



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoC Lab 1

RVfpga-SoC 소개

표 1. RVfpga 용어

Name	Description
Courses	
RVfpga	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.
RVfpga-SoC	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.
Cores and SoCs	
SweRV EH1 Core	Western Digital 에서 개발한 오픈 소스 상용 RISC-V 코어 (https://github.com/chipsalliance/Cores-SweRV).
SweRV EH1 Core Complex	추가된 메모리(ICCM, DCCM 및 명령 캐시), 프로그래밍 가능한 인터럽트 컨트롤러(PIC), 버스 인터페이스 및 디버그 장치가 있는 SweRV EH1 코어 (https://github.com/chipsalliance/Cores-SweRV).
SweRVolfX	RVfpga 과정에서 사용하는 System on Chip 으로 SweRVolf 의 확장입니다. SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf): 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. SweRVolfX : SweRVolf 에 GPIO, PTC, 추가 SPI 및 8 자리 7-세그먼트 디스플레이 컨트롤러, 4 가지 새로운 주변 장치를 추가합니다.
RVfpgaNexys	Nexys A7 보드 및 주변 장치를 대상으로 하는 SweRVolfX SoC 입니다. DDR2 인터페이스, CDC(clock domain crossing) 장치, BSCAN 로직(JTAG 인터페이스용) 및 클럭 생성기를 추가합니다. RVfpgaNexys 는 SweRVolf Nexys(https://github.com/chipsalliance/Cores-SweRVolf)와 동일하지만 SweRVolf Nexys 는 SweRVolf 를 기반으로 합니다.

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

1. 소개

1. 시스템 온 칩 소개

이 실습에서는 빌딩 블록에서 RISC-V 시스템 온 칩(SoC)을 구축하는 방법을 보여줍니다. **SoC** 는 전체 전자 또는 컴퓨터 시스템을 통합하는 집적 회로 또는 IC 입니다. SoC 에는 운영 체제를 로드하고 프로그램을 실행하는 데 필요한 코어와 모든 주변 장치 및 인터페이스가 포함됩니다.

그림 1 은 프로세서 코어로 시작하여 코어를 중심으로 구축된 SoC, 마지막으로 시스템 및 보드 인터페이스로 구성된 임베디드 시스템의 일반적인 계층 구조를 보여줍니다.



그림 1. 일반적인 임베디드 시스템

SoC 의 설계 프로세스는 FPGA 에서 프로토타이핑하는 것으로 시작됩니다. 우리의 초점은 SoC 를 FPGA 로 타겟팅하는 것입니다.

우리가 사용할 RISC-V CPU 는 Western Digital 의 **SweRV EH1 Core Complex** 이며 이 실습에서 설계할 SoC 는 **Nexys A7-100T** 보드를 대상으로 하는 **SweRVolfX** 의 하위 집합입니다. 그림 2 는 다양한 구성 요소와 이들이 어떻게 결합되는지 보여줍니다.

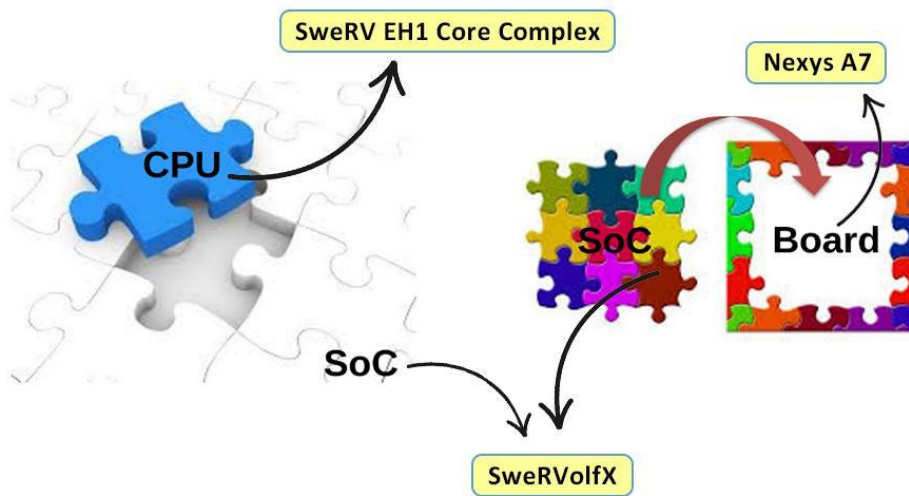


그림 2. RVfpga 시스템 기반 임베디드 시스템

2. SweRVVolFX 및 RVfpga 시스템 소개

이 실습을 시작하기 전에 RVfpga 과정의 시작 안내서를 살펴보고 전체 RVfpga 시스템을 이해하는 것이 좋습니다.

다음은 RVfpga 과정에서 도입된 RVfpga 시스템에 대한 간략한 설명입니다.

표 1 은 SweRV EH1 Core 에서 RVfpgaNexys 및 RVfpgaSim 까지의 RVfpga 시스템의 계층 구조를 보여줍니다. RVfpga 시스템에 사용되는 SoC(System on Chip)는 **SweRVVolFX** 라고 부르며, **SweRV EH1 Core Complex** 에서 빌드된 **SweRVVolFX** 버전 0.7.3 (<https://github.com/chipsalliance/Cores-SweRVVolFX/releases/tag/v0.7.3>) 을 그림 3 에서 설명하고 있습니다.

SweRV EH1 Core Complex 외에도 SweRVVolFX SoC 에는 Boot ROM, UART, 시스템 컨트롤러 및 SPI 컨트롤러도 포함됩니다. SweRV EH1 Core 는 AXI 버스를 사용하고 주변 장치는 Wishbone 버스를 사용하고 SoC 에는 AXI-Wishbone Bridge 도 있습니다.

RVfpga 시스템에서 SweRVVolFX SoC 는 다른 SPI 컨트롤러(SPI2), GPIO(범용 입/출력) 컨트롤러, PTC(PWM/타이머/카운터) 모듈과 같은 몇 가지 추가 기능으로 확장됩니다. (그림 3 은 이러한 새로운 주변 장치를 빨간색으로 보여줍니다). 이 System on Chip 은 **SweRVVolFX**(X 는 eXtended 를 나타냄)라고 합니다.

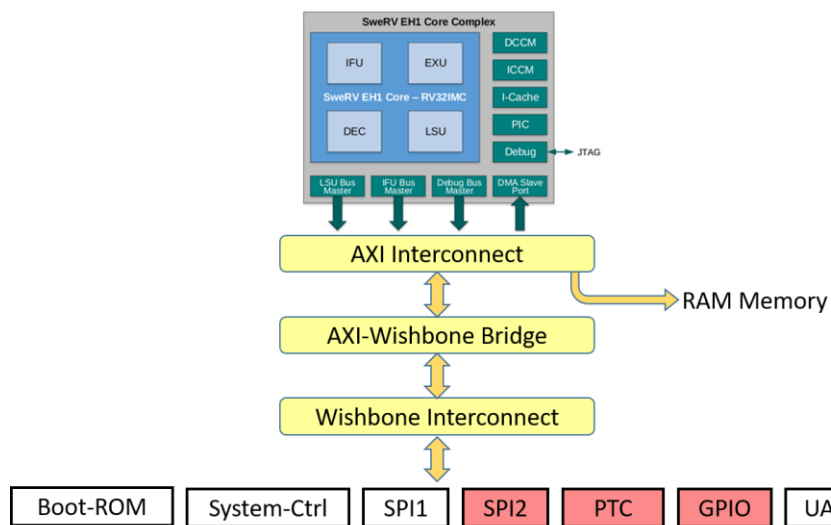


그림 3. SweRVolfX

표 4 는 Wishbone 상호 연결을 통해 SweRV EH1 코어에 연결된 주변 장치의 메모리 매핑된 주소를 제공합니다.

표 4. SweRVolfX 의 MEMORY-MAPPED 주소

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

* SweRVolfX 에 추가된 주변 장치

3. RVfpga-SoC 소개

RVfpga 에서 SweRVolfX 의 생성 방법에 대하여 구체적인 정보 없이 SweRVolfX 를 소개하였습니다. RVfpga-SoC 과정에서는 SweRV 코어, 메모리 및 주변 장치와 같은 빌딩 블록을 사용하여 처음부터 SweRVolfX SoC 의 하위 집합을 구축하는 방법을 보여줍니다.

이 실습은 CPU(SweRV EH1 Core Complex)로 시작하여 SoC 를 구축하는 방법을 보여주는 단계별 가이드가 될 것입니다.

우리는 Vivado Block Design Tool 을 사용할 것입니다. Vivado 의 블록 디자인 도구는 배선 구성 요소를 그래픽으로 용이하게 하여 프로세스를 더 쉽게 이해하고 시각화할 수 있도록 합니다. 이 시각적 접근 방식은 또한 각 모듈이 다른 모듈과 연결되어 SoC 를 형성하는 방법을 보여줍니다.

모듈은 세 가지 주요 블록 또는 범주로 분류할 수 있습니다:

1. CPU(SweRV EH1 Core Complex)
2. 상호 연결(AXI-Interconnect, AXI2WB 및 WB-Interconnect)
3. 주변 장치(Boot-ROM, GPIO 컨트롤러 및 시스템 컨트롤러)

SweRVolfX 에는 다양한 모듈이 있으며 일부는 주요(barebones) RISC-V SoC 에 필요하지 않습니다. 따라서 추가 모듈은 실험실을 단순화하고 CPU 코어를 활성화하는 데 필요한 주요 기능에 중점을 두어 정리되었습니다.

그림 4 는 포함하지 않을 모듈(UART, PTC, SPI1 및 SPI2)을 보여줍니다.

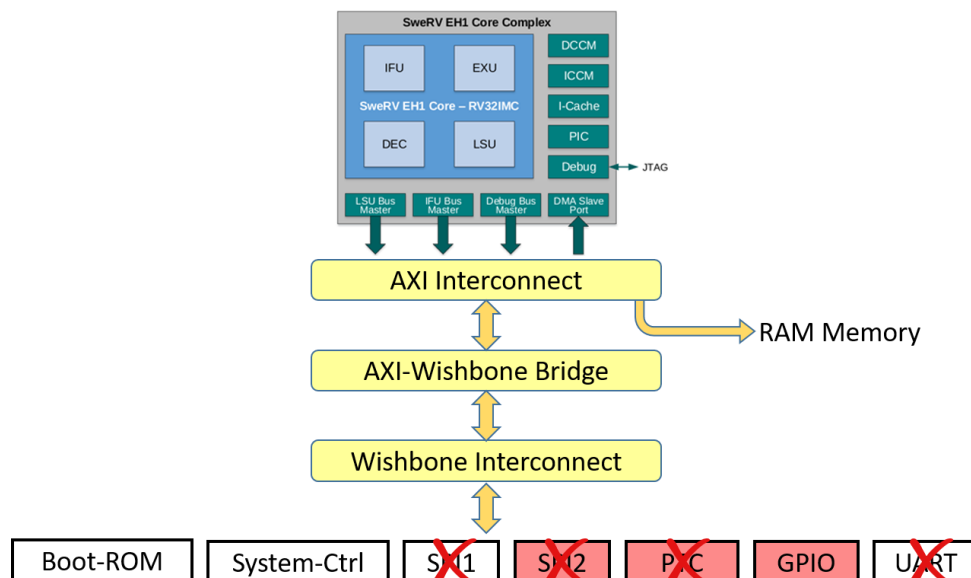


그림 4. SweRVolfX 의 하위 집합

그림 5 는 우리가 구현할 SoC 의 개략적인 블록 다이어그램을 보여줍니다.

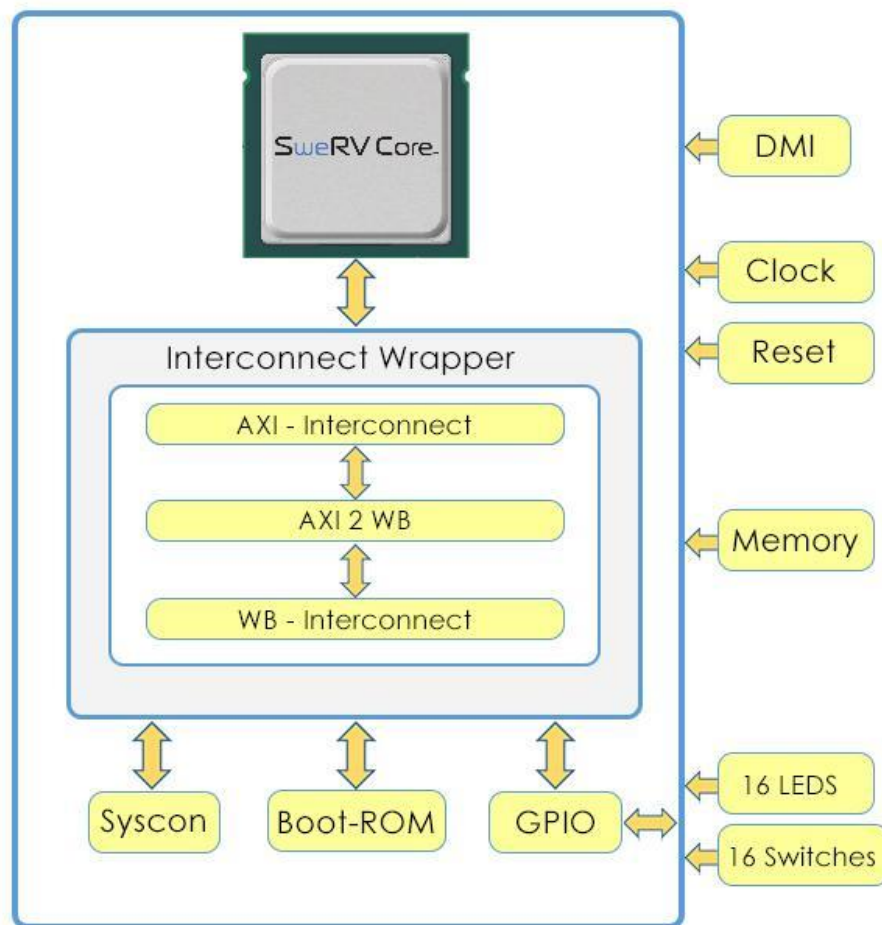


그림 5. Lab 1 SoC 의 상위 레벨 블록 다이어그램

학습과 이해의 편의를 위해 Interconnect 를 구성하는 일부 구성 요소(AXI Interconnect, Wishbone Interconnect 및 AXI to Wishbone 브리지)가 하나의 Interconnect 래퍼 모듈에 래핑 되었습니다.

CPU 와 CPU 내부에 중점을 둔 실습의 경우 RVfpga 과정을 참조하십시오. RVfpga(RISC-V FPGA) 과정은 상용 RISC-V 프로세서 및 SoC 를 FPGA(Field-Programmable Gate Array)로 대상화한 다음 이를 사용 및 확장하여 학습하기 위한 지침, 도구 및 실습이 포함된 패키지로 컴퓨터 아키텍처, 디지털 디자인, 임베디드 시스템 및 프로그래밍으로 확장하여 공부할 수 있습니다. RVfpga 에 대한 자세한 내용은 <https://university.imgtec.com/rvfpga/>를 참조하십시오.

2. 요구사항

이 실습을 완료하려면 다음 소프트웨어가 설치되어 있어야 합니다:

- Vivado 2019.2 Web Pack (설치 가이드 참조 (Page No.04))
- Digilent Board Files (설치 가이드 참조 (Page No.05))

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

아직 설치하지 않았다면 RVfpga-SoC 설치 가이드의 지침에 따라 Xilinx의 Vivado를 설치하십시오. Imagination University Programme (IUP)에서 다운로드한 **RVfpgaSoC** 폴더를 컴퓨터에 복사했는지 확인합니다.

3. Vivado 프로젝트 생성

Xilinx의 Vivado Design Suite를 사용하여 시스템을 정의하는 Verilog 파일인, RTL을 사용하여 SweRVofX 하위 집합을 빌드합니다. 아래에서 자세히 설명하고 있는 다음 단계에 따라 Vivado 프로젝트를 생성하십시오.

1 단계. Vivado 열기

RVfpga-SoC 설치 가이드에 설명된 대로 시스템에 Vivado를 설치하지 않았다면 지금 설치하십시오. 보드 파일도 설치하십시오.

이제 Vivado를 실행합니다(**Linux**에서는 터미널을 열고 vivado를 입력합니다. **Windows**에서는 시작 메뉴에서 Vivado를 엽니다). Vivado 환영 화면이 열립니다. 프로젝트 만들기를 클릭합니다(그림 6 참조).

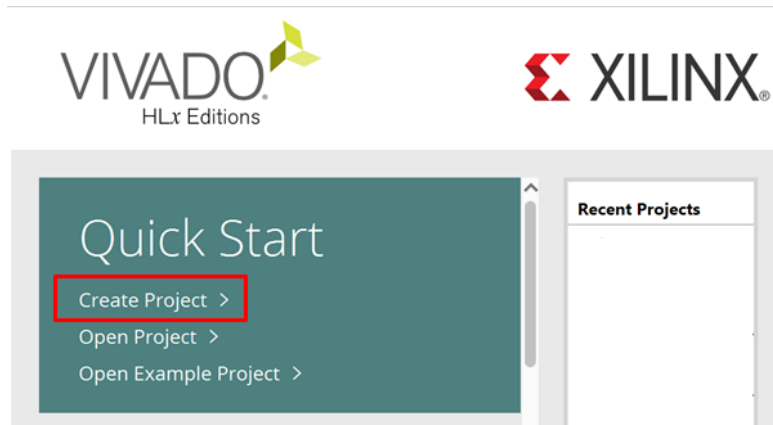


그림 6. Vivado 시작 화면: Create Project

2 단계. 새 RTL 프로젝트 생성

이제 Create a New Vivado Project Wizard가 열립니다(그림 7 참조). 다음을 클릭합니다.

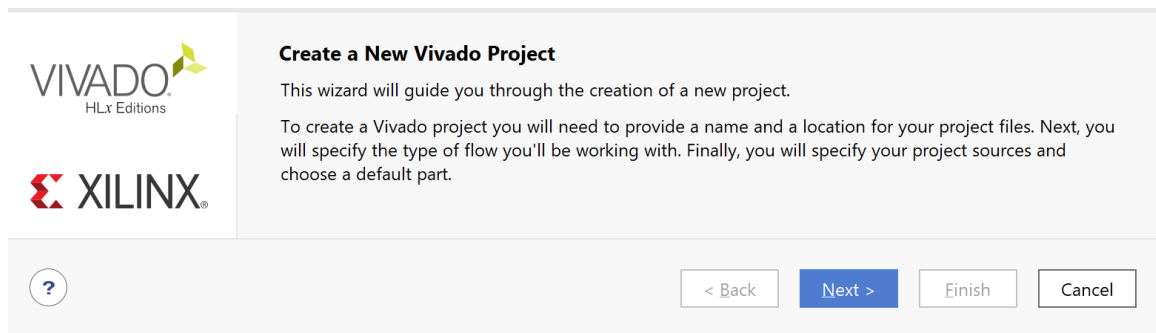


그림 7. 새 Vivado 프로젝트 마법사 만들기

프로젝트 이름을 공백 없이 "Lab1"으로 입력합니다. 그런 다음 Next 를 클릭합니다(그림 8 참조).

다음 프로젝트 위치 경로를 선택하십시오:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabProjects/Lab1

"LabProjects" 폴더에 "Lab1"이라는 폴더가 이미 있으므로 프로젝트 하위 디렉토리 생성 확인란의 선택을 취소합니다.

