, 아직 설치하지 않았다면 RVfpga-SoC 설치 가이드의 지침에 따라 Verilator와 Xilinx의 Vivado를 설치하십시오. Imagination의 대학 프로그램(IUP)에서 다운로드한 RVfpga-SoC 폴더를 컴퓨터에 복사했는지 확인합니다.

## 3. FuseSoC란?

FuseSoC는 수상 경력에 빛나는 패키지 관리자이며, HDL(Hardware Description Language) 코드를 빌드하는 툴 세트입니다. 주요 목적은 IP(지적 재산권) 코어의 재사용을 늘리고 SoC 솔루션을 생성, 구축 및 시뮬레이션하는 것입니다.

FuseSoC의 기본 엔터티는 코어입니다. 코어는 로컬 또는 원격 위치에서 FuseSoC 패키지 관리자에 의해 검색되고 빌드 시스템에 의해 전체 하드웨어 설계로 결합될 수 있습니다.

FuseSoC 코어는 반드시 프로세서일 필요는 없지만 FIFO 구현과 같이 합리적으로 자체 포함되고 재사용 가능한 IP 부분입니다. FuseSoC는 이를 패키지라고도 합니다. 다른 시스템에서는 이러한 재사용 가능한 하드웨어 조각을 모듈이라고 합니다.

FuseSoC에 대한 자세한 내용은

<https://fusesoc.readthedocs.io/en/latest/user/overview.html#understanding-fusesoc>에서 설명서를 읽을 수 있습니다.

FuseSoC는 "SweRVolf"(SweRV RISC-V 코어를 FuseSoC 기반 SoC)를 라이브러리에 패키지로 제공합니다. FuseSoC를 사용하여 "**swervolf**" 라이브러리를 추가한 다음 "sim"(시뮬레이션) 또는 "Nexys\_a7"(보드)과 같은 특정 대상에 대해 빌드할 수 있습니다.

#### 4. SweRVolf : SweRV Eh1 용 FuseSoC 기반 SoC

SweRVolf 는 SweRV RISC-V 코어용 FuseSoC 기반 SoC 입니다. SweRVolf 는 시뮬레이터나 FPGA 보드에서 RISC-V 적합성 테스트, Zephyr OS 또는 기타 소프트웨어를 실행할 수 있습니다. FuseSoC 는 이식성, 확장성 및 사용 용이성을 지원하고 향상시키는 것을 목표로 합니다. SweRV 사용자가 소프트웨어를 빠르게 실행할 수 있도록 하려면 SoC 를 필요에 맞게 수정하거나 새 대상 장치에 이식할 수 있습니다.

그림 1 은 SweRVolf 의 상위 수준 블록 다이어그램을 보여줍니다.

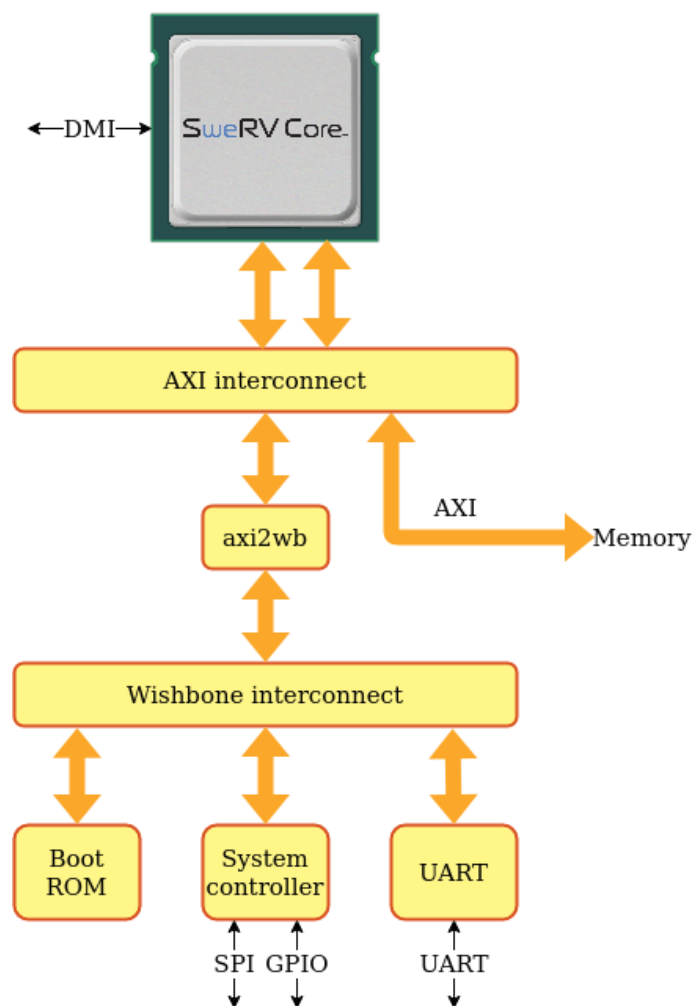


그림 1. SweRVolf 코어

SweRVolf 의 코어는 Boot ROM 이 있는 SweRV CPU, AXI4 상호 연결, UART, SPI, RISC-V 타이머 및 GPIO 로 구성됩니다. 코어는 RAM 을 포함하지 않지만, 대신 대상 특정 래퍼가 적절한 메모리 컨트롤러에 연결할 메모리 버스를 제공합니다. 다른 외부 연결로는 클럭, 리셋, UART, GPIO, SPI 및 DMI(디버그 모듈 인터페이스)가 있습니다.

표 4 는 Wishbone 인터커넥트 및 AXI 인터커넥트를 통해 SweRV EH1 코어에 연결된 주변 장치의 메모리 매핑된 주소를 제공합니다.

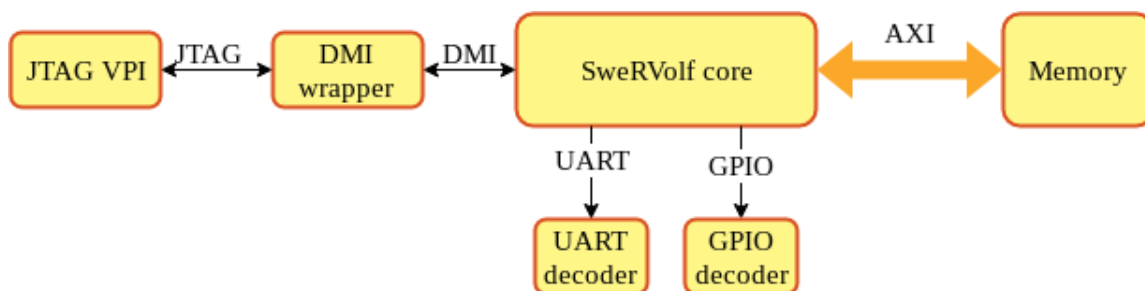
**표 4. SweRVolf 의 MEMORY-MAPPED 주소**

System	Address
RAM	0x00000000-0x07FFFFFF
Boot ROM	0x80000000-0x80000FFF
System Controller	0x80001000-0x80001FFF
UART	0x80002000-0x80002FFF
GPIO	0x80001010-0x80001013

실습 2 의 SystemVerilog 모듈 RVfpgaSim 및 RVfpgaNexys 와 유사하게 SweRVolf 는 SweRVolf 의 두 가지 버전인 시뮬레이션용 SweRVolf Sim 과 Digilent Nexys A7 보드용 SweRVolf Nexys 도 제공합니다.

