

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga-SoC Lab 1**

**Introduction to RVfpga-SoC**

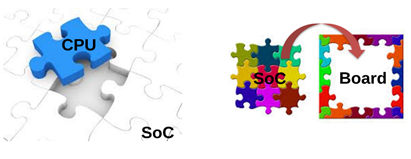
**Table 1. RVfpga Terms**

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

# Introduction

**Introduction to a System on a Chip**

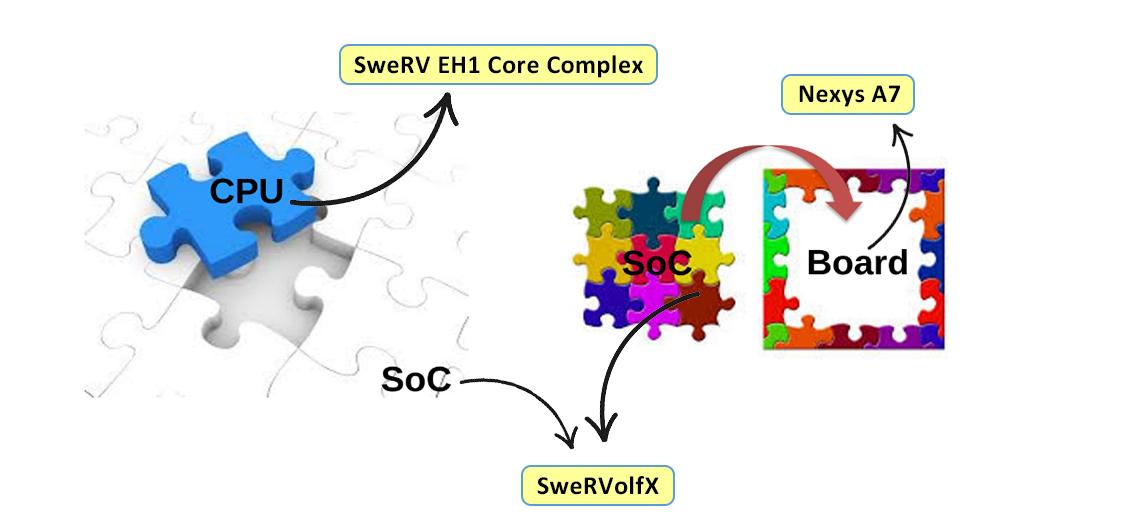
In this lab, we will show how to build a RISC-V system on a chip (SoC) from building blocks. An **SoC** is an integrated circuit or an IC that integrates an entire electronic or computer system onto it. An SoC includes a core and all of the peripherals and interfaces necessary to load an operating system 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.



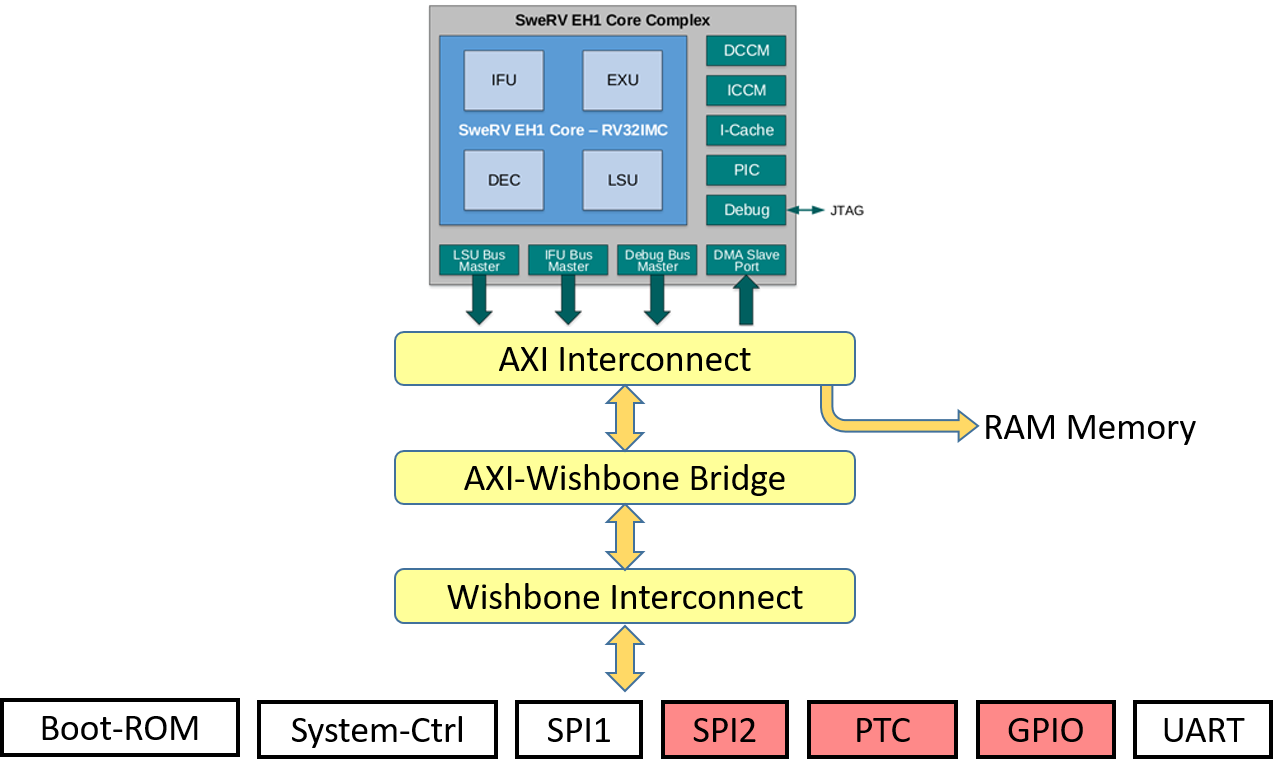
**Figure 2. RVfpga System based Embedded system**