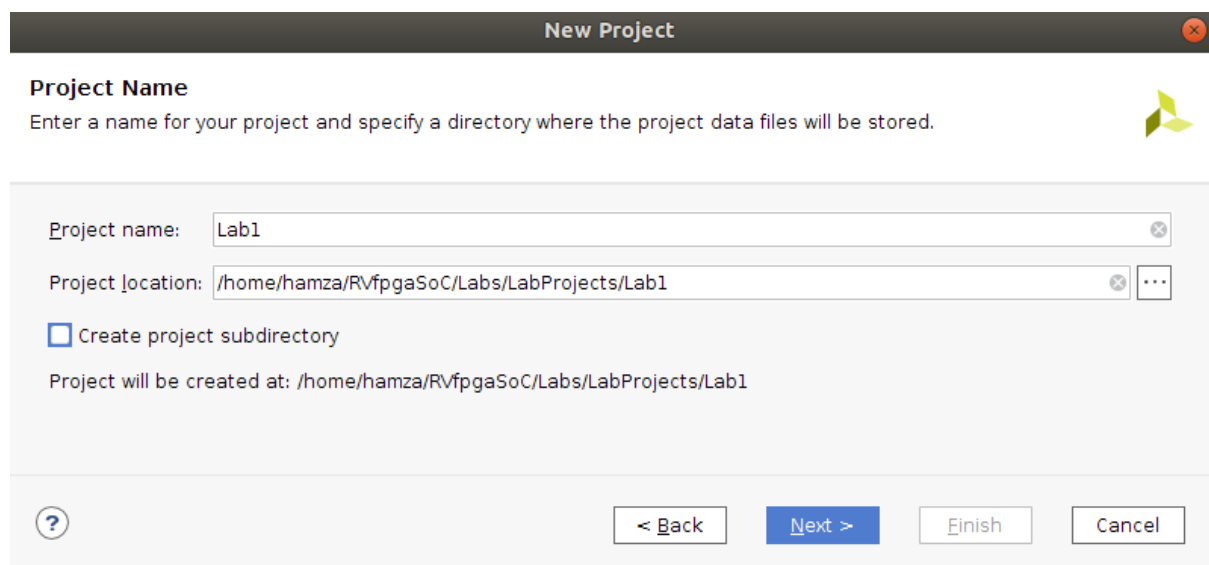


그림 8. 프로젝트 이름

프로젝트 유형을 RTL 프로젝트로 선택하고 Next 를 클릭합니다(그림 9 참조).

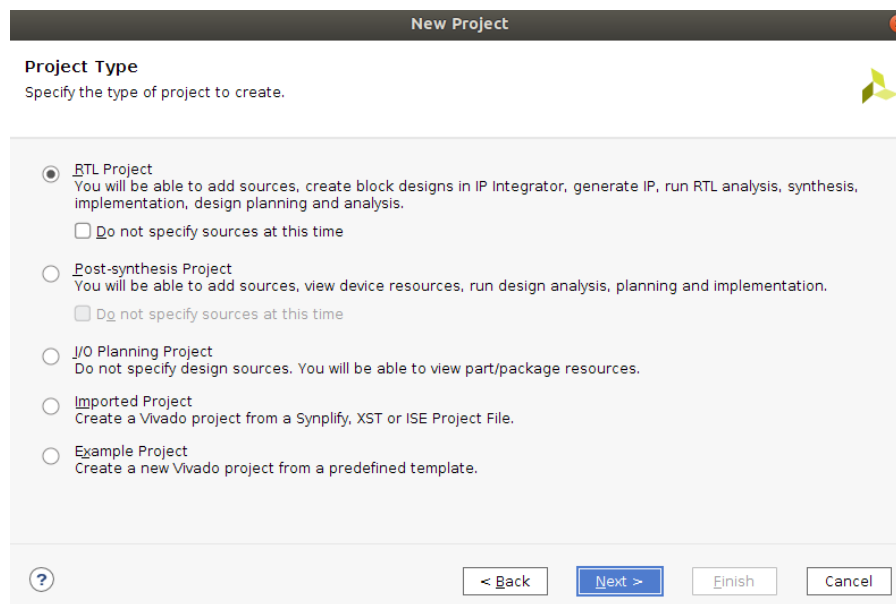


그림 9. RTL 프로젝트

3 단계. RTL 소스 파일 및 constraint files 추가

소스 추가 창에서 "디렉토리 추가"를 클릭합니다(그림 10 참조).

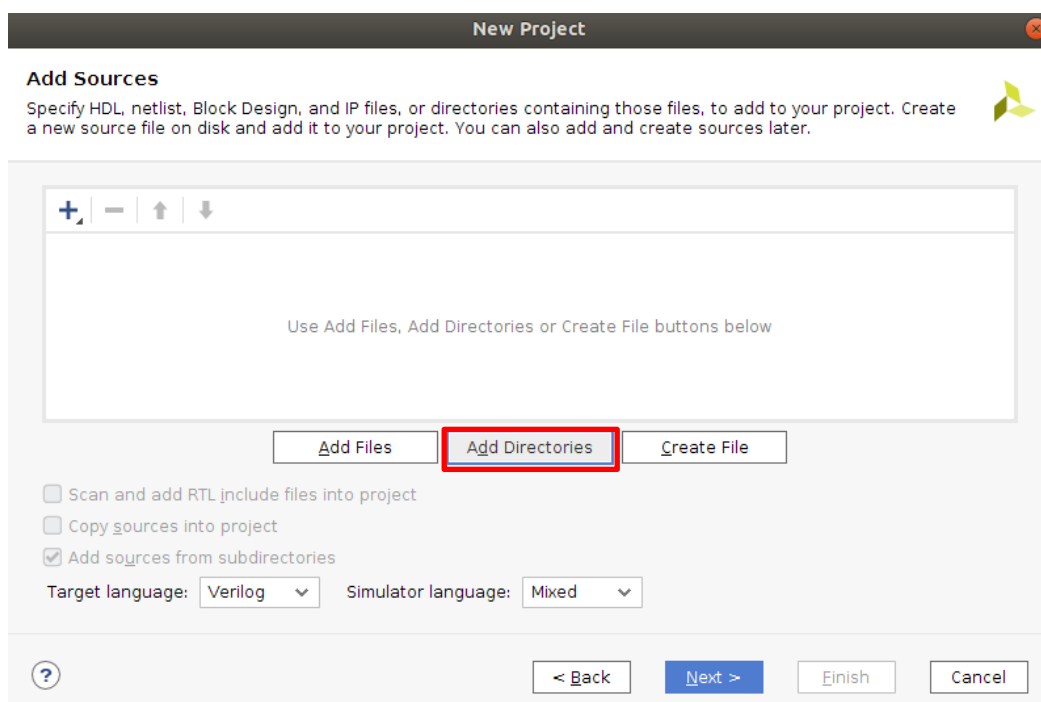


그림 10. Sources 디렉토리 추가

이제 [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src 경로에서 "src" 디렉토리를 선택합니다(그림 11 참조).

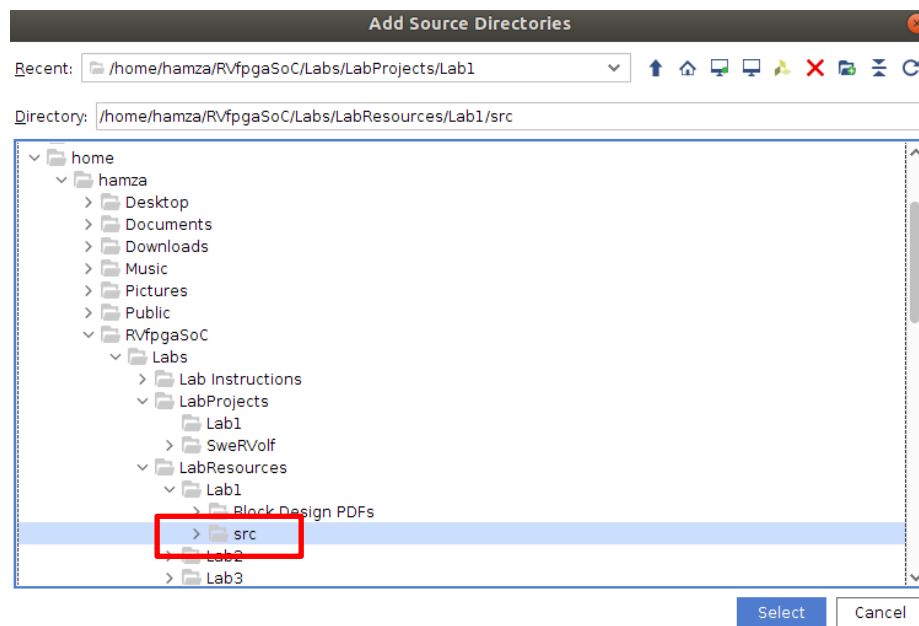


그림 11. "src" 디렉토리 선택

Select 를 클릭합니다.

그런 다음 **"Add Files"** 버튼을 클릭합니다.



그림 12. 파일 추가

파일 유형을 **"All Files"**로 선택합니다. 이제 방금 추가한 **src** 디렉토리 내의 **LiteDRAM** 디렉토리로 이동합니다.

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/mem_1.init
- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/litdram_core.init

".init" 파일을 모두 선택하고 확인을 클릭하여 둘 다 추가합니다(그림 13 참조).

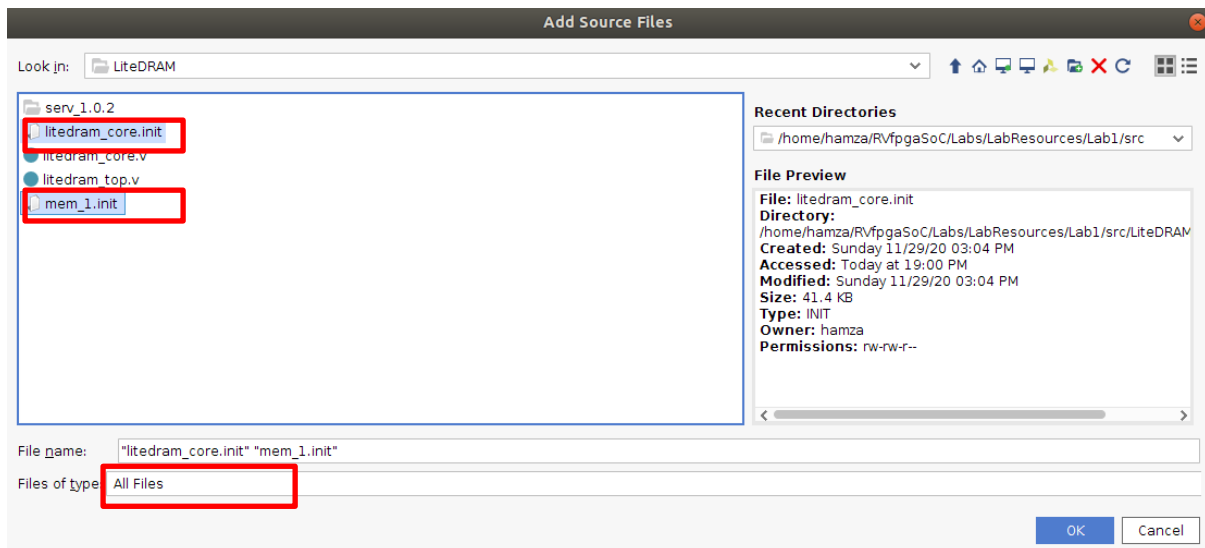


그림 13. LiteDram 소스 파일 추가

세 개의 확인란이 모두 선택되어 있는지 확인합니다(그림 14 참조).

"Next"를 클릭하여 다음 단계로 진행합니다.

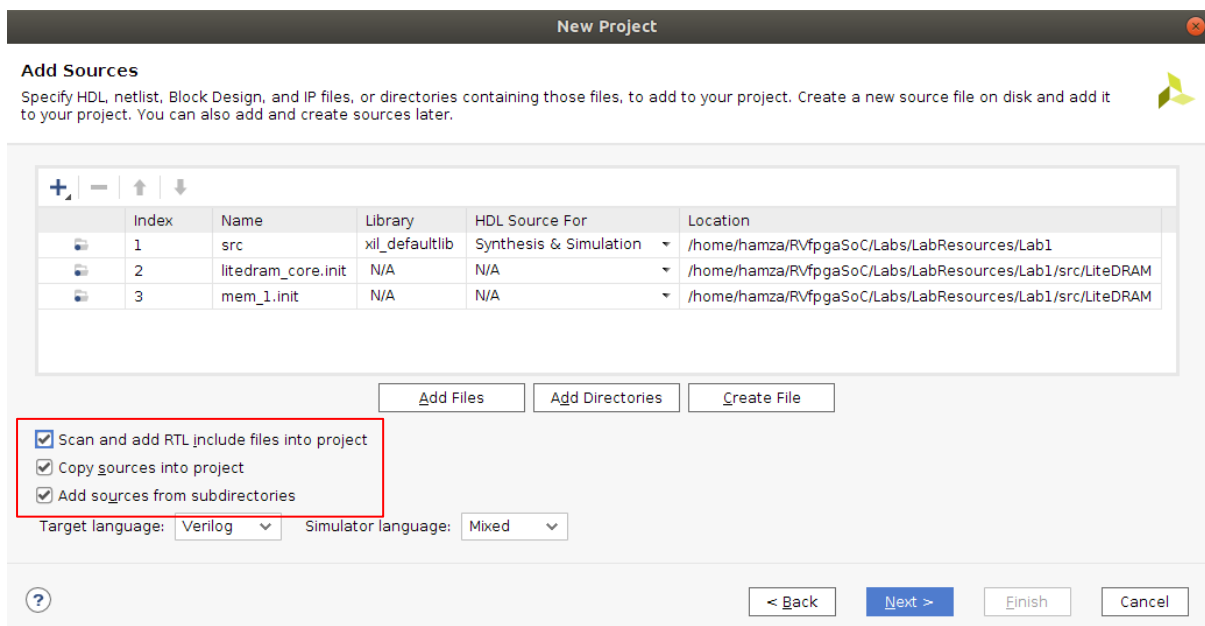


그림 14. 소스 추가

이제 시스템에 대한 constraints 를 추가합니다. 이 파일은 신호 이름을 보드에 있는 핀에 매핑합니다. 예를 들어 Nexys A7 FPGA 보드의 LED 는 PCB 트레이스를 통해 보드의 FPGA 핀에 연결됩니다. Vivado 는 RTL 의 올바른 신호 이름을 올바른 FPGA 핀에 매핑하기 위해 이것을 알아야 합니다. 예를 들어, Xilinx 디자인 constraints file 인 `[RVfpgaSoCPath]/RVfpgaSoC/src/rvfpga.xdc` 파일의 다음 줄은 FPGA 핀 H17 이 최하위 LED(`o_led[0]`)에 매핑되고, 이는 LVCMOS 3.3 V 신호를 사용합니다.

```
set_property -dict { PACKAGE_PIN H17    IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
```


신호 이름 `o_led` 는 RVfpga 의 Verilog 코드에서 Nexys A7 보드의 LED 를 구동하는 데 사용되는 이름입니다.

Constraints 추가 창에서 **"Add Files"**를 클릭하고 다음 두 파일을 선택합니다(그림 15 참조).

`[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/rvfpga.xdc`
`[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/litedram.xdc`

NEXT 를 클릭합니다.



그림 15. Constraints 추가

4 단계. 대상 보드로 Nexys A7 선택

기본 파트(Default Part) 창에서 Boards 를 클릭한 다음 Nexys A7-100T 를 선택합니다(그림 16 참조).

검색 상자를 사용하여 결과 범위를 좁힐 수 있습니다. 또한 실제 타겟 FPGA 의 이름이 파트 열, xc7a100tcsg324-1 에 나열되어 있음을 알 수 있습니다. 이는 CSG(chip-scale grid) 패키지과 324 핀이 포함된 100k 등가 게이트가 있는 Xilinx Artix-7 FPGA 임을 나타냅니다.

Next 를 클릭합니다.

Default Part
Choose a default Xilinx part or board for your project.

Parts: **Boards**

[Reset All Filters](#) Update Board Repositories

Vendor: All Name: All Board Rev: Latest

Search: (5 matches)

Display Name	Preview	Vendor	File Version	Part	I/C
Nexys A7-100T		digilentinc.com	1.0	xc7a100tcsg324-1	32
Nexys A7-50T		digilentinc.com	1.0	xc7a50ticsg324-1L	32
Nexys4		digilentinc.com	1.1	xc7a100tcsg324-1	32
Nexys4 DDR		digilentinc.com	1.1	xc7a100tcsg324-1	32

? < Back Next > Finish Cancel

그림 16. 대상 보드 선택: Nexys A7-100T

새 프로젝트 요약 창에서 **Finish** 를 클릭합니다(그림 17 참조).

VIVADO
HLx Editions

XILINX

New Project Summary

- A new RTL project named 'Lab1' will be created.
- 208 source files will be added.
- 2 constraints files will be added.
- The default part and product family for the new project:
 Default Board: Nexys A7-100T
 Default Part: xc7a100tcsg324-1
 Product: Artix-7
 Family: Artix-7
 Package: csg324
 Speed Grade: -1

To create the project, click Finish

? < Back Next > Finish Cancel

그림 17. 새 프로젝트 요약 창

프로젝트 설정이 완료되면 Syntax error 가 있는 파일이 있음을 나타냅니다. 이는 다음 단계에서 수정됩니다.

5 단계. rvfpga 를 최상위 모듈로 설정

프로젝트가 초기화됩니다. 이제 rvfpga 모듈을 최상위 모듈로 설정합니다. Sources 창에서 Design Sources 아래로 스크롤하고 rvfpga 모듈을 마우스 오른쪽 버튼으로 클릭하고 Set as Top 을 선택합니다(그림 18 참조).

그림과 같이 검색 상자에 이 이름을 입력하여 rvfpga 모듈을 찾을 수도 있습니다. 이것은 rvfpga 를 계층에서 가장 높은 레벨의 모듈로 설정하고 FPGA 에 합성 및 구현될 타겟을 설정합니다. rvfpga 를 최상위 모듈로 설정하면 계층이 업데이트됩니다.

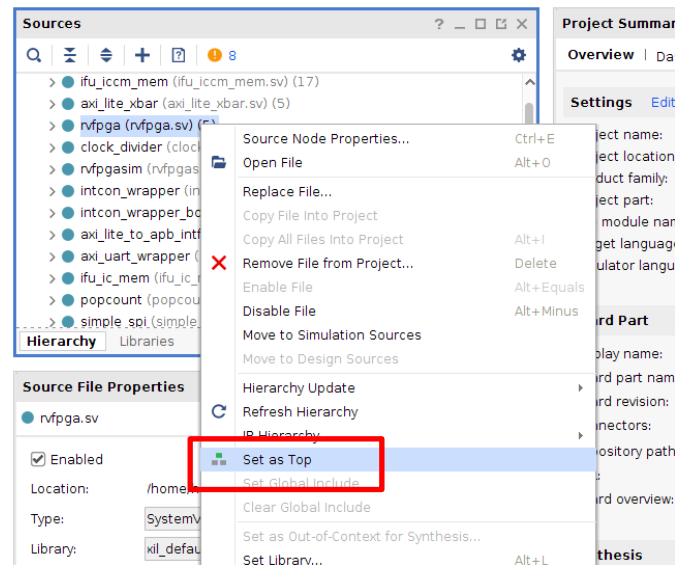


그림 18. rvfpga 를 최상위 모듈로 설정

6 단계. Verilog 헤더 파일을 전역(Global) 포함 파일로 설정

이제 Design Sources 아래의 Sources 창에서 Non-modules 파일 그룹을 확장하고 common_defines.vh 를 클릭합니다. 그러면 파일의 속성이 소스 창 바로 아래의 소스 파일 속성 창에서 열립니다. 전역 포함(Global Include)을 클릭하여 해당 상자를 선택합니다(그림 19 참조). 이제 계층이 업데이트되고 해당 파일이 Design Sources/Global Include 에 포함됩니다.