## 1. SweRVolf 시뮬레이션

SweRVolf Sim 은 Verilator 또는 QuestaSim 과 같은 기타 이벤트 구동 시뮬레이터에서 사용할 테스트벤치에 SweRVolf 코어를 래핑하는 시뮬레이션 대상입니다. SweRV 프로세서에서 실행되는 프로그램을 실행하는 전체 시스템 시뮬레이션에 사용할 수 있습니다. 또한 OpenOCD 및 JTAG VPI 를 통한 디버거 연결을 지원합니다(그림 2 참조).



**그림 2. SweRVolf 시뮬레이션**

## 2. SweRVolf Nexys

SweRVolf Nexys 는 Digilent Nexys A7 보드를 대상으로 하는 SweRVolf SoC 버전입니다. RAM 용으로 온보드 128MB DDR2 를 사용하고, GPIO 가 LED 에 연결되어 있으며, SPI 플래시에서 부팅을 지원하며, UART 및 JTAG 통신을 위해 마이크로 USB 포트를 사용합니다. SweRVolf Nexys 대상의 기본 부트로더는 기본적으로 SPI Flash 에 저장된 프로그램을 로드하려고 시도합니다(그림 3 참조).

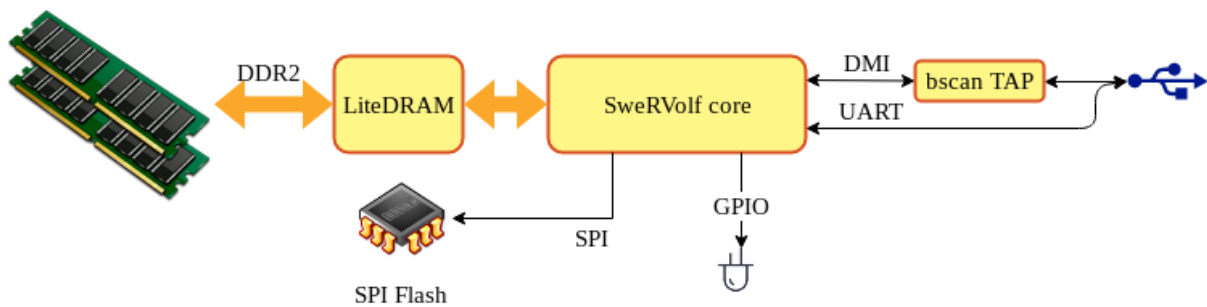


그림 3. SweRVolf Nexys

SweRVolf SoC 는 시뮬레이션 또는 하드웨어, 즉 Digilent Nexys A7 보드에서 실행할 수 있습니다. 두 경우 모두 FuseSoC 를 사용하여 시뮬레이션 또는 빌드를 시작하거나, FPGA build 를 실행할 수 있습니다.

이전 실습의 RVfpgaSim 과 마찬가지로 SweRVolf Sim 은 SweRVolf 코어를 Verilator 에서 사용할 테스트벤치로 래핑하는 시뮬레이션 대상입니다. 마찬가지로 이전 실습의 RVfpgaNexys 는 Digilent Nexys A7 보드를 대상으로 하는 SweRVolf SoC 버전인 SweRVolf Nexys 와 유사합니다.

## 5. 환경설정

이 섹션에서는 Sim 빌드 및 Nexys 빌드를 위한 환경을 설정하는 방법을 보여줍니다.

**Step 1.** 프로젝트의 루트로 사용할 **"SweRVolf"** 디렉토리로 이동합니다.

이 디렉토리는 지금부터 **\$WORKSPACE** 라고 불립니다. 달리 명시되지 않는 한 모든 추가 명령은 \$WORKSPACE 에서 실행됩니다. 작업공간 디렉토리 입력 후 실행합니다.

```
> export WORKSPACE=$(pwd)
```

\$WORKSPACE 쉘 변수를 설정하려면(그림 4 참조) 다음 명령을 사용하여 쉘 변수가 성공적으로 추가되었는지 확인할 수 있습니다.

```
> printenv WORKSPACE
```

```
hamza@imagination:~$ cd RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export WORKSPACE=$(pwd)
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ printenv WORKSPACE
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

그림 4. 작업 공간 설정

**2 단계.** FuseSoC 가 이미 설치되어 있는지 확인합니다. FuseSoC 가 없는 경우, 다음을 실행하여 설치합니다.

```
> sudo pip3 install fusesoc
```



**3 단계.** FuseSoC 기본 라이브러리를 작업 공간에 추가합니다.

```
> fusesoc library add fusesoc-cores
https://github.com/fusesoc/fusesoc-cores
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add fusesoc-cores https://github.com/fusesoc/fusesoc-cores
INFO: Cloning library into fusesoc_libraries/fusesoc-cores
Cloning into 'fusesoc_libraries/fusesoc-cores'...
remote: Enumerating objects: 202, done.
remote: Counting objects: 100% (202/202), done.
remote: Compressing objects: 100% (140/140), done.
remote: Total 651 (delta 85), reused 163 (delta 50), pack-reused 449
Receiving objects: 100% (651/651), 136.59 KiB | 81.00 KiB/s, done.
Resolving deltas: 100% (229/229), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

그림 5. FuseSoC 기본 라이브러리 추가

**4 단계.** swervolf 라이브러리를 추가합니다.

```
> fusesoc library add swervolf
https://github.com/chipsalliance/Cores-SweRVolf
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf
INFO: Cloning library into fusesoc_libraries/swervolf
Cloning into 'fusesoc_libraries/swervolf'...
remote: Enumerating objects: 130, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 1035 (delta 52), reused 89 (delta 42), pack-reused 905
Receiving objects: 100% (1035/1035), 1.17 MiB | 206.00 KiB/s, done.
Resolving deltas: 100% (580/580), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

그림 6. Swervolf 라이브러리 추가

**5 단계.** swervolf 디렉토리를 "SWERVOLF\_ROOT" 셸 변수로 설정합니다.

```
> export
SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

그림 7. 셸 변수 설정

## 6. 시뮬레이션 빌드

이 섹션에서는 다음 단계인 FuseSoC 를 사용하여 SweRVolf Sim 을 구축하는 방법을 보여줍니다.

실행할 항목을 선택하려면 --target 매개변수와 함께 "fusesoc run" 명령을 사용하십시오.

**6 단계.** 시뮬레이션에서 실행하려면

```
> fusesoc run --target=sim swervolf
```

이 명령은 다음 경로 내에 "**Vswervolf\_core\_tb**"라는 시뮬레이션 바이너리를 생성합니다.

*SweRVolf/build/swervolf\_0.7.3/sim-verilator/*

이는 *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM* 경로에서 make 명령을 사용하여 "**Vrvfpgasim**" 시뮬레이션 바이너리를 생성한, Lab 2 에서 수행한 작업과 유사합니다.

그러나 Lab 2 에서 사용된 SoC 는 SweRVolfX 의 하위 집합인 반면 여기에서 사용된 SoC 는 완전한 SweRVolf 입니다.

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
WARNING: Unknown item compilation_mode in section Xsim
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing ::jtag_vpi:0-r5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
=====
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
=====
INFO: Setting up project
=====
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from hello.vh
Releasing reset
SweRV+FuseSoC rocks

Finito
- ./src/swervolf_0.7.3/rtl/swervolf_syscon.v:151: Verilog $finish
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

## 그림 8. SweRVolf Sim 빌드

이 명령은 Workspace 디렉터리에 다음 계층을 생성합니다.