**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](https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7)), 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. SweRVolfX**

**Introduction to RVfpga-SoC**

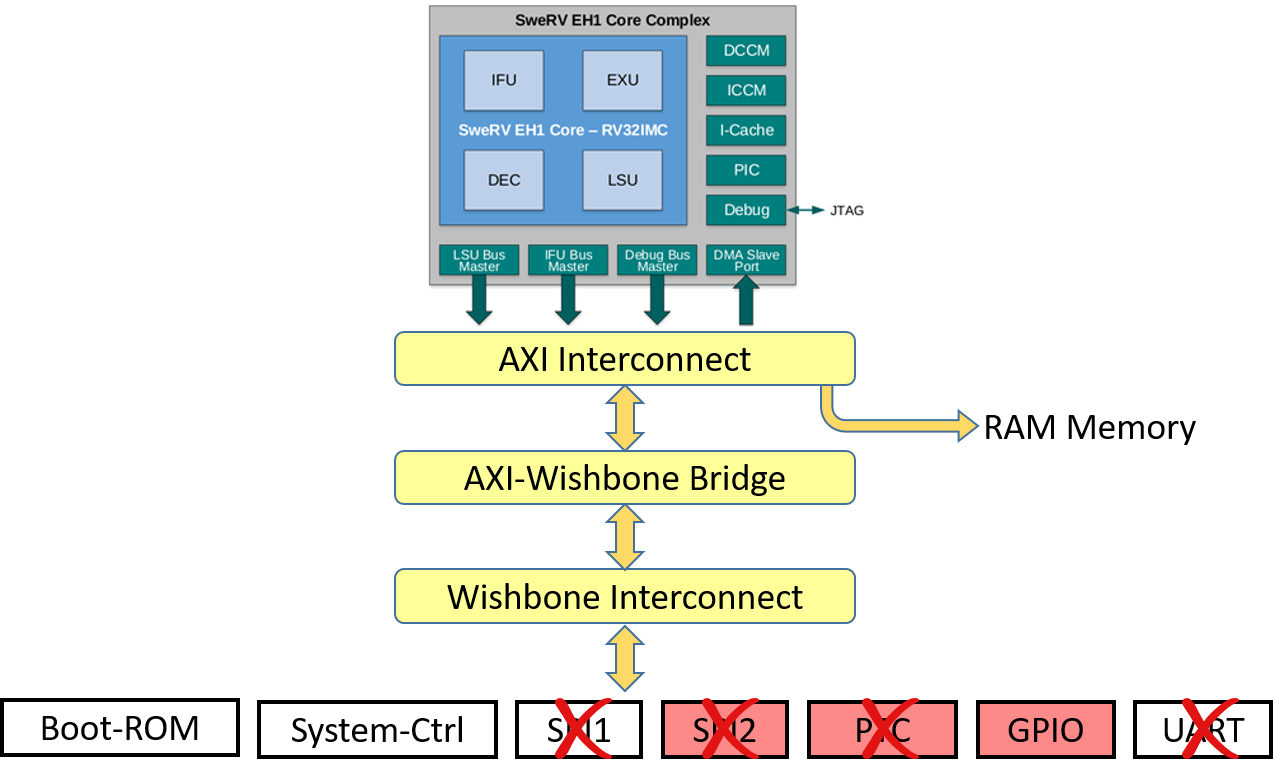
In RVfpga, SweRVolfX was introduced without any detail regarding how SweRVolfX was created. The RVfpga-SoC course shows how to build a subset of SweRVolfX SoC from scratch using building blocks such as the SweRV core, memories, and peripherals.