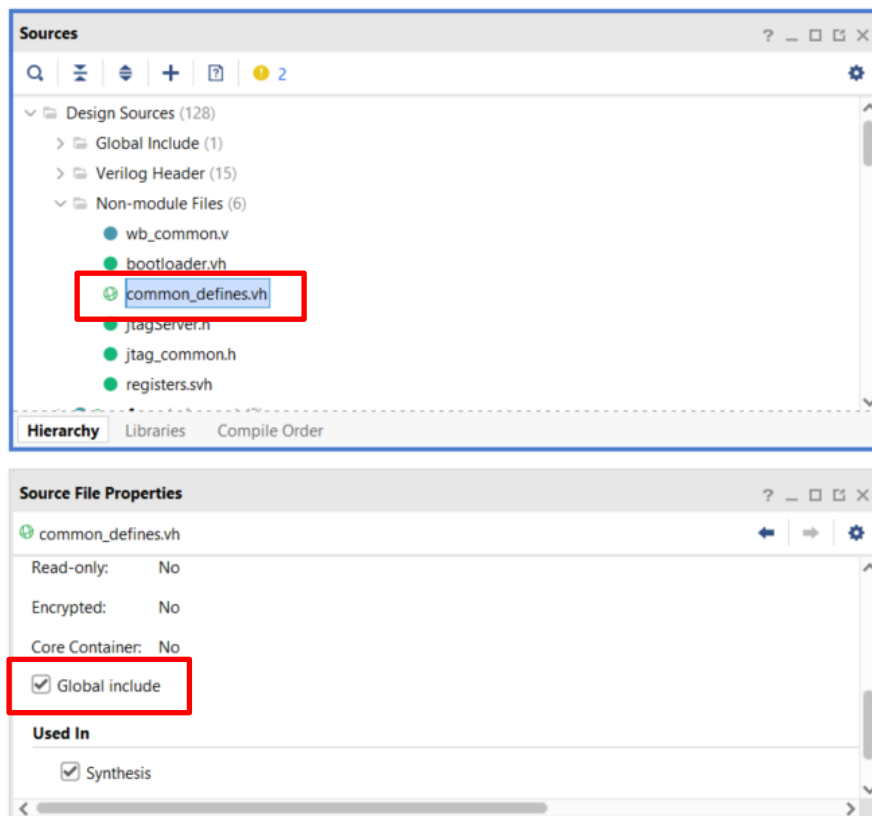


그림 19. common_defines.vh 를 Global include file 로 설정

마찬가지로 "assign.svh", "registers.svh" 및 "typedef.svh" SystemVerilogHeader 파일을 global include files 로 설정합니다(그림 20 참조).

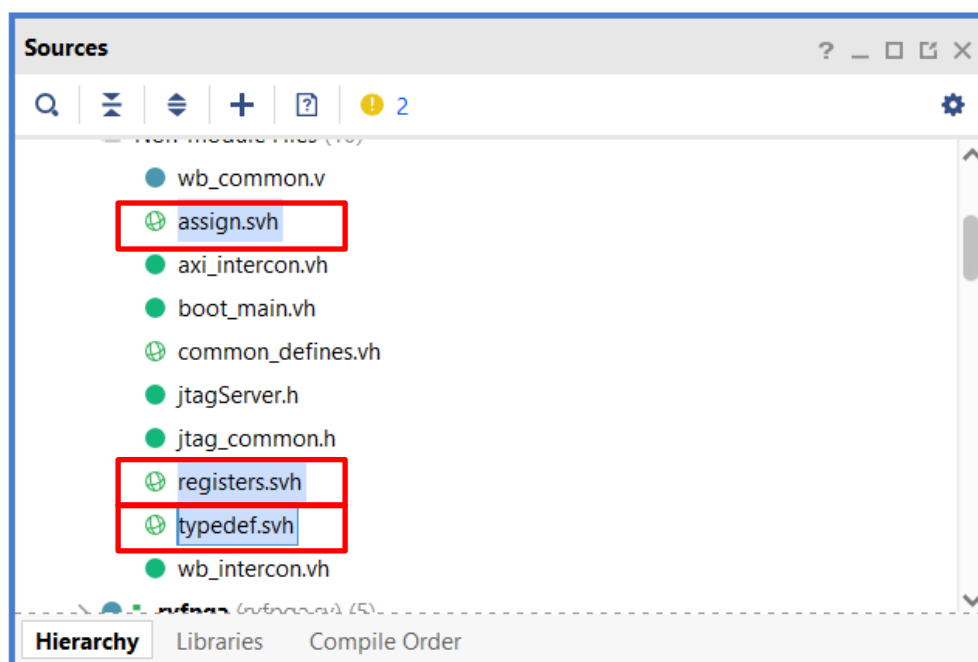


그림 20. ".svh" 파일을 Global include file 로 설정

이제 **"unknown"** 파일 그룹을 확장하고 **"litedram_core.init"** 를 클릭합니다. 그런 다음 소스 파일 속성 패널에서 일반 버튼 옆에 있는 속성 버튼을 클릭합니다. **"IS_Global_INCLUDE"**를 클릭하여 해당 상자를 선택합니다(그림 21 참조).

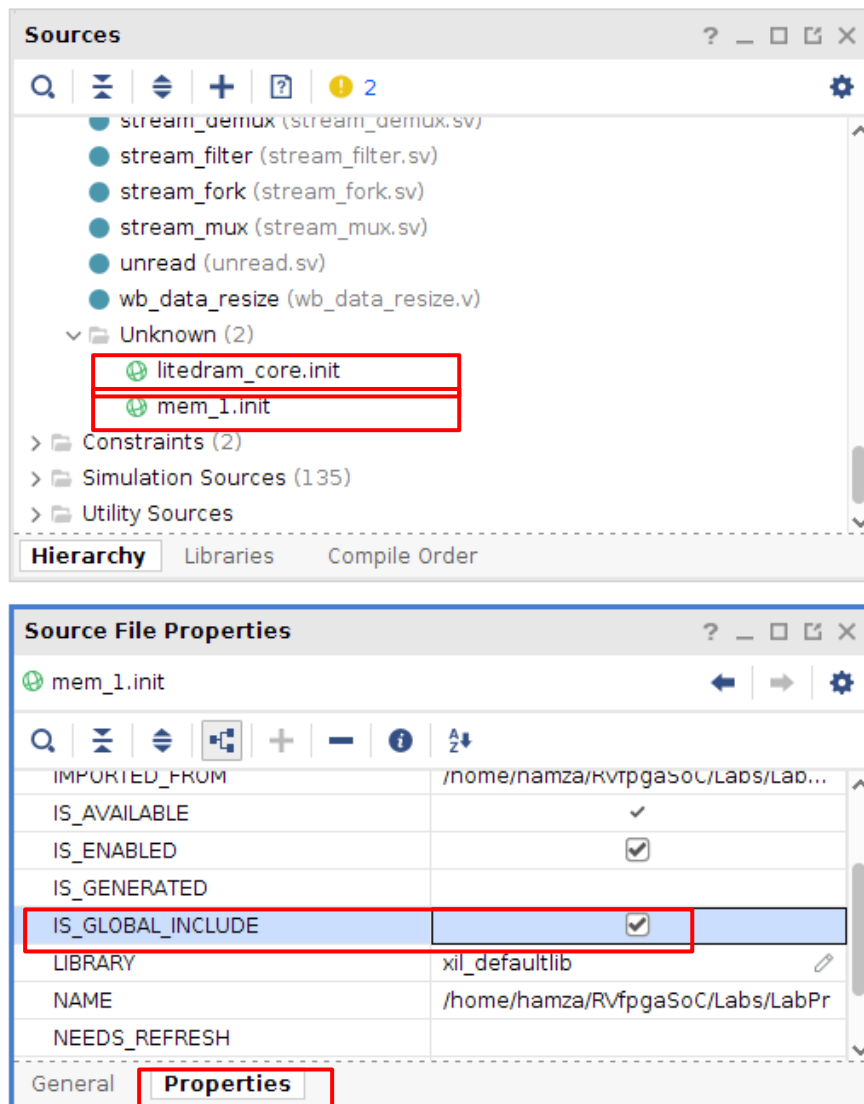


그림 21. litedram_core.init 를 전역 포함 파일로 설정

이제 **"mem_1.init"** 파일에 대해 동일한 작업을 수행하고 해당 파일을 **"litedram_core.init"** 파일과 마찬가지로 전역 포함 파일로 설정합니다.

7 단계. 프로젝트에 boot_main.mem 추가

Flow Navigator 창에서 Add Sources 를 클릭하고, 기본 옵션 ("Add or create design sources")을 그대로 두고, Add Files 를 클릭합니다(그림 22 참조).

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/SweRVolfSoC/BootROM/sw 로 이동하여

boot_main.mem 을 선택합니다(그림 22 참조). 계층 구조가 업데이트되고 해당 파일이 Design Sources/Memory File 에 포함됩니다.

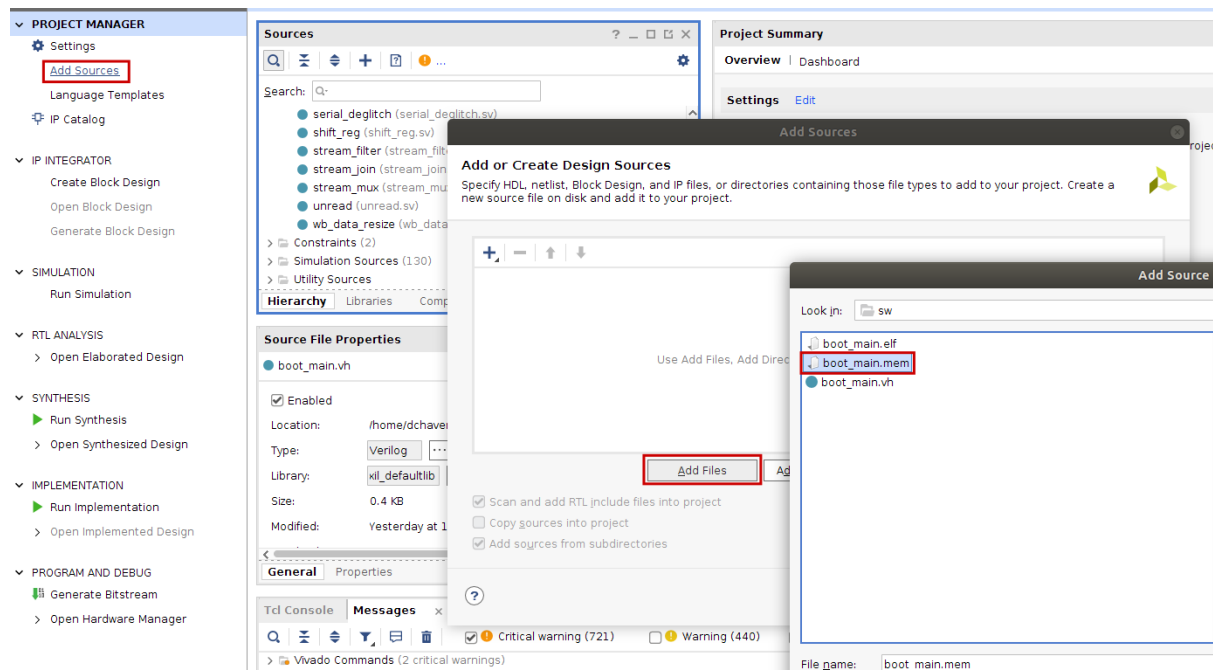


그림 22. 메모리 추가 *boot_main.mem* 파일

이제 디자인 소스 파일이 추가되었으며 이제 블록 디자인 생성을 시작할 수 있습니다.

4. 블록 디자인 생성

우리는 Vivado 의 블록 디자인 기능을 사용하여 SweRVofX 하위 집합을 생성하는 데 필요한 모듈을 추가한 다음 모듈을 서로 연결합니다.

1 단계. 블록 디자인 생성 클릭

IP Integrator 제목 아래의 Create Block Design 을 클릭하여 Flow Navigator 에서 새로운 블록 디자인을 만듭니다(그림 23 참조).

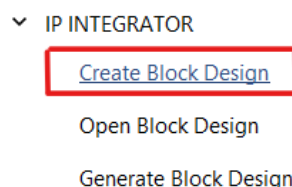


그림 23. 블록 디자인 생성

2 단계. 블록 디자인 이름 선택

나중에 Lab 에서 이름 충돌을 방지하려면 디자인 이름을 "BD"로 선택하십시오(그림 24 참조).

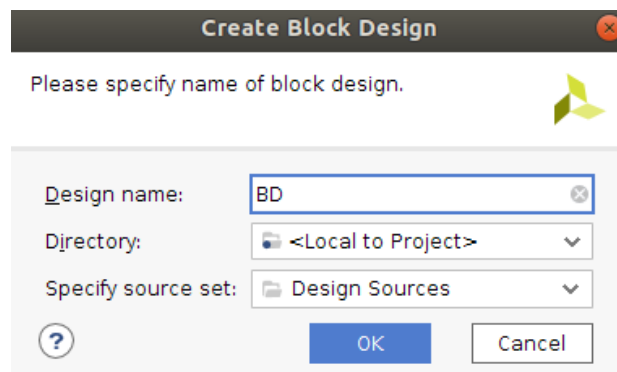


그림 24. 블록 디자인의 이름과 디렉토리 선택

이제 빈 블록 디자인 다이어그램 패널이 표시됩니다. (그림 25 참조)

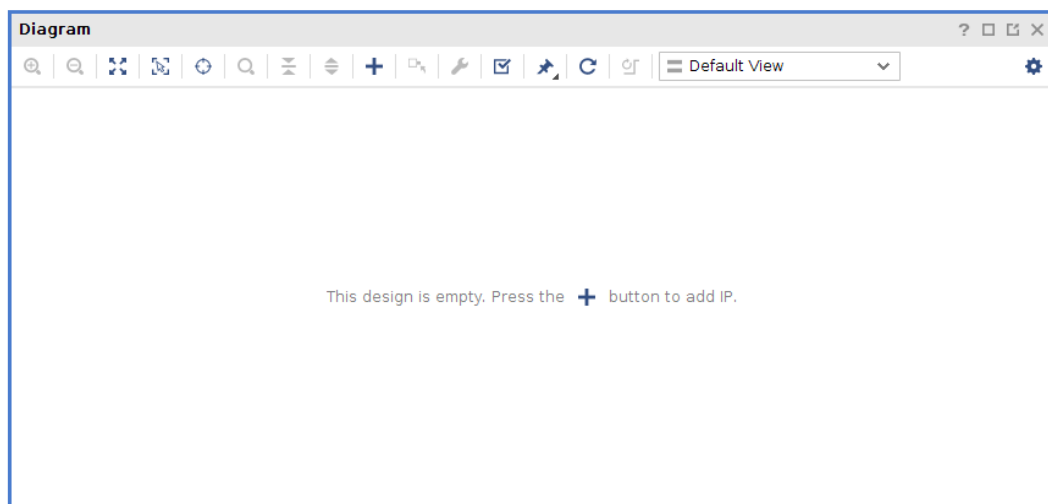


그림 25. 빈(비어 있는) 블록 디자인

3 단계. 블록 디자인에 모듈 추가

이제 블록 디자인에 모듈을 추가할 수 있습니다. 빈 블록 디자인을 마우스 오른쪽 버튼으로 클릭하고 **"Add Module"** 옵션을 선택하면 됩니다(그림 26 참조).

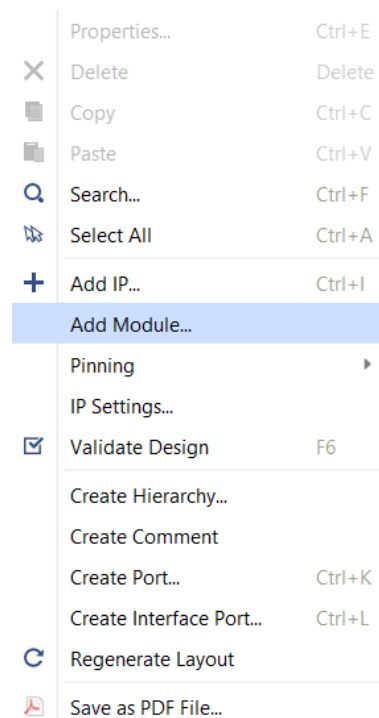


그림 26. 모듈 추가

대화 상자가 나타납니다. 아래로 스크롤하거나 검색 상자에 추가하려는 필수 모듈의 이름을 입력할 수 있습니다. SweRV EH1 Core Complex 를 추가하는 것으로 시작하겠습니다.

"swerv_wrapper_verilog"를 선택하고 확인을 클릭합니다.

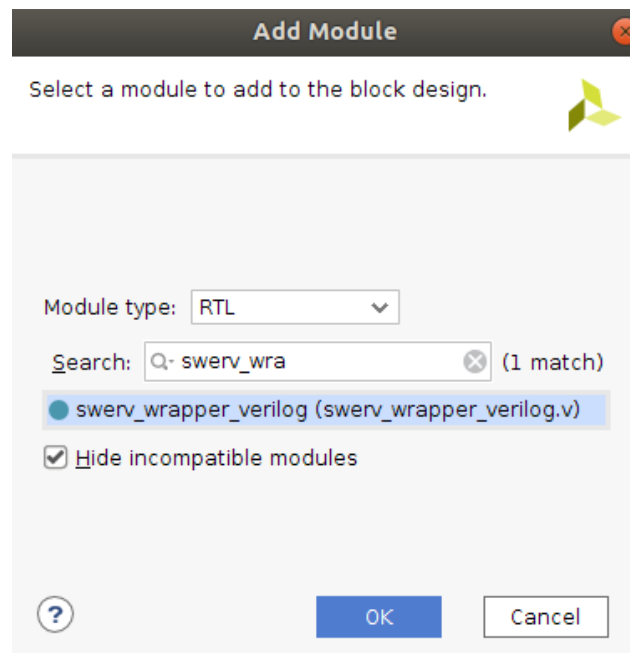


그림 27. swerv_wrapper_verilog 추가

심각한 경고 메시지가 나타납니다(그림 28 참조). 이 경고 메시지를 무시하려면 OK 를 클릭하십시오.

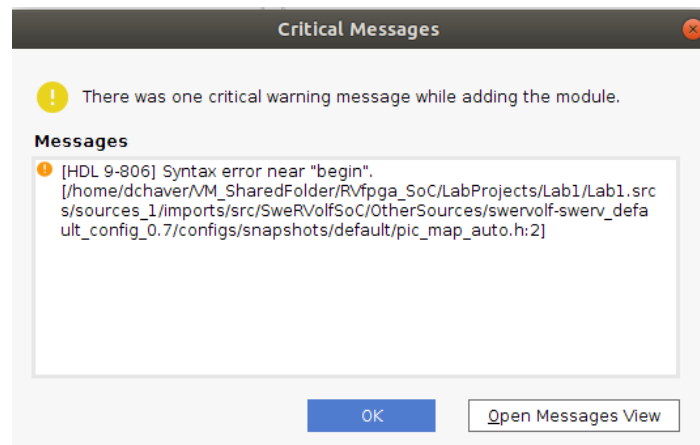


그림 28. 심각한 경고 메시지

모듈을 추가한 후 모듈의 "+" 아이콘을 클릭하여 "ifu_axi", "lsu_axi" 또는 "sb_axi"의 모든 핀을 시각화하고 액세스할 수 있습니다.

