



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoC Lab 1

Introduction to RVfpga-SoC

Table 1. RVfpga Terms

Name	Description
Courses	
RVfpga	A course that shows how to use RVfpgaNexys and RVfpgaSim, RISC-V system-on-chips (SoCs), to run programs and extend the system by adding peripherals (RVfpga Labs 1-10), and explore the core and memory system by running simulations, measuring performance, adding instructions, and modifying the memory system (RVfpga Labs 11-20). Throughout the course, users are also shown how to use the RISC-V toolchain (compilers and debuggers) and simulators, the Verilator HDL simulator, and Western Digital's Whisper instruction set simulator (ISS).
RVfpga-SoC	A course that shows how to build a subset SweRVolfX SoC from scratch using building blocks such as the SweRV core, memories, and peripherals. The course also shows how to load the Zephyr real-time operating system (RTOS) onto SweRVolf and run programs including Tensorflow Lite's hello world example on top of the operating system.
Cores and SoCs	
SweRV EH1 Core	Open-source commercial RISC-V core developed by Western Digital (https://github.com/chipsalliance/Cores-SweRV).
SweRV EH1 Core Complex	SweRV EH1 core with added memory (ICCM, DCCM, and instruction cache), programmable interrupt controller (PIC), bus interfaces, and debug unit (https://github.com/chipsalliance/Cores-SweRV).
SweRVolfX	The System on Chip that we use in the RVfpga course. It is an extension of SweRVolf. SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf): An open-source SoC built around the SweRV EH1 Core Complex. It adds a boot ROM, UART interface, system controller, interconnect (AXI Interconnect, Wishbone Interconnect, and AXI-to-Wishbone bridge), and an SPI controller. SweRVolfX : It adds four new peripherals to SweRVolf: a GPIO, a PTC, an additional SPI, and a controller for the 8 Digit 7-Segment Displays.
RVfpgaNexys	The SweRVolfX SoC targeted to the Nexys A7 board and its peripherals. It adds a DDR2 interface, CDC (clock domain crossing) unit, BSCAN logic (for the JTAG interface), and clock generator. RVfpgaNexys is the same as SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf.

RVfpgaSim	<p>The SweRVolfX SoC with a testbench wrapper and AXI memory intended for simulation.</p> <p>RVfpgaSim is the same as SweRVolf Sim, (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf.</p>
------------------	---

1. Introduction

1. Introduction to a System on a Chip

In this lab, we will show how to build a RISC-V system on a chip (SoC) from building blocks. An **SoC** is an integrated circuit or an IC that integrates an entire electronic or computer system onto it. An SoC includes a core and all of the peripherals and interfaces necessary to load an operating system and run programs. Figure 1 illustrates the typical hierarchical organization of an embedded system starting with the processor core, then the SoC built around the core, and finally the system and board interface.

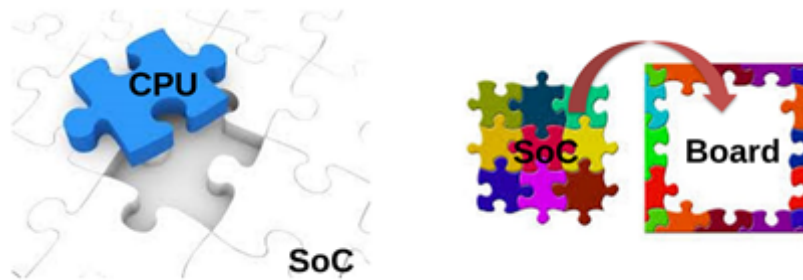


Figure 1. Typical Embedded system

The design process of an SoC starts with prototyping on an FPGA. Our focus will be on targeting an SoC to an FPGA.

The RISC-V CPU that we will use is Western Digital's **SweRV EH1 Core Complex**, and the SoC that we will design in this lab will be a subset of **SweRVolfX**, which we will target to the **Nexys A7-100T** board. Figure 2 illustrates the various components and how they fit together.

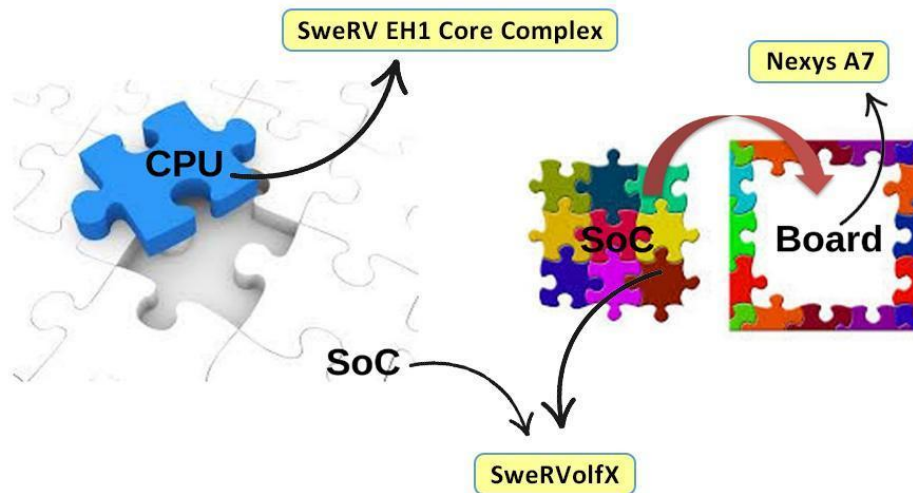


Figure 2. RVfpga System based Embedded system

2. Introduction to SweRVolfX and the RVfpga System

Before starting this lab, we highly encourage students to have gone through the RVfpga course Getting Started Guide and understand the overall RVfpga System. The following is a brief description of the RVfpga system introduced in the RVfpga course.

Table 1 shows the hierarchical organization of the RVfpga system, from the SweRV EH1 Core up to the RVfpgaNexys and RVfpgaSim. The System on Chip (SoC) used in the RVfpga system, called **SweRVolfX** and illustrated in Figure 3, is based on, **SweRVolf** version 0.7.3 (<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>), which is built on top of the **SweRV EH1 Core Complex**. In addition to the SweRV EH1 Core Complex, the SweRVolf SoC also includes Boot ROM, a UART, a System Controller, and an SPI controller. SweRV EH1 Core uses an AXI bus, and the peripherals use a Wishbone bus; the SoC also has an AXI-Wishbone Bridge.

In the RVfpga system, the SweRVolf SoC is extended with some more functionality, such as another SPI controller (SPI2), a GPIO (General Purpose Input/Output) controller, a PTC (PWM/Timer/Counter) module. (Figure 3 shows these new peripherals in red). This System on a Chip is called **SweRVolfX** (the X stands for eXtended).

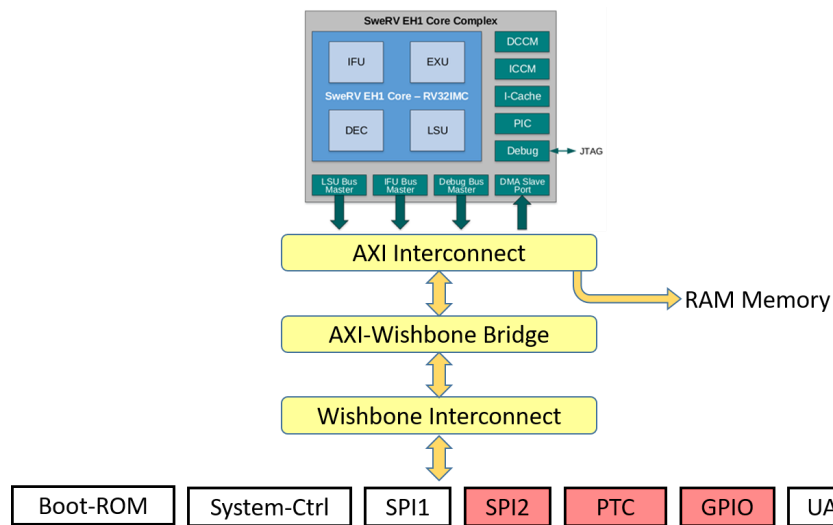


Figure 3. SweRVVolFX

Table 4 gives the memory-mapped addresses of the peripherals that are connected to the SweRV EH1 core via the Wishbone interconnect.

Table 4. MEMORY-MAPPED Addresses of SweRVolFX

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

* Peripherals added in SweRVolFX

3. Introduction to RVfpga-SoC

In RVfpga, SweRVolFX was introduced without any detail regarding how SweRVolFX was created. The RVfpga-SoC course shows how to build a subset of SweRVolFX SoC from scratch using building blocks such as the SweRV core, memories, and peripherals.

This Lab will be a step-by-step guide that shows how to start with a CPU (the SweRV EH1 Core Complex) and then build it into an SoC. We will be using the Vivado Block Design Tool. Vivado's block design tool facilitates wiring components graphically, making the process easier to understand and visualize. This visual approach also illustrates how each module is connected with the others to form an SoC.

The modules can be classified into three major blocks or categories:

1. CPU (SweRV EH1 Core Complex)
2. Interconnect (AXI-Interconnect, AXI2WB, and WB-Interconnect)

3. Peripherals (Boot-ROM, GPIO controller, and System controller)

SweRVolfX has many different modules and some are not necessary for a barebones RISC-V SoC. Hence, the extra modules have been trimmed to simplify the Lab and focus on the barebone functionality needed to bring a CPU core alive. Figure 4 shows the modules we will not be including (UART, PTC, SPI1, and SPI2).

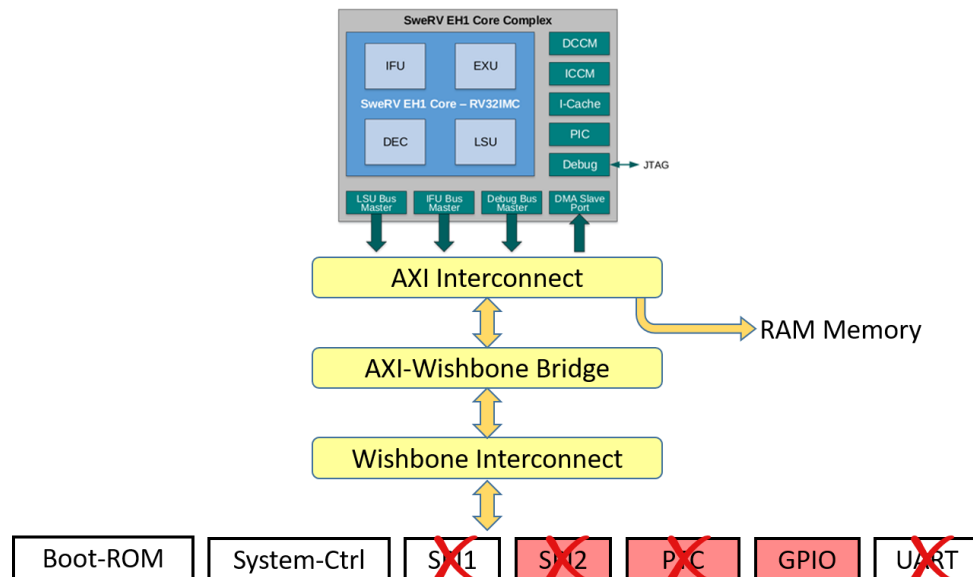


Figure 4. A subset of the SweRVolfX

Figure 5 shows a high-level block diagram of the SoC that we will be implementing.

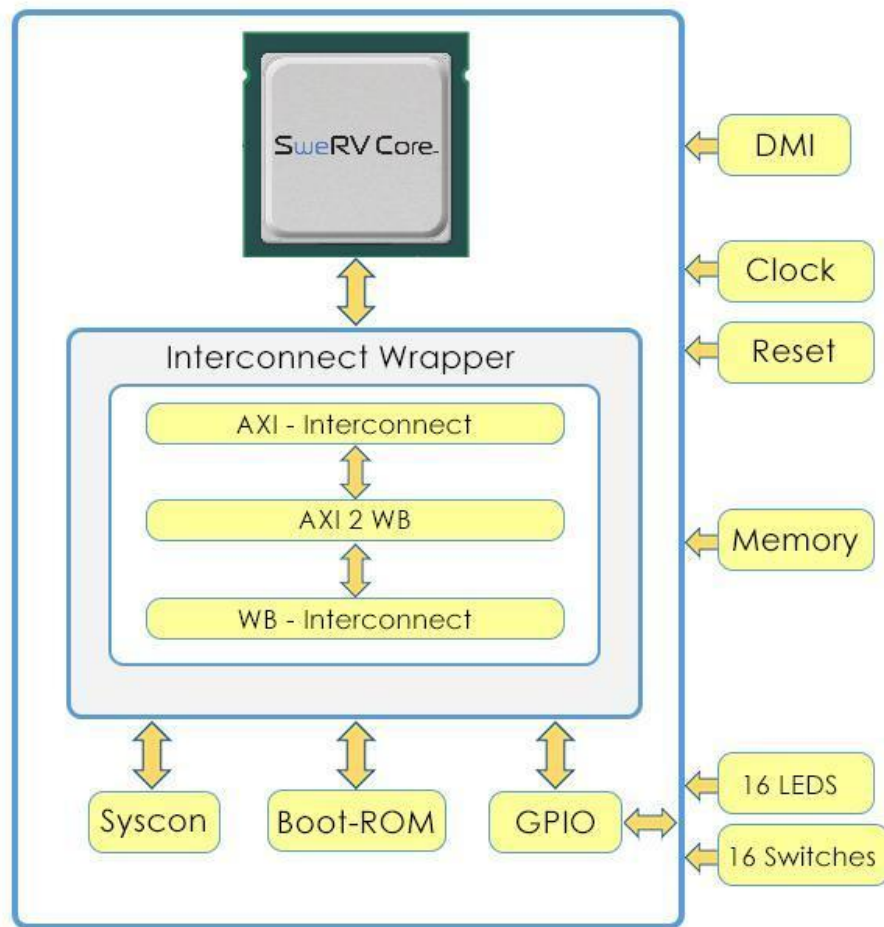


Figure 5. High-level block diagram of Lab 1 SoC

For the sake of ease of learning and understanding, some components that make up the Interconnect (AXI interconnect, Wishbone Interconnect, and AXI to Wishbone bridge) have been wrapped into one Interconnect wrapper module.

For Labs that focus on the CPU and inside the CPU, please refer to the RVfpga course. The RVfpga (also written RISC-V FPGA) course is a package that includes instructions, tools, and labs for targeting a commercial RISC-V processor and SoC to a field-programmable gate array (FPGA) and then using and expanding it to learn about computer architecture, digital design, embedded systems, and programming. For more information about RVfpga, visit <https://university.imgtec.com/rvfpga/>

2. Requirements

To complete this lab, you will need to have the following software installed:

- Vivado 2019.2 Web Pack (Refer to Installation Guide (Page No.04))
- Digilent Board Files (Refer to Installation Guide (Page No.05))

IMPORTANT: Before starting RVfpga-SoC Labs, we highly recommend completing the RVfpga-SoC Installation Guide.

For example, if you have not already, install Xilinx's Vivado following the instructions in the RVfpga-SoC Installation Guide. Make sure that you have copied the *RVfpgaSoC* folder that you downloaded from Imagination's University Programme to your machine.

3. Create Vivado Project

You will use Xilinx's Vivado Design Suite to build the SweRVofX subset using the RTL, the Verilog files that define the system. Follow these steps, detailed below, to create a Vivado project.

Step 1. Open Vivado

If you did not install Vivado on your machine as described in the RVfpga-SoC Installation Guide, do so now. Be sure to install the board files as well.

Now, run Vivado (in **Linux**, open a terminal and type: **vivado**; in **Windows**, open Vivado from the Start menu). The Vivado welcome screen will open. Click on Create Project (see Figure 6).

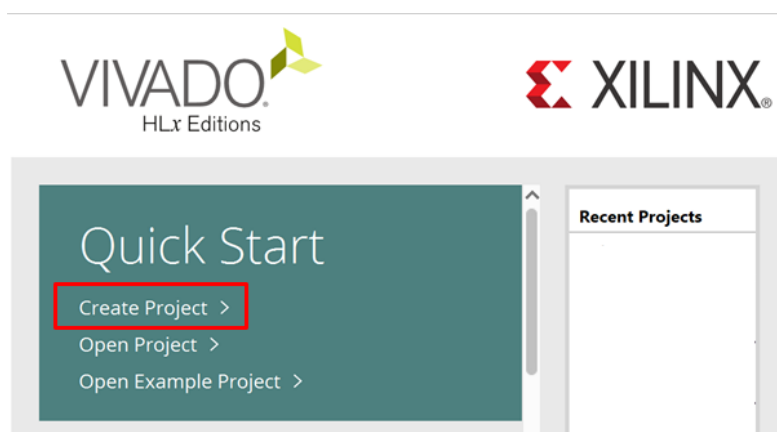


Figure 6. Vivado welcome screen: Create Project

Step 2. Create a new RTL project

The Create a New Vivado Project Wizard will now open (see Figure 7). Click Next.

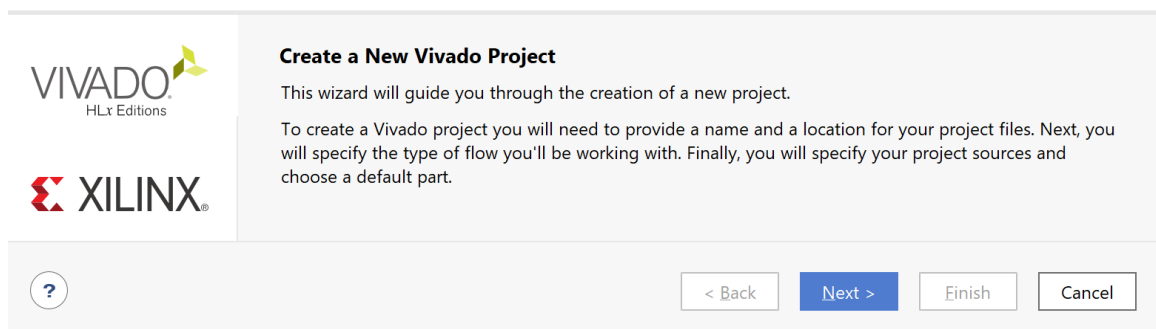


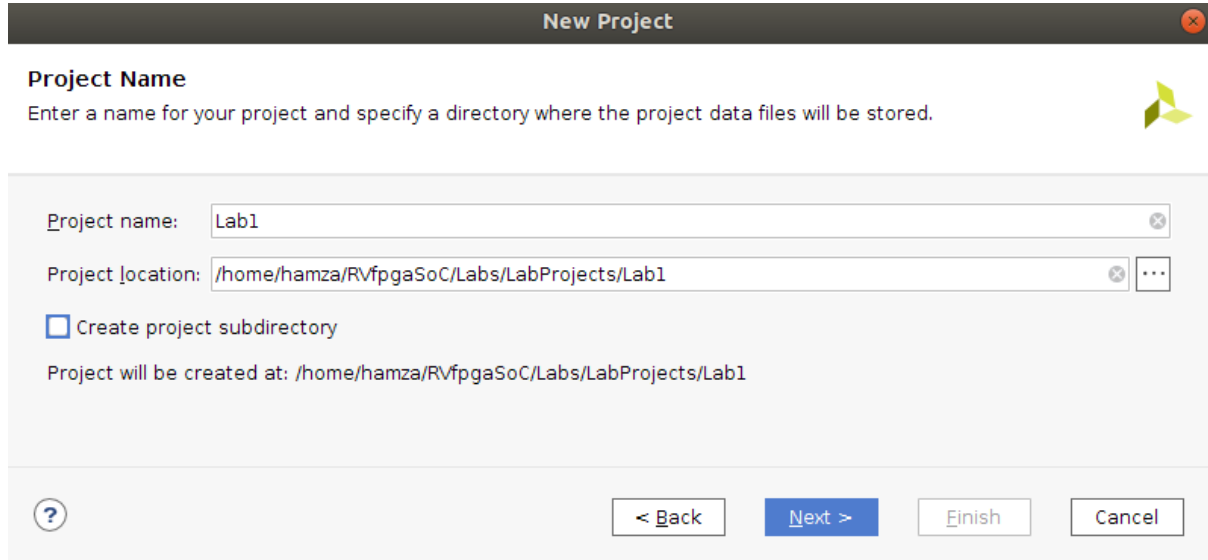
Figure 7. Create a New Vivado Project Wizard

Enter the name of the project as “**Lab1**” with no spaces. Then click Next (see Figure 8).