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:

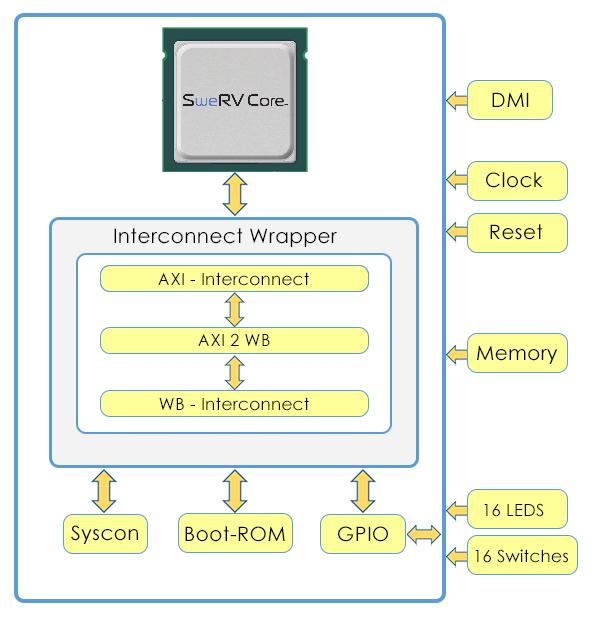
1. CPU (SweRV EH1 Core Complex)
2. Interconnect (AXI-Interconnect, AXI2WB, and WB-Interconnect)
3. Peripherals (Boot-ROM, GPIO controller, and System controller)

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/> |

# 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.

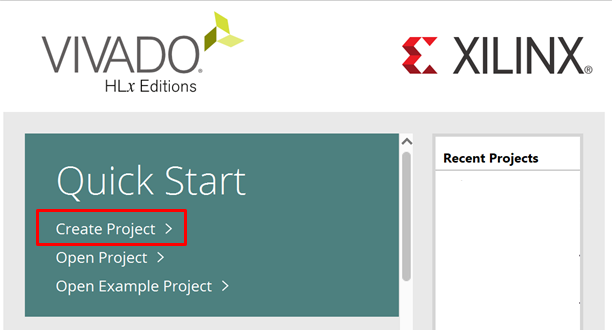
# Create Vivado Project

You will use Xilinx’s Vivado Design Suite to build the SweRVolfX 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.

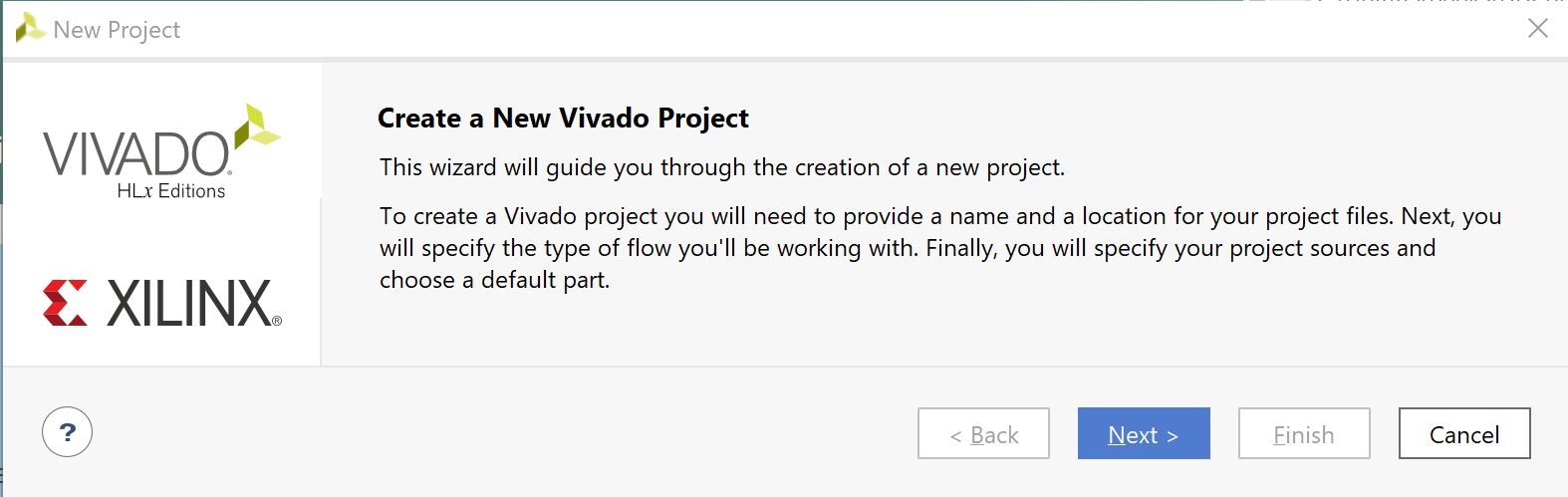
Now, run Vivado (in **Linux**, open a terminal and type: **vivado**; in **Windows**, open Vivado from the Start menu). The Vivado welcome screen will open. Click on Create Project (see Figure 6).