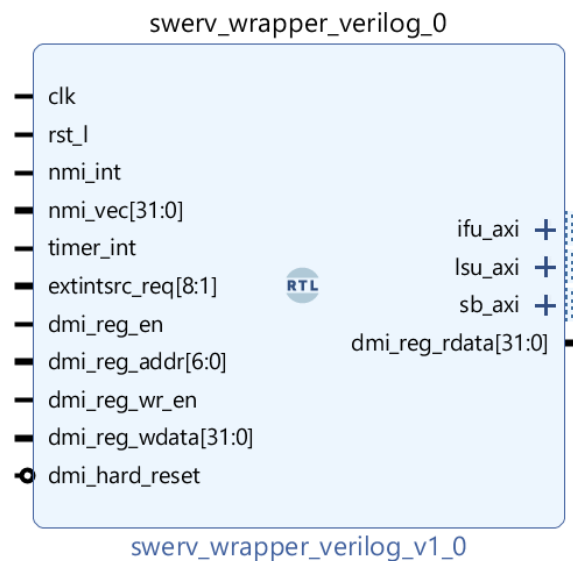
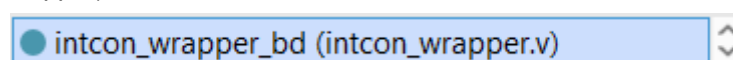


그림 29. "swerv_wrapper_verilog" 모듈

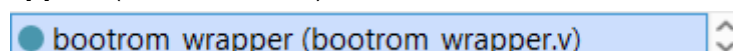
마찬가지로 이제 다음 모듈을 추가합니다.

- "intcon_wrapper_bd"(Interconnect Wrapper Module): 3 개의 Interconnect 모듈을 모두 포함하는 래퍼(wrapper) 모듈입니다.



이제 SoC 에 필요한 주변 장치를 추가합니다:

- "bootrom_wrapper" (Boot-ROM 모듈)



- “gpio_wrapper” (GPIO 탭 모듈)

● gpio_wrapper (gpio_wrapper.v)

- “syscon_wrapper” (시스템 컨트롤 모듈)

● syscon_wrapper (syscon_wrapper.v)

GPIO 모듈과 연결할 32 개의 "bidirec" 모듈을 추가합니다. 이 중 16 개는 LED 용이고 16 개는 스위치용입니다.

- "bidirec"(양방향 GPIO 모듈)

● bidirec (bidir.v)

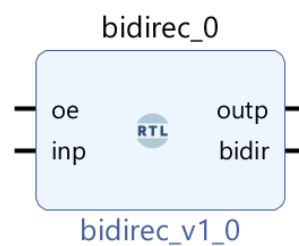


그림 30. bidir GPIO 모듈

유사하게, 우리는 이 모듈 중 32 개를 블록 디자인에 추가할 것입니다.

이 32 개의 모듈을 추가하는 빠른 방법은 다이어그램의 블록을 복사하여 붙여 넣는 것입니다. 먼저 1 개의 "bidirec" 블록을 복사하여 붙여 넣은 다음, 다시 2 개의 블록을 복사하여 붙여 넣고, "bidirec" 모듈의 32 개 블록이 블록 디자인에 추가될 때까지 복사 및 붙여 넣기 과정을 반복합니다.

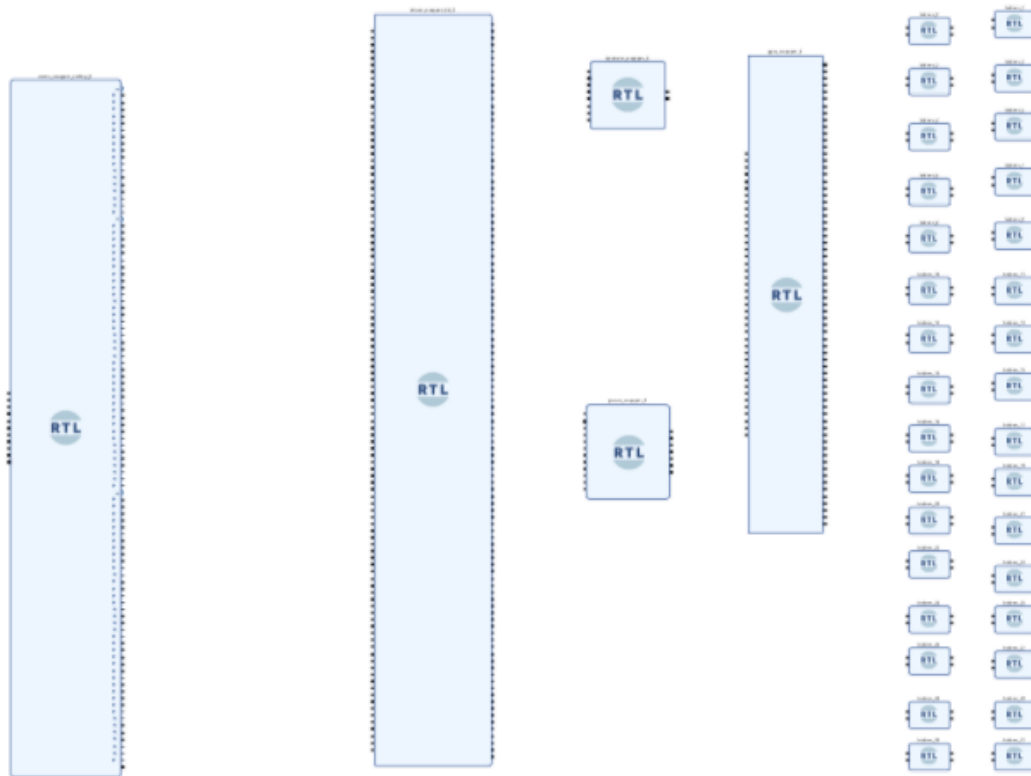


그림 31. 블록 디자인에 추가된 필수 모듈

(그림 31 참조) 왼쪽부터 시작하여 **SweRV Core** 모듈(`swerv_wrapper_verilog_0`)을 봅니다. 그런 다음 오른쪽에서 **Interconnect Wrapper** 모듈(`intcon_wrapper_bd_0`)과 **Boot-ROM**(`bootrom_wrapper_0`) 모듈, **System Controller** (`syscon_wrapper_0`) 모듈, **GPIO** (`gpio_wrapper_0`) 모듈인 4 개의 주변 장치 모듈을 봅니다. 가장 오른쪽에 32 개의 **Bidirec** (`bidirec_x`) 모듈이 표시됩니다.

4 단계. 모듈 연결

이제 모듈을 핀별로 또는 경우에 따라 버스별로 배선합니다. "`swerv_wrapper_verilog`"를 "`intcon_wrapper_bd`"와 연결하기 시작합니다.

코어의 다음 하위 모듈과 관련된 이러한 모듈 간에 세 가지 다른 핀 세트를 연결해야 합니다.

- **IFU** (Instruction Fetch Unit)
- **LSU** (Load Store Unit)
- **SB** (Store Byte).

먼저 IFU 와 관련된 핀을 연결하는 것부터 시작합니다. "`swerv_wrapper_verilog`" 모듈의 "`ifu_axi_arid[2:0]`" 핀을 "`intcon_wrapper_bd`"의 "`i_ifu_arid[2:0]`" 핀에 연결합니다.

비슷하게,

`ifu_axi_araddr[31:0]`은 `i_ifu_araddr[31:0]`에 연결되며,

`ifu_axi_arlen[7:0]`은 `i_ifu_arlen[7:0]`에 연결되며,

`ifu_axi_arsize[2:0]`은 `i_ifu_arsize[2:0]` 등에 연결됩니다(그림 32 참조).

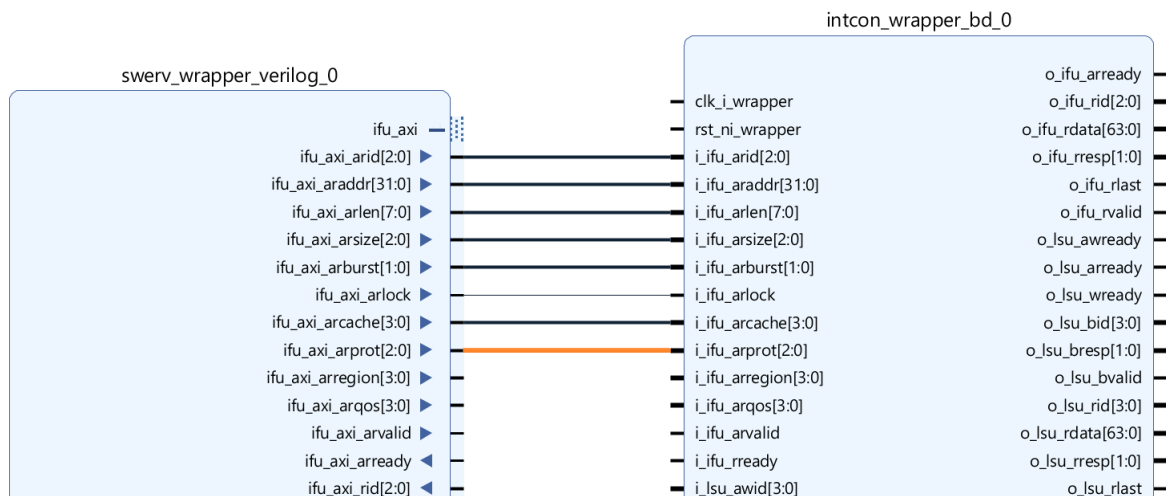


그림 32. 해당 핀 연결

마찬가지로 "swerv_wrapper_verilog"의 모든 IFU(Instruction Fetch Unit) 핀을 "intcon_wrapper_bd"의 IFU 핀과 연결합니다(그림 33 참조).

PDF: 배선 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다:
 [RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
 /InternalConnections/1_SwervW_IntconW_IFU.pdf

이제 **"swerv_wrapper_verilog"**의 모든 **LSU(Load Store Unit)** 핀을 **"intcon_wrapper_bd"**의 **LSU** 핀에 연결하도록 이동합니다. **IFU** 핀에 대해 수행한 것과 동일한 프로세스를 수행하여 **"swerv_wrapper_verilog"** 모듈의 각 LSU 핀을 **"intcon_wrapper_bd"** 모듈의 해당 핀과 연결합니다(그림 34 참조).

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/](#)
[InternalConnections/2 SwervW IntconW LSU.pdf](#)

이제 **SB** 핀 연결을 진행합니다. 유사하게 "**swerv_wrapper_verilog**"의 모든 **SB** 핀을 "**intcon_wrapper_bd**"의 해당 **SB** 핀에 연결합니다(그림 35 참조).

Imagination University Programme – RVfpga Lab 1: Introduction to RVfpga-SoC
Version 1.0 – 15th July 2021
© Copyright Imagination Technologies

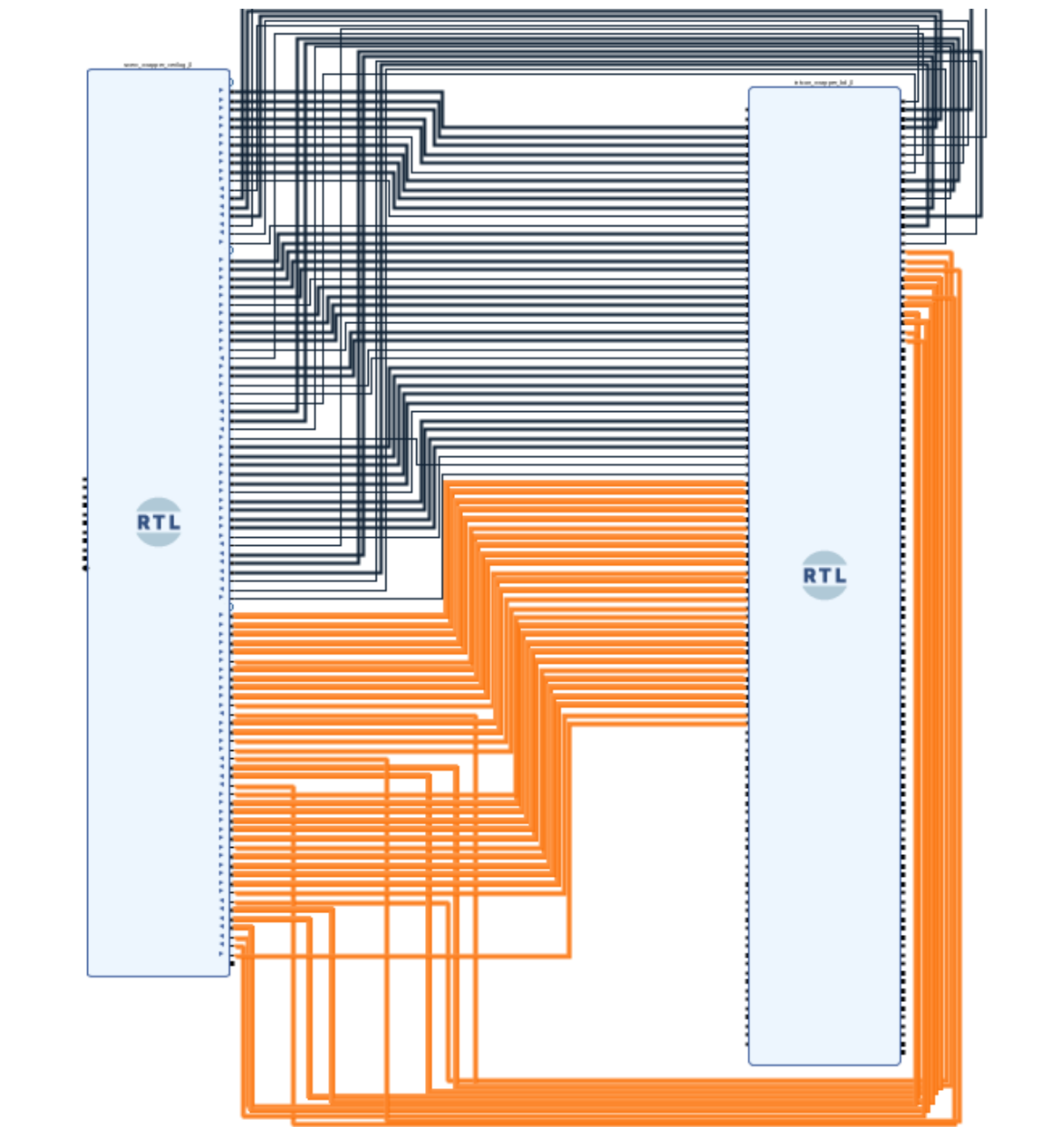


그림 35. 모든 SB 핀 연결

다음으로 "Intcon_wrapper_bd"로 주변 장치를 연결합니다. "Intcon_wrapper_bd"의 "wb_rom_xxx_x" 와이어를 결합하여 "bootrom_wrapper" 모듈부터 시작합니다(그림 36 참조).

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /InternalConnections/4_BootRomW_IntconW.pdf](#)

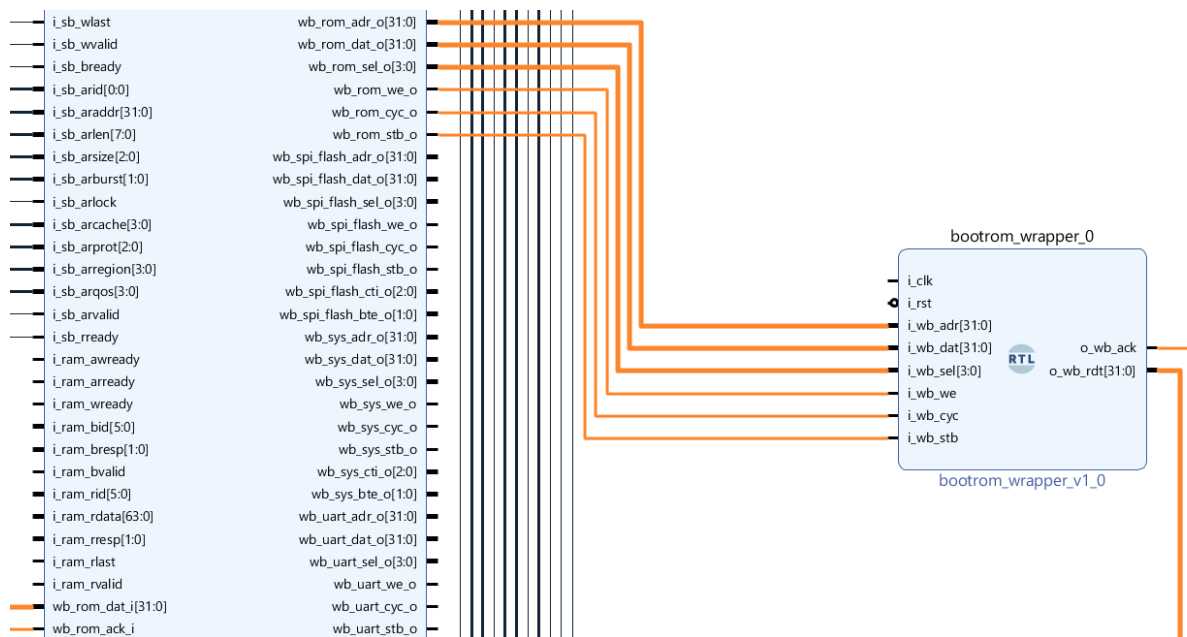


그림 36. BootROM 모듈을 Interconnect Wrapper 모듈과 연결

이제 "syscon_wrapper" 모듈을 "Intcon _wrapper_bd" 모듈과 연결합니다(그림 37 참조).

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /InternalConnections/5_SysconW_IntconW.pdf

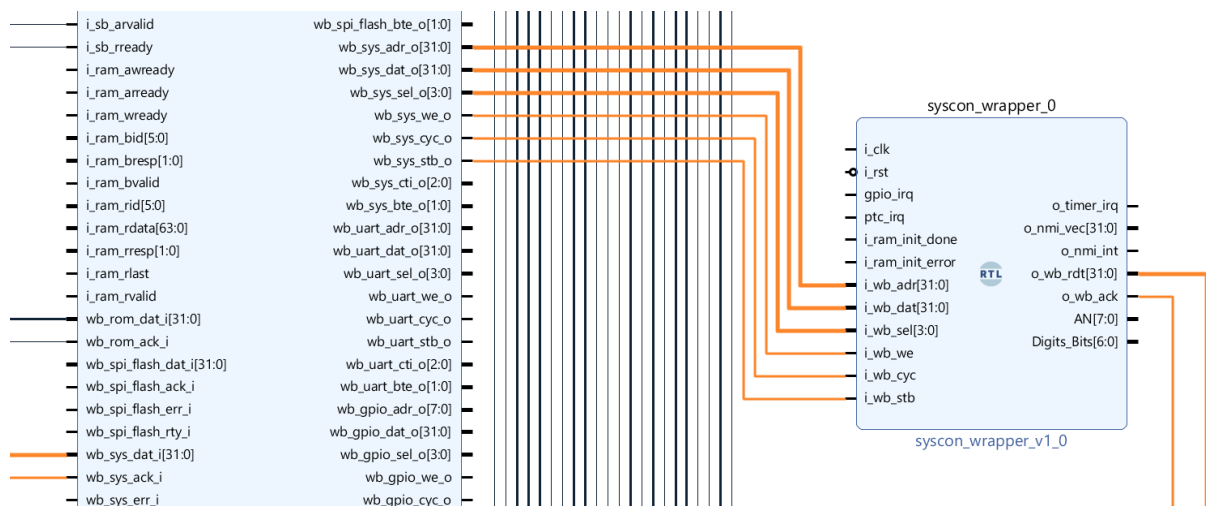


그림 37. WB Interconnect 핀으로 Syscon 연결

"syscon_wrapper"의 다음 핀은 "swerv_wrapper_verilog"에 연결됩니다(그림 38 참조).