Select the following Project Location Path :

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabProjects/Lab1

Uncheck the create project subdirectory checkbox because there is already a folder called “**Lab1**” in the “**LabProjects**” folder.



New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

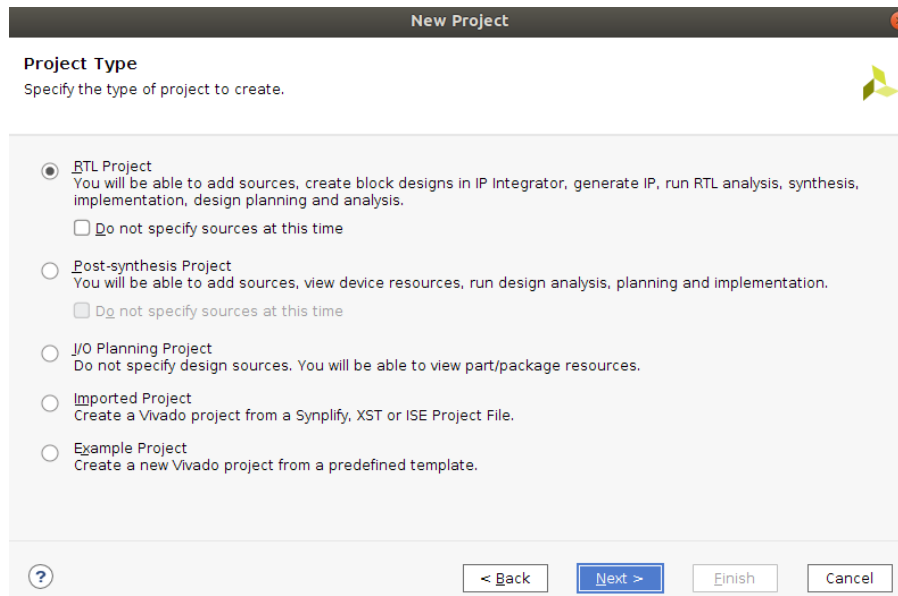
Project location:

☐ Create project subdirectory

Project will be created at: /home/hamza/RVfpgaSoC/Labs/LabProjects/Lab1

Figure 8. Project Name

Select the project type as RTL Project, and click Next (see Figure 9).



New Project

Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

☐ **Post-synthesis Project**
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**
Create a new Vivado project from a predefined template.

Figure 9. RTL Project

Step 3. Add the RTL source files and the constraint files

In the Add Sources window, click on “**Add Directories**” (see Figure 10).

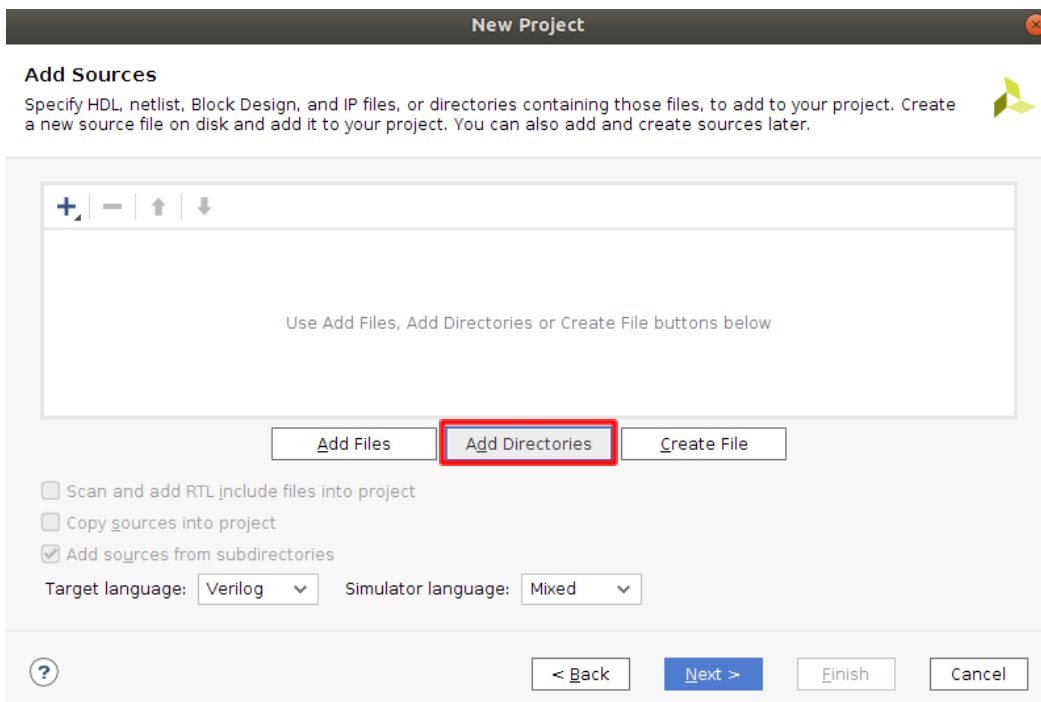


Figure 10. Add Sources directory

Now select the “src” directory at the following path
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/src (See Figure 11).

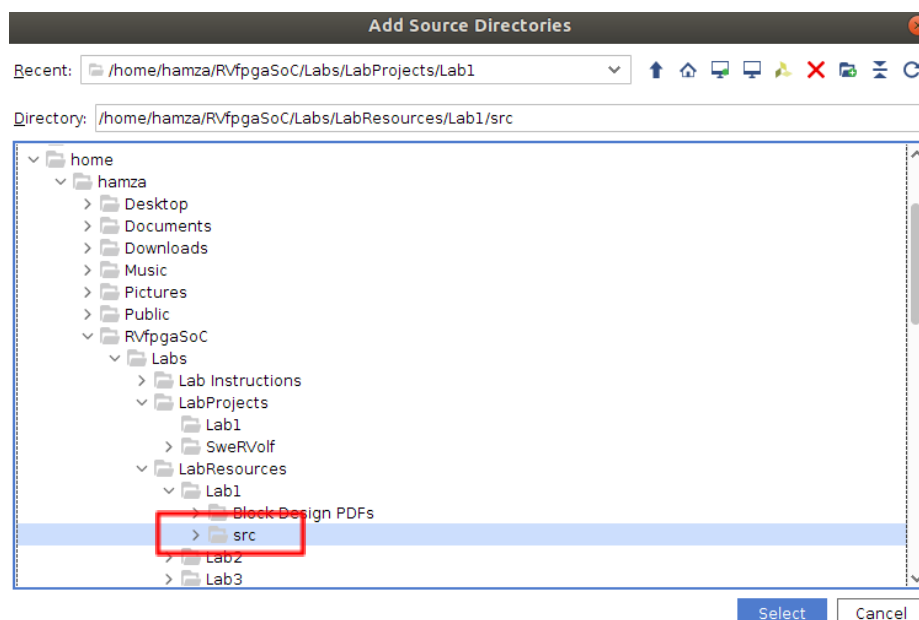


Figure 11. Select the “src” directory

Click Select.
Then click on the “**Add Files**” button.

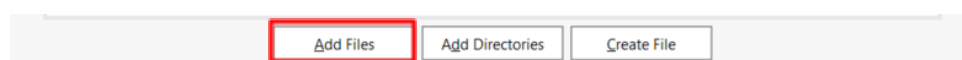


Figure 12. Add Files

Select the Files type to “**All Files**”. Now navigate to the **LiteDRAM** directory inside the **src** directory that we have just added.

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

Select both the “.init” files and click OK to add them both (see Figure 13).

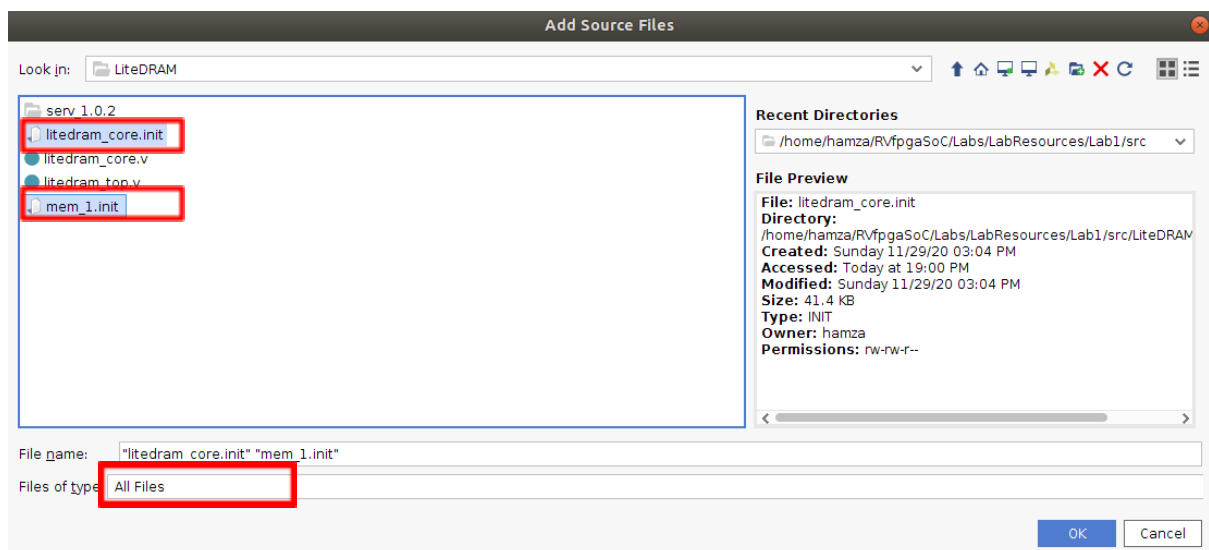


Figure 13. Add LiteDram Sources Files

Make sure all three of the checkboxes are checked (see Figure 14). Click “**Next**” to proceed to the next step.

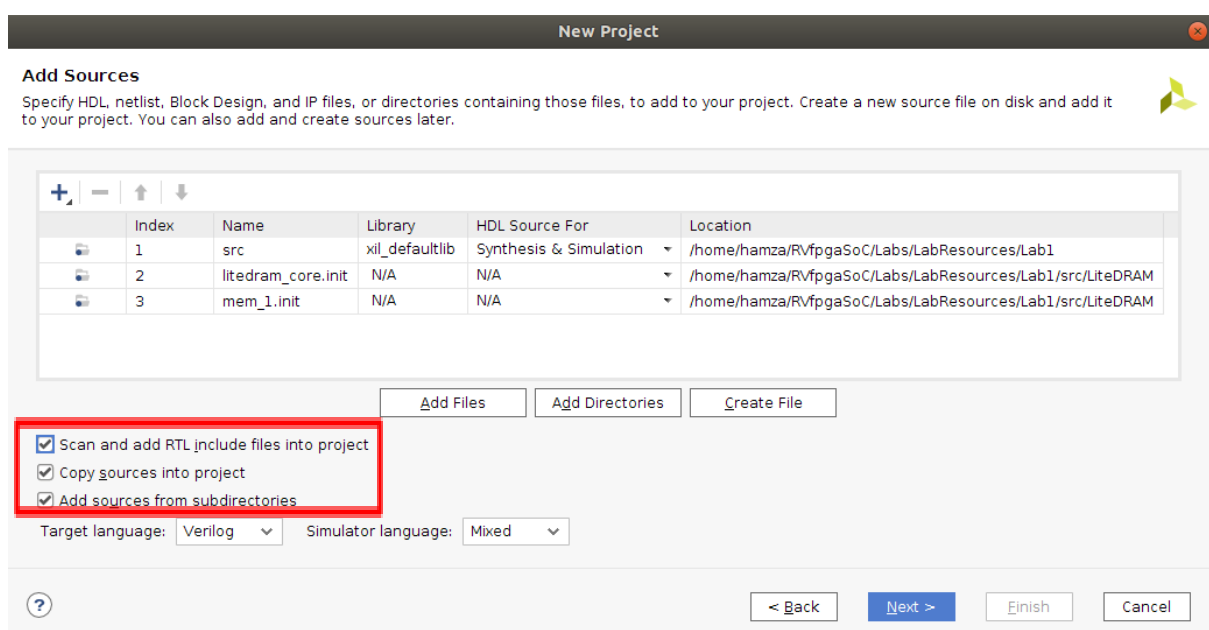


Figure 14. Add Sources

You will now add the constraints for the system. These files map the signal names to the pins on the board. For example, the Nexys A7 FPGA board's LEDs are connected to FPGA pins on the board through the PCB traces. Vivado must know this to map the correct signal name in the RTL to the correct FPGA pin. For example, the following line in the `[RVfpgaSoCPath]/RVfpgaSoC/src/rvfpga.xdc` file, a Xilinx design constraints file, indicates that FPGA pin H17 maps to the least significant LED (`o_led[0]`) and that it uses LVCMOS 3.3V signalling:

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

Note that the signal name `o_led` is the name used in RVfpga's Verilog code to drive the Nexys A7 board's LEDs.

In the Add Constraints window, click on **"Add Files"** and select the following two files (see Figure 15):

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

Then click **Next**.

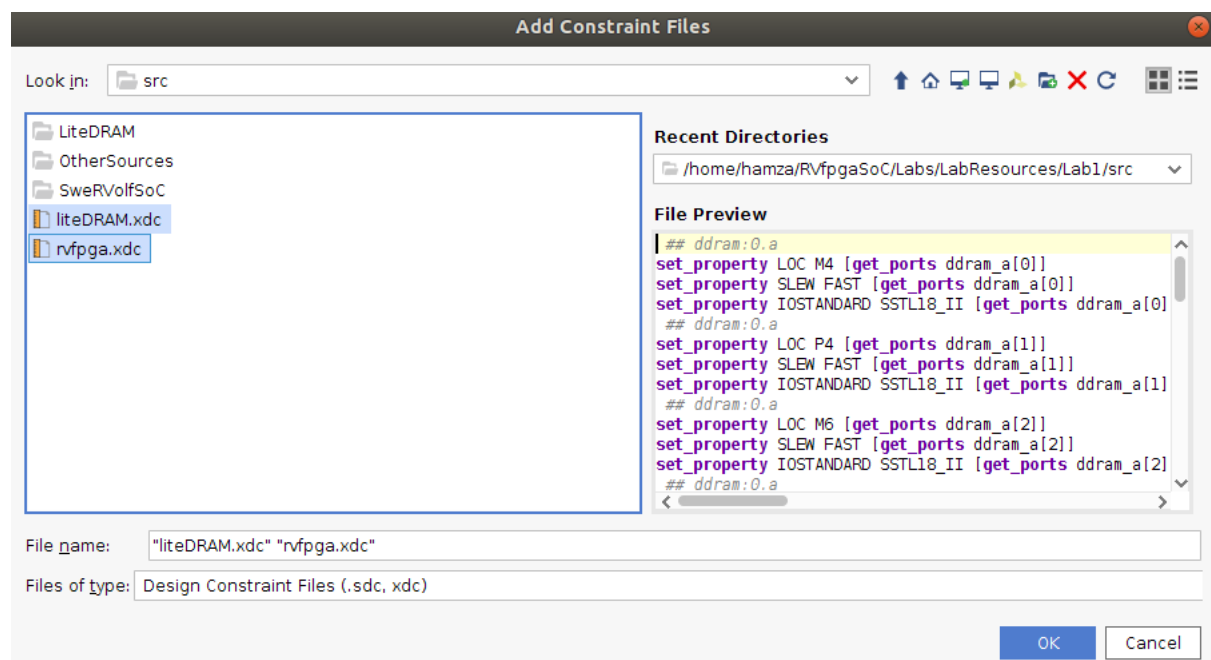


Figure 15. Add Constraints

Step 4. Select Nexys A7 as the target board

In the Default Part window, click on Boards and then select Nexys A7-100T (See Figure 16). You may use the Search box to narrow down the results. You will also notice that the name of the actual target FPGA is listed in the Part column: xc7a100tcsg324-1. This indicates that it is a Xilinx Artix-7 FPGA with 100k equivalent gates with a CSG (chip-scale grid) package and 324 pins.

Click Next.

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	xc7a100tcs324-1	32
Nexys A7-50T		digilentinc.com	1.0	xc7a50tics324-1L	32
Nexys4		digilentinc.com	1.1	xc7a100tcs324-1	32
Nexys4 DDR		digilentinc.com	1.1	xc7a100tcs324-1	32

? < Back Next > Finish Cancel

Figure 16. Select target board: Nexys A7-100T

In the New Project Summary window, click **Finish** (see Figure 17).

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: xc7a100tcs324-1
 Product: Artix-7
 Family: Artix-7
 Package: csg324
 Speed Grade: -1

To create the project, click Finish

? < Back Next > Finish Cancel

Figure 17. New Project Summary Window

Note that once the project completes being set up, it will indicate that files exist with Syntax Errors – this will be fixed in the next step.

Step 5. Set rvfpga as Top Module

The project will initialize. You will now set the rvfpga module as the top module. In the Sources pane, scroll down under Design Sources, right-click on the rvfpga module, and select Set as Top (see Figure 18). You can also find the rvfpga module by typing this name in the search box, as shown. This sets rvfpga as the highest-level module in the hierarchy and the target to be synthesized and implemented onto the FPGA. After setting rvfpga as the top module, the hierarchy will update.