- \$WORKSPACE
  - fusesoc\_libraries
  - build
    - swervolf\_0.7.3

- `sim-verilator`
  - `Vswervolf_core_tb`
- `src`

## 7. Nexys A7 빌드

이 섹션에서는 FuseSoC 를 사용하여 SweRVolf Nexys 를 구축하는 방법을 보여줍니다.

**1 단계.** Nexys A7 보드용 이미지를 빌드(및 선택적으로 프로그래밍)하려면 다음을 실행합니다.

```
➤ fusesoc run --target=nexys_a7 swervolf
```

**참고:** 이 명령을 실행하기 전에 Nexys A7 보드를 컴퓨터에 연결하고 보드가 켜져 있는지 확인하십시오.

이 명령은 다음 경로 내에서 "**swervolf\_0.7.3.bit**"라는 FPGA(".bit"로 끝남)를 프로그래밍하는데 사용되는 bitstream 파일을 생성합니다.

*SweRVolf/build/swervolf\_0.7.3/nexys\_a7-vivado/*

또한 우리 컴퓨터에 연결된 Nexys A7 보드에 비트스트림을 업로드합니다.

이것은 "Generate Bitstream" 옵션을 사용하여 블록 디자인을 생성한 후 Vivado 에서 "**rvfpga.bit**" 파일을 생성할 때 Lab 1 에서 했던 것과 유사합니다. 나중에 Lab 2 에서 PlatformIO 를 사용하여 Nexys A7 보드에 "**rvfpga.bit**" 파일을 업로드했습니다.

**FuseSoC** 는 위에서 언급한 명령을 사용하여 Lab 1 및 Lab 2 에서 수동으로 수행했던 이 모든 작업을 수행합니다.

```

hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=nexys_a7 swervolf
WARNING: Unknown item compilation_mode in section Xsim
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::serv:1.0.2
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing swervolf:litedram:0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
=====
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
=====
INFO: Setting up project

```

```

.....
FuseSoC Xilinx FPGA Programming Tool
=====
INFO: Programming part xc7a100tcs324-1 with bitstream swervolf_0.7.3.bit
INFO: [Labtools 27-2285] Connecting to hw_server url TCP:localhost:3121
INFO: [Labtools 27-2222] Launching hw_server...
INFO: [Labtools 27-2221] Launch Output:

***** Xilinx hw_server v2019.2
**** Build date : Nov 6 2019 at 22:13:42
** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: [Labtools 27-3415] Connecting to cs_server url TCP:localhost:3042
INFO: [Labtools 27-3417] Launching cs_server...
INFO: [Labtools 27-2221] Launch Output:

***** Xilinx cs_server v2019.2.0
**** Build date : Nov 07 2019-10:41:48
** Copyright 2017-2019 Xilinx, Inc. All Rights Reserved.

INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcs324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A

INFO: SUCCESS! FPGA xc7a100tcs324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$

```

그림 9. SweRVolf Nexys 빌드

앞에서 언급한 대로 명령이 완료되면 Workspace 디렉터리는 다음과 같이 표시됩니다.

- \$WORKSPACE
  - fusesoc\_libraries

- build
  - swervolf\_0.7.3
    - sim-verilator
    - nexys\_a7-vivado
      - **swervolf\_0.7.3.bit**
    - src

FuseSoC 는 **Vivado** 를 사용하여 프로젝트를 생성하고, 전역 변수를 설정하고, 제약 파일(constraints files)을 추가하고, 비트스트림을 자동으로 생성합니다.

Lab 1 에서 본 것처럼 계층 구조를 시각화하기 위해 Vivado 프로젝트를 열 수 있습니다.

**2 단계.** 다음 명령으로 "**nexys\_a7-vivado**" 디렉토리를 입력합니다.

```
➤ cd build/swervolf_0.7.3/nexys_a7-vivado/
```

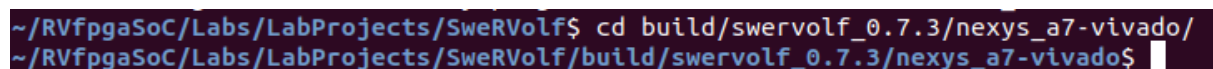


그림 10. "**nexys\_a7-vivado**" 디렉토리로 이동

**3 단계.** 다음 명령을 입력하여 Vivado 프로젝트를 엽니다.

```
➤ vivado swervolf_0.7.3.xpr
```

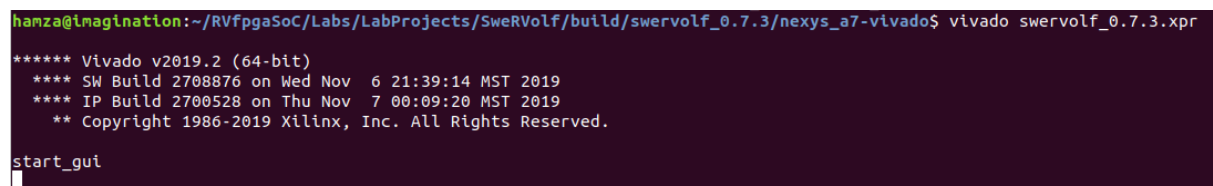


그림 11. Vivado 프로젝트 열기

소스 패널에 설정된 전역 변수 및 constraints files 을 시각화할 수 있습니다.

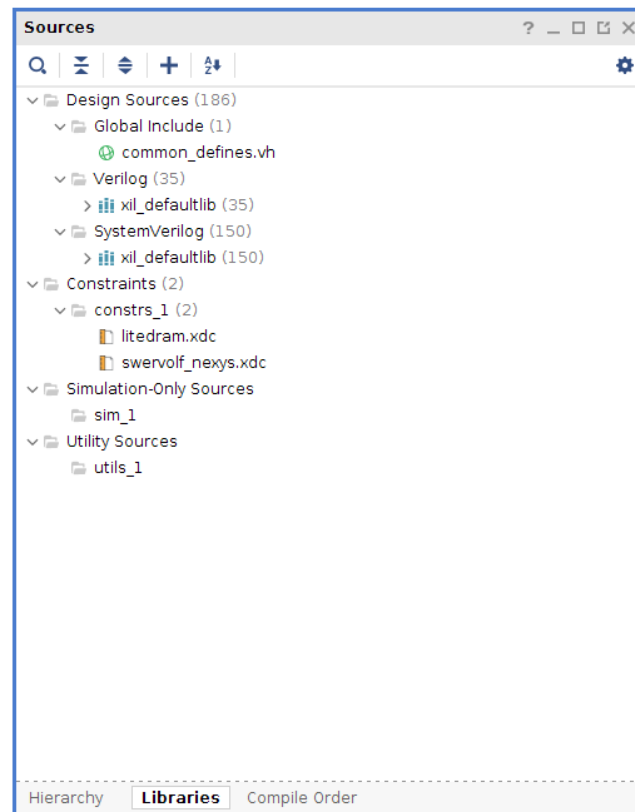


그림 12. 소스 패널

"Design Runs" 탭에서 합성 및 구현이 이미 성공적으로 완료된 것을 볼 수 있습니다.