- o_timer_irq
- o_nmi_vec[31:0]
- o_nmi_int

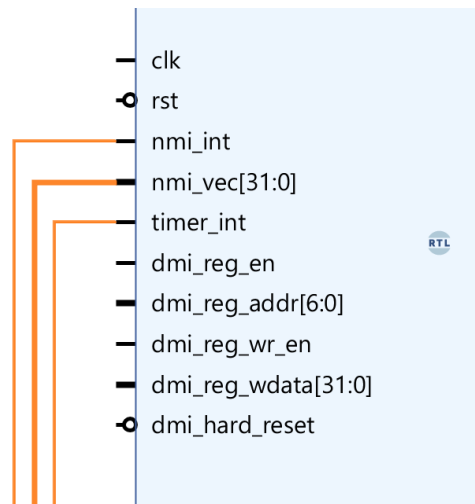


그림 38. syscon_wrapper 를 swerv_wrapper_verilog 핀과 연결

이제 "gpio_wrapper" 모듈을 "intcon_wrapper_bd"와 연결합니다. "intcon_wrapper_bd" 모듈의 "wb_gpio_XXX_x" 핀을 "gpio_wrapper" 모듈 핀과 연결합니다(그림 39 참조).

"gpio_wrapper" 모듈의 "wb_inta_o" 핀을 "syscon_wrapper" 모듈의 "gpio_irq" 핀과 연결합니다.

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/InternalConnections/6_GpioW_IntconW.pdf

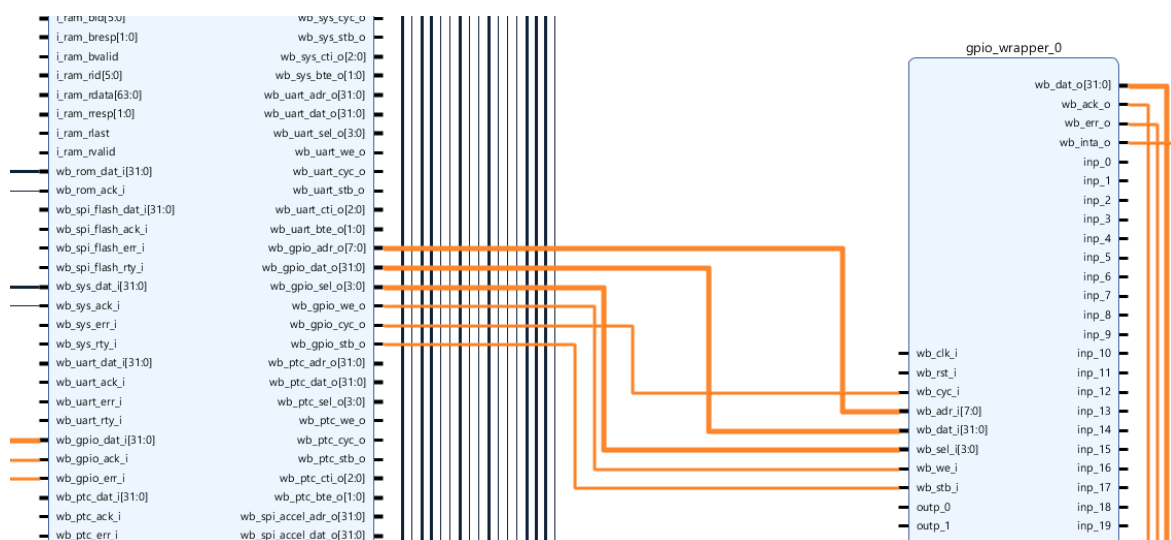


그림 39. gpio_wrapper 를 intcon_wrapper 핀과 연결

32 개의 GPIO "bidirec" 모듈을 이미 연결한 "gpio_wrapper" 모듈과 연결할 것입니다. 특히 "gpio_wrapper" 모듈을 "bidirec_x" 모듈과 연결합니다. 여기서 x 는 1 에서 32 사이의 숫자입니다. 연결은 다음과 같이 진행됩니다.

“gpio_wrapper_0”의 “inp_0” 핀은 “bidirec_0”의 “inp”에 연결되고,
 “gpio_wrapper_0”의 “inp_1” 핀은 “bidirec_1”의 “inp”에 연결되고, 이러한 연결은
 “gpio_wrapper”의 “inp_31”인 마지막 “bidirec_31”의 “inp” 연결까지 갈 것입니다.

비슷하게,

“gpio_wrapper_0”의 “oe_0” 핀은 “bidirec_0”의 “oe”에 연결되고,
 “gpio_wrapper_0”의 “oe_1” 핀은 “bidirec_1”의 “oe”에 연결되며, 이러한 연결은 “gpio_wrapper”의
 “oe_31”인 마지막 “oe” 연결까지 이어지며 “bidirec_31”의 “oe”에 연결됩니다.

그리고 마찬가지로 비슷하게,

“gpio_wrapper_0”의 “outp_0” 핀은 “bidirec_0”의 “outp”에 연결되고,
 “gpio_wrapper_0”의 “outp_1” 핀은 “bidirec_1”의 “outp”에 연결되고, 이러한 연결은
 “gpio_wrapper”의 “outp_31”인 마지막 “bidirec_31”의 “outp” 연결까지 갈 것입니다.

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF는 여기에서 사용할 수 있습니다.:
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /InternalConnections/7_GpioW_32xBidirec.pdf](#)

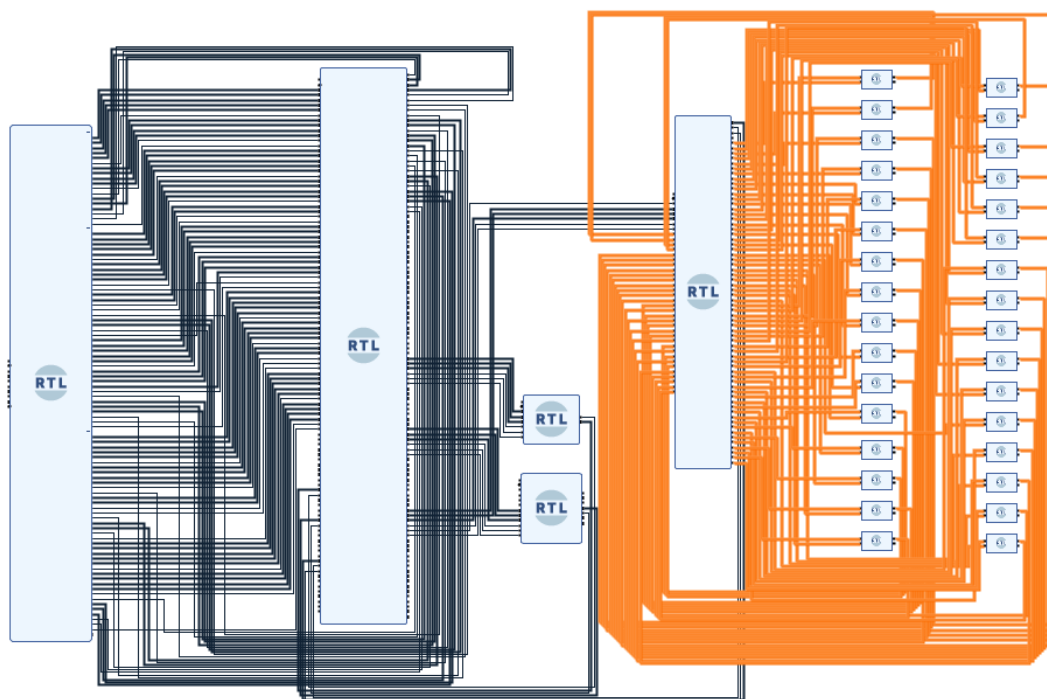


그림 40. gpio_wrapper 모듈에 연결된 모든 GPIO Bidirec 모듈

모듈 간의 모든 내부 연결을 연결했으므로 이제 외부 연결을 만듭니다.

5 단계. IOPin 에 대한 외부 연결 만들기

이제 블록 디자인으로 들어오는 핀을 입력으로 연결하거나 블록 디자인에서 나가는 핀을 출력으로 연결할 차례입니다. 이 핀을 외부 핀/포트로 연결합니다. 이러한 외부 핀에는 **RAM(DDR)**, **CLK(Clock)**, **RST(Reset)** 및 **DMI(Debug Module interface)** 핀이 포함됩니다.

"clk"핀을 연결하는 것으로 시작합니다. "**swerv_wrapper_verilog**" 모듈로 이동하여 "**clk**" 핀을 마우스 오른쪽 버튼으로 클릭하면 드롭다운이 표시됩니다(그림 41 참조).

모든 드롭다운 옵션 중에서 외부 만들기(Make External) 옵션을 선택합니다. 핀을 마우스 왼쪽 버튼으로 클릭하고 단축키 "**CTRL + T**"를 사용하여 핀을 외부로 만들 수도 있습니다.

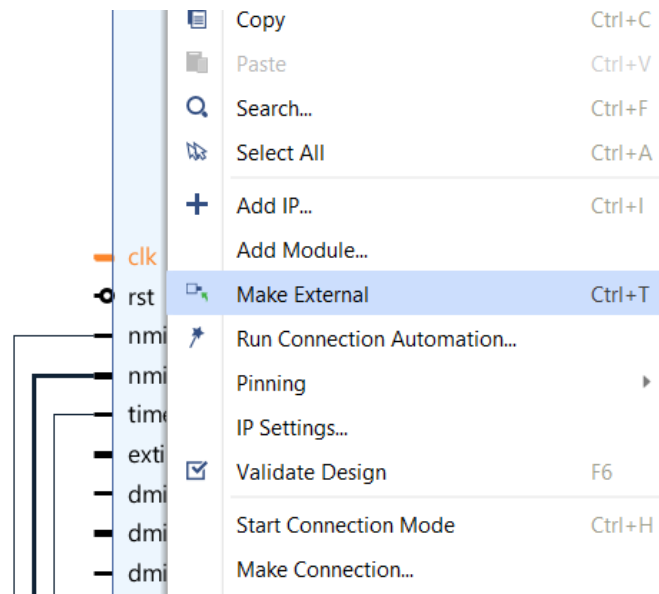


그림 41. "clk" 외부 연결 만들기

이제 외부 핀 "**clk_0**"에 연결된 "**swerv_wrapper_verilog**"의 "**clk**" 핀을 볼 수 있습니다(그림 42 참조).

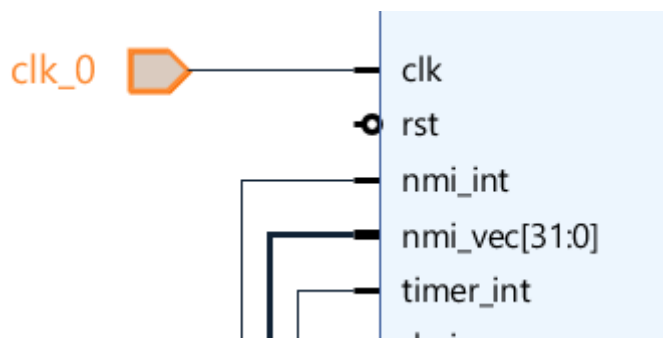


그림 42. "clk"가 외부 연결이 됨

이제 "**clk**" 외부 핀을 **intcon_wrapper_bd**, **syscon_wrapper**, **bootrom_wrapper** 및 **gpio_wrapper** 를 포함한 나머지 모듈에 연결할 수 있습니다.

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/ExternalConnections/1_Clock.pdf

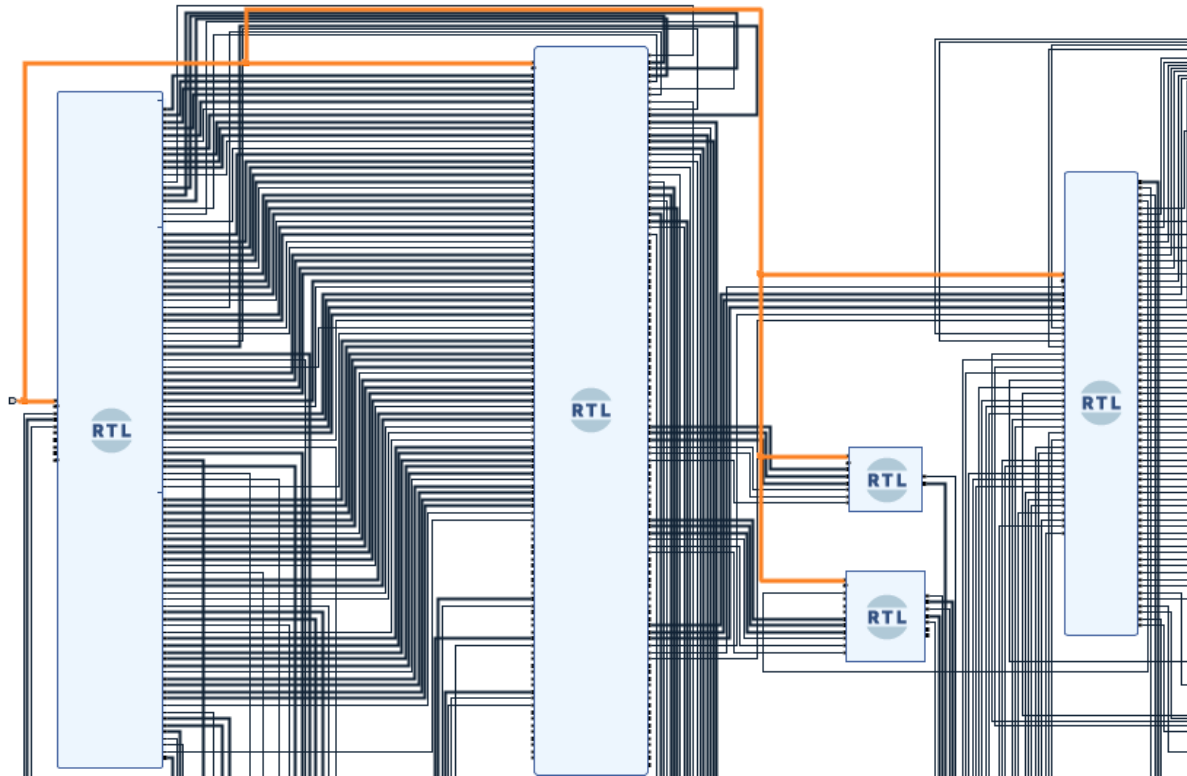


그림 43. 모든 모듈에 연결된 신호 clk

마찬가지로 "rst" 핀을 모든 모듈에 연결할 수 있습니다.

"clk"용으로 만든 외부 핀과 마찬가지로 "rst"용을 하나 만듭니다. 이제 다시 "swerv_wrapper_verilog" 모듈로 이동하여 "rst" 핀을 마우스 오른쪽 버튼으로 클릭하고 외부 핀으로 만듭니다.

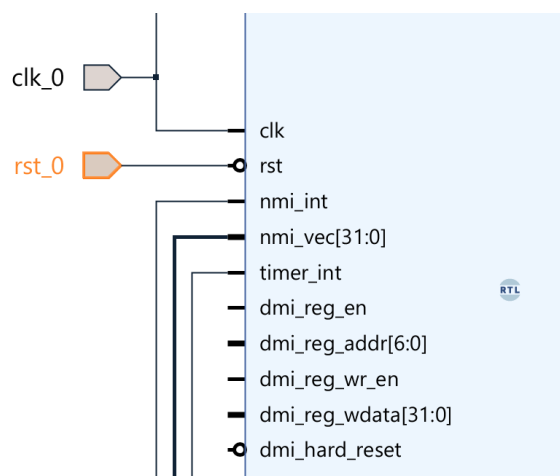


그림 44. rst_1 을 외부 핀으로 만들기

이제 **"rst_0"** 외부 핀을 `intcon_wrapper`, `syscon_wrapper`, `bootrom_wrapper` 및 `gpio_wrapper` 를 포함한 나머지 모듈에 연결합니다.

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/2_Reset.pdf](#)

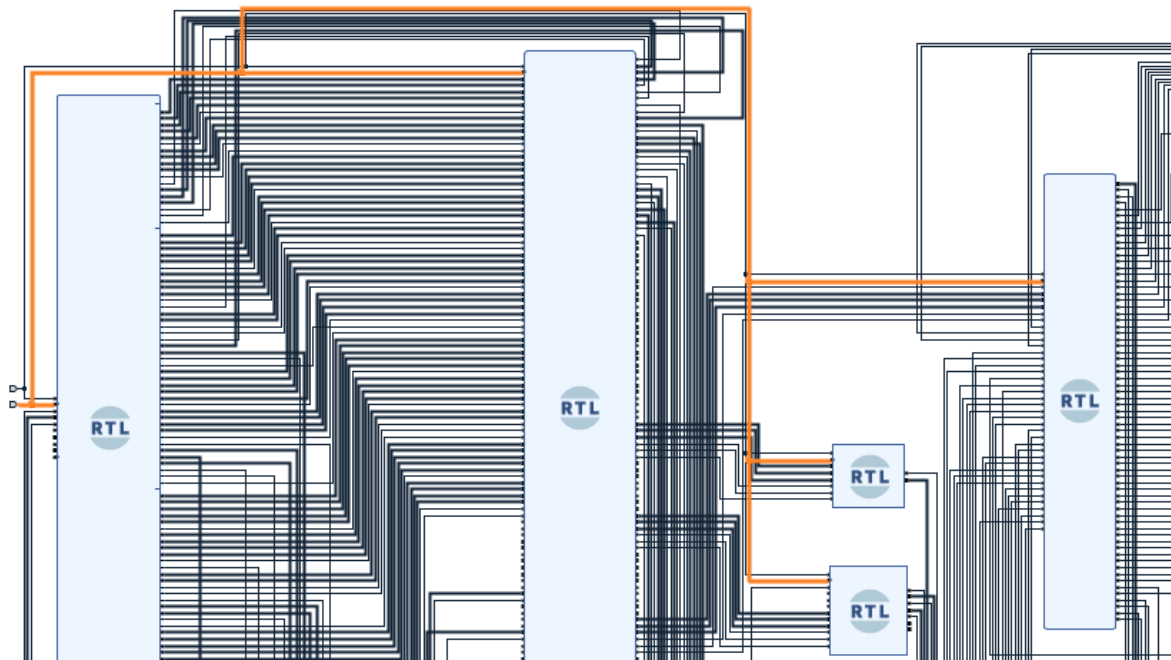


그림 45. 반전된 "rst_0" 핀을 나머지 모듈에 연결

이제 다음 단계를 완료하여 **"Intcon_wrapper_bd"** 모듈의 모든 RAM(DDR) 핀을 외부 RAM 핀에 연결합니다.