**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.

****

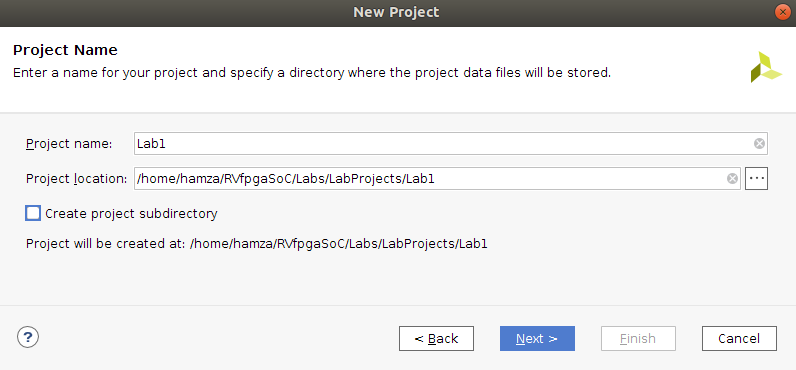
**Figure 7. Create a New Vivado Project Wizard**

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

Select the following Project Location Path :

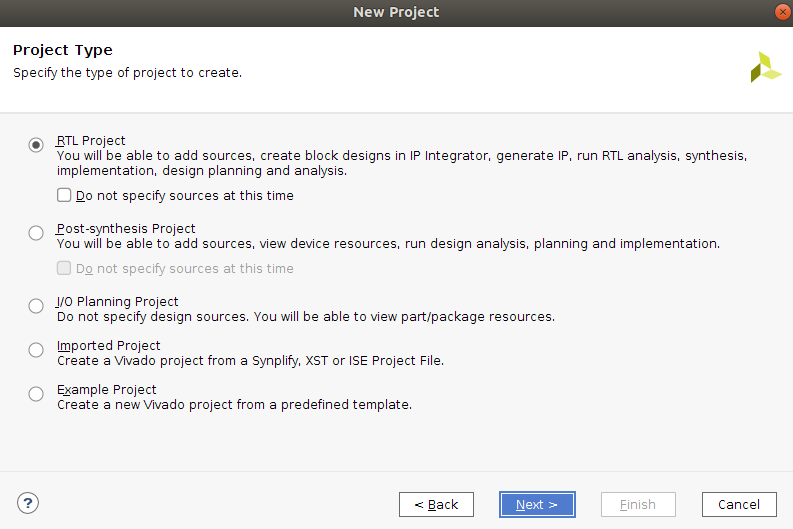
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/Lab1

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



**Figure 8. Project Name**

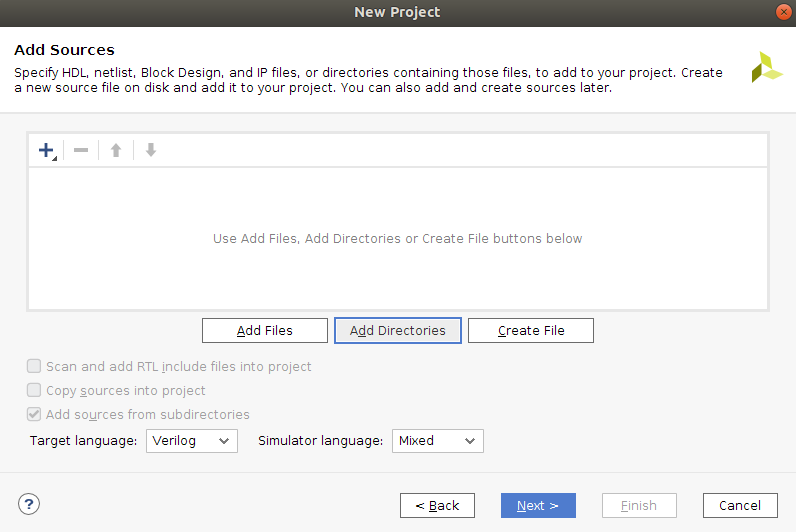
Select the project type as RTL Project, and click Next (see Figure 9).

****

**Figure 9. RTL Project**

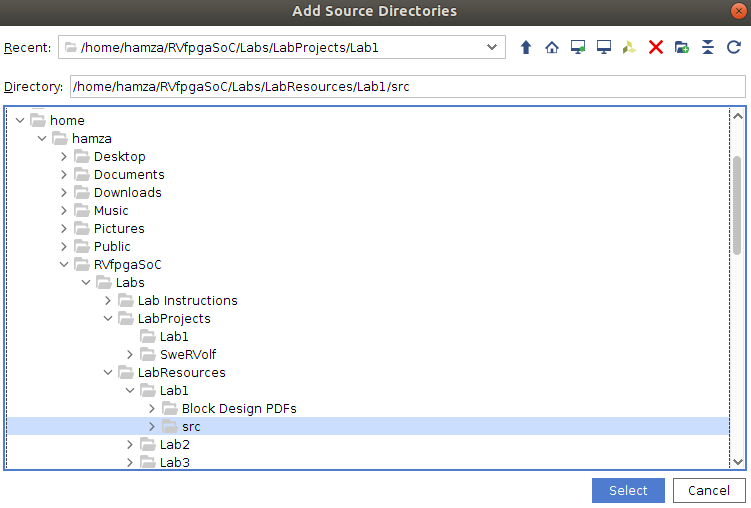
**Step 3. Add the RTL source files and the constraint files**