Tcl Console   Messages   Log   Reports   Design Runs x															
Q   Z   S   I   <<   >>   +   %															
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	
✓ synth_1	constrs_1	synth_design Complete!								34263	18537	44.0	0	4	
✓ impl_1	constrs_1	write_bitstream Complete!	0.453	0.000	0.052	0.000	0.000	0.929	0	33663	18538	44.0	0	4	

그림 13. 설계 실행

## 8. SweRVolf Sim 에서 AL\_Operations 예제 실행

이 섹션에서는 Verilator 를 사용하여 SweRVolf Sim 에서 AL\_Operations 프로그램을 실행하는 방법을 보여줍니다.

**1 단계.** 경로가 다음과 같은 examples 디렉토리를 입력합니다.

`[RvfpgaSoCPath]/RvfpgaSoC/Labs/LabResources/Lab3/examples/  
AL_Operations/CommandLine/`

```
> cd ../../LabResources/Lab3/examples/AL_Operations/commandLine/
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd ../../LabResources/Lab3/examples/AL_Operations/commandLine/
~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

그림 14. AL\_Operations 디렉토리

2 단계. 시뮬레이션을 위한 16 진수 프로그램 생성:

```
> make clean
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make clean
rm -f *.elf *.bin *.vh *.dis *.mem *.vcd
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

그림 15. Make clean

```
> make AL_Operations.vh
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make AL_Operations.vh
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-gcc -nostartfiles -march=rv32i -mabi=ilp32 -
Tlink.ld -oAL_Operations.elf AL_Operations.S
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-objcopy -O binary AL_Operations.elf AL_Opera
tions.bin
python3 makehex.py AL_Operations.bin > AL_Operations.vh
rm AL_Operations.bin AL_Operations.elf
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

그림 16. AL\_Operations.vh 파일 생성

3 단계. 시뮬레이터 실행:

```
> ../../../../../../LabProjects/SweRVolf/build/swervolf_0.7.3/sim-
verilator/Vswervolf_core_tb +ram_init_file=AL_Operations.vh
+vcd=1
```

"ram\_init\_file" 매개변수는 초기 on-chip RAM 내용으로 사용할 Verilog 16 진 파일을 로드합니다. 따라서 해당 옵션(+ram\_init\_file)을 사용하여 방금 생성한 Verilog 16 진 파일 "**AL\_Operations.vh**"를 로드하고, 시뮬레이션 바이너리 "**Vswervolf\_core\_tb**"를 실행합니다. 또한 vcd(value change dump)를 1로 설정하여 모듈 내의 모든 변수를 덤프합니다.

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ ../../../../../../LabProjects/SweRVolf
/build/swervolf_0.7.3/sim-verilator/Vswervolf_core_tb +ram_init_file=AL_Operations.vh +vcd=1
Loading RAM contents from AL_Operations.vh
Releasing reset
```

그림 17. 시뮬레이터 실행

시뮬레이션을 중지하려면 "ctrl + c"를 누르십시오. "**trace.vcd**" 파일이 생성되어야 합니다.

4 단계. 추적 파일을 엽니다.

```
> gtkwave trace.vcd
```

```

hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ gtkwave trace.vcd
GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

Warning! File size is 758 MB. This might fail in recoding.
Consider converting it to the FST database format instead. (See the
vcd2fst(1) manpage for more information.)
To disable this warning, set rc variable vcd_warning_filesize to zero.
Alternatively, use the -o, --optimize command line option to convert to FST
or the -g, --giga command line option to use dynamically compressed memory.

[0] start time.
[2481540] end time.

```

그림 18. gtkwave 에서 trace.vcd 파일 열기

이제 **gtkwave** 가 실행됩니다. 이제 Lab 2 의 과정을 반복하여 그래프에 신호를 추가하고 분석해야 합니다.

**5 단계.** *GTKWave* 의 왼쪽 상단 창에서 SoC 계층을 확장하여 그래프에 신호를 추가할 수 있습니다. 계층 구조를 **TOP** → **swervolf\_core\_tb** → **swervolf** → **rvtop** → **swerv** 로 확장하고 모듈 **ifu** 를 클릭하고 signal *clk*(코어에 사용되는 클럭)를 선택하고 오른쪽의 흰색 Signals 창 또는 검은색 Waves 창으로 끌어다 놓습니다..

**6 단계.** **ifu** 모듈이 강조 표시되었는지 확인하고(그림 19 참조) 필터 검색에 "inst"를 입력한 다음 "ifu\_i0\_instr[31:0]" 및 "ifu\_i1\_instr[31:0]" 신호를 삽입합니다.

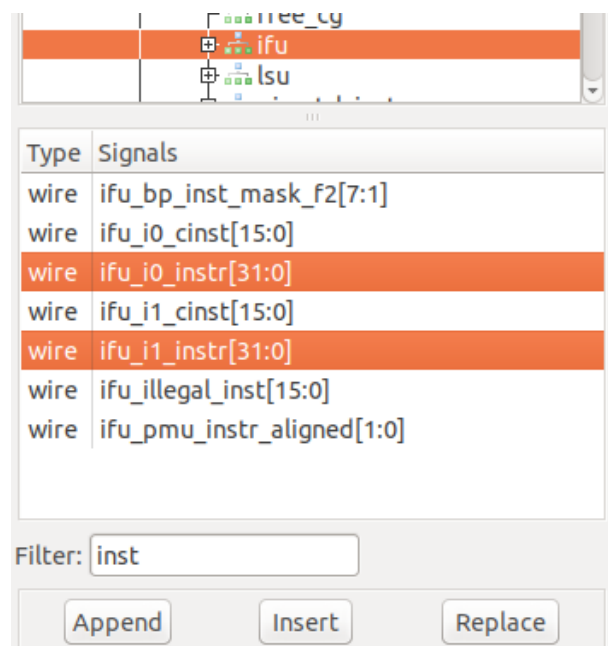
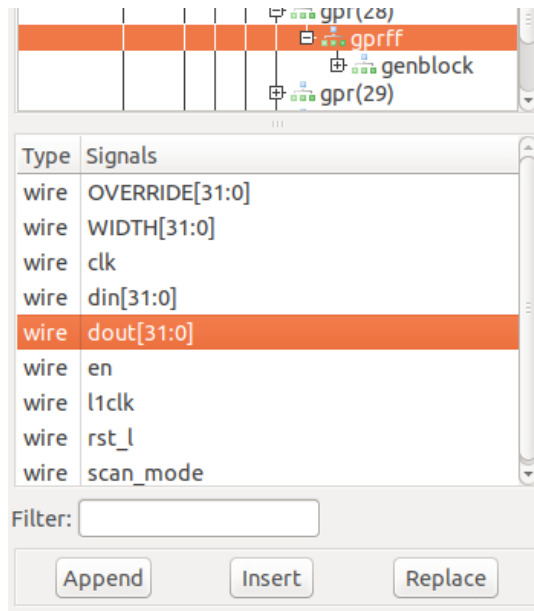