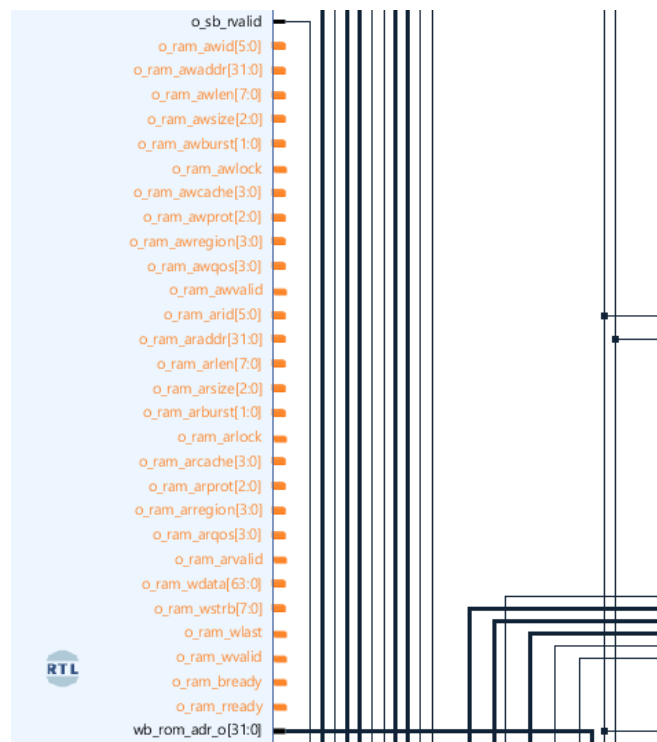


그림 46. Intcon 래퍼 오른쪽 RAM 핀

이제 "Intcon_wrapper_bd" 모듈의 모든 오른쪽 RAM 핀을 외부 핀으로 만듭니다(그림 47 참조).

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/ExternalConnections/3_RAM_R.pdf

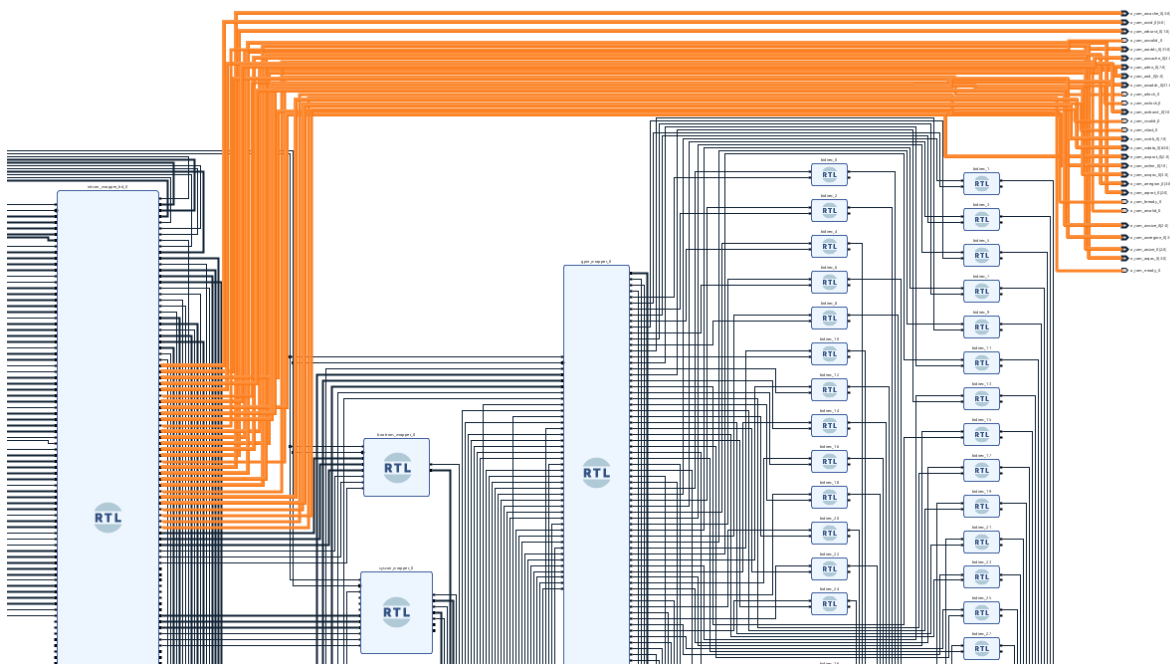


그림 47. 모든 오른쪽 RAM 핀을 외부로 설정

이제 "Intcon_wrapper_bd"의 왼쪽 RAM 핀을 외부 핀으로 만듭니다.

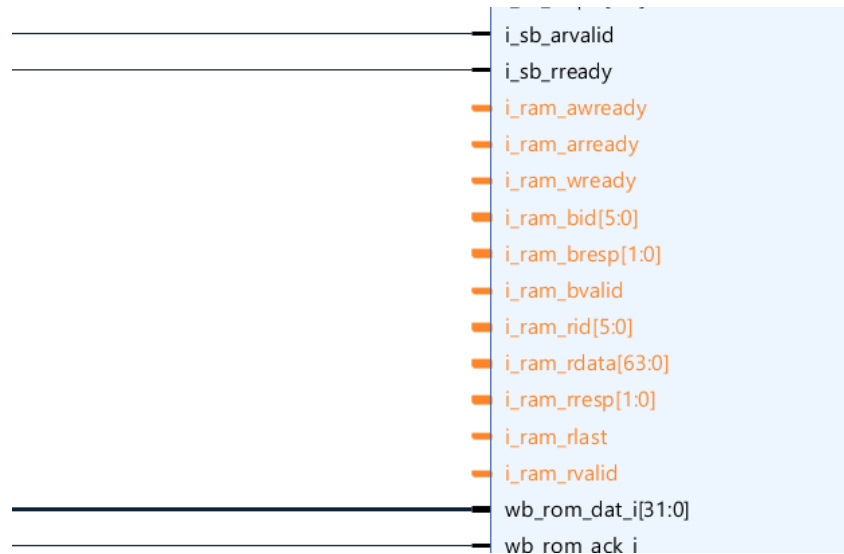


그림 48. 상호 연결 래퍼의 왼쪽 RAM 핀

이 모든 RAM 핀을 외부 핀으로 만들 것입니다(그림 49 참조).

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/4_RAM_L.pdf](#)

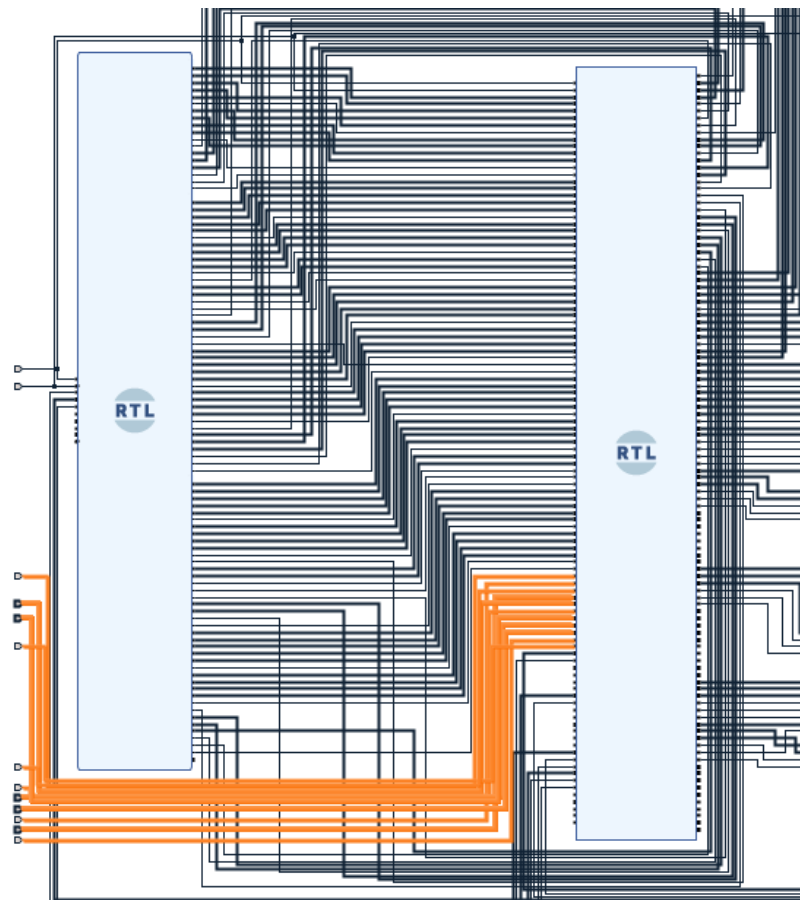


그림 49. 모든 왼쪽 RAM 핀을 외부로 설정

이제 "swerv_wrapper_verilog" 모듈의 DMI 핀을 외부 핀과 연결합니다.

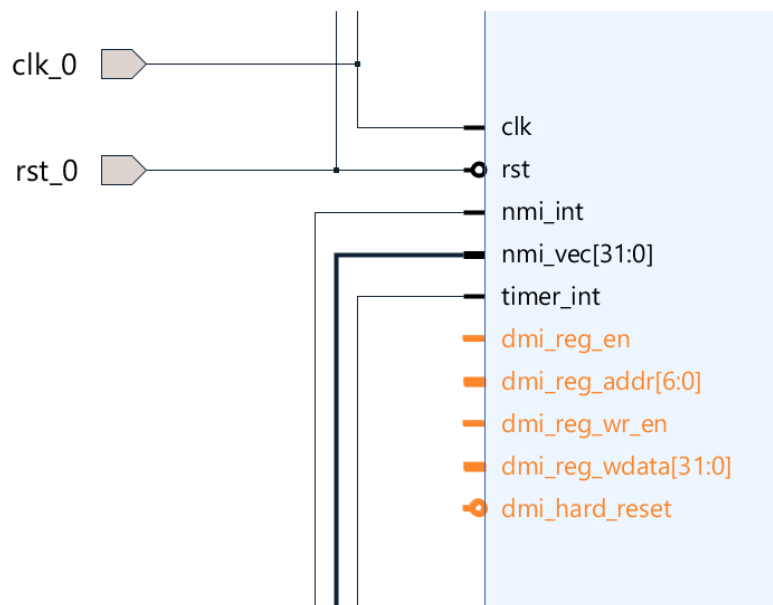


그림 50. dmi 핀 op swerv_wrapper_verilog(왼쪽)

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다:
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/ExternalConnections/5_DMI.pdf

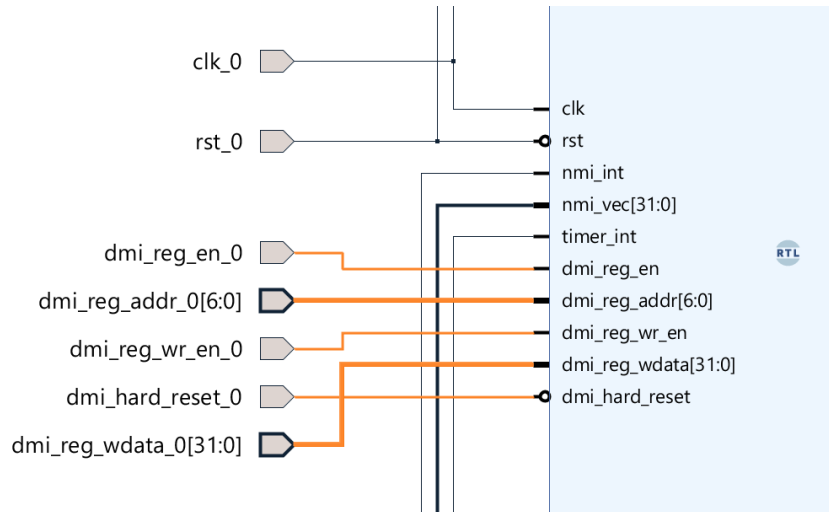


그림 51. dmi 핀을 외부 핀으로 만들기

"swerv_wrapper_verilog" 모듈의 오른쪽 하단에 있는 외부 핀에 핀을 하나 더 연결합니다. 이 핀은 "dmi_reg_rdata[31:0]"입니다..

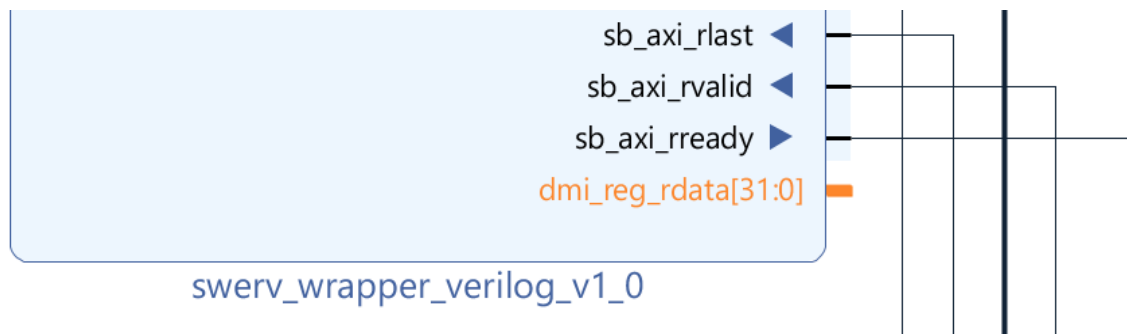


그림 52. "dmi_reg_rdata[31:0]" 핀(swerv_wrapper_verilog의 오른쪽)

"dmi_reg_rdata[31:0]"도 외부 핀으로 만들 것입니다.

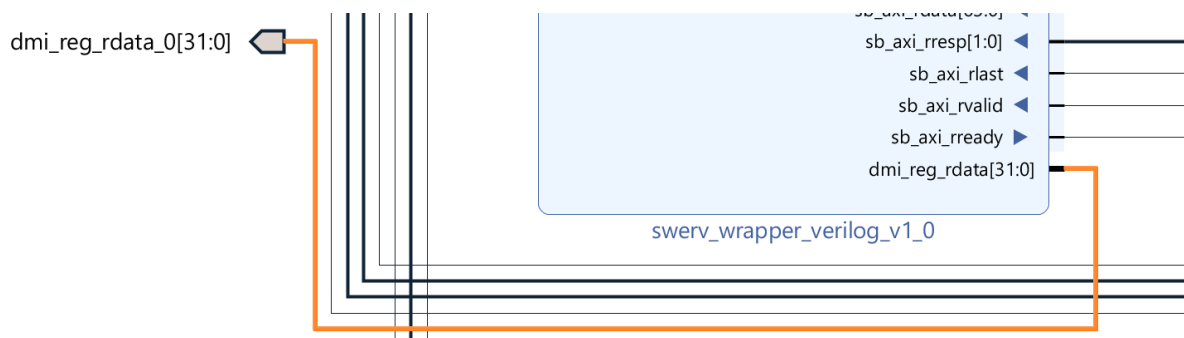


그림 53. "dmi_reg_rdata[31:0]" 핀을 외부 핀으로 만들기

이제 "syscon_wrapper" 모듈의 다음 핀을 외부 핀으로 만듭니다.

- i_ram_init_done
- i_ram_init_error
- AN[7:0]
- Digital_Bits[6:0]

PDF: 배선의 세부 사항을 보여주는 블록 설계의 상세 PDF 는 여기에서 사용할 수 있습니다.:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/ExternalConnections/6_SysconW_External.pdf

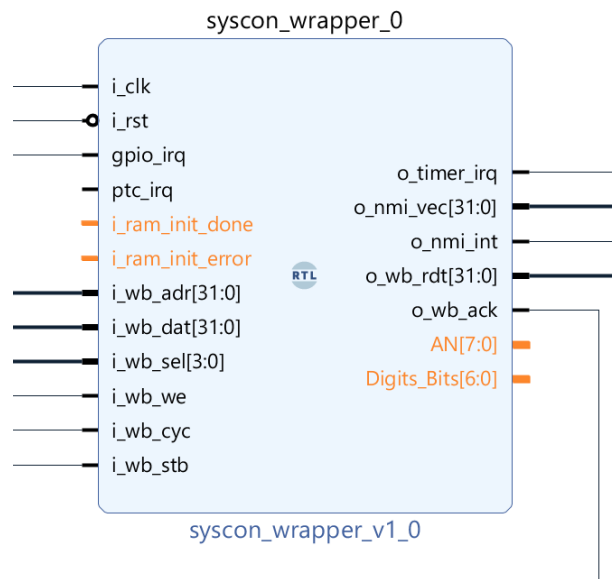


그림 54. syscon_wrapper 의 외부 핀

마지막으로 남은 연결은 모든 "bidirec" 모듈의 모든 "bidir" 핀을 외부 핀으로 만드는 것입니다.

참고: "bidirec_0" 모듈에서 시작하여 이러한 연결을 하나씩 외부로 만듭니다.

따라서 "bidirec_0" 모듈의 "bidir" 핀이 외부 핀인 "bidir_0" 에 연결됩니다.

그런 다음 "bidirec_1"의 "bidir" 핀으로 이동하고 계속하여 연결합니다.

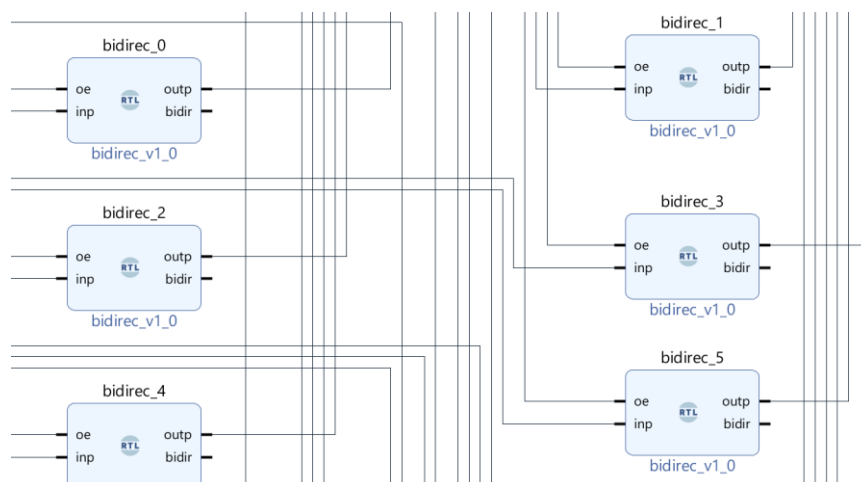


그림 55. GPIO Bidirec 모듈의 "bidir" 핀을 외부 핀으로 만들기

PDF: 배선의 세부 사항을 보여주는 블록 설계의 클로즈업된 PDF 는 여기에서 사용할 수 있습니다.:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/7_Bidir.pdf

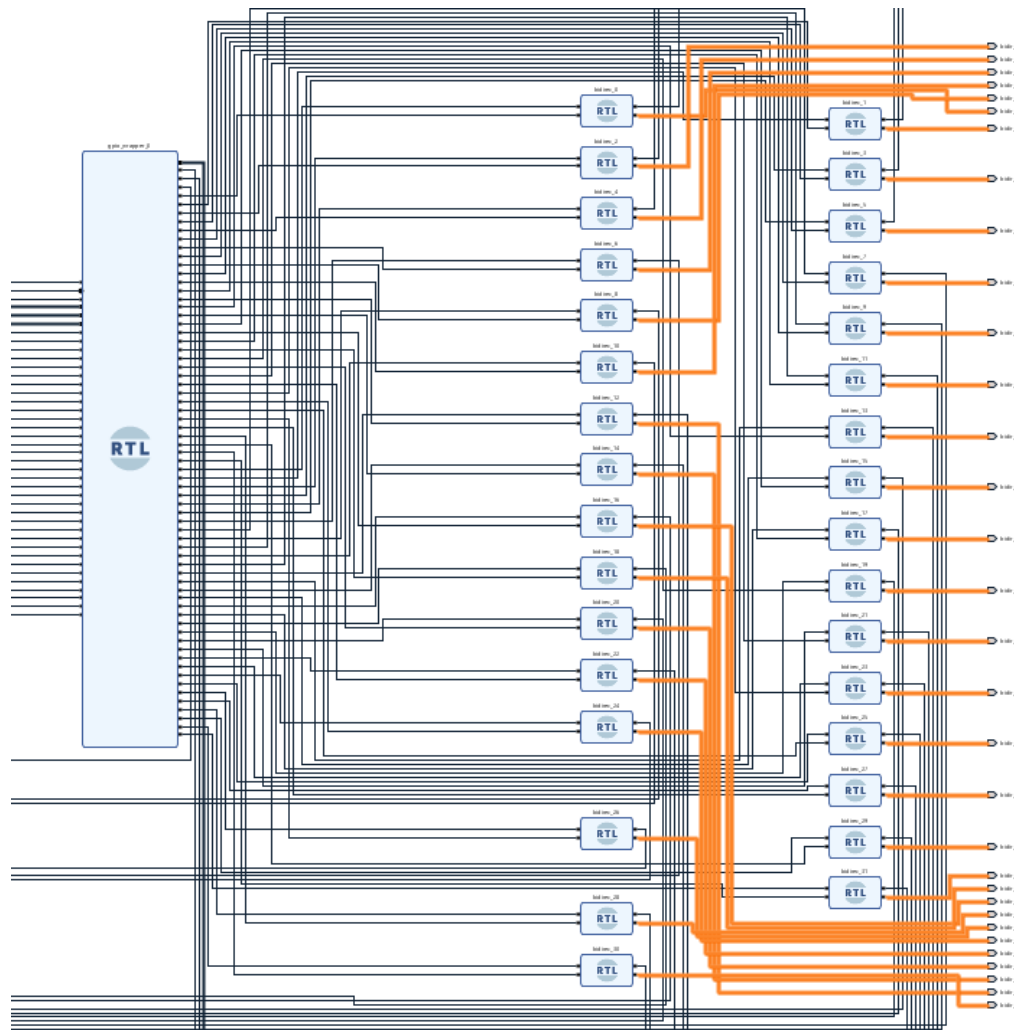


그림 56. "bidir" 외부 연결 만들기

이제 블록 설계 SoC 에 대한 내부 및 외부 연결을 모두 완료했습니다. "Ctrl + S"를 눌러 블록 디자인을 저장합니다.