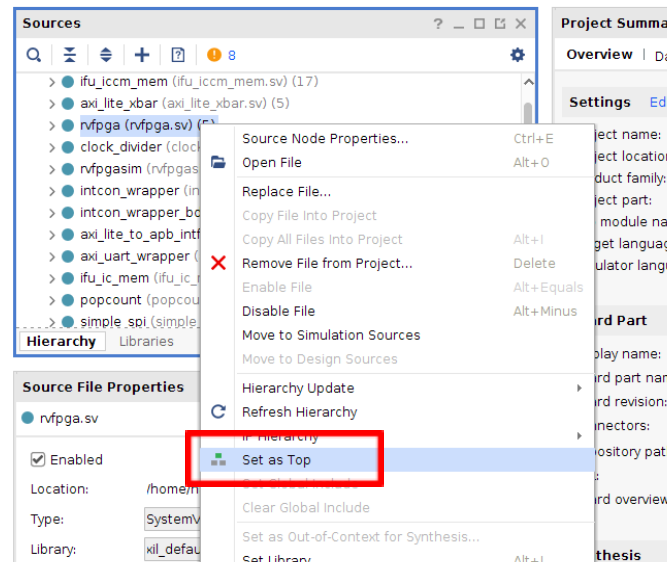


Figure 18. Set rvfpga as top module

Step 6. Set Verilog header files as global include Files

Now, still in the Sources pane under Design Sources, expand the Non-modules filegroup and click on common_defines.vh. The properties of the file will then open in the Source File Properties pane, just below the Sources pane. Click on Global Include to tick that box (see Figure 19). The hierarchy will now update and include that file in Design Sources/Global Include.

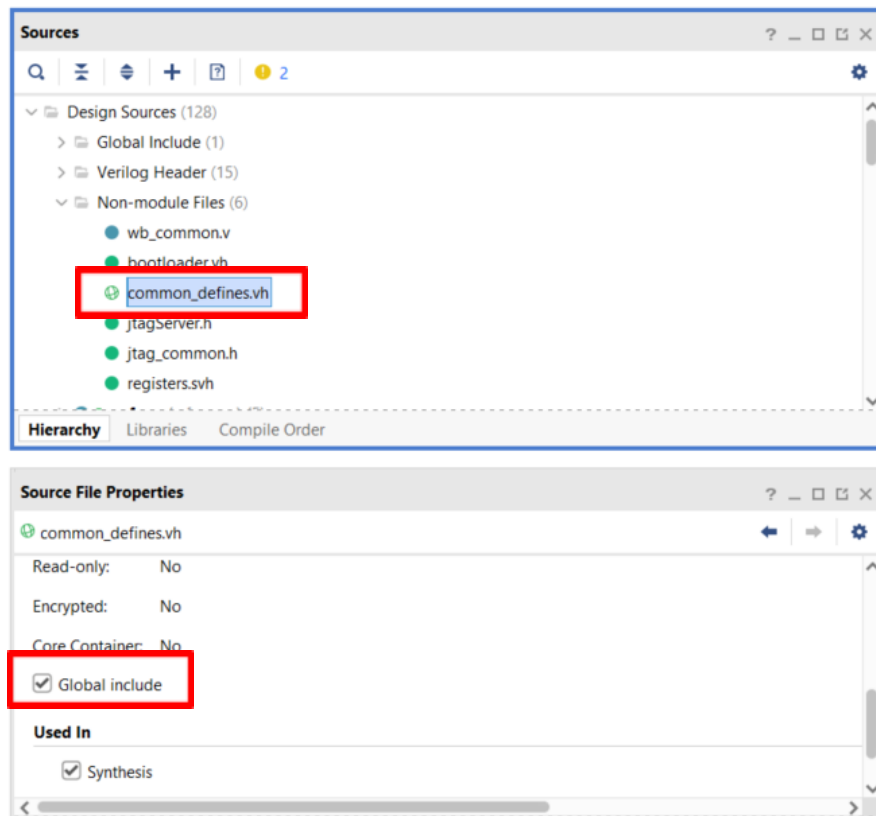


Figure 19. Set common_defines.vh as a Global include file

Similarly, set the “**assign.svh**”, “**registers.svh**”, and “**typedef.svh**” SystemVerilogHeader files as global include files (See Figure 20).

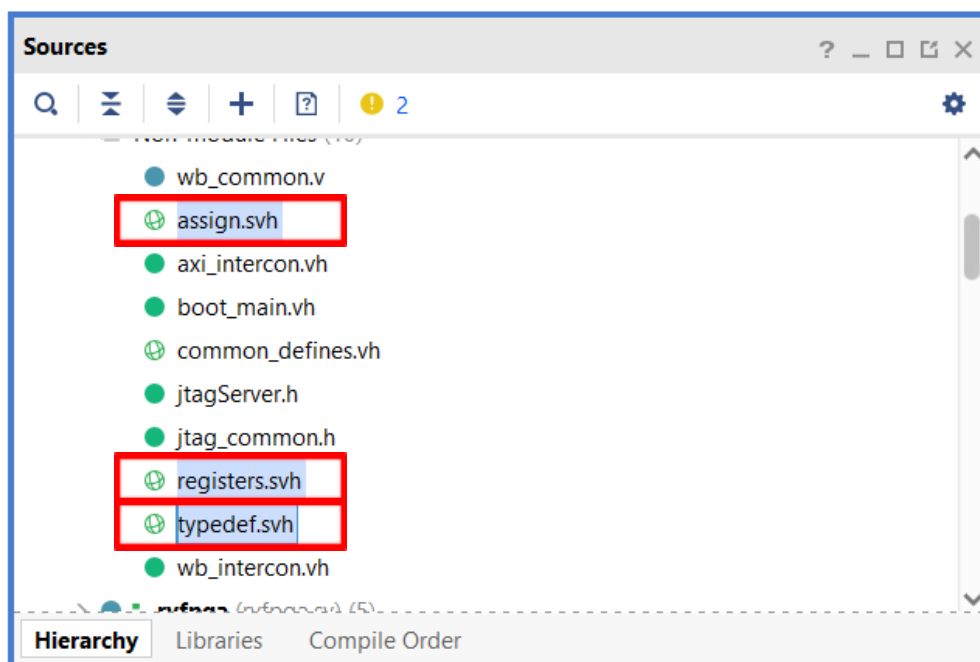


Figure 20. Set “.svh” files as a Global include file

Now expand the “**unknown**” filegroup and click on “**litedram_core.init**”. Then click on the Properties button next to the General button in the Source File Properties panel. Click on “**IS_Global_INCLUDE**” to tick that box (See Figure 21).

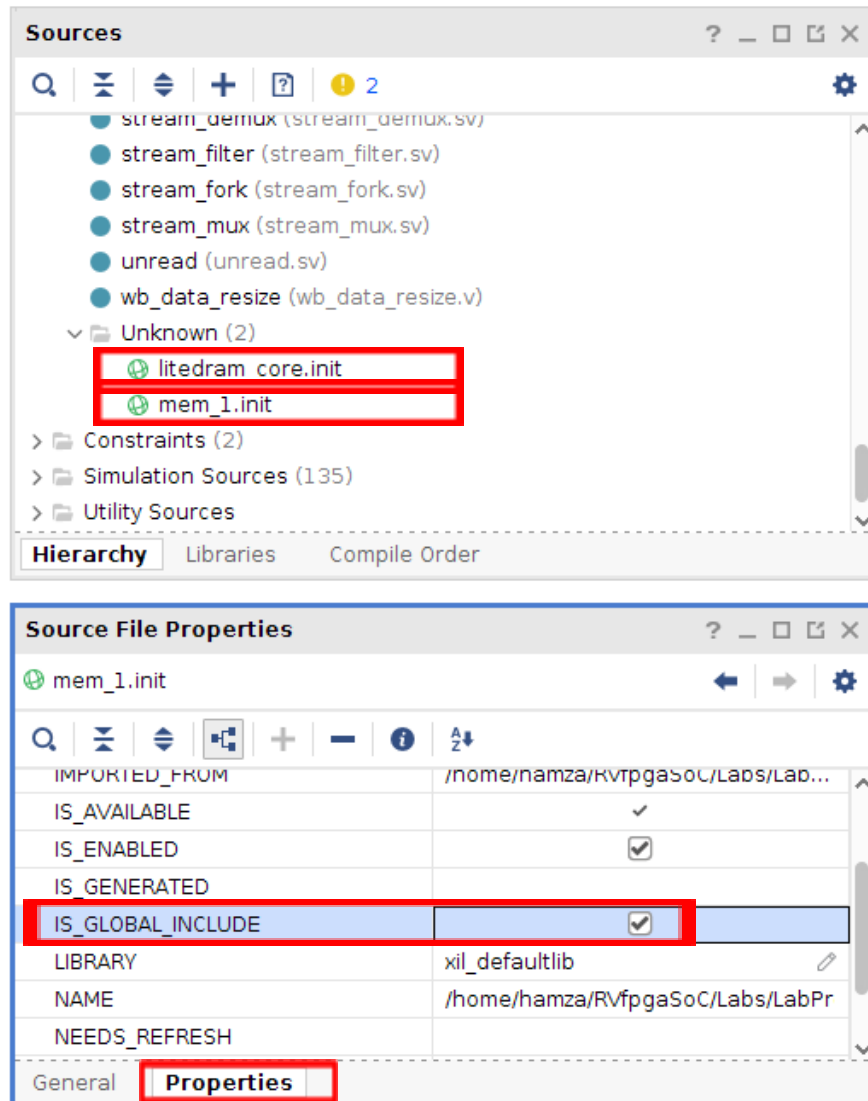


Figure 21. Set litedram_core.init as a Global include file

Now do the same for the “**mem_1.init**” file and set that file as a Global include file as well, just as did for the “**litedram_core.init**” file.

Step 7. Add boot_main.mem to the project

In the Flow Navigator pane, click on Add Sources, leave the default option (“Add or create design sources”), and click on Add Files (see Figure 22). Navigate to *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/SweRVolfSoC/BootROM/sw* and select *boot_main.mem* (as shown in Figure 22). The hierarchy will update and include that file in the Design Sources/Memory File.

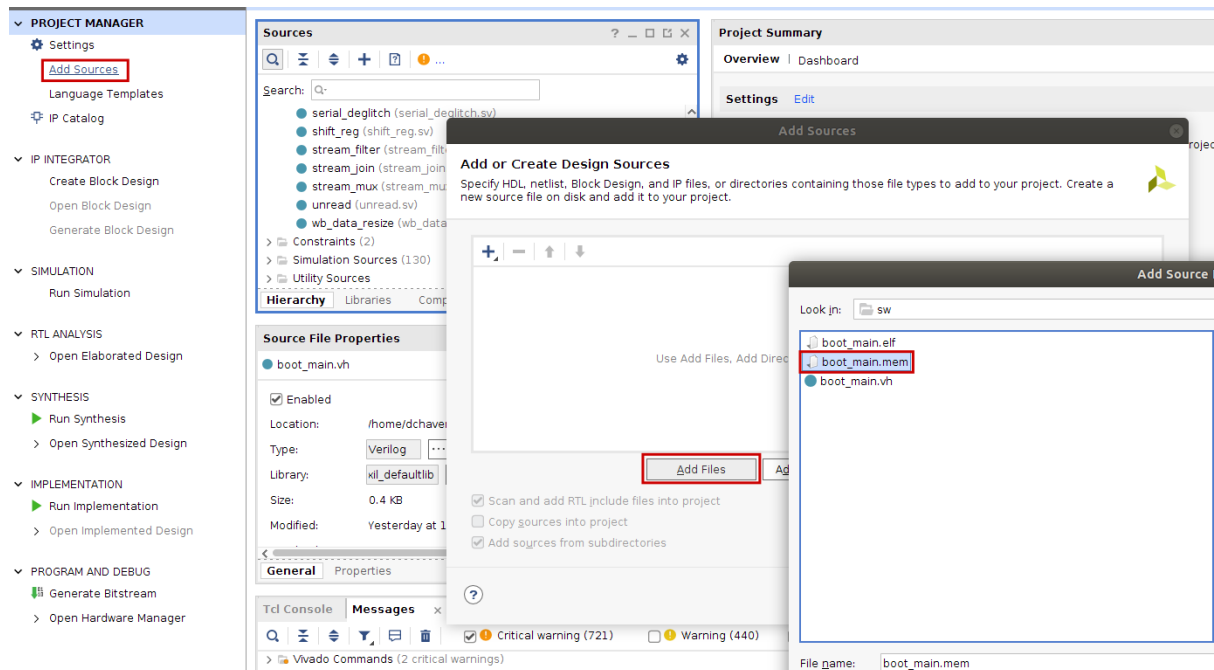


Figure 22. Add Memory File boot_main.mem

The design source files have now been added and now we can go ahead and start creating the block design.

4. Create Block Design

We will be using Vivado's Block Design feature to add the modules required to create the SwERVolfX subset and then wire the modules with each other.

Step 1. Click on Create Block Design

Create a new block design in the Flow Navigator by clicking on Create Block Design under the IP Integrator heading (see Figure 23).

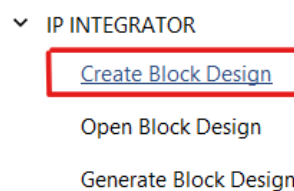


Figure 23. Create Block Design

Step 2. Select Block Design's Name

Select the Design name as "BD" to avoid any naming conflicts later in the Lab (See Figure 24).

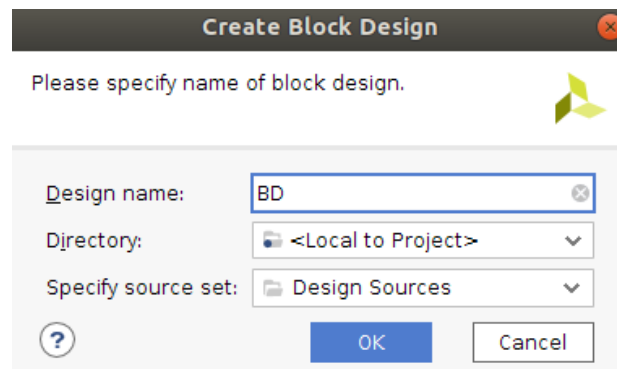


Figure 24. Select Block Design's Name and Directory

Now you will see a blank block design Diagram panel. (see Figure 25)

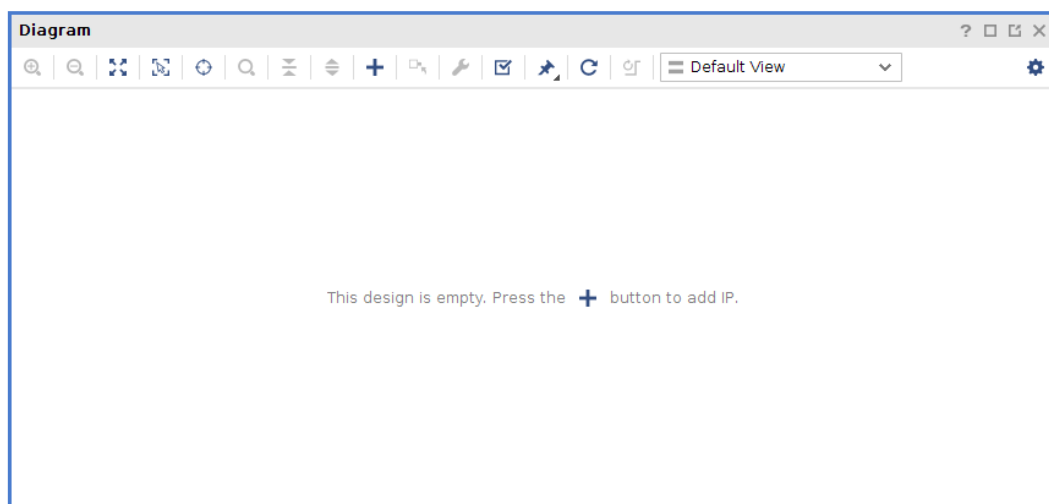


Figure 25. Blank Block Design

Step 3. Add Modules to the Block Design

Now we can start adding modules to our Block Design. We can do that by right-clicking on the blank Block design and select the **"Add Module"** option (See Figure 26).

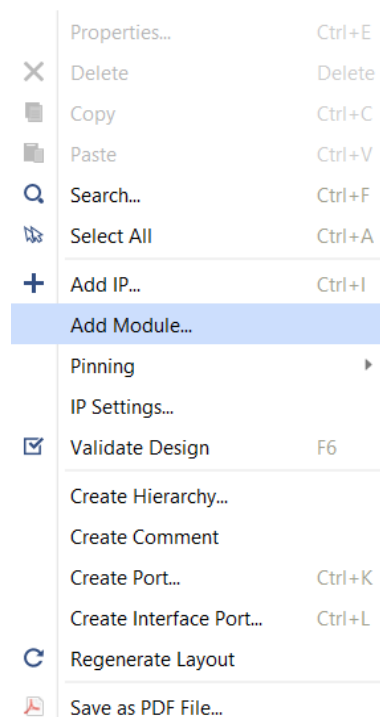


Figure 26. Add Module

A dialogue box will appear; you can either scroll down or type in the search box the name of the required modules you would like to add. We will start by adding the SweRV EH1 Core Complex.

Select “**swerv_wrapper_verilog**” and click OK.

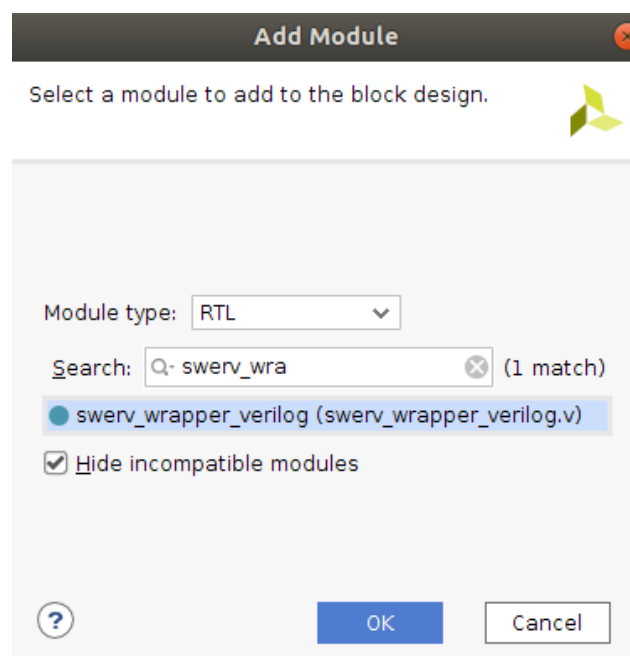


Figure 27. Add swerv_wrapper_verilog

A critical warning message will pop up (see Figure 28). Click OK to ignore this warning message.