그림 19. "instr" 신호

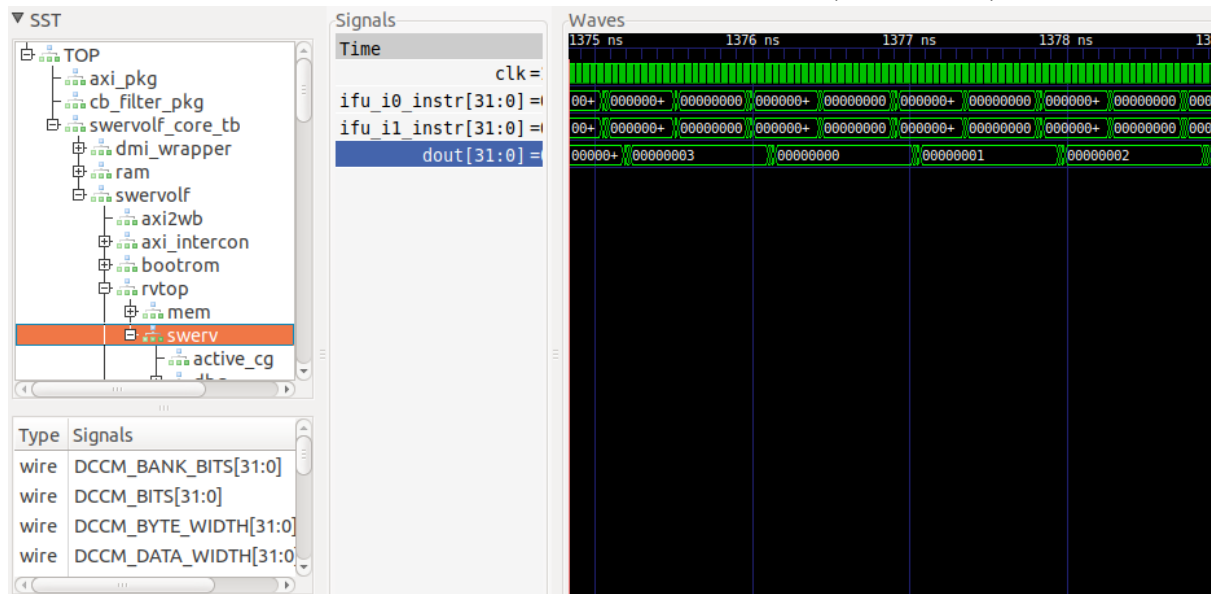
**7 단계.** 이제 **TOP** → **swervolf\_core\_tb** → **swervolf** → **rvtop** → **swerv** → **dec** → **arf** → **gpr\_banks(0)** → **gpr(28)** → **gprff** 로 이동하고 "dout[31:0]" 신호를 클릭하고 이를 삽입합니다. (그림 20 참조).





### 그림 20. "dout" 신호

이제 "+" 버튼을 사용하여 신호를 확대하고 파형을 분석할 수 있습니다(그림 21 참조).



## 그림 21. 신호 분석

## 9. SweRVolf Nexys 에서 Blinky 예제 실행

이 섹션에서는 SweRVolf Nexys 에서 Blinky 프로그램을 실행하는 방법을 보여줍니다. 이 실습의 앞부분인 FuseSoC 에서 생성한 ".bit" 파일을 사용할 것입니다.

**참고:** 실습 1 및 실습 2 에서 사용된 SweRVofX 모듈용 LED 및 I/O 의 주소 메모리 맵은 SweRVofX(SweRV EH1 용 FuseSoC 기반 SoC)와 다릅니다. 따라서 실습 2 와 실습 3 에는 메모리

주소가 변경된 동일한 예제 프로그램(Blinky)의 별도 예제 디렉토리가 있습니다.

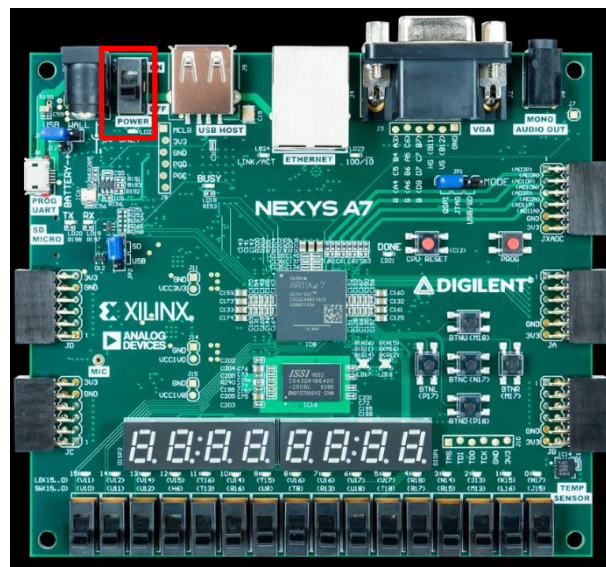
**표 5. SweRVolf 및 SweRVolfX 의 MEMORY-MAPPED GPIO 주소**

SoC	System	Address
SweRVolfX (Lab 1)	GPIO	0x80001400 - 0x8000143F
SweRVolf (Lab 3)	GPIO	0x80001010 - 0x80001013

다음 단계를 완료하여 SweRVolf Nexys 로 Nexys A7 보드를 프로그래밍한 다음 Blinky 프로그램을 실행하십시오:

**1 단계.** Nexys A7 보드를 컴퓨터에 연결합니다.

**2 단계.** 왼쪽 상단의 스위치를 사용하여 Nexys A7 보드를 켭니다. (그림 22 참조)



**그림 22. Nexys A7 보드 ON/OFF 버튼**

**3 단계.** VSCode 및 PlatformIO 가 아직 열려 있지 않은 경우 엽니다.

**4 단계.** 상단 메뉴 모음에서 파일 → 폴더 열기(그림 23 참조)를 클릭하고 `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab3/examples/` 디렉토리로 이동합니다.

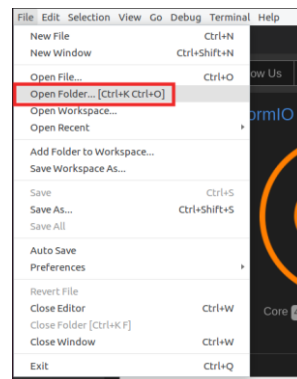


그림 23. 폴더 열기

**5 단계.** *Blinky\_FuseSoC* 디렉토리를 선택하고(열지 말고 선택만 하십시오) 창 상단의 확인을 클릭하십시오. 이제 PlatformIO 가 예제를 엽니다.

**6 단계.** 왼쪽 사이드바에서 *platformio.ini* 를 클릭하여 *platformio.ini* 파일을 엽니다(그림 24 참조). 다음 줄을 편집하여 시스템에서 RVfpga 비트스트림의 경로를 설정합니다(그림 24 참조).

**7 단계.** FuseSoC 를 사용하여 생성된 "**swervolf\_0.7.3.bit**" 파일은 다음 경로에 있습니다.

```
board_build.bitstream_file =
/home/<Username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0
.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
```