SweRVolfX SoC 가 완성된 후, 모델링된 블록 디자인에는 다음과 같은 연결된 모듈이 포함됩니다.

- 1 SweRV Core (swerv_wrapper_verilog)
- 1 Interconnect Wrapper (intcon_wrapper_bd)
- 1 Boot-ROM (bootrom_wrapper)
- 1 GPIO Top Module (gpio_wrapper)
- 1 System Controller (syscon_wrapper)
- 32 Bidirec Gpio Module (bidirec) (그림 57 참조).

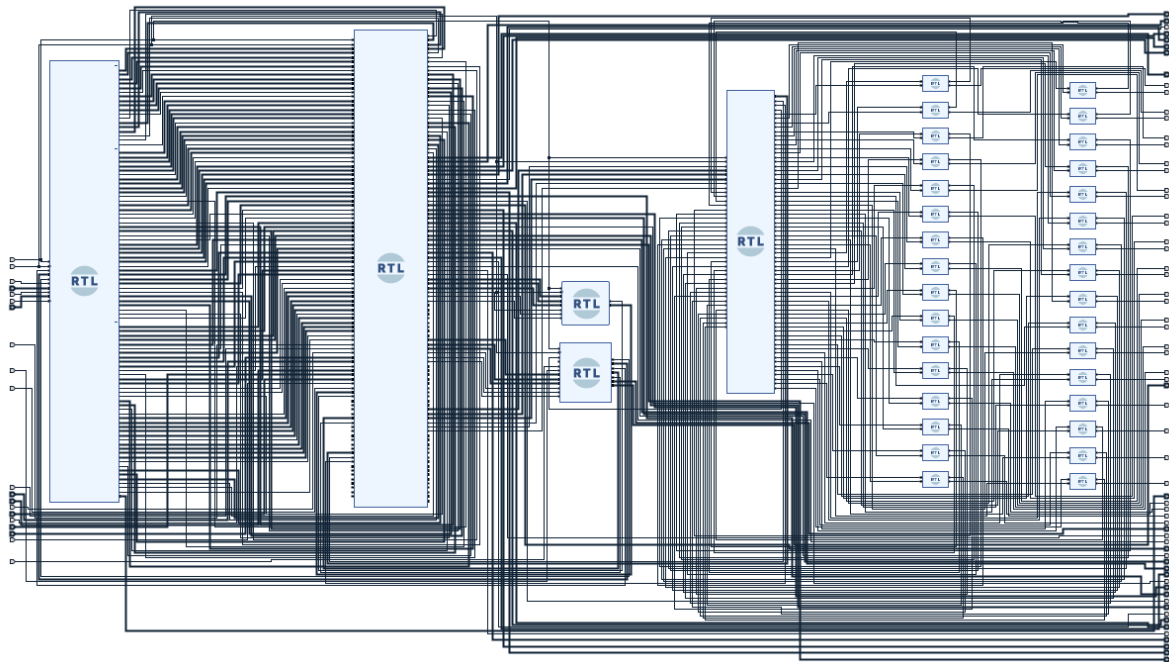


그림 57. 블록 설계 SoC 완성

5. 블록 디자인 모듈 Verilog 파일 생성

이제 생성한 블록 디자인의 Verilog 모듈 파일을 생성합니다.

1 단계. 소스 패널로 이동하여 방금 생성한 블록 디자인 모듈 "BD"를 찾습니다.

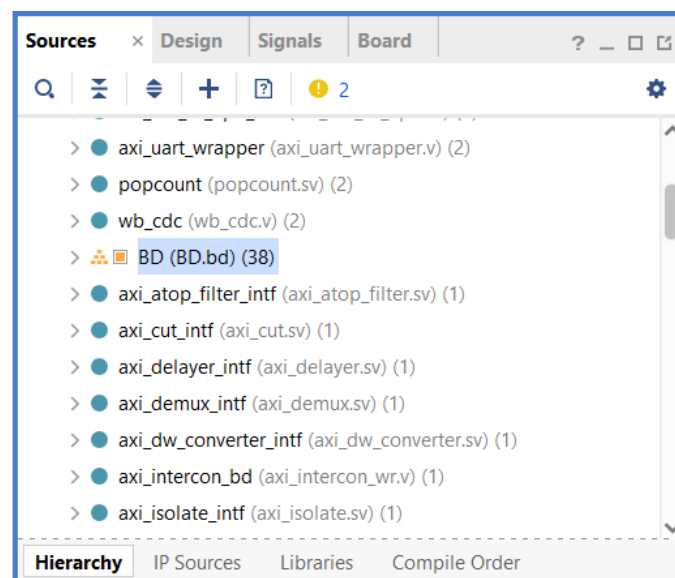


그림 58. 소스에서 "BD" 찾기

2 단계. 이제 해당 블록 디자인(BD)을 마우스 오른쪽 버튼으로 클릭한 다음 " **Create HDL Wrapper** "을 선택합니다(그림 59 참조).

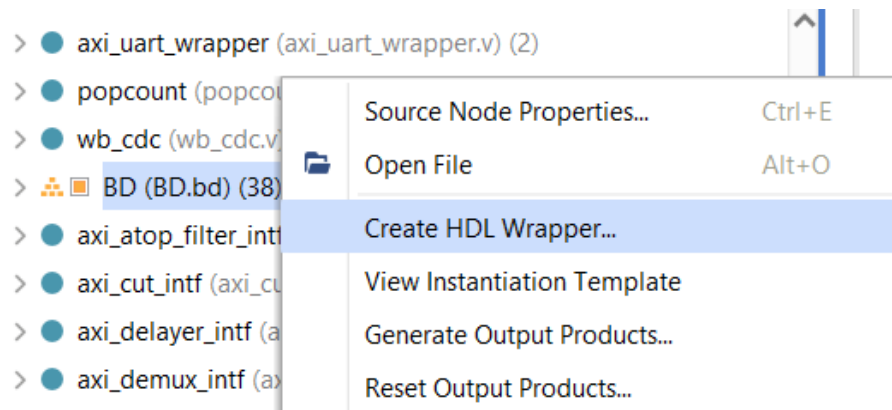


그림 59. HDL 래퍼 생성

3 단계. "Let Vivado manage wrapper and auto-update" 옵션을 선택하고 확인을 클릭하여 계속 진행합니다.

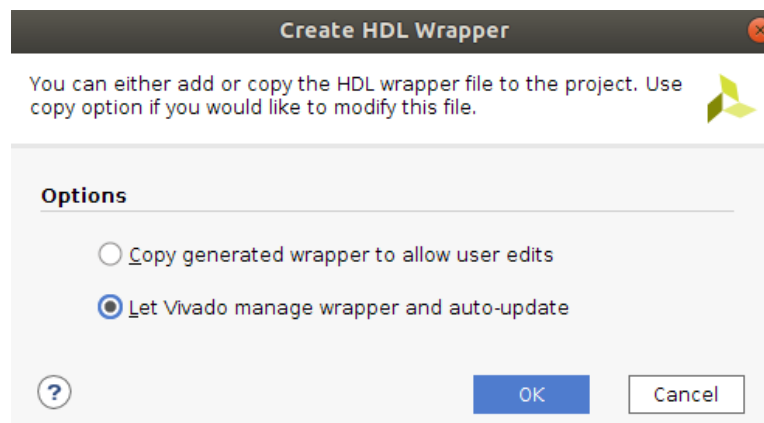


그림 60. 두 번째 옵션 선택

블록 설계에서 여러 핀을 연결하지 않은 상태로 두어 이러한 핀이 자동으로 "0"(접지)에 연결되기 때문에 중요한 경고 팝업이 표시됩니다.

OK 를 클릭합니다.

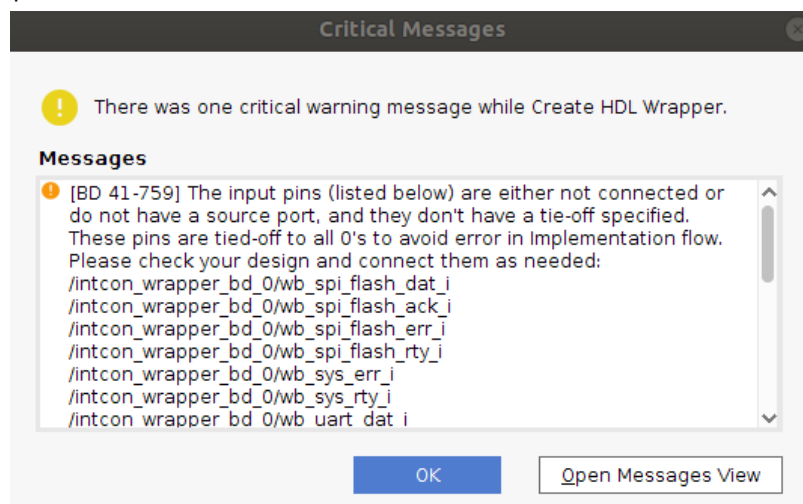


그림 61. 경고 팝업

이제 블록 디자인의 HDL 래퍼가 생성되었습니다. 소스 패널로 이동하여 " **BD_wrapper** "가 표시될 때까지 아래로 스크롤할 수 있습니다. 옆에 있는 드롭다운 아이콘을 클릭한 다음 " **BD_i** "에 대해 다시 클릭합니다.

이제 " **BD (BD.v)** " 파일을 두 번 클릭하여 엽니다(그림 62 참조).

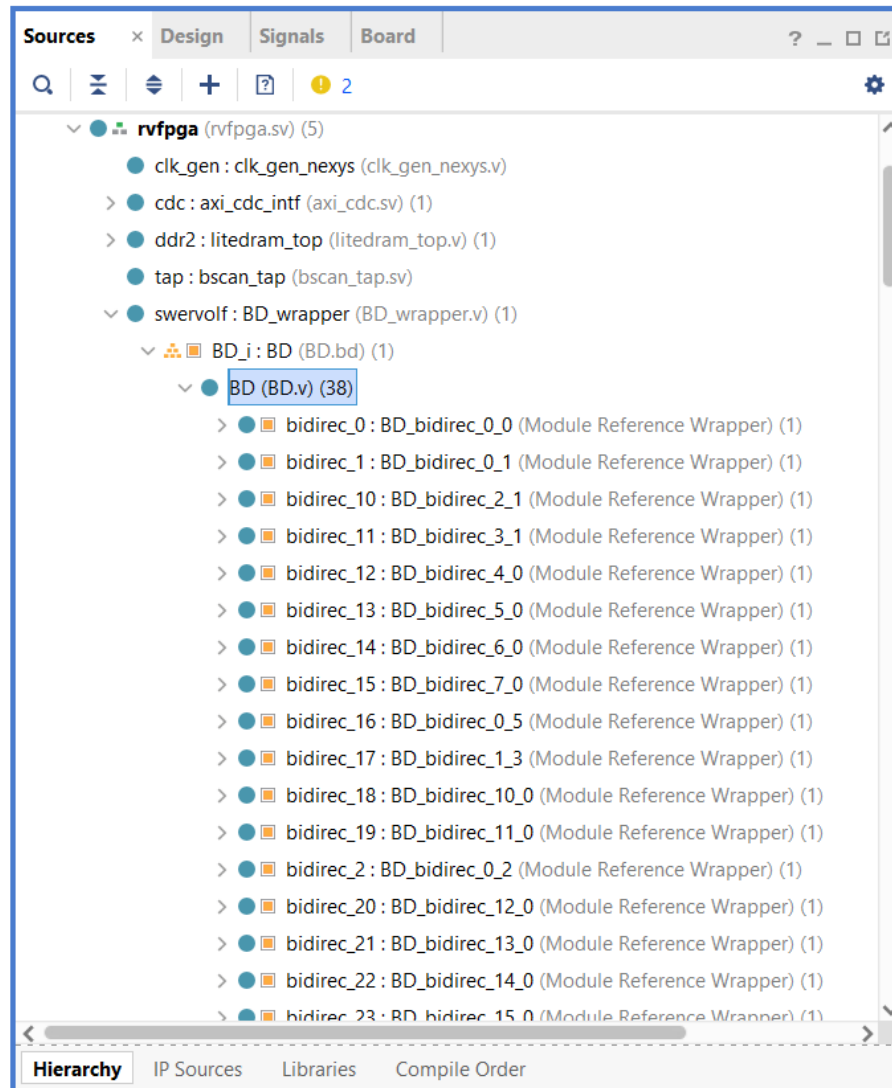


그림 62. 소스 패널에서 "BD.v" 찾기

여기에서 Vivado의 블록 디자인 도구를 사용하여 생성된 " **BD.v** " Verilog 파일을 볼 수 있습니다.



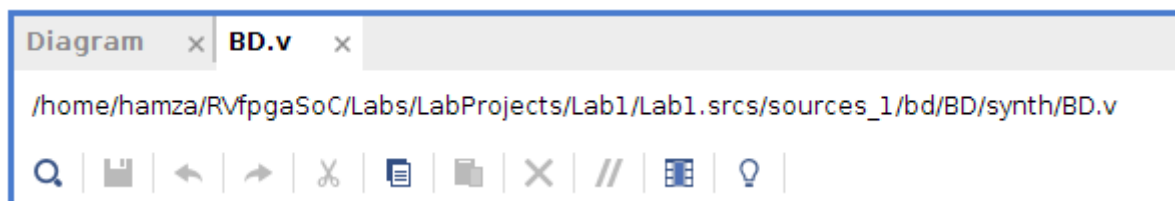
```

1 //Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
2 //
3 //Tool Version: Vivado v.2019.2 (lin64) Build 2708876 Wed Nov 6 21:39:14 MST 2019
4 //Date       : Wed Mar 24 20:17:38 2021
5 //Host       : hamza-lenovo running 64-bit Ubuntu 18.04.5 LTS
6 //Command    : generate_target BD.bd
7 //Design     : BD
8 //Purpose    : IP block netlist
9 //
10 timescale 1 ps / 1 ps
11
12 (* CORE_GENERATION_INFO = "BD,IP_Integrator,{x_ipVendor=xilinx.com,x_ipLibrary=BlockDiagram,x_ipName=BD,x_ipVersion=1.00.a,x_ipLanguage="
13 module BD
14 (AN 0,
15  Digits_Bits_0,
16  bidir_0,
17  bidir_1,
18  bidir_10,
19  bidir_11,
20  bidir_12,
21  bidir_13,
22  bidir_14,
23  bidir_15,
24  bidir_16,
25  bidir_17,
26  bidir_18,
27  bidir_19,
28  bidir_2,
29  bidir_20,
30  bidir_21,
31  bidir_22,
32  bidir_23,
33  bidir_24,
34  bidir_25,
35  bidir_26,
36  )

```

그림 63. "BD.v"

새로 생성된 파일의 경로는 파일 상단에서 확인할 수 있습니다. 다음 실습에서는 이 경로를 사용하여 이 "BD.v" 파일에 액세스합니다.



```

/home/hamza/RVfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources_1/bd/BD/synth/BD.v

```

그림 64. "BD.v" 파일의 경로

6. 비트스트림(Bitstream) 생성

이제 Vivado의 블록 디자인 도구를 사용하여 SweRVolfX 하위 집합을 생성하고 Verilog 래퍼를 생성했으므로 FPGA를 구성하는 데 사용할 비트스트림을 생성할 준비가 되었습니다. 비트스트림을 생성하려면 먼저 다음 단계를 완료하여 Vivado에서 일부 설정을 조정해야 합니다.

1 단계. 설정으로 이동합니다.

Vivado 탐색 바의 왼쪽 상단에 있는 "Tools"로 이동한 다음 옵션에서 "Setting"을 선택합니다.

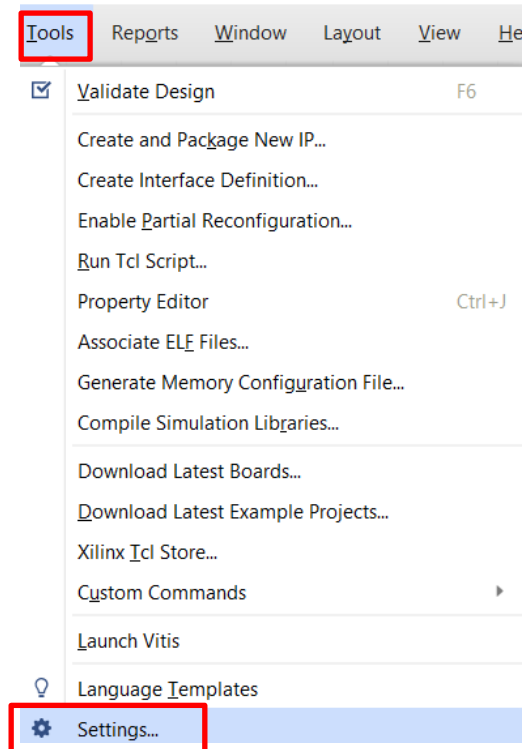


그림 65. 설정으로 이동

2 단계. 일반 탭으로 이동

" **General** " 탭으로 이동한 다음 언어 옵션 섹션에서 " **Verilog options** "을 선택합니다.