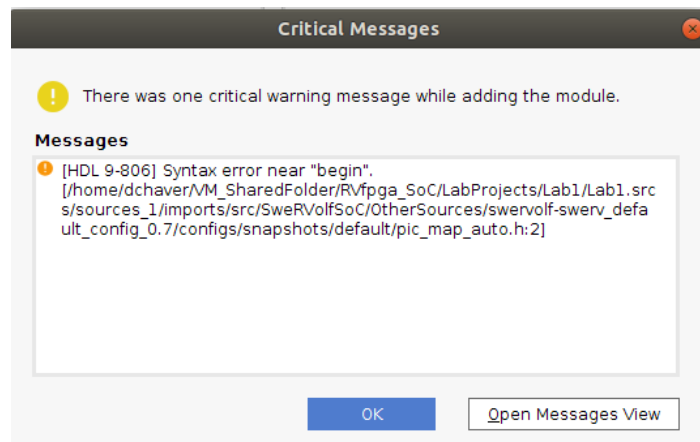


Figure 28. critical warning message

After we have added the module, we can visualize and access all the pins of “ifu_axi”, “lsu_axi” or “sb_axi” by clicking on the “+” icon on the module.

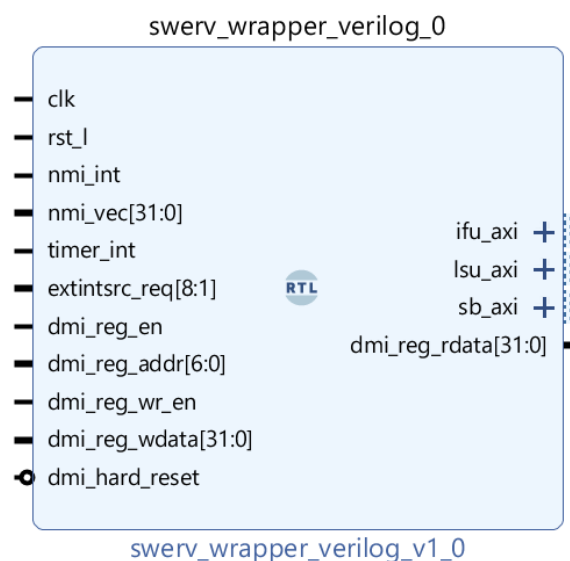
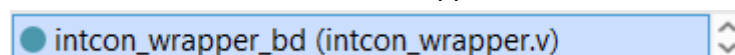


Figure 29. “swerv_wrapper_verilog” module

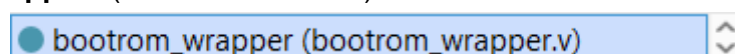
Similarly, we will now add the following modules:

- “**intcon_wrapper_bd**” (Interconnect Wrapper Module): It is a wrapper module that contains all the three interconnect modules wrapped into it.



Now we will add the peripherals needed for our SoC :

- “**bootrom_wrapper**” (Boot-ROM Module)



- “**gpio_wrapper**” (GPIO Top Module)

● gpio_wrapper (gpio_wrapper.v)

- “syscon_wrapper” (System Controller Module)

● syscon_wrapper (syscon_wrapper.v)

We will add the 32 “**bidirec**” modules to attach with our GPIO module. 16 of these will be for the LEDs, and 16 will be for the switches.

- “**bidirec**” (Bidirectional GPIO module)

● bidirec (bidir.v)

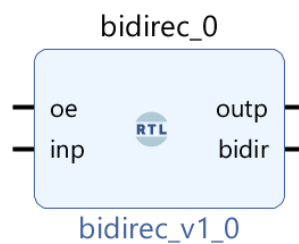


Figure 30. bidir GPIO module

Similarly, we will add 32 of these modules to the block design.

A quick way to add these 32 modules is to copy-paste the blocks in the Diagram. First Copy 1 “**bidirec**” block then paste it, then copy 2 blocks and paste them, then repeat the process of copying and pasting until you have 32 blocks of “**bidirec**” module added to your block design.

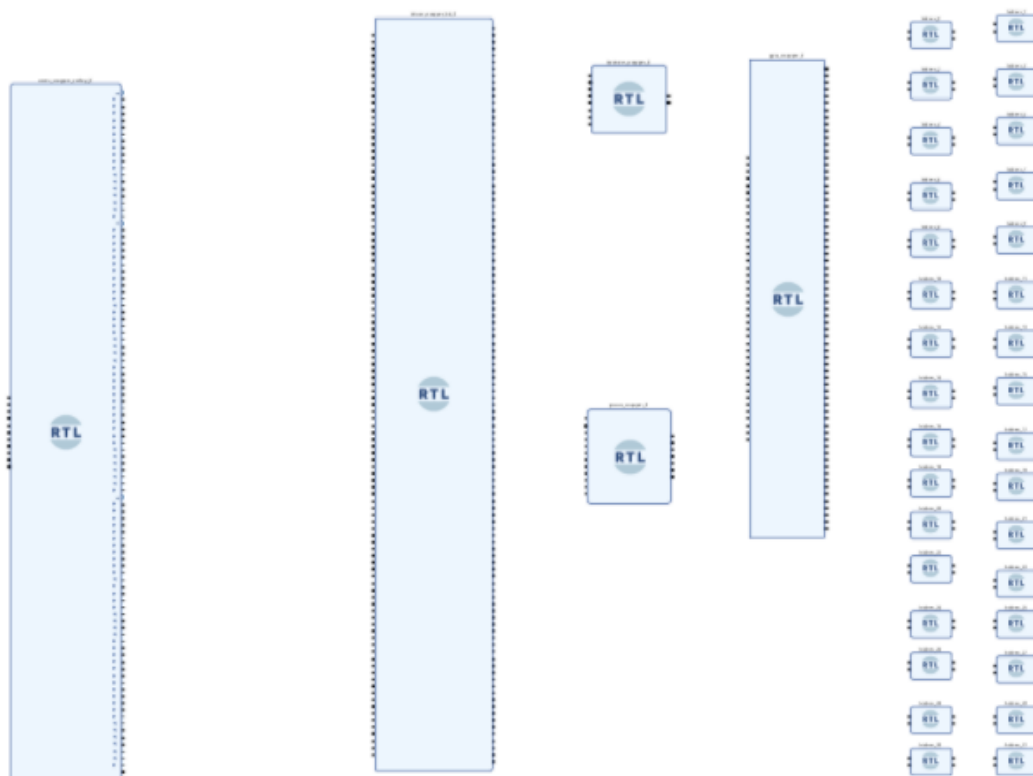


Figure 31. Required modules have been added to the Block Design

(see Figure 31) Starting from the left-hand side, view the **SweRV Core** module (swerv_wrapper_verilog_0); then, to the right, view the **Interconnect Wrapper** module (intcon_wrapper_bd_0) and the four peripheral modules, which are the **Boot-ROM** (bootrom_wrapper_0) module, **System Controller** (syscon_wrapper_0) module, **GPIO** (gpio_wrapper_0) module. On the rightmost side, you will see the 32 **Bidirec** (bidirec_x) modules.

Step 4. Wire up the modules

We now wire the modules to each other pin-by-pin or, in some cases, bus-by-bus. We will begin connecting the “swerv_wrapper_verilog” with the “intcon_wrapper_bd”. Three different sets of pins need to be connected between these modules related to the following submodules of the core:

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

We will first start with connecting the pins related to the IFU. Connect the pin “ifu_axi_arid[2:0]” of the “swerv_wrapper_verilog” module to the “i_ifu_arid[2:0]” pin of the “intcon_wrapper_bd”.

Similarly,

ifu_axi_araddr[31:0] will be connected to *i_ifu_araddr[31:0]*,

ifu_axi_arlen[7:0] will be connected to *i_ifu_arlen[7:0]*,

ifu_axi_arsize[2:0] will be connected to *i_ifu_arsize[2:0]* and so on (see Figure 32).

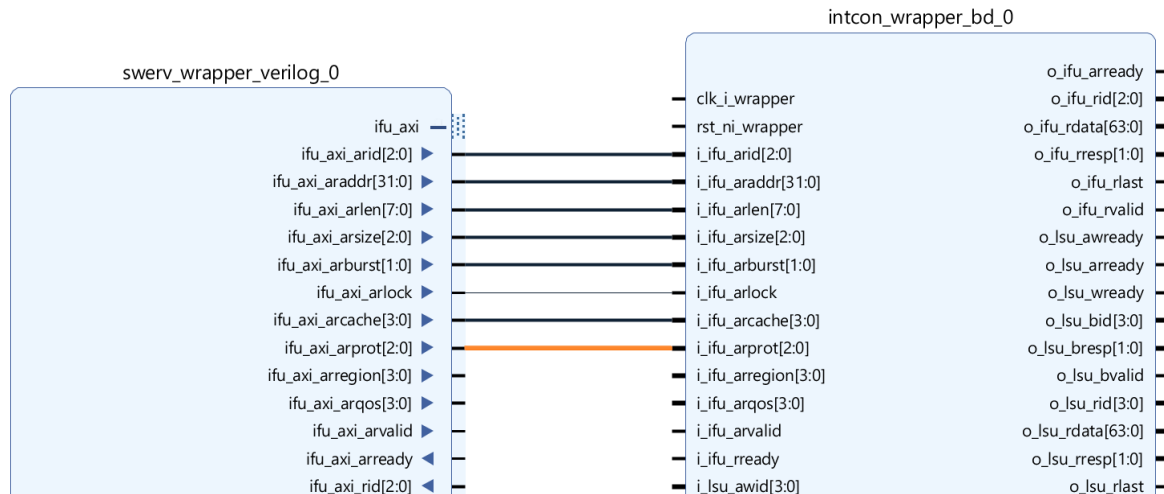


Figure 32. Connect the relevant pin

Similarly, we will connect all the **IFU (Instruction Fetch Unit)** pins of “swerv_wrapper_verilog” with the IFU’s pins of “intcon_wrapper_bd” (see Figure 33).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/InternalConnections/1_SwervW_IntconW_IFU.pdf

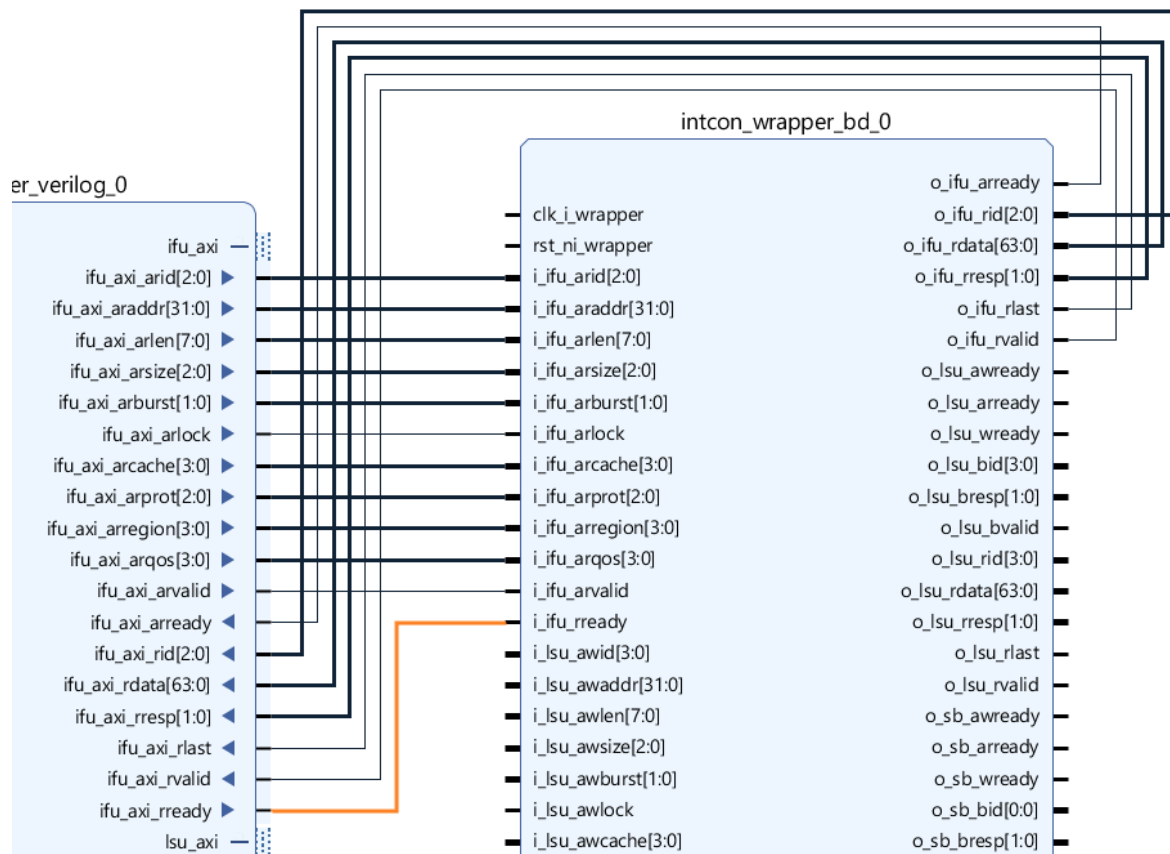


Figure 33. Connect all the IFU pins

Now we will move to connect all the **LSU (Load Store Unit)** pins of the “**swerv_wrapper_verilog**” to the **LSU’s** pins of “**intcon_wrapper_bd**”. We will perform the same process as we did for the **IFU** pins, connecting each **LSU** pin of the “**swerv_wrapper_verilog**” module with its respective pin on the “**intcon_wrapper_bd**” module (see Figure 34).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/InternalConnections/2_SwervW_IntconW_LSU.pdf

Now we proceed to connect the **SB** pins. We similarly connect all the **SB** pins of the “**swerv_wrapper_verilog**” with its respective **SB**’s pins of “**intcon_wrapper_bd**” (see Figure 35).

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/InternalConnections/3 SwervW IntconW SB.pdf

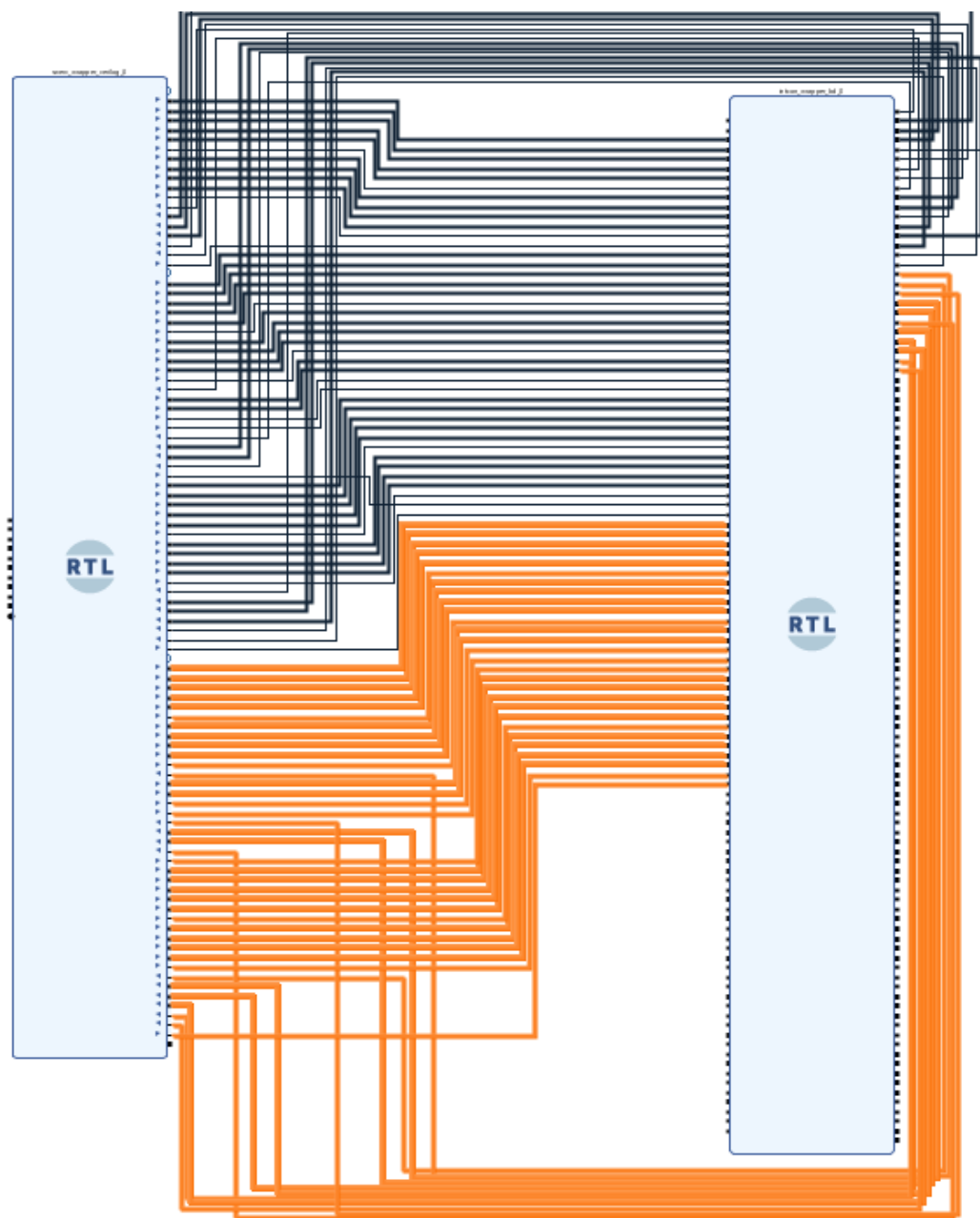


Figure 35. Connect all the SB pins

Next, we will connect the peripherals with the “**Intcon_wrapper_bd**”. We start with the “**bootrom_wrapper**” module by joining the “wb_rom_xxx_x” wires of the “**Intcon_wrapper_bd**” (see Figure 36).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/4_BootRomW_IntconW.pdf](#)

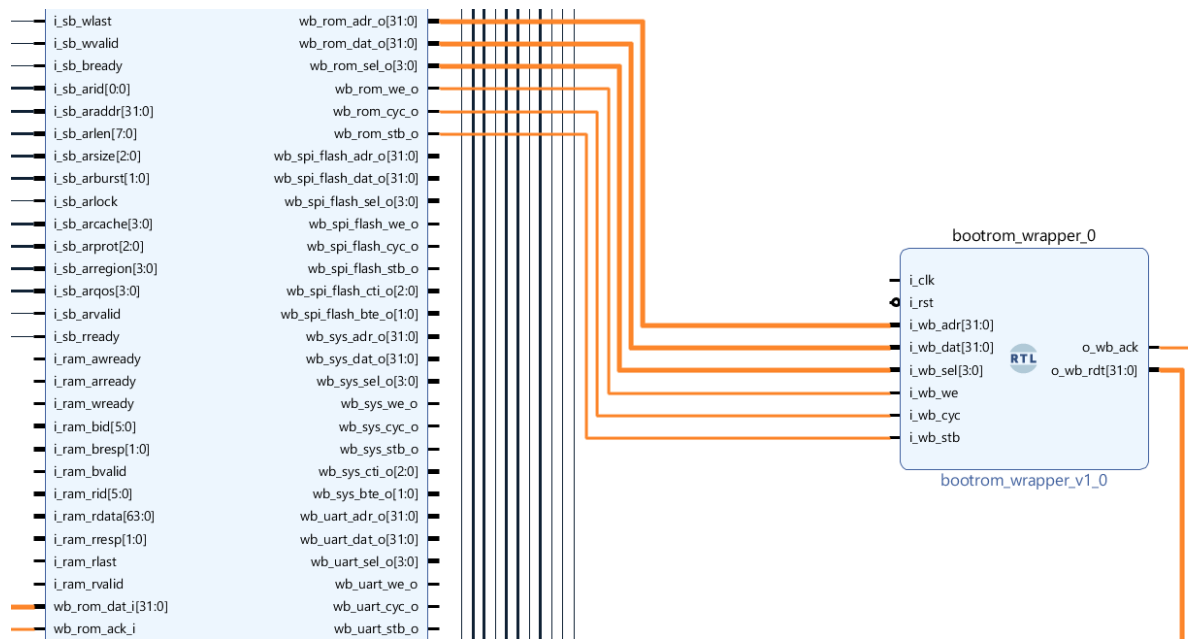


Figure 36. Connect the BootROM module with the Interconnect Wrapper module

Now we will connect the “**syscon_wrapper**” module with the “**Intcon_wrapper_bd**” module (see Figure 37).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /InternalConnections/5_SysconW_IntconW.pdf](#)