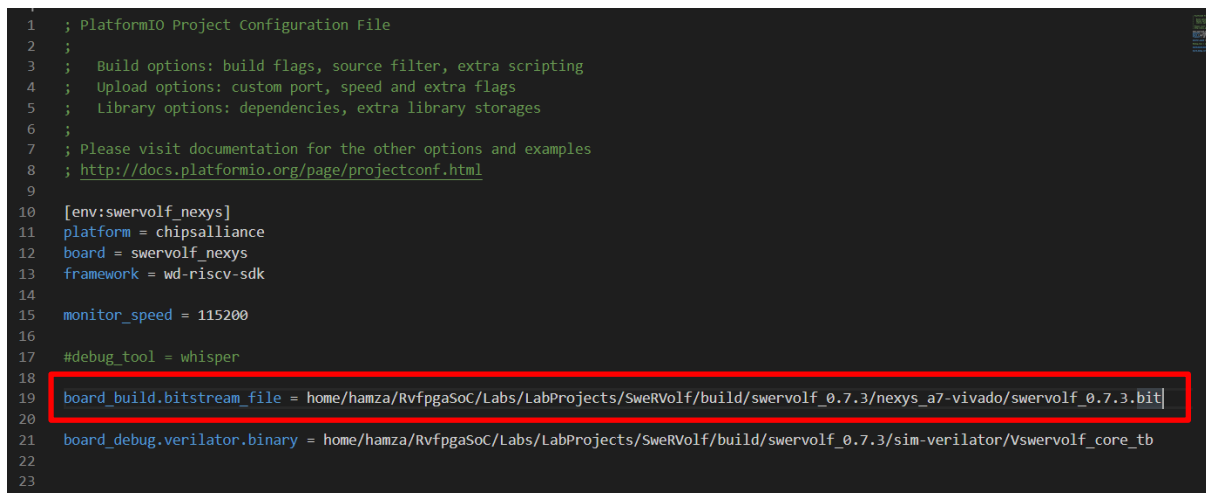


그림 24. Platformio 초기화 파일: platformio.ini



**8 단계.** 왼쪽 메뉴 리본에서 PlatformIO 아이콘  을 클릭합니다(그림 25 참조).



그림 25. PlatformIO 아이콘

프로젝트 작업 창이 비어 있는 경우(그림 26) 먼저  을 클릭하여 프로젝트 작업을 새로 고쳐야 합니다. 몇 분이 소요될 수 있습니다.

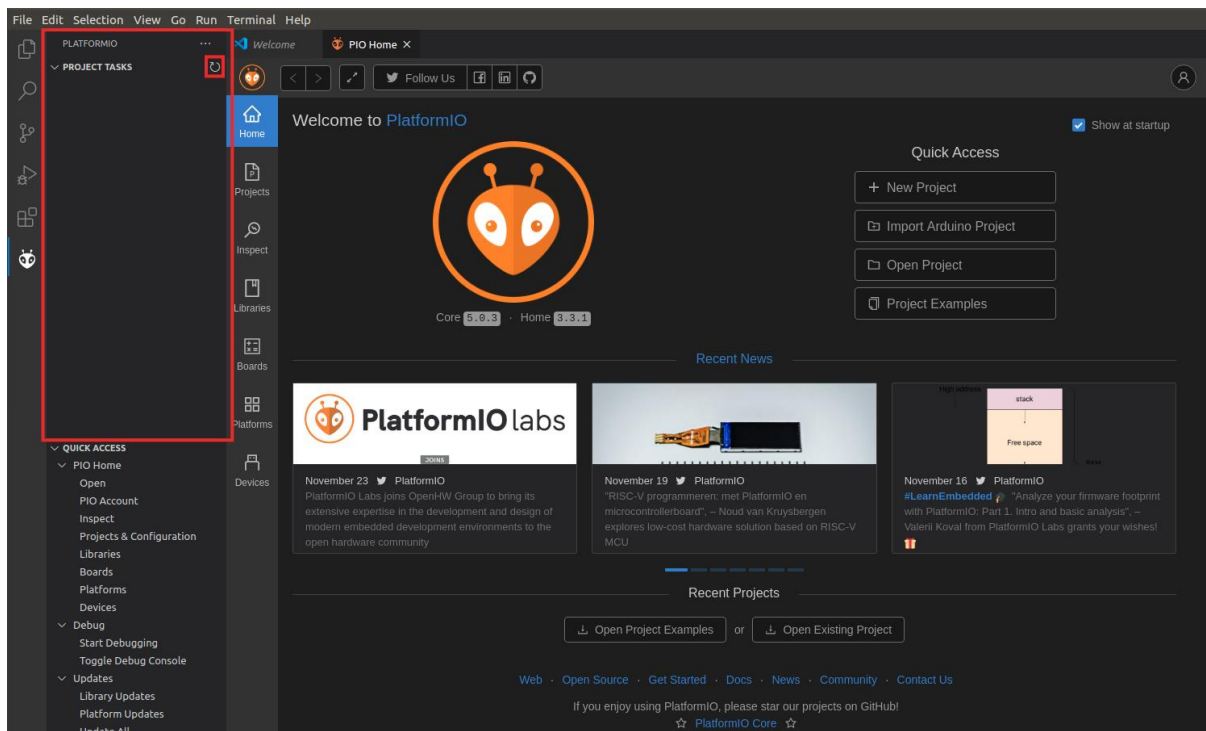


그림 26. PROJECT TASKS 창 비어 있음 – 새로 고침

9 단계. 그림 27 과 같이 Project Tasks → env:swervolf\_nexys → Platform 을 확장하고 비트스트림 업로드를 클릭합니다.

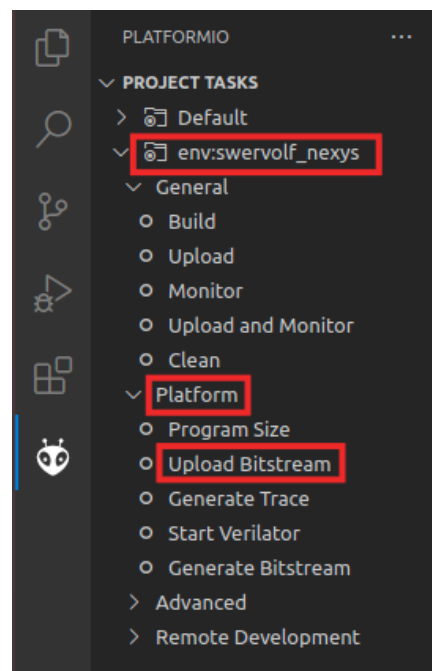


그림 27. 비트스트림 업로드

이제 비트스트림이 업로드 되었으므로 디버깅 프로세스를 시작합니다.