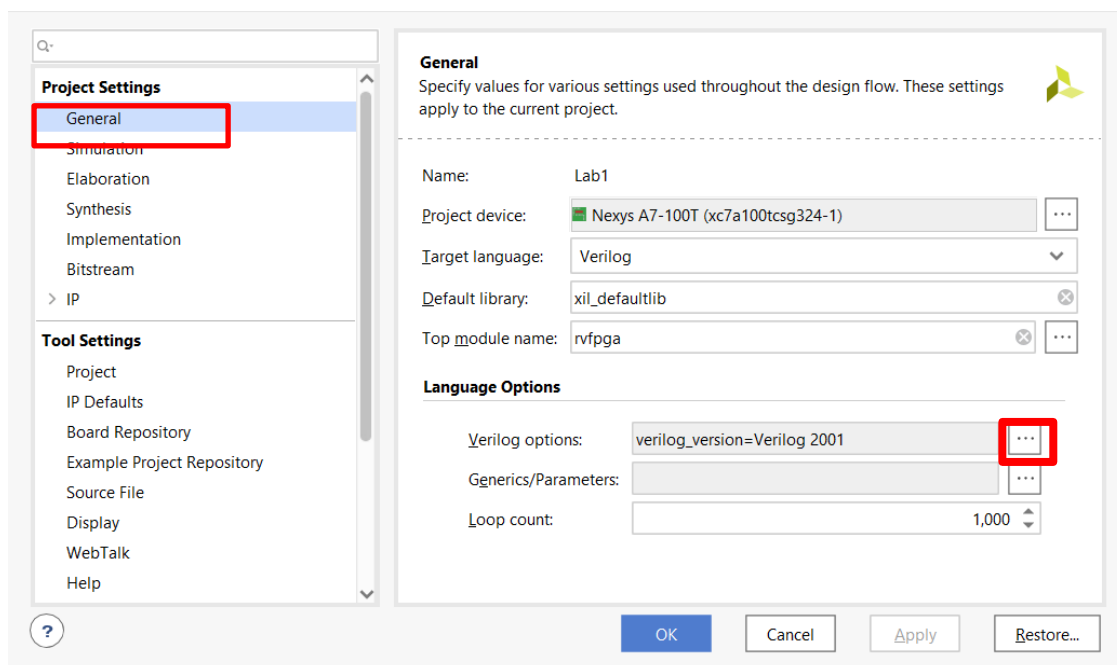


그림 66. 일반 설정

3 단계. 포함 파일의 경로를 추가합니다.

"+" 버튼을 클릭하여 Verilog 검색 경로 **Include Files** 를 추가합니다.

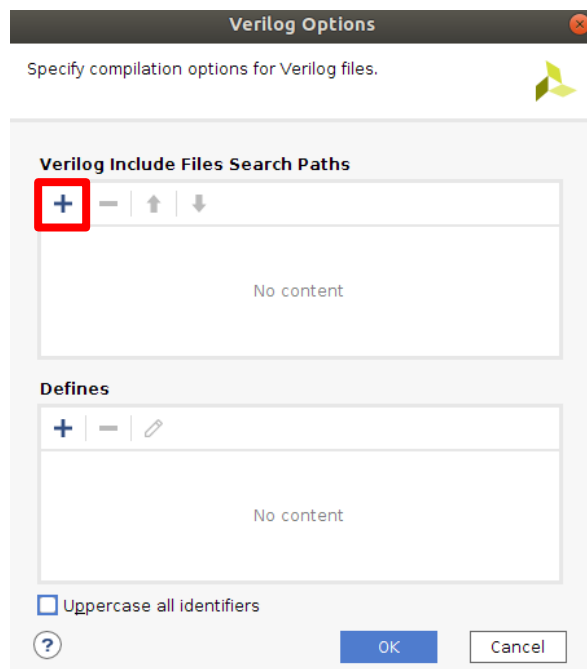


그림 67. Verilog 옵션

이제 다음 세 가지 경로를 추가합니다.

- [RVfpgaSoCPath] /RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs /sources_1/imports/src/SweRVolfSoC/Interconnect /AxiInterconnect/pulp-platform.org__axi_0.25.0/include
- [RVfpgaSoCPath] /RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs /sources_1/imports/src/OtherSources /pulp-platform.org__common_cells_1.20.0/include
- [RVfpgaSoCPath] /RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs /sources_1/imports/src/SweRVolfSoC/SweRVeh1CoreComplex/include

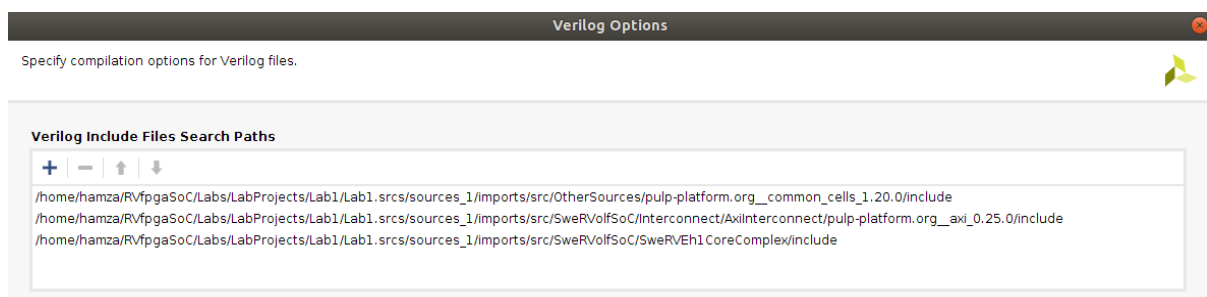


그림 68. Verilog 포함 파일 경로

OK 를 클릭합니다.

4 단계. Bitstream 탭으로 이동

"Bitstream" 탭으로 이동한 다음 "tcl.pre" 버튼을 클릭합니다.

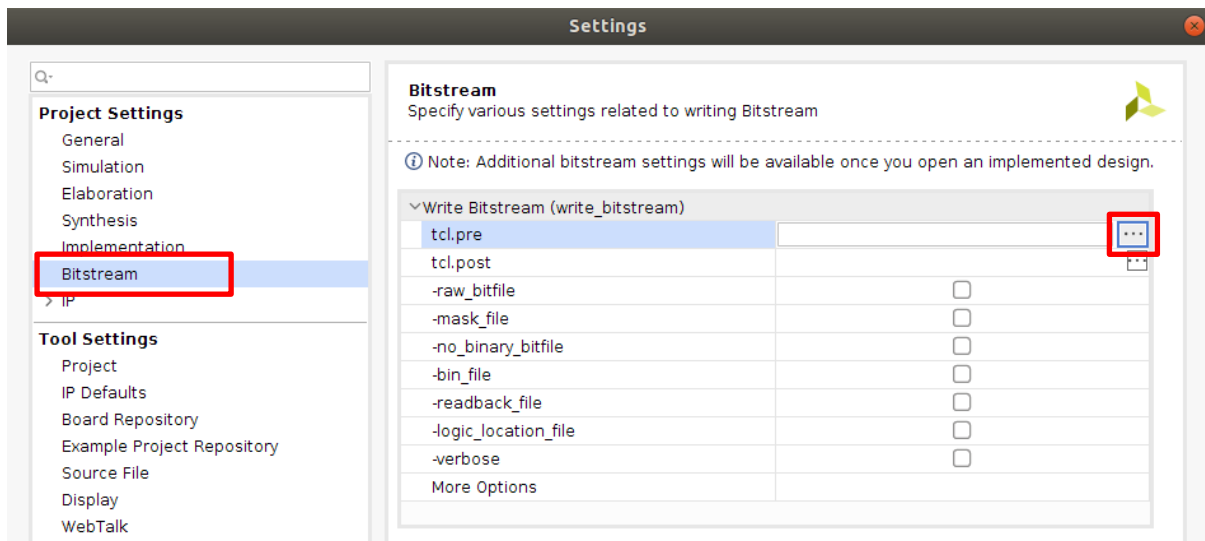


그림 69. 비트스트림 설정

"New script" 옵션을 선택하십시오.

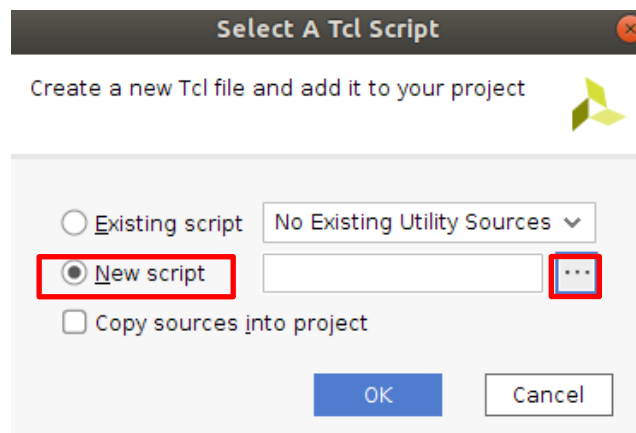


그림 70. 새로운 Tcl 스크립트

다음 경로로 이동하여 "script.tcl" 파일을 선택합니다. (그림 71 참조)

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/script.tcl

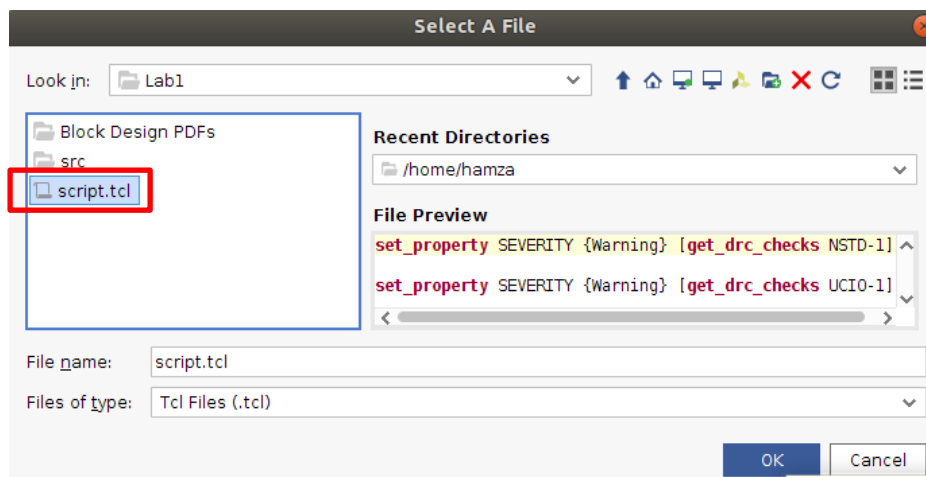


그림 72. "script.tcl" 파일 가져오기

OK 를 클릭하고 변경 사항을 적용합니다.

4 단계. 비트스트림을 생성합니다.

이제 그림 73 과 같이 Flow → Generate Bitstream 을 클릭합니다.

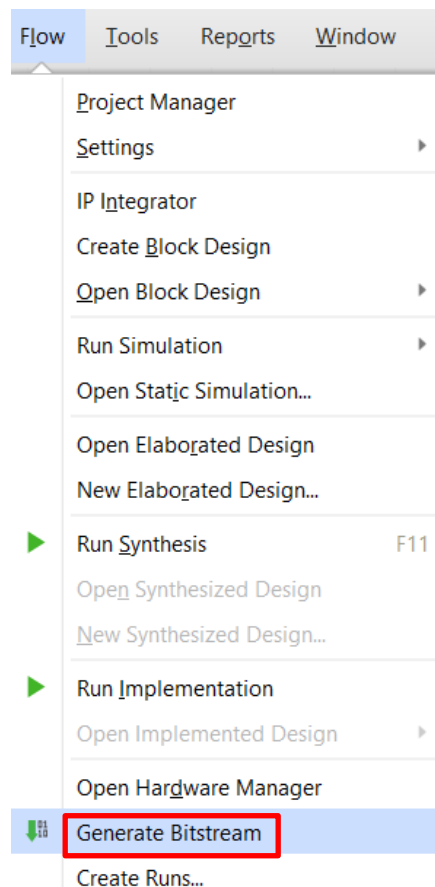


그림 73. 비트스트림 생성

사용 가능한 구현 결과가 없고 합성 및 구현을 시작하도록 요청하는 창이 나타날 수 있습니다. YES 를 클릭합니다(그림 74 참조).

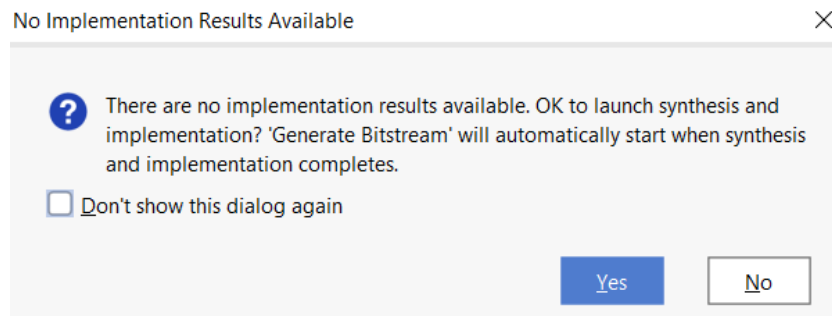


그림 74. 합성 및 구현 창 시작

Launch Runs 창이 화면에 팝업됩니다(그림 75 참조). OK 를 클릭합니다.

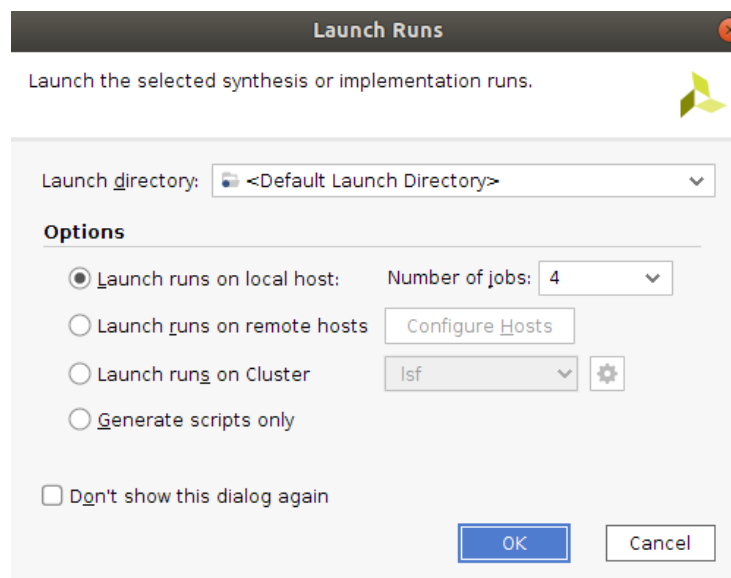


그림 75. Runs 실행

이제 연결하지 않은 핀이 자동으로 "0"에 연결된다는 경고 목록이 표시됩니다. OK 를 클릭합니다. (그림 76 참조).

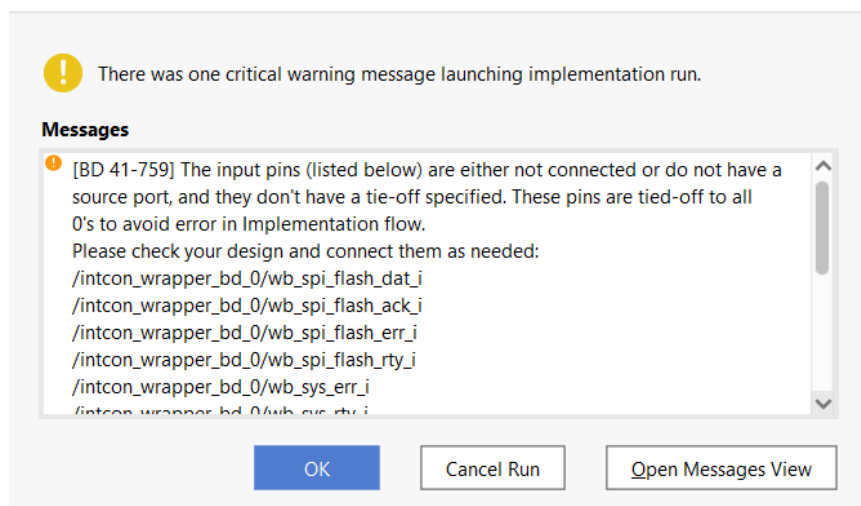


그림 76. 시작 실행 경고 메시지

이 단계에서는 **RVfpgaNexys**(프로젝트의 Verilog 및 SystemVerilog 파일에 정의됨)를 합성하고 이를 FPGA 에 매핑하고 비트스트림을 생성합니다.

Tcl Console Messages Log Reports Design Runs																
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	constrs_1	Queued...														00:00:00
impl_1	constrs_1	Queued...														00:00:00
Out-of-Context Module Runs																
BD		Running Submodule Runs													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_7	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_5	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_8	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_4	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_2	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_3	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_9	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_5	Running synth_design...													3/3/21, 2:55 PM	00:01:50
BD_bidirec_	BD_bidirec_7	Queued...														00:00:00
BD_bidirec_	BD_bidirec_0	Queued...														00:00:00
BD_bidirec_	BD_bidirec_3	Queued...														00:00:00
BD_bidirec_	BD_bidirec_2	Queued...														00:00:00

그림 77. 설계 실행

참고: 다음과 같은 오류가 발생하는 경우 Gtk-Message: Failed to load module "canberra-gtk-module"

문제를 해결하려면 다음 명령으로 패키지를 설치하십시오.

```
sudo apt install libcanberra-gtk-module libcanberra-gtk3-module
```

VM 을 사용하는 경우 낮은 RAM 할당으로 인해 합성 중에 Vivado 와 충돌할 수 있습니다. Vivado 가 충돌하는 경우 VM 에 더 많은 RAM 을 할당하는 것이 좋습니다.

이 프로세스는 컴퓨터 속도에 따라 몇 분이 소요될 수 있습니다.

Tcl Console Messages Log Reports Design Runs																
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	constrs_1	synth_design Complete!								3368	3331	13.0	0	0	5/4/21, 3:22 PM	00:02:13
impl_1	constrs_1	write_bitstream Complete!	0.327	0.000	0.050	0.000	0.000	0.934	0	33637	18546	44.0	0	4	5/4/21, 3:25 PM	00:12:32
Out-of-Context Module Runs																
BD		Submodule Runs Complete													5/4/21, 3:03 PM	00:19:17

그림 78. Verilog 포함 파일 경로

비트스트림이 생성되면 그림 79 와 같은 창이 나타납니다. 오른쪽 상단 모서리에 있는 X 버튼을 클릭하여 창을 닫습니다.

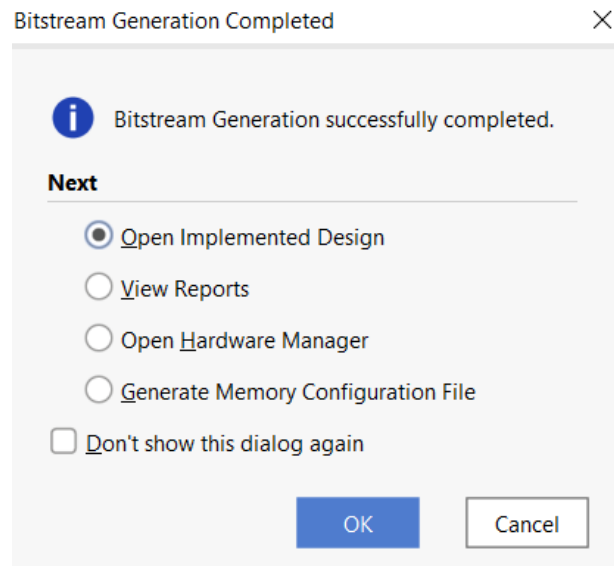


그림 79. 비트스트림 생성 완료

이제 비트스트림이 생성되었으므로 다음 실습에서는 이 비트스트림을 PlatformIO 를 통해 Nexys A7 보드에 업로드하는 방법을 보여주고, 이번 실습에서 방금 구축한 SweRVolfX 하위 집합에서 예제 프로그램을 실행하는 방법을 보여줍니다.