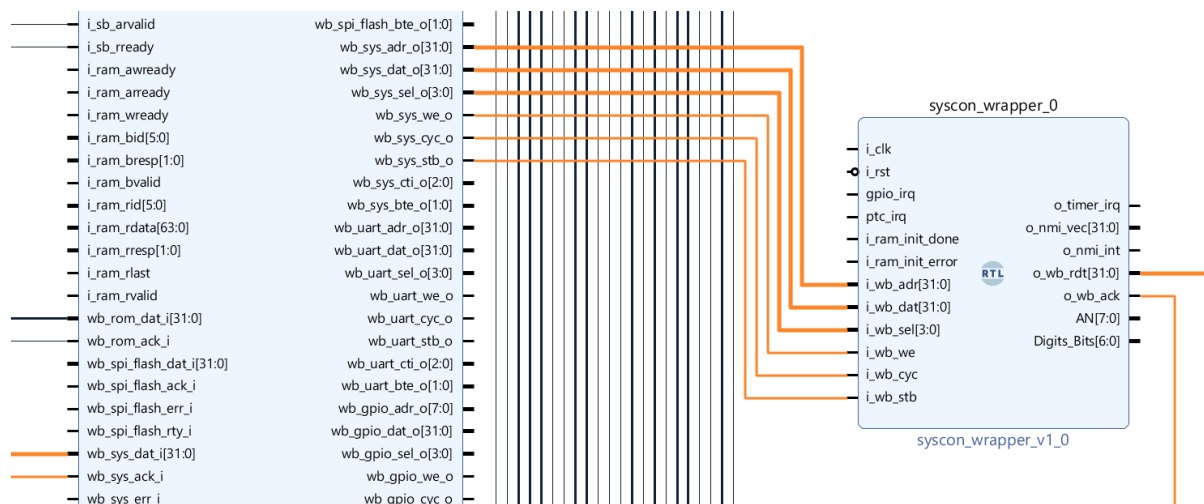


Figure 37. Connect the Syscon with the WB Interconnect Pins

The following pins of the “**syscon_wrapper**” will be connected to the “**swerv_wrapper_verilog**” (see Figure 38).

- o_timer_irq
- o_nmi_vec[31:0]
- o_nmi_int

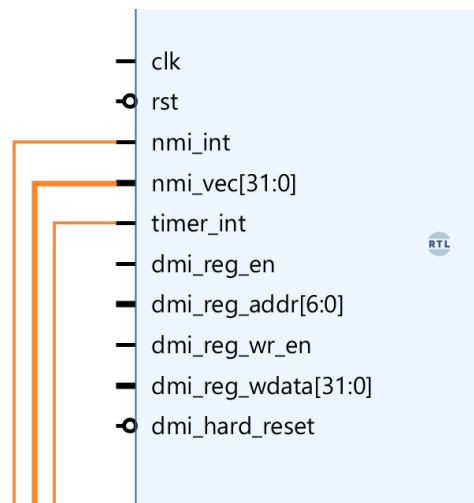


Figure 38. Connect the syscon_wrapper with the swerv_wrapper_verilog pins

Now we will connect the “**gpio_wrapper**” module with the “**intcon_wrapper_bd**”. Connect the “**wb_gpio_XXX_x**” pins of the “**intcon_wrapper_bd**” module with the “**gpio_wrapper**” module pins (see Figure 39).

Connect the “**wb_inta_o**” pin of the “**gpio_wrapper**” module with the “**gpio_irq**” pin of the “**syscon_wrapper**” module.

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /InternalConnections/6_GpioW_IntconW.pdf](#)

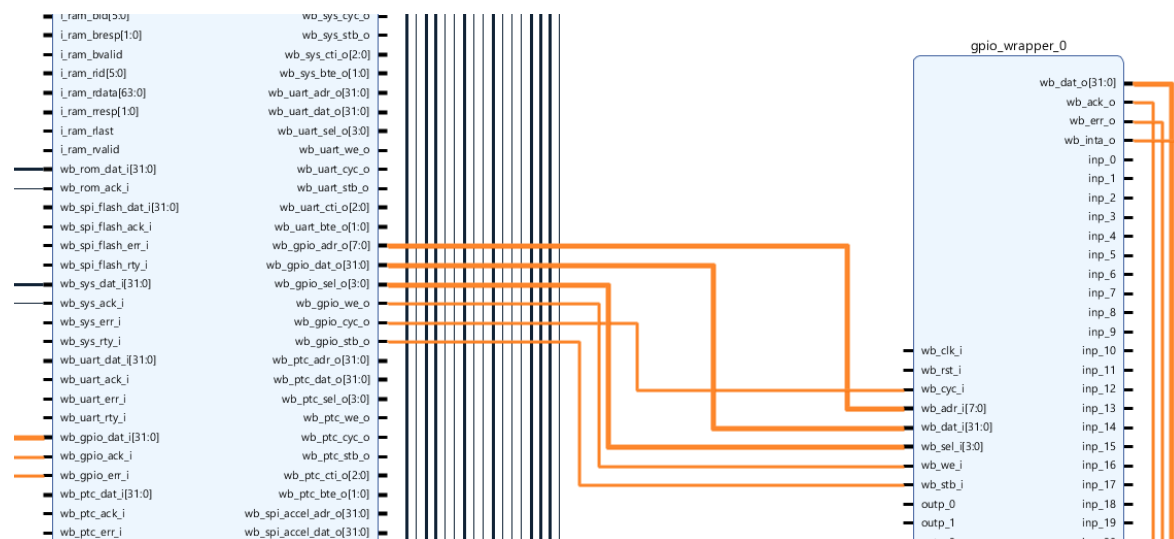


Figure 39. Connect the gpio_wrapper with the intcon_wrapper pins

We will connect the 32 GPIO “**bidirec**” modules with our “**gpio_wrapper**” module that we have already connected. Specifically, we will connect the “**gpio_wrapper**” module with the “**bidirec_x**” modules, where x is a number from 1 to 32. The connections will go as follows:

“inp_0” pin of “gpio_wrapper_0” will be connected to “inp” of “bidirec_0”,
“inp_1” pin of “gpio_wrapper_0” will be connected to “inp” of “bidirec_1”, and similarly
these connections will go till the last “inp” connection, which is “inp_31” of “gpio_wrapper”
will be connected to “inp” of “bidirec_31”.

Similarly,

“oe_0” pin of “gpio_wrapper_0” will be connected to “oe” of “bidirec_0”,
“oe_1” pin of “gpio_wrapper_0” will be connected to “oe” of “bidirec_1”, and similarly
these connections will go till the last “oe” connection, which is “oe_31” of “gpio_wrapper”
will be connected to “oe” of “bidirec_31”.

And similarly again,

“outp_0” pin of “gpio_wrapper_0” will be connected to “outp” of “bidirec_0”,
“outp_1” pin of “gpio_wrapper_0” will be connected to “outp” of “bidirec_1”, and similarly
these connections will go till the last “outp” connection, which is “outp_31” of
“gpio_wrapper” will be connected to “outp” of “bidirec_31”.

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are
available here:
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs
/InternalConnections/7_GpioW_32xBidirec.pdf](#)

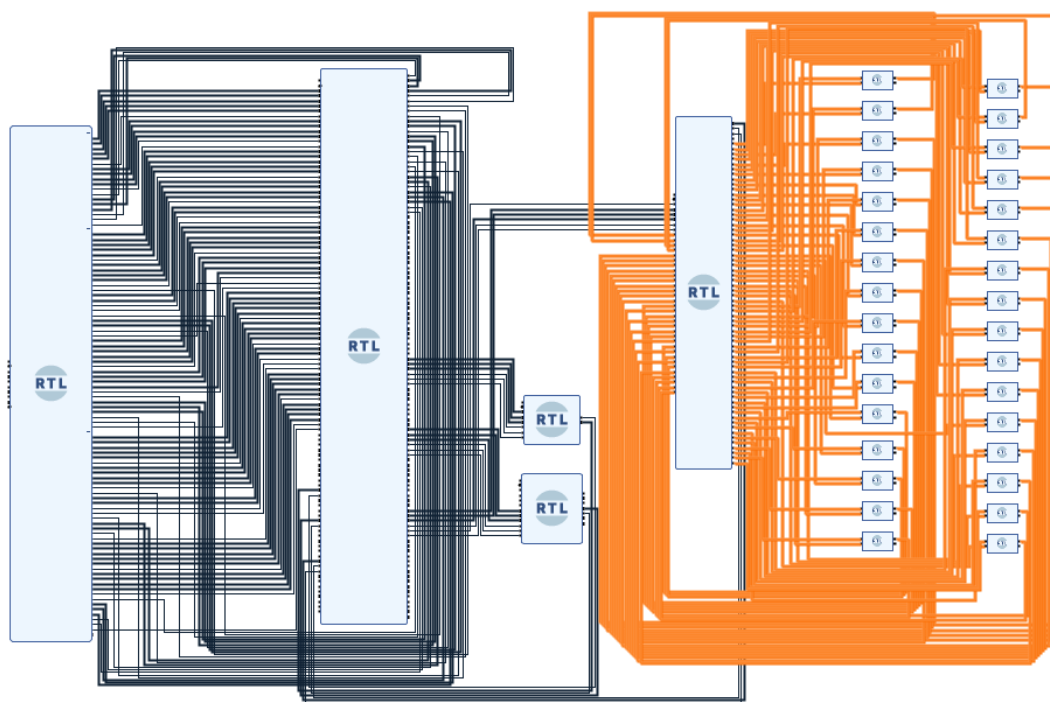


Figure 40. All GPIO Bidirec Modules connected to gpio_wrapper module

Now that we have connected all the internal connections between the modules, we will now make the external connections.

Step 5. Make External Connections for I/O Pins

Now it is time to connect the pins coming into our block design as an Input or going out of our block design as an Output. We will connect these pins as the external Pins/Ports. These

external pins include the pins of **RAM** (DDR), **CLK** (Clock), **RST** (Reset), and **DMI** (Debug Module interface).

We begin by connecting the “clk” pin. Go to the “**swerv_wrapper_verilog**” module, right-click on the “clk” pin, and you will see a dropdown (see Figure 41). Select the option of **Make External** from among all the dropdown options. You can also left-click on the pin and use the shortcut key “CTRL + T” to make the pin External.

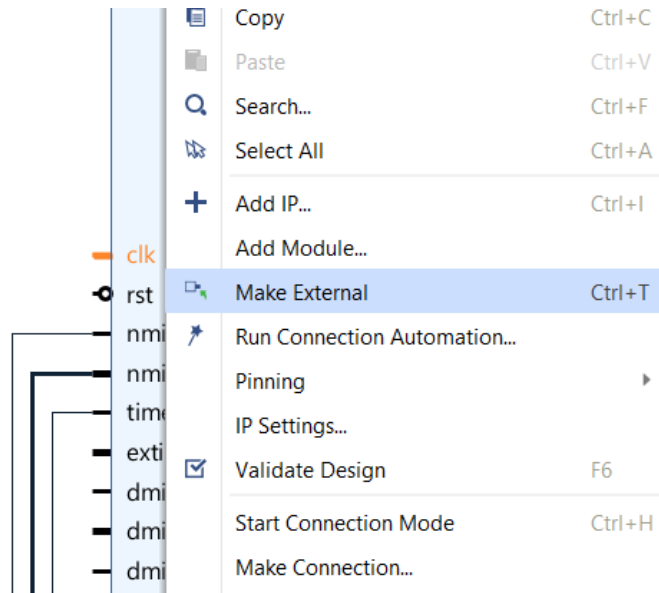


Figure 41. Make “clk” an external connection

You will now see the “clk” pin of “**swerv_wrapper_verilog**” connected to an external pin “**clk_0**” (see Figure 42).

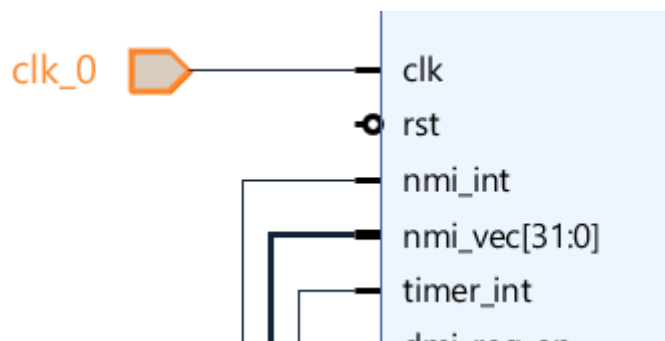


Figure 42. “clk” becomes an external connection

Now we can connect the “clk” external pin to the rest of the modules, including the **intcon_wrapper_bd**, **syscon_wrapper**, **bootrom_wrapper**, and **gpio_wrapper**.

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/1_Clock.pdf

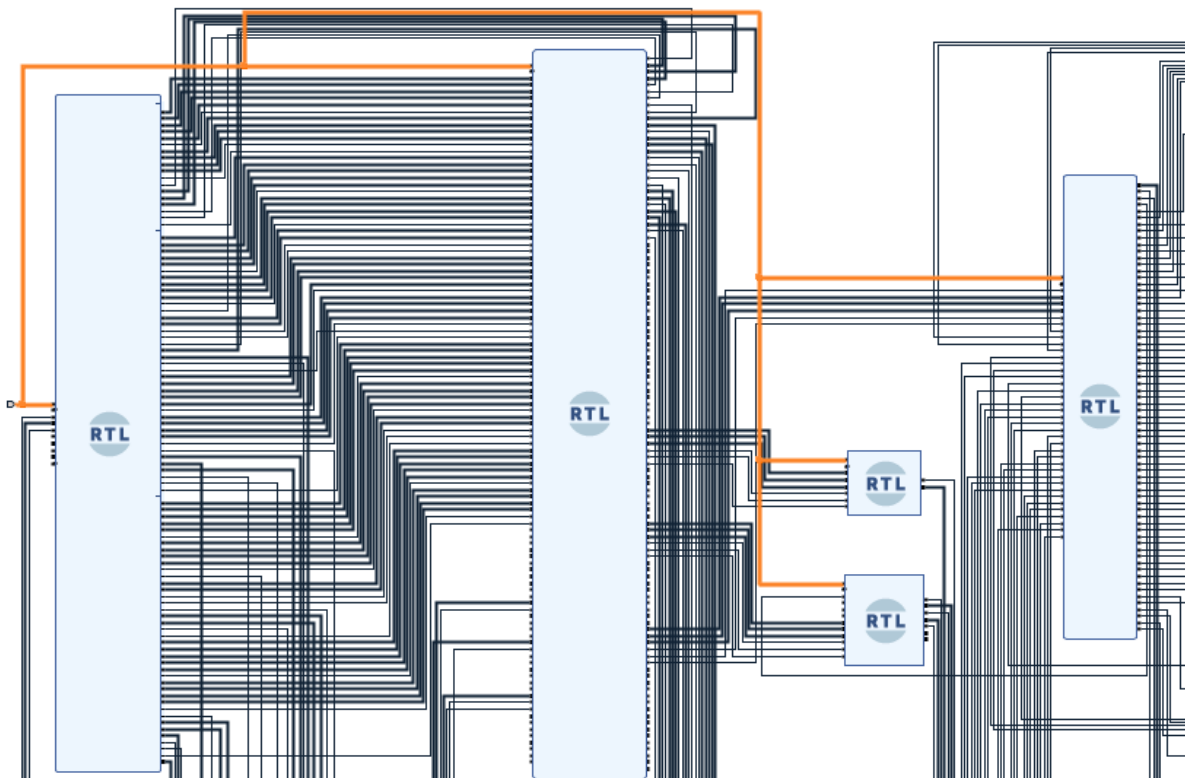


Figure 43. Signal clk Connected to all modules

Similarly, we can connect the “**rst**” pin to all the modules.

Like the external pin we created for “**clk**”, we will create one for “**rst**”. Now again, go to the “**swerv_wrapper_verilog**” module and right-click on the “**rst**” pin, and make it external.

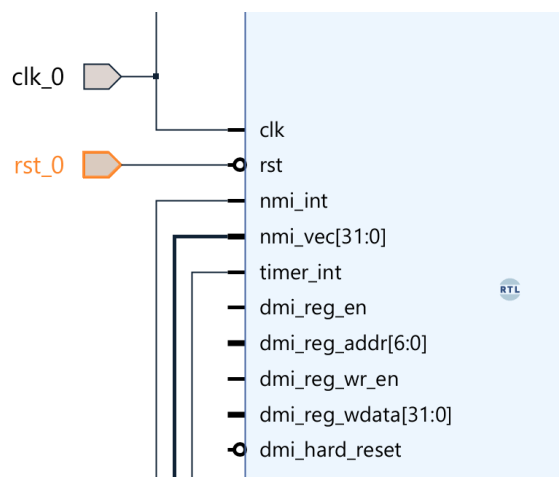


Figure 44. Make rst_0 as an external pin

Now we will connect the “**rst_0**” external pin to the rest of the modules, including the **intcon_wrapper**, **syscon_wrapper**, **bootrom_wrapper**, and **gpio_wrapper**.