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

****

**Figure 10. Add Sources directory**

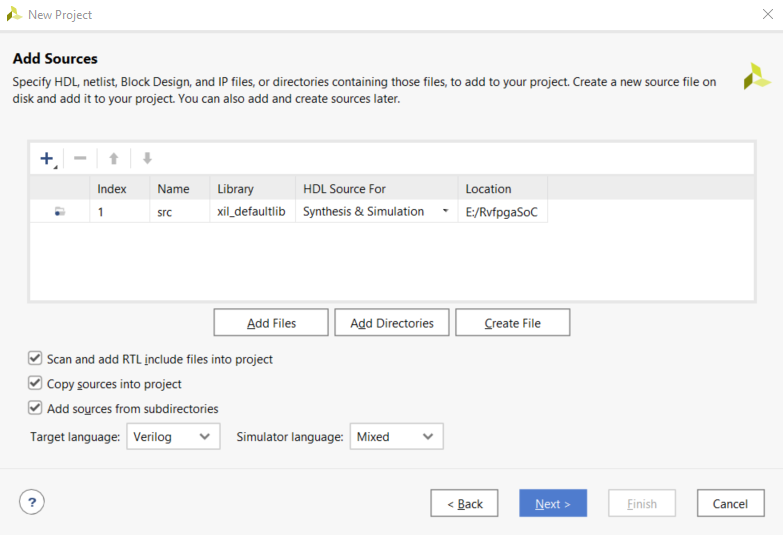
Now select the “src” directory at the following path [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src (See Figure 11).



**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.

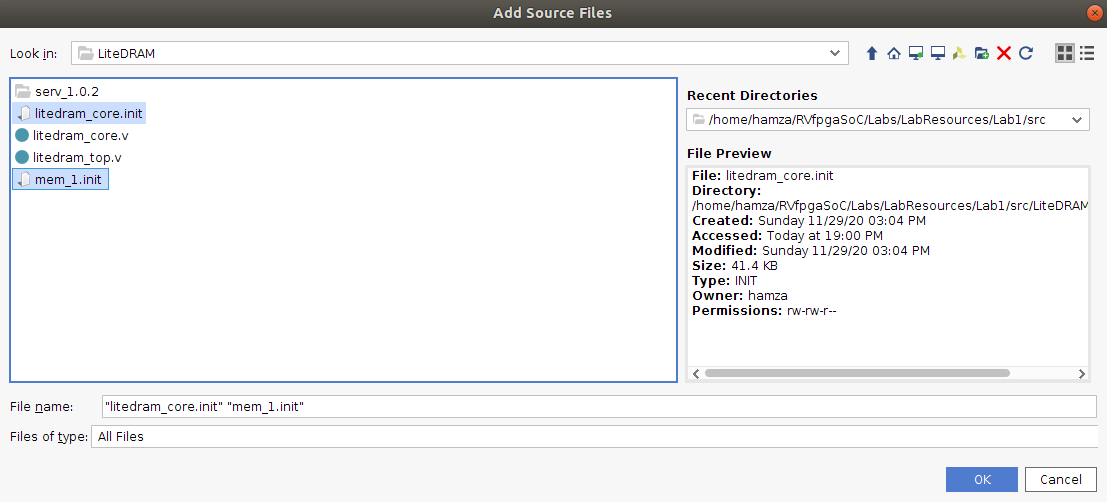
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM

/mem\_1.init

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM

/litdram\_core.init

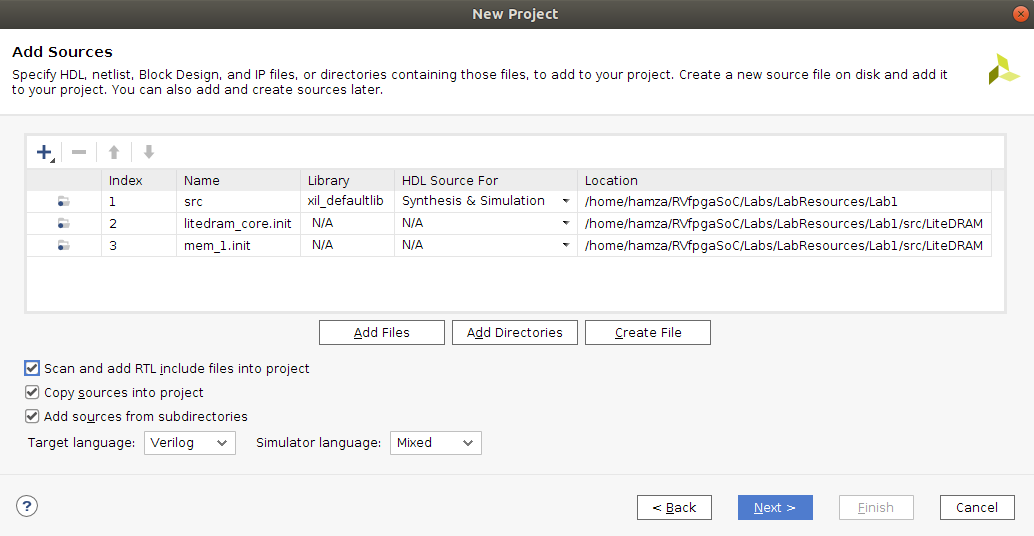
Select both the “**.init**” files and click OK to add them both (See Figure 13).