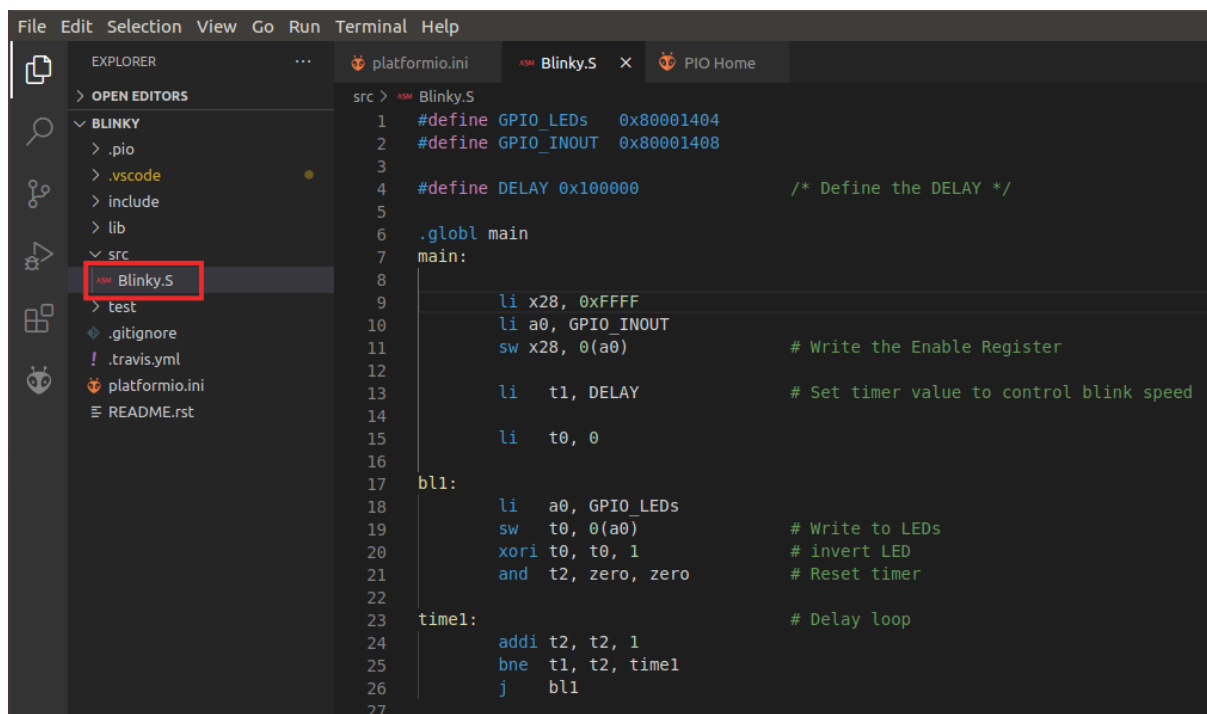


그림 28. PlatformIO 의 blinky.S

10 단계. 프로그램을 실행하고 디버깅하려면  를 클릭하십시오. 그런 다음 재생 버튼

RUN ▶ PIO Debug

을 클릭하여 디버깅을 시작합니다. PlatformIO 는 주 기능의 시작 부분에 임시 중단점을 설정합니다. 따라서 계속 버튼 ▶ 을 클릭하여 프로그램을 실행합니다.

**11 단계.** 보드에서 가장 오른쪽에 있는 LED 가 깜박이기 시작하는 것을 볼 수 있습니다.

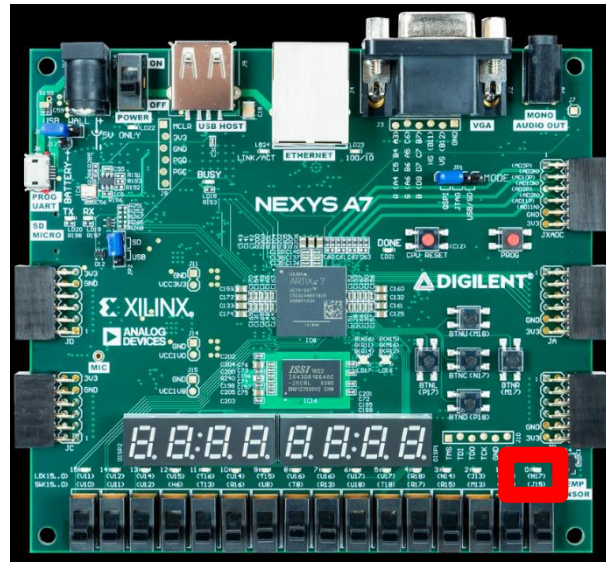
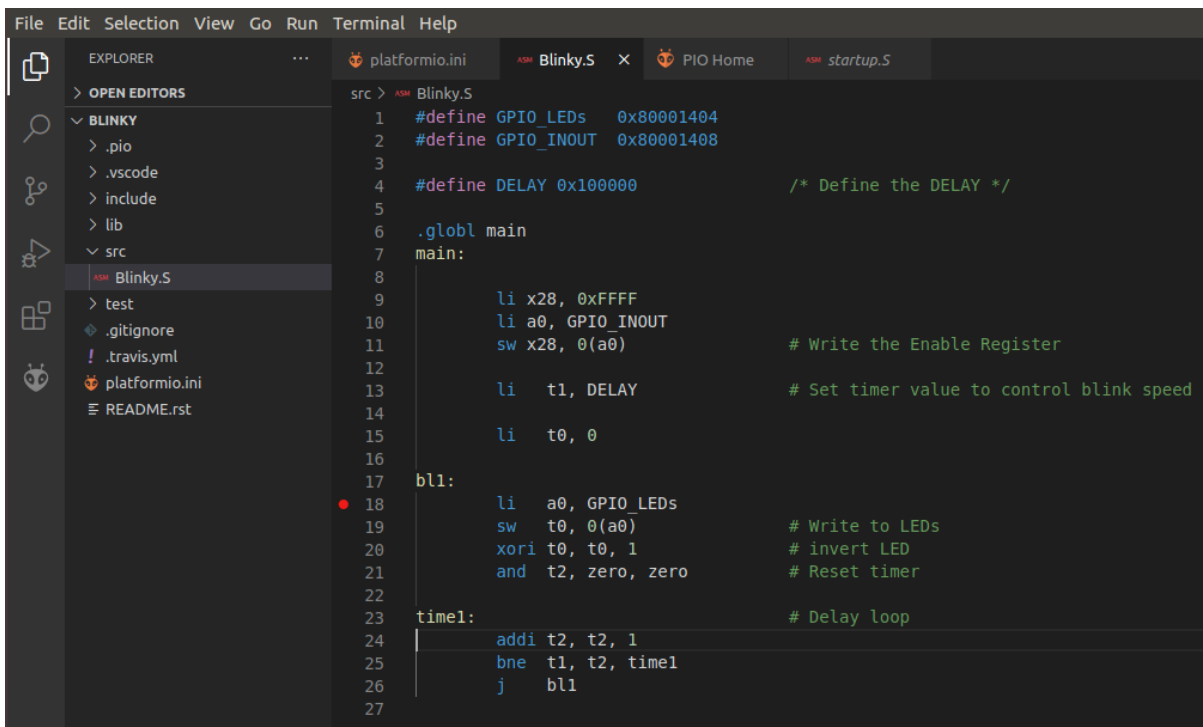


그림 29. 맨 오른쪽 LED 깜박임



**12 단계.** 일시 중지 버튼 을 클릭하여 실행을 일시 중지합니다. 실행은 무한 루프 내부(아마도 `time1` 지연 루프 내부)에서 중지됩니다.

**13 단계.** 줄 번호 18 의 왼쪽을 클릭하여 중단점(breakpoint)을 설정합니다. 빨간색 점이 나타나고 중단점이 BREAKPOINTS 탭에 추가됩니다(그림 30 참조).



```

src > Blinky.S
1  #define GPIO_LEDS    0x80001404
2  #define GPIO_INOUT   0x80001408
3
4  #define DELAY 0x100000      /* Define the DELAY */
5
6  .globl main
7  main:
8
9      li x28, 0xFFFF
10     li a0, GPIO_INOUT
11     sw x28, 0(a0)          # Write the Enable Register
12
13     li  t1, DELAY          # Set timer value to control blink speed
14
15     li  t0, 0
16
17 bl1:
18     li  a0, GPIO_LEDS
19     sw  t0, 0(a0)          # Write to LEDs
20     xori t0, t0, 1         # invert LED
21     and t2, zero, zero    # Reset timer
22
23 timel:                      # Delay loop
24     addi t2, t2, 1
25     bne t1, t2, timel
26     j   bl1
27

```

그림 30. blinky.S 에서 중단점 설정하기

**14 단계.** 그런 다음 계속 버튼  을 클릭하여 실행을 계속합니다. 실행은 계속되고 가장 오른쪽 LED 에 1(또는 0)을 쓰는 저장 워드(sw) 명령 후에 중지됩니다.

**단계 15.** 여러 번 실행을 계속하십시오. 가장 오른쪽에 있는 LED 에 대한 값 구동이 매번 변경되는 것을 볼 수 있습니다.

**16 단계.** 디버깅을 중지  하고  를 클릭하여 탐색기 창으로 돌아갑니다. **파일** → **폴더 단기**를 선택하여 프로그램을 닫습니다.

## 10. SweRVolf 디버깅

SweRVolf 는 하드웨어와 시뮬레이션 모두에서 디버깅을 지원합니다. 디버거를 연결하는 방법에는 여러 가지 절차가 있지만 일단 연결되면 동일한 명령을 사용할 수 있습니다(프로그램이 시뮬레이션에서 훨씬 느리게 실행됩니다).