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/2_Reset.pdf

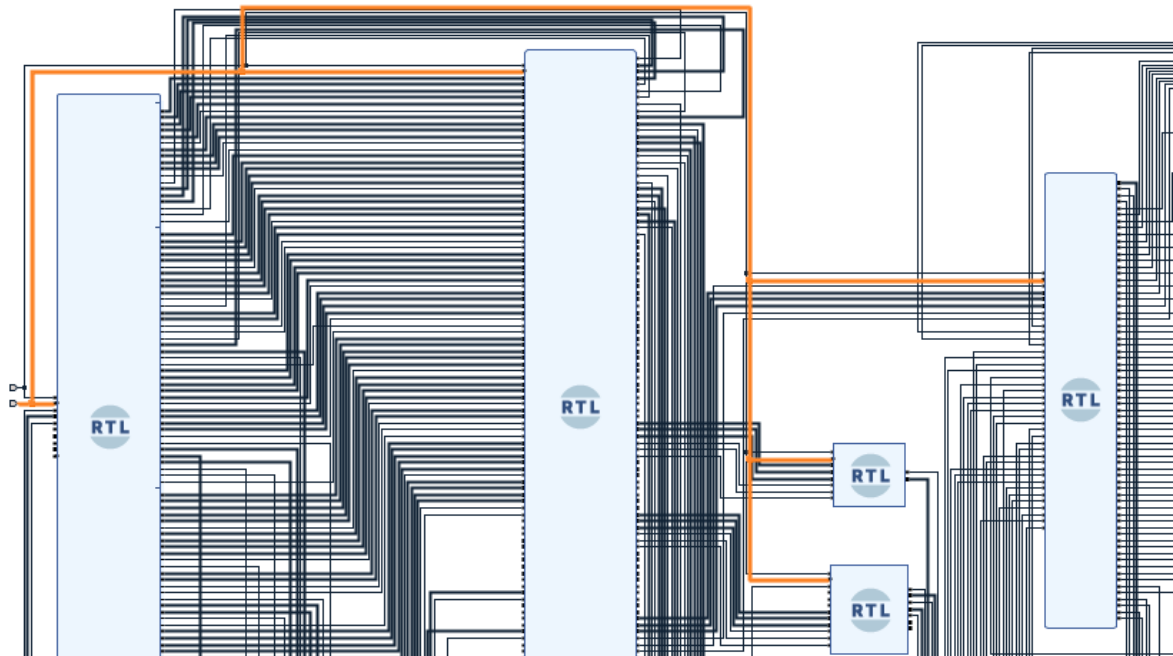


Figure 45. Connect the Inverted “rst_0” pin with the rest of the modules

Now we will connect all the RAM (DDR) pins of the “Intcon_wrapper_bd” module to the external RAM pins by completing the following steps.

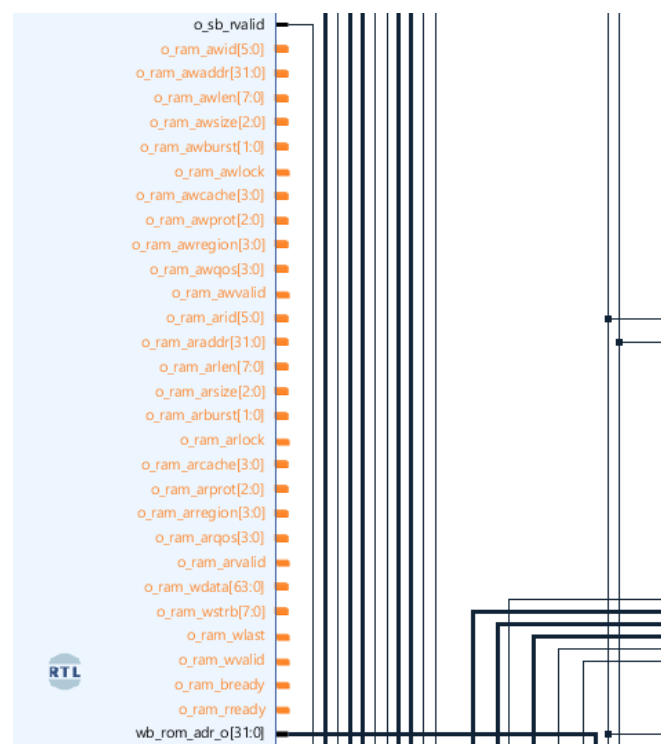


Figure 46. Intcon wrapper right-hand side RAM pins

We will now make all the right-hand side RAM pins in the “**Intcon_wrapper_bd**” module as external pins (See Figure 47).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/3_RAM_R.pdf

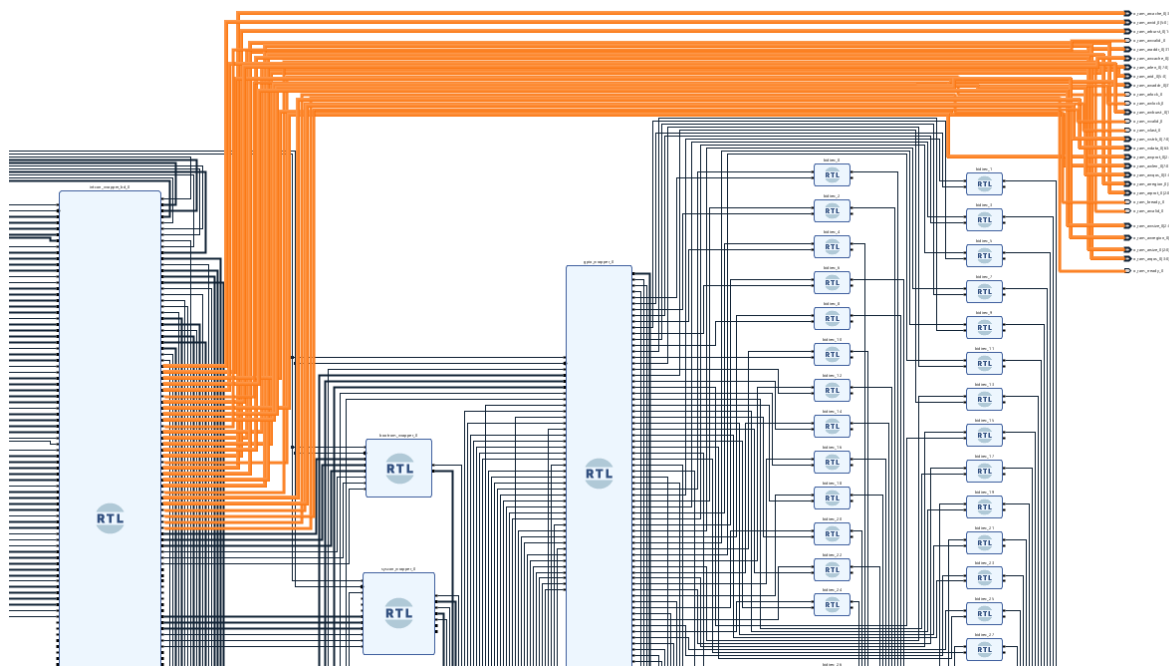


Figure 47. Make all the right-hand side RAM pins as External

Now we will make the left-hand side RAM pins of “**Intcon_wrapper_bd**” into external pins.

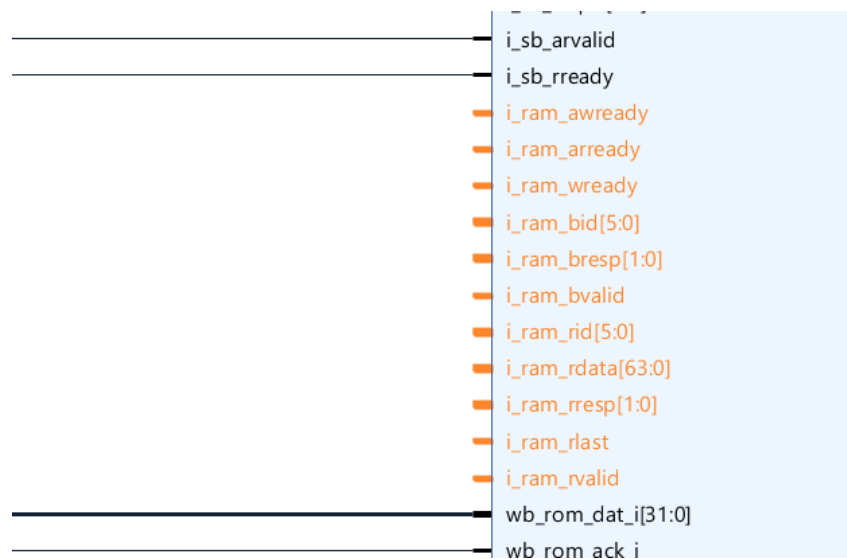


Figure 48. Left Side RAM Pins of Interconnect Wrapper

We will make all these RAM pins as external pins (see Figure 49).

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/4_RAM_L.pdf

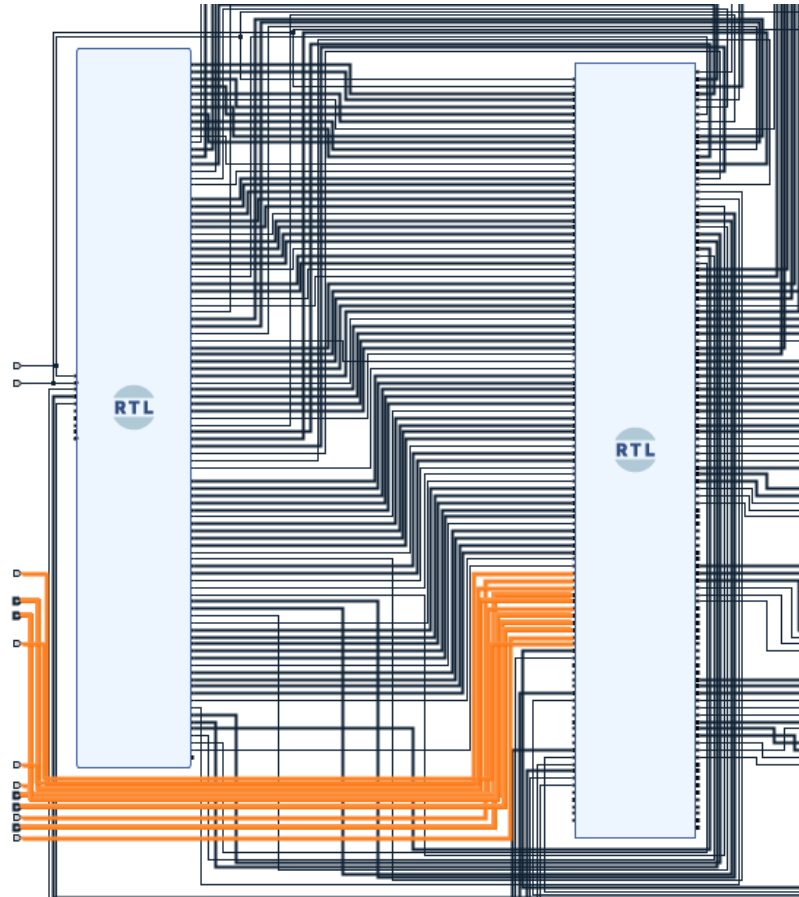


Figure 49. Make all the left-hand side RAM pins as External

Now we will connect the **DMI** pins of the “**swerv_wrapper_verilog**” module with the external pins.

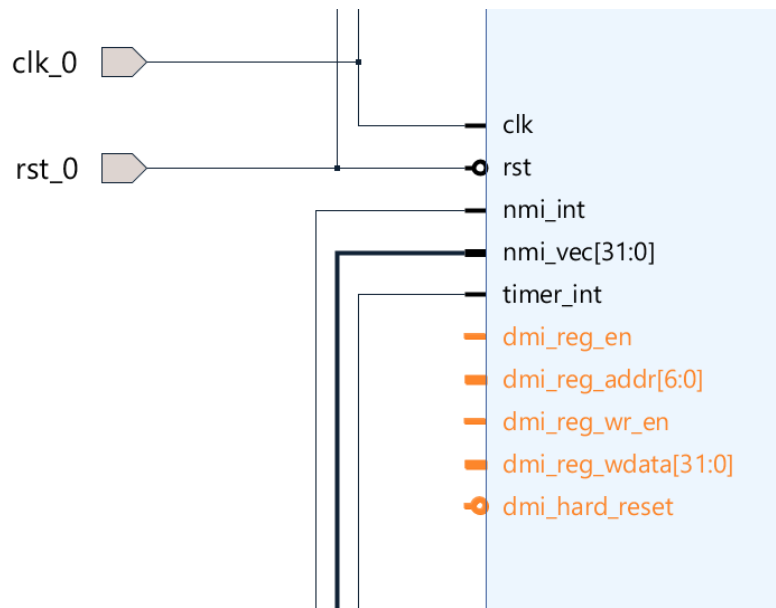


Figure 50. dmi pins on swerv_wrapper_verilog (Left Side)

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/5_DMI.pdf

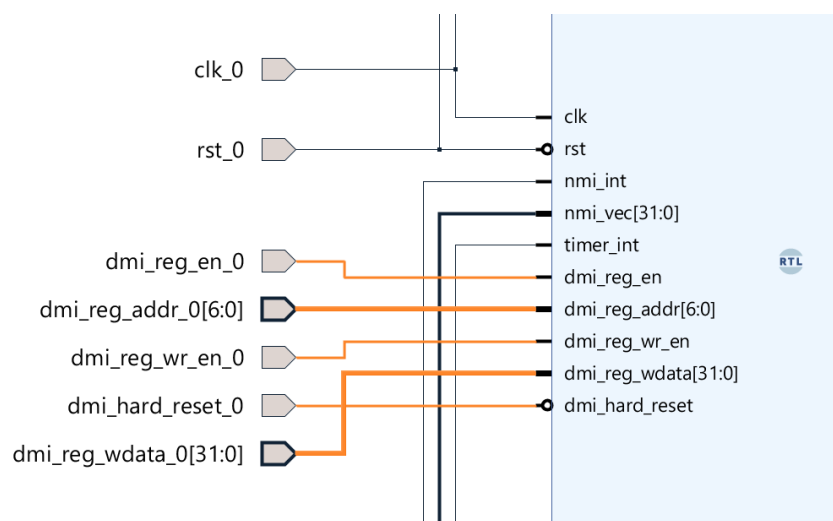


Figure 51. Making dmi pins as External Pins

We will connect one more pin with the external pin on the bottom right-hand side of the “swerv_wrapper_verilog” module. This pin is “dmi_reg_rdata[31:0]”.

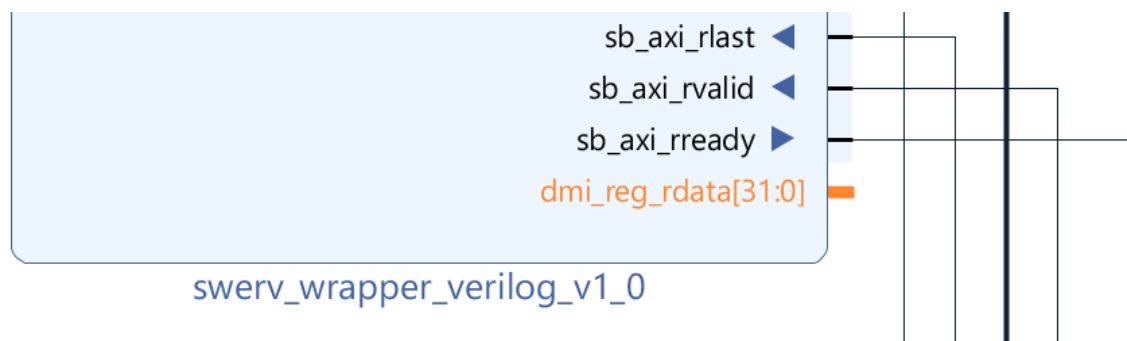


Figure 52. “dmi_reg_rdata[31:0]” Pin (Right Side of swerv_wrapper_verilog)

We will make “dmi_reg_rdata[31:0]” as an external pin as well.

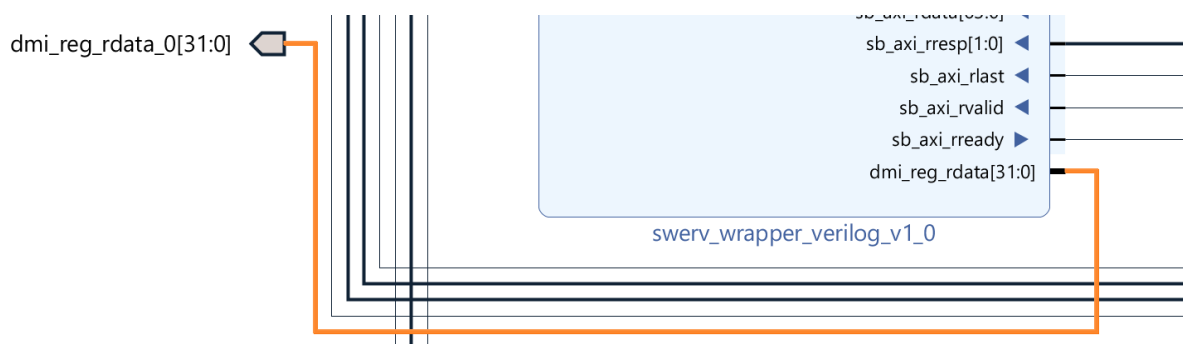


Figure 53. Make “dmi_reg_rdata[31:0]” Pin as an External Pin

We will now make the following pins of the “syscon_wrapper” module as external pins.

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

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/6_SysconW_External.pdf](#)

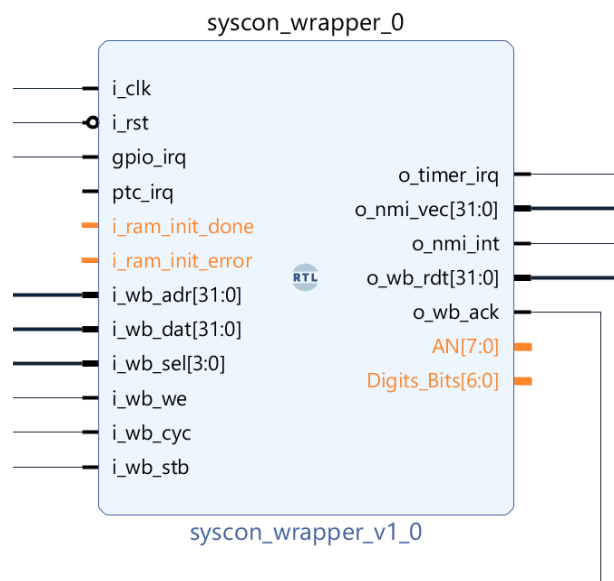


Figure 54. syscon_wrapper's external pins

The last connection left is to make all the “bidir” pins of all the “**bidirec**” modules as external pins.

Note: Make these connections external one by one starting from the “**bidirec_0**” module so the “**bidir**” pin of the “**bidirec_0**” module will be connected to an external pin “**bidir_0**”. Then go to the “**bidir**” pin of “**bidirec_1**”, and so on.