**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] }]

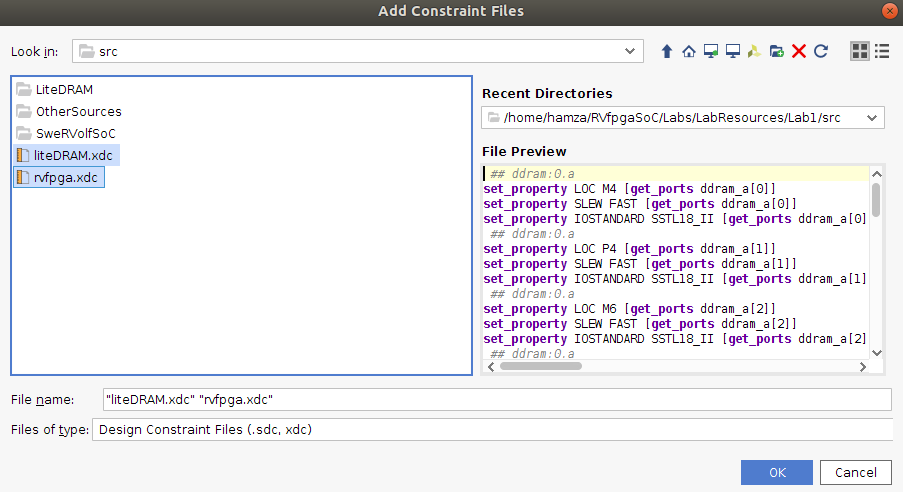
Note that the signal name o\_led is the name used in RVfpga’s Verilog code to drive the Nexys A7 board’s LEDs.

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**.