### 시뮬레이션:

**1 단계.** WORKSPACE 디렉토리 "SweRVolf"를 입력한 다음 시뮬레이션 명령을 실행합니다.

```
> fusesoc run --target=sim swervolf --jtag_vpi_enable
```



"--jtag\_vpi\_enable" 매개변수는 OpenOCD가 연결할 수 있는 JTAG 서버를 활성화합니다. SweRVolf 시뮬레이션이 "--jtag\_vpi\_enable"로 시작되면 클라이언트가 연결하고 JTAG 명령을 보내기를 기다리는 JTAG 서버가 시작됩니다.

```
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Starting jtag_vpi server: interface 127.0.0.1 (loopback), port 5555/tcp ...
jtag_vpi server created.
Waiting for client connection...
```

**그림 31. 시뮬레이션 명령 실행**

**2 단계.** "Ctrl + Shift + t"를 사용하여 새 터미널을 연 다음 Workspace 디렉터리에서 다음을 실행하여 데이터 디렉터리로 이동합니다.

```
> cd fusesoc_libraries/swervolf/data/
```

**3 단계.** 이제 다음 명령을 실행하여 OpenOCD를 시뮬레이션 인스턴스에 연결합니다.

```
> openocd -f swervolf_sim.cfg
```

성공하면 OpenOCD는 다음을 출력해야 합니다(그림 32 참조).

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/fusesoc_libraries/swervolf/data$ openocd -f swervolf_sim.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : Set server port to 5555
Info : Set server address to 127.0.0.1
Info : Connection to 127.0.0.1 : 5555 succeed
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

**그림 32. OpenOCD 실행**

**4 단계.** "Ctrl + Shift + t"를 사용하여 세 번째 터미널을 열고 OpenOCD를 통해 디버그 세션에 연결합니다.

```
> telnet localhost 4444
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/fusesoc_libraries/swervolf/data$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
>
```



### 그림 33. telnet

**5 단계.** 이제 명령을 입력하여 지시를 내리십시오. 이제 세 번째 터미널을 통해 SweRVolf 에 실시간 지침을 제공할 수 있습니다.

이 FuseSoC 기반 SoC 에서 16 개의 LED 는 주소 `0x80001010-0x80001011` 에서 메모리 매핑된 GPIO 에 의해 제어됩니다. 이 주소는 실습 1 과 2 에서 사용한 RVfpga 시스템 모듈과 다릅니다.

따라서 메모리의 해당 주소에 "1"의 값을 쓸 수 있습니다.

```
> mwb 0x80001010 1
```

이제 첫 번째 터미널을 열면 gpio0 이 켜져 있는 것을 볼 수 있습니다(그림 34 참조).

```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
```

그림 34. gpio0 이 켜져 있습니다.

다시 세 번째 터미널로 돌아가 동일한 메모리 주소에 "0" 값을 작성하여 gpio0 을 끕니다.

```
> mwb 0x80001010 0
```

```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
528686180: gpio0 is off
```

그림 35. gpio0 이 꺼져 있습니다.

이제 Nexys A7 보드에서 동일한 프로세스를 완료합니다.

## HARDWARE:

**1 단계.** Nexys A7 보드를 컴퓨터에 연결한 다음 Workspace 디렉터리에서 FPGA 빌드 명령을 실행합니다. Nexys A7 보드용으로 다시 빌드하지 않고 비트스트림을 업로드하려면 "--run " 매개변수를 추가할 수 있습니다.

➤ `fusesoc run --target=nexys_a7 --run swervolf`

```
INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcs9324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: SUCCESS! FPGA xc7a100tcs9324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

### 그림 36. FPGA 빌드 실행

**2 단계.** OpenOCD 를 사용하여 보드 프로그래밍

➤ `openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit" -f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg`

그림 37.과 같이 출력 됩니다.

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit"
-f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdf (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
shutdown command invoked
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

### 그림 37. OpenOCD 실행

**3 단계.** OpenOCD 를 SweRVolf 에 연결

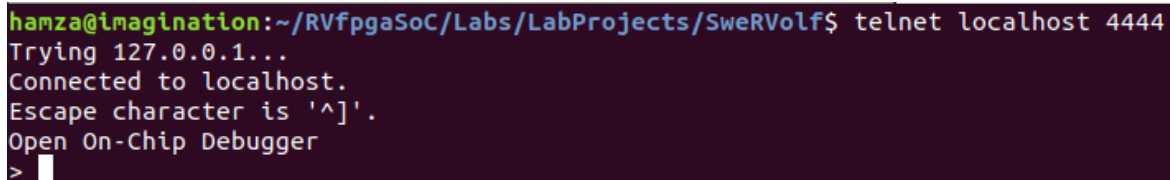
➤ `openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdf (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=
0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info : hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

### 그림 38. OpenOCD 연결됨

**4 단계.** "Ctrl + Shift + t"를 사용하여 세 번째 터미널을 열고 OpenOCD 를 통해 디버그 세션에 연결합니다.

```
> telnet localhost 4444
```



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> 
```

### 그림 39. telnet

**5 단계.** 이제 명령을 입력하여 보드에 직접 지시를 내립니다. 세 번째 터미널을 통해 SweRVolf 에 실시간 지침을 제공할 수 있습니다.

16 개의 LED 는 주소 **0x80001010 - 0x80001011** .에서 메모리 매핑된 GPIO 에 의해 제어됩니다.

가장 오른쪽 LED 를 켜기 위해 메모리에 "1"의 값을 0x80001010 에 씁니다.

```
> mwb 0x80001010 1
```

이제 다음 명령을 실행하여 맨 오른쪽 LED 를 끕니다.

```
> mwb 0x80001010 0
```

이것은 보드의 가장 오른쪽에 있는 LED 를 끌 것입니다.

이 실습을 빠르게 요약하기 위해 먼저 블록 설계 코드를 수동으로 조정한 코드와 비교했습니다. 그런 다음 우리는 SweRVolf(SweRV Eh1 용 FuseSoC 기반 SoC) 패키지를 포함하는 패키지 관리자인 FuseSoC 를 소개했습니다.

그런 다음 "SweRVolf Sim"과 "SweRVolf Nexys"의 두 가지 버전을 살펴보았습니다.

이전 실습의 **RVfpgaSim** 과 마찬가지로 **SweRVolf Sim** 은 SweRVolf 코어를 Verilator 에서 사용할 테스트벤치로 래핑하는 시뮬레이션 대상입니다. 마찬가지로 이전 실습의 **RVfpgaNexys** 는 Digilent Nexys A7 보드를 대상으로 하는 SweRVolf SoC 버전인 **SweRVolf Nexys** 와 유사합니다.

그런 다음 FuseSoC 실행을 사용하여 시뮬레이션용 SweRVolf Sim 과 Nexys A7 보드용 SweRVolf Nexys 를 구축했습니다. 나중에 RVfpgaSim 및 RVfpgaNexys 에 대한 이전 실습에서 실행한 것과 동일한 예제를 이러한 빌드에서 실행합니다.