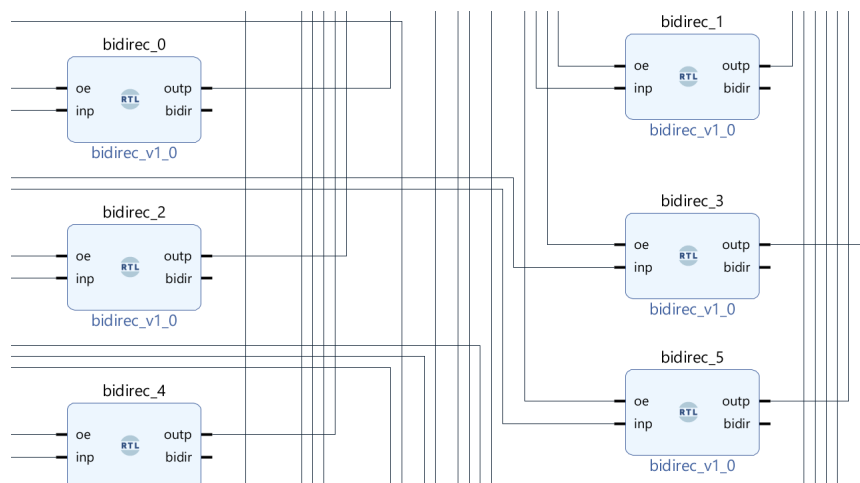


Figure 55. Make “bidir” Pin of our GPIO Bidirec Modules as External Pins

PDF: High-quality PDFs of the Block Design showing close-up details of the wiring are available here:
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/7_Bidir.pdf](#)

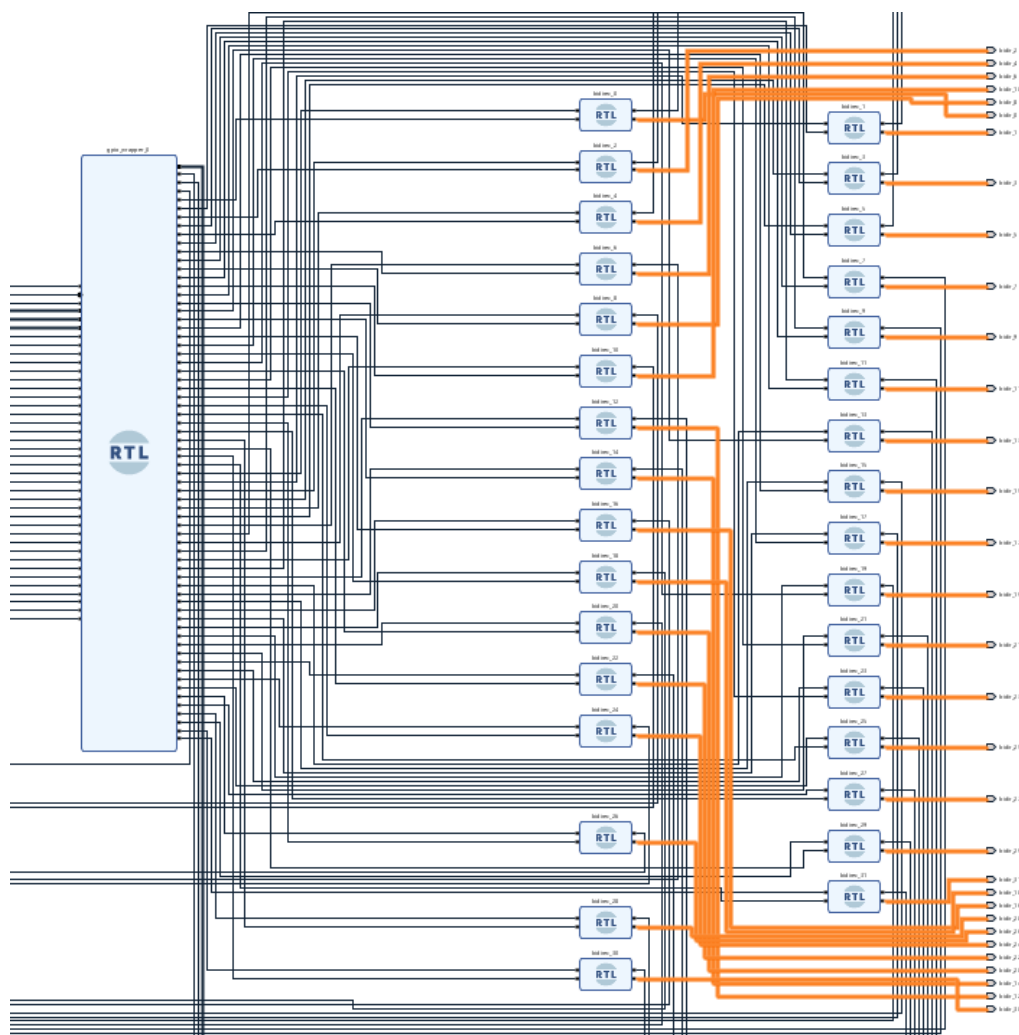


Figure 56. Make “bidir” External connections

Now we have completed all of both internal and external connections for the Block design SoC. Press **“Ctrl + S”** to save the block design.

Our block design, which is modeled after the SweRVofX SoC has been completed, It now contains the following connected modules:

- 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)

(See Figure 57).

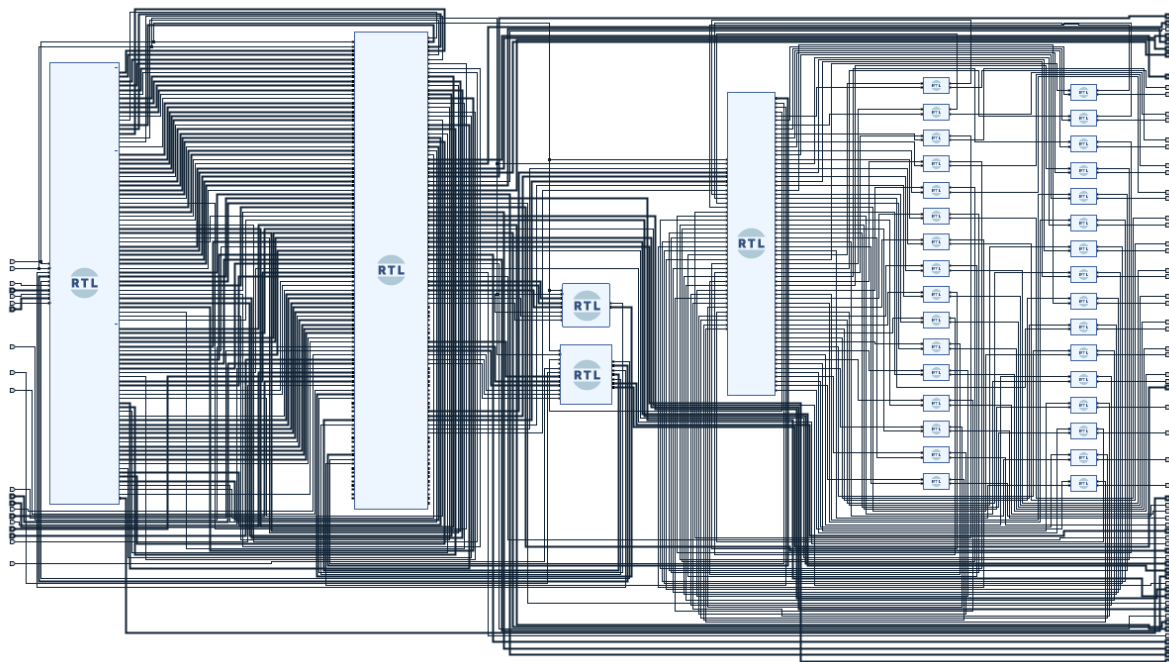


Figure 57. Block design SoC completed

5. Generating The Block Design Module Verilog File

We will now generate the Verilog module file of the block design that we have created.

Step 1. Navigate to the sources panel and find the block design module “BD” that we just created.

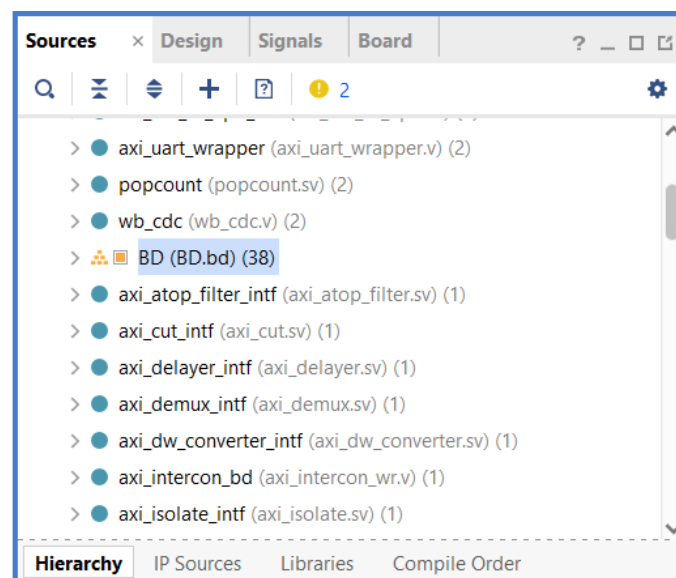


Figure 58. Find “BD” in Sources

Step 2. Now right-click on that Block Design (BD) and then select “Create HDL Wrapper” (see Figure 59).

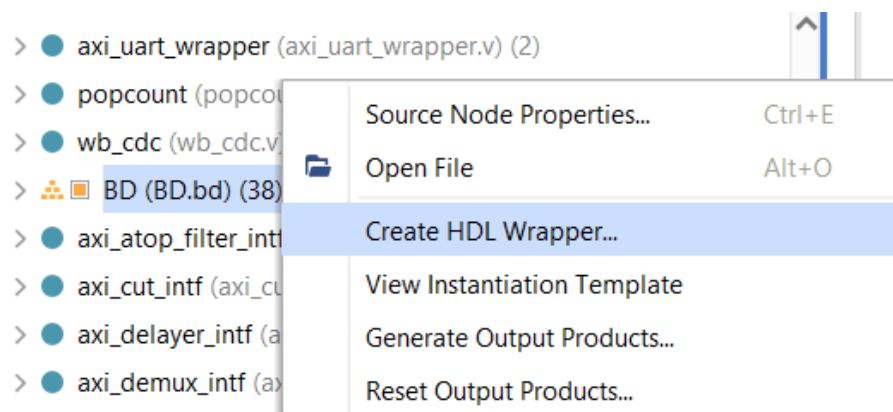


Figure 59. Create HDL Wrapper

Step 3. Select the “Let Vivado manage wrapper and auto-update” option and click OK to proceed.

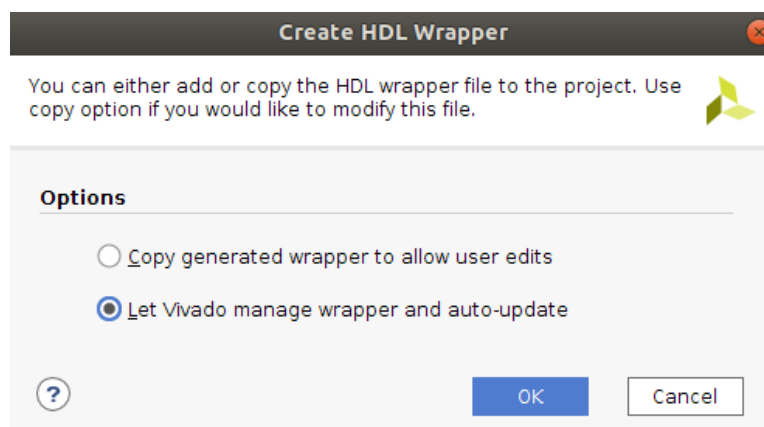


Figure 60. Select the second Option

You will see a pop-up of critical warnings because we have left several pins in our block design unconnected so that these pins will be automatically connected to “0” (ground).

Click OK.

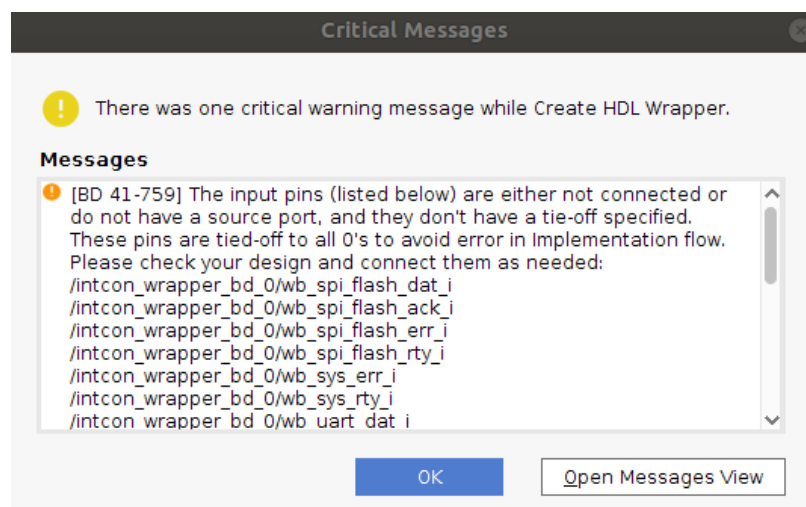


Figure 61. Warning Pop-Up

Now the Block design's HDL Wrapper has been created. You can navigate to the Sources panel and scroll down until you see “**BD_wrapper**”. Click on the dropdown icon next to it and then again for “**BD_i**”.

Now open the “**BD (BD.v)**” file by double-clicking on it (see Figure 62).

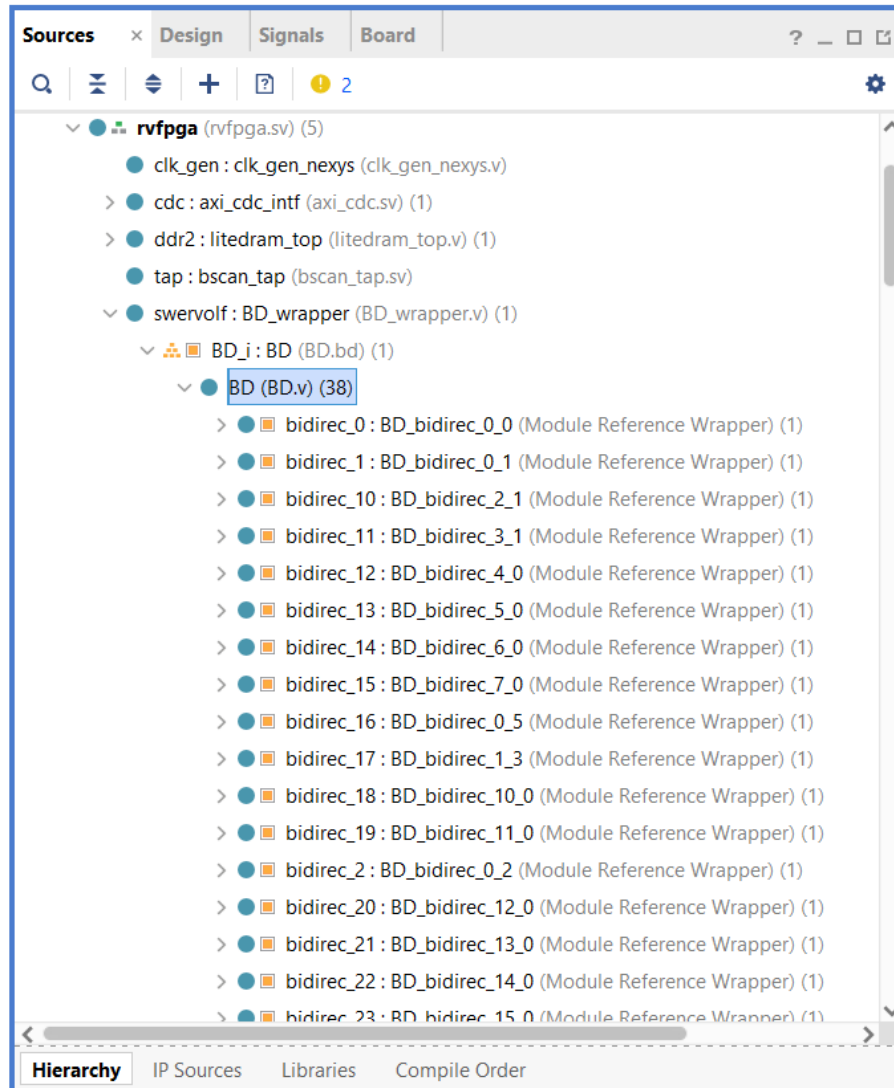
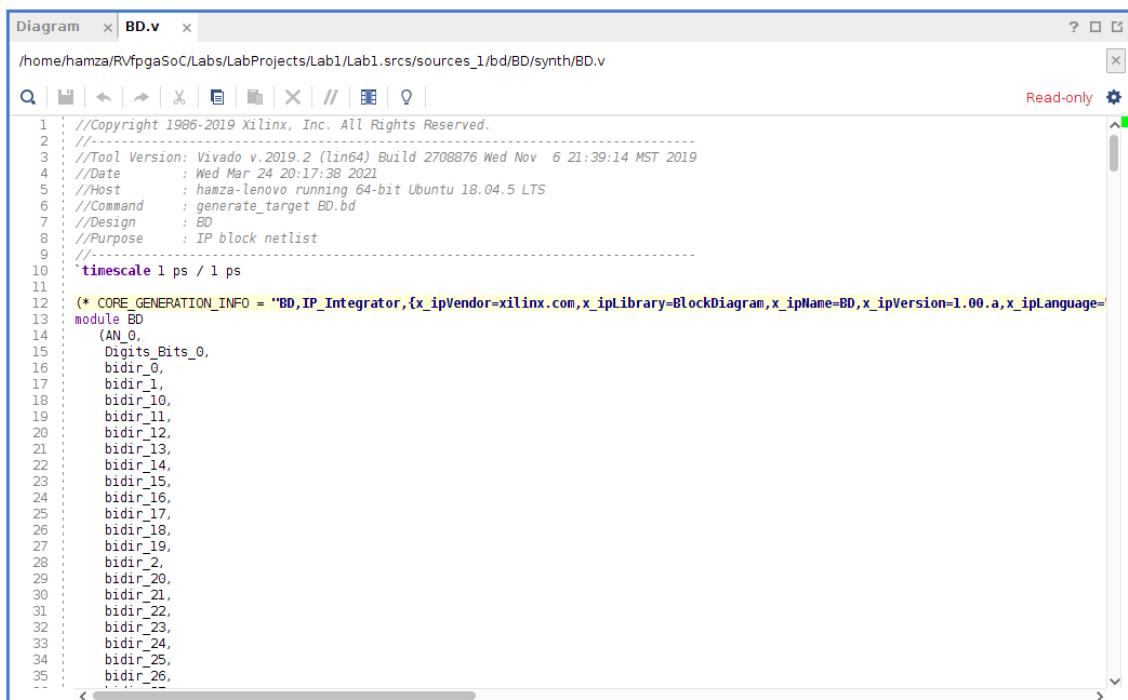


Figure 62. Find “BD.v” in the sources panel

Here you see the “**BD.v**” Verilog file that has been created using Vivado’s Block Design tool.