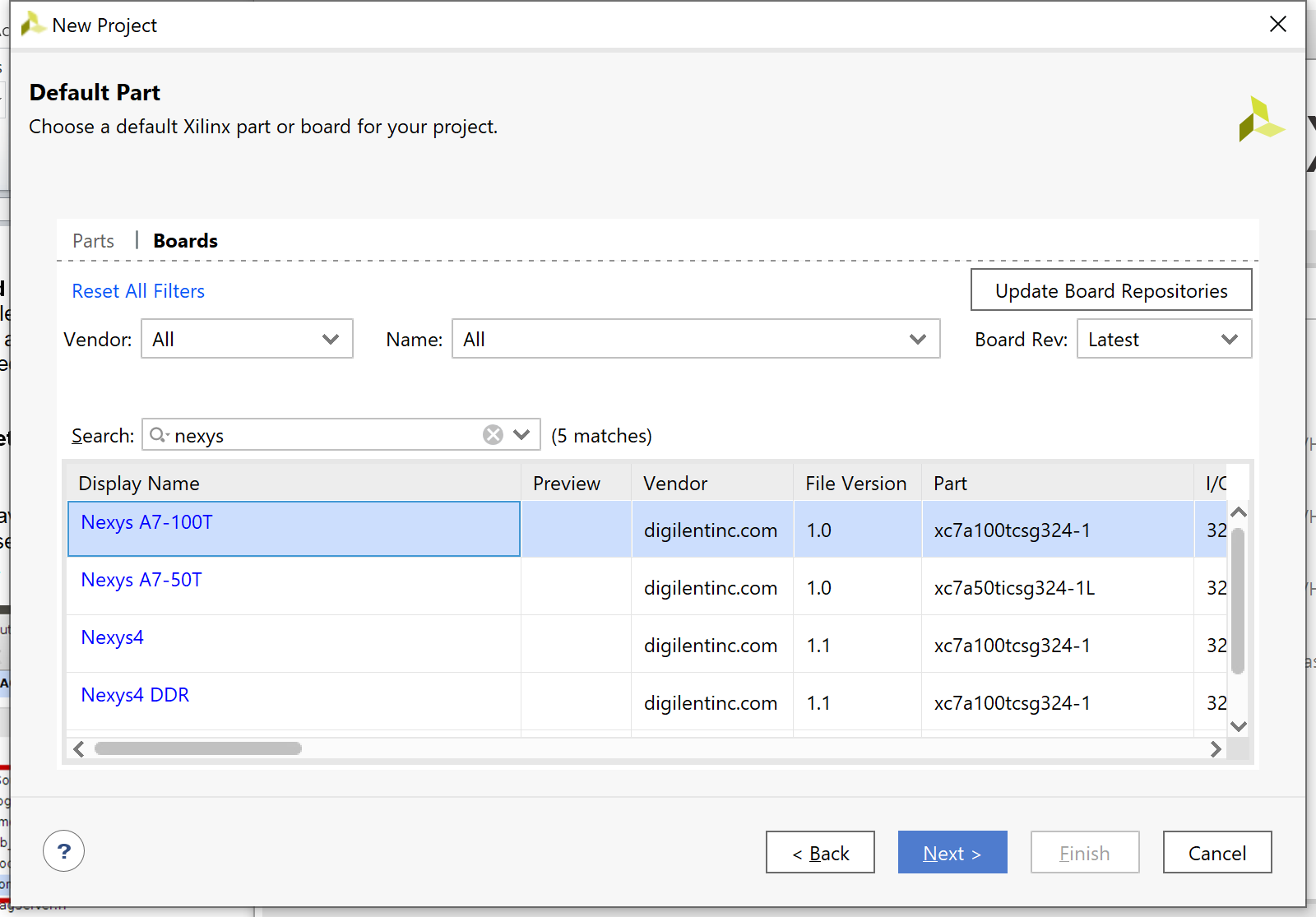
****

**Figure 15. Add Constraints**

**Step 4. Select Nexys A7 as the target board**

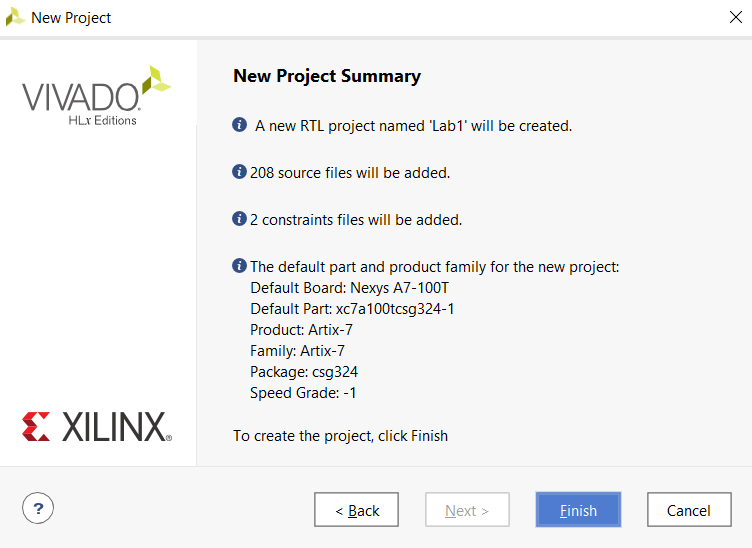
In the Default Part window, click on Boards and then select Nexys A7-100T (See Figure 16). You may use the Search box to narrow down the results. You will also notice that the name of the actual target FPGA is listed in the Part column: xc7a100tcsg324-1. This indicates that it is a Xilinx Artix-7 FPGA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

Click Next.