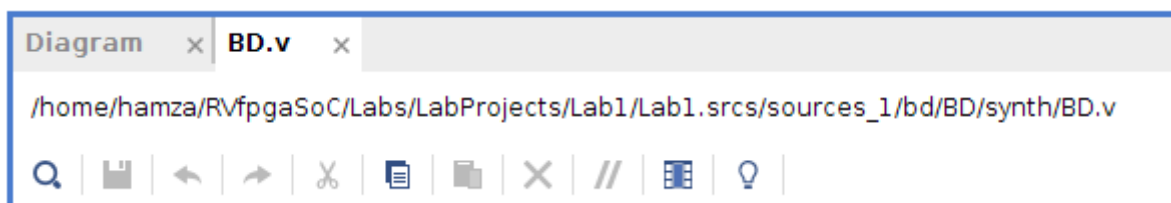
```

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,

```

Figure 63. “BD.v”

You can see this newly created file’s path at the top of the file. In the next Lab, we will use this path to access this “BD.v” file.



```

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

```

Figure 64. Path of the “BD.v” file

6. Generate Bitstream

Now that we have created the SweRVolfX subset using Vivado’s Block Design tool and generated a Verilog wrapper, we are ready to generate the bitstream which we will use to configure the FPGA. To generate the bitstream, we will first need to adjust some settings in Vivado by completing the following steps.

Step 1. Navigate to Settings.

Go to “Tools” in the upper left side of the Navigation Bar of Vivado, then select “Settings” from the options.

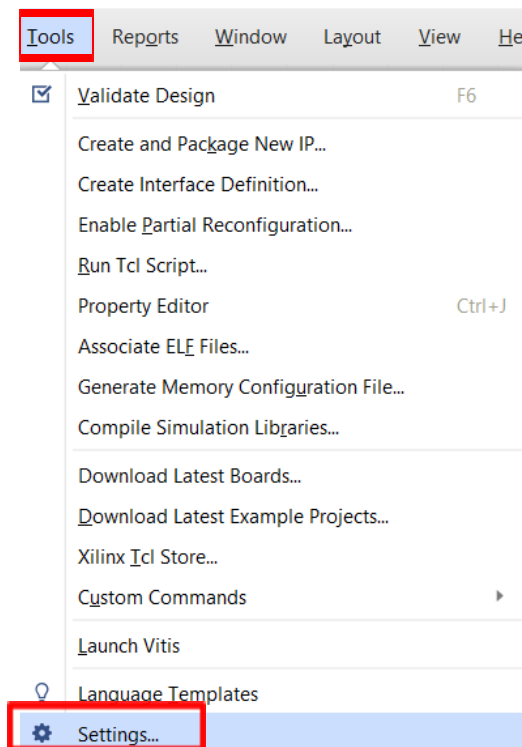


Figure 65. Go to Settings

Step 2. Navigate to the General tab

Go to the “**General**” tab, then select “**Verilog options**” from the language options section.

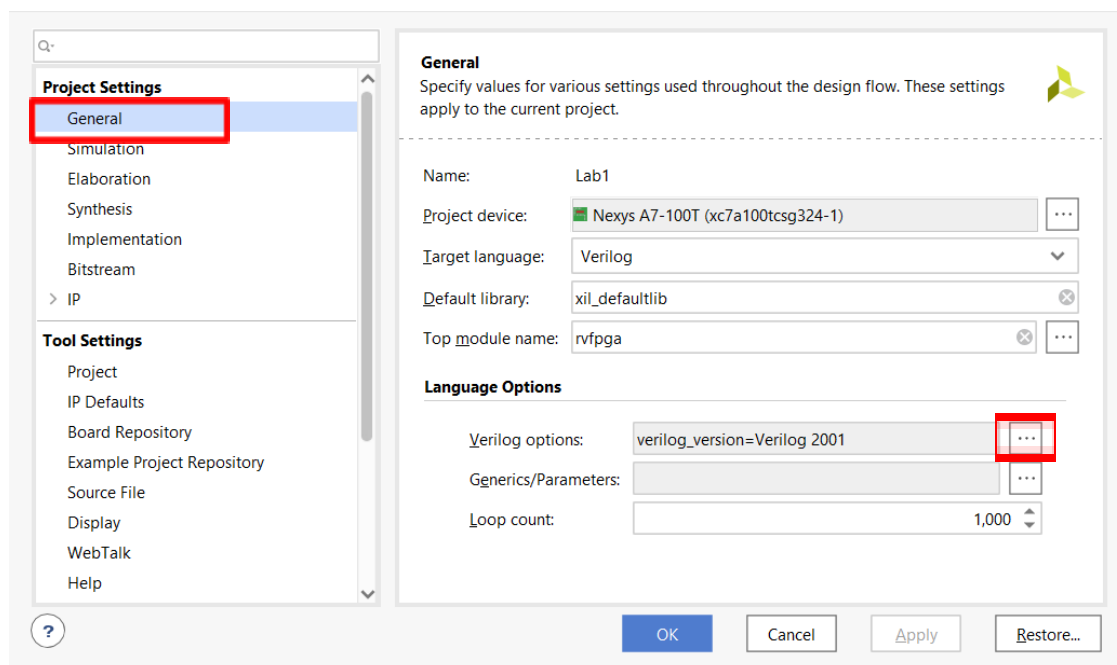


Figure 66. General Settings

Step 3. Add the path to the include files.

Click on the “+” button to add the Verilog search path **Include Files**.

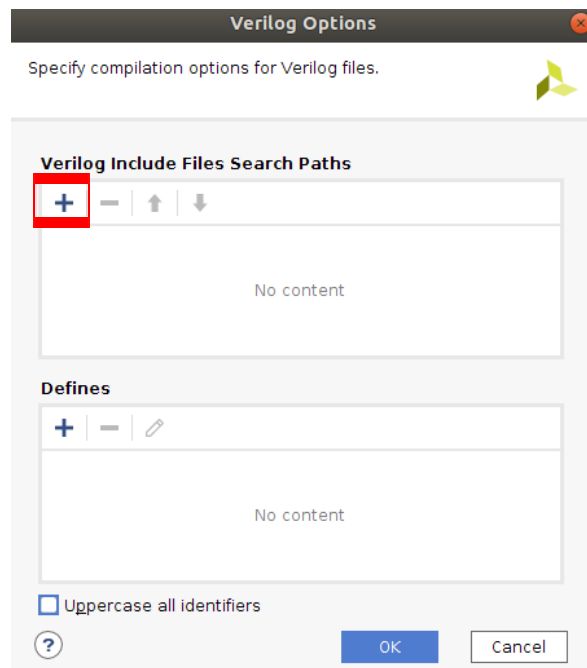


Figure 67. Verilog Options

Now add the following three paths :

- [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



Figure 68. Verilog Include Files Paths

Click OK.

Step 4. Navigate to the Bitstream tab

Go to the “**Bitstream**” tab, then click on “**tcl.pre**” button.

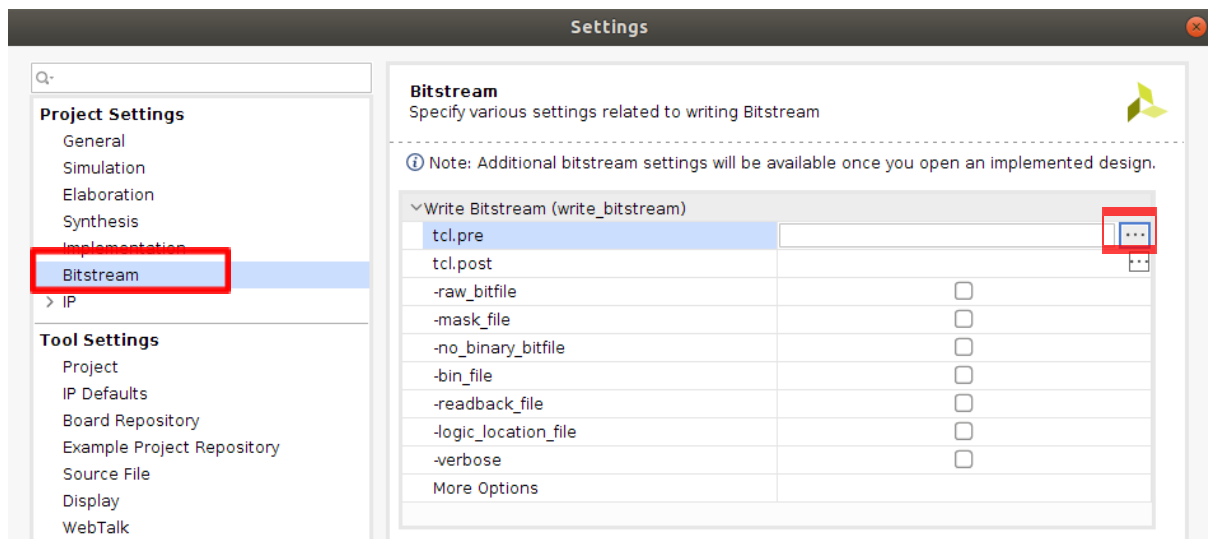


Figure 69. Bitstream setting

Select the “New script” option.

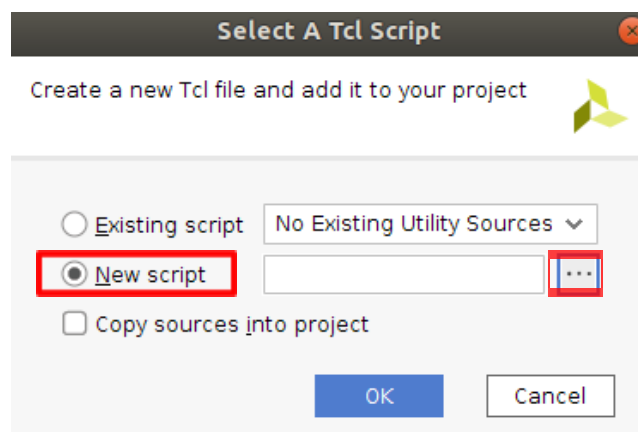


Figure 70. New Tcl script

Navigate to the following path and select the “script.tcl” file. (see Figure 71)
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/script.tcl

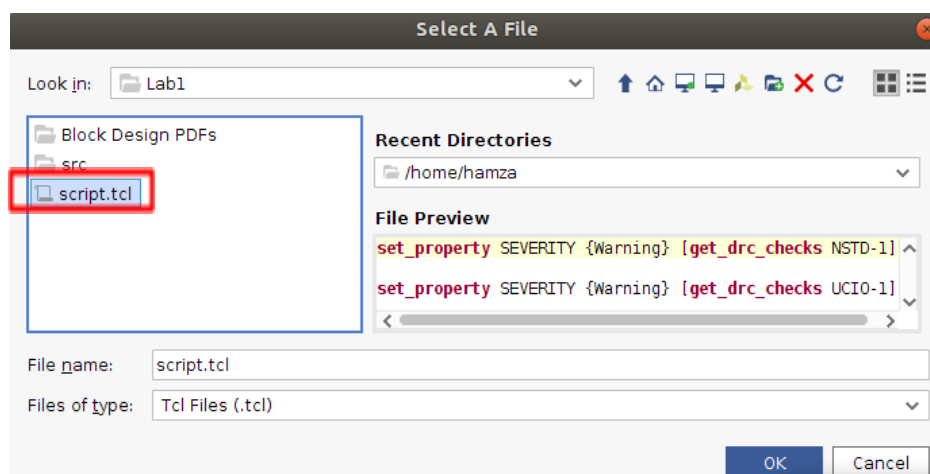


Figure 72. import “script.tcl” file

Click OK and apply the changes.

Step 4. Generate Bitstream.

Now Click on Flow → Generate Bitstream, as shown in Figure 73.

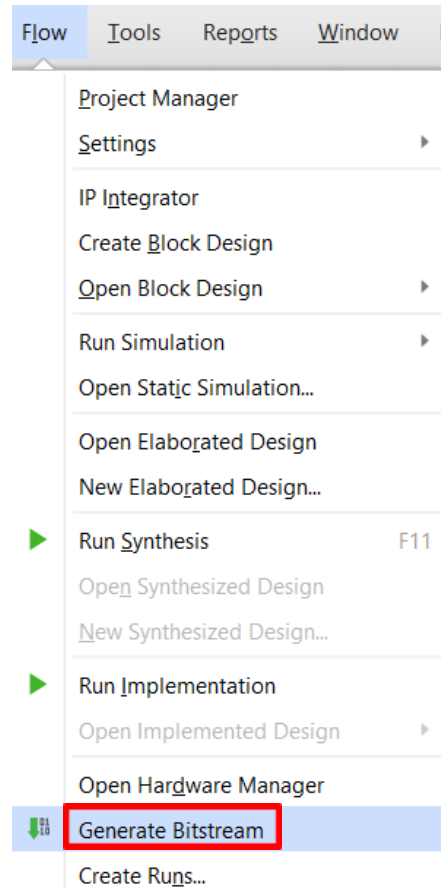


Figure 73. Generate Bitstream

A window might pop up that says there are no implementation results available and ask to launch synthesis and implementation.

Click Yes (see Figure 74).

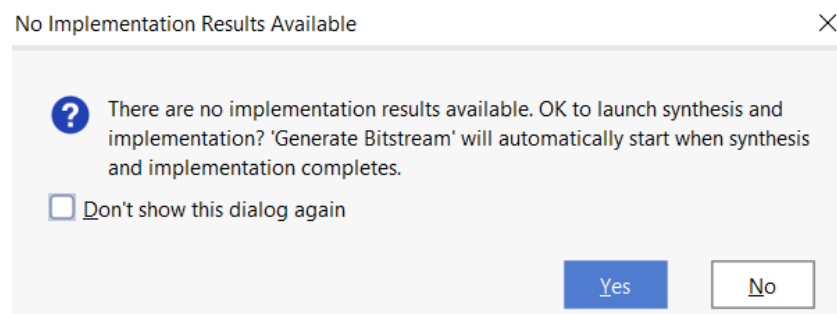


Figure 74. Launch synthesis and implementation window

The **Launch Runs** window will pop up on the screen (see Figure 75). Click OK.

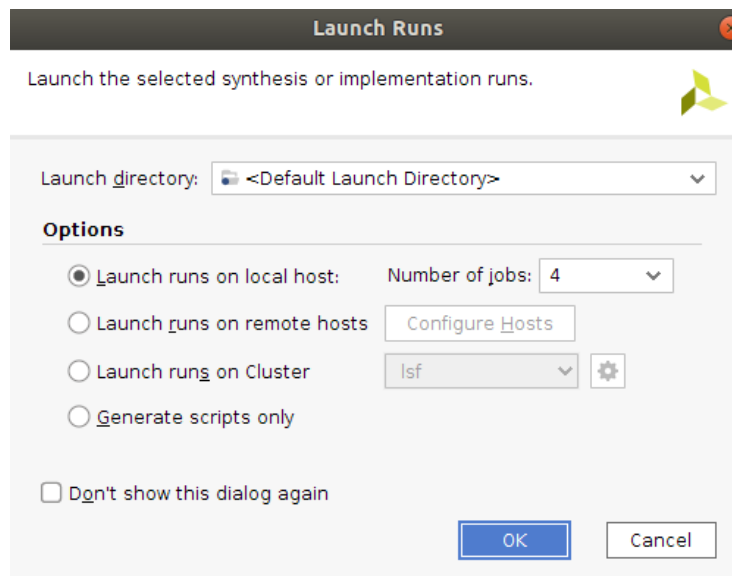


Figure 75. Launch Runs

Now we will see a list of warnings that tell us about the pins we left unconnected will be automatically connected to "0". We will click OK. (see Figure 76).

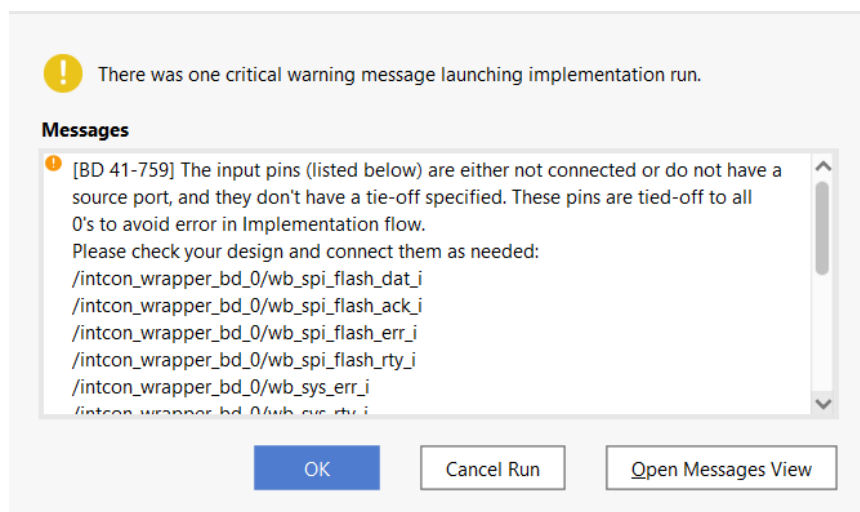


Figure 76. Launch Runs Warning Messages

This step synthesizes **RVfpgaNexys** (as defined by the Verilog and SystemVerilog files in the project), maps it onto the FPGA, and creates the bitstream.

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

Figure 77. Design Runs

Note: If you get an error like: Gtk-Message: Failed to load module "canberra-gtk-module"
Install a package by the following command to solve the issue.

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

If you are using a VM, Vivado might crash while synthesis due to low RAM allocation. It is recommended to allocate more RAM to the VM if Vivado crashes.

This process may take several minutes, depending on your computer's speed.

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

Figure 78. Verilog Include Files Path

After the bitstream has been generated, a window will pop up, as shown in Figure 79. Click on the X button in the top-right corner to close the window.

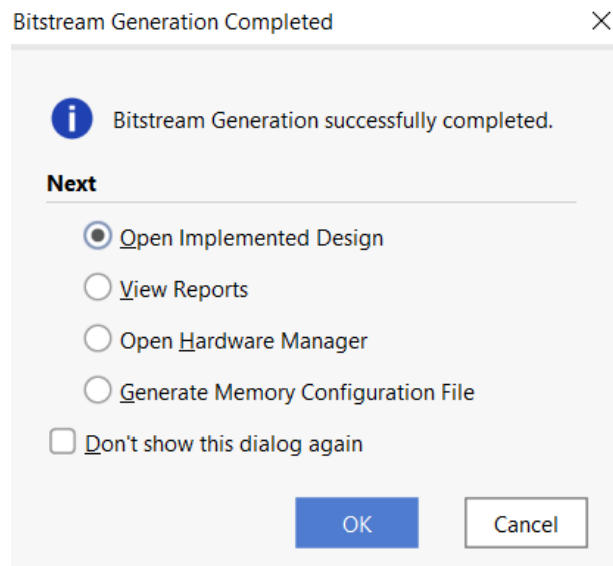


Figure 79. Bitstream Generation Completed

Now that the bitstream has been created, in the next Lab, we will show how to upload this bitstream onto a Nexys A7 board via PlatformIO, and then we will show how to run example programs on the SweRVolfX subset that we have just built in this lab.