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

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

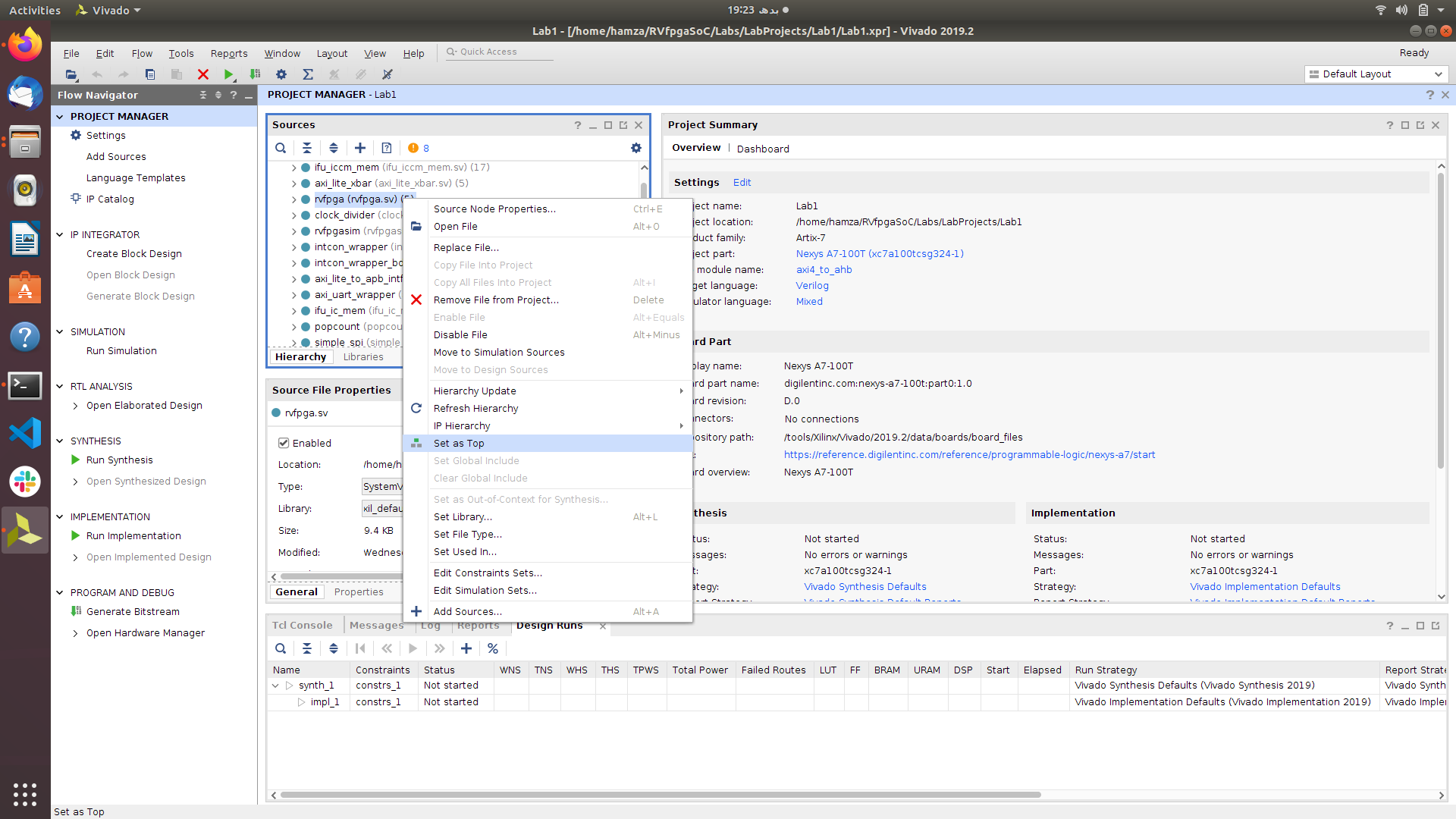


**Figure 17. New Project Summary Window**

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**

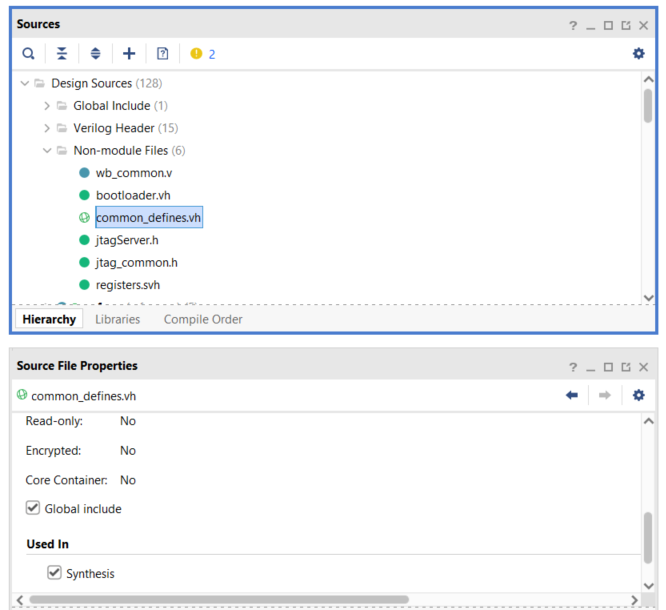
The project will initialize. You will now set the rvfpga module as the top module. In the Sources pane, scroll down under Design Sources, right-click on the rvfpga module, and select Set as Top (see Figure 18). You can also find the rvfpga module by typing this name in the search box, as shown. This 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**

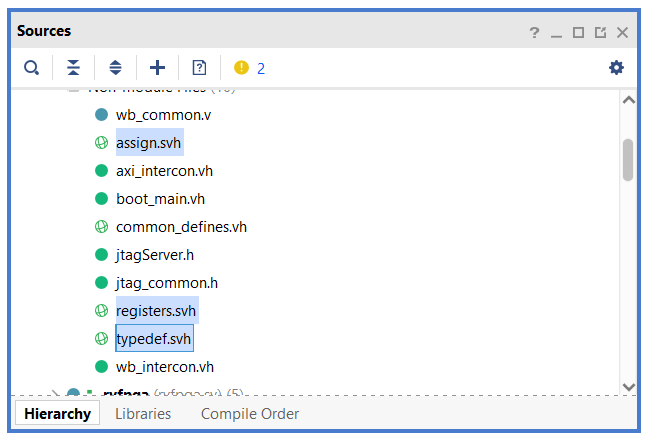
**Step 6. Set Verilog header files as global include Files**

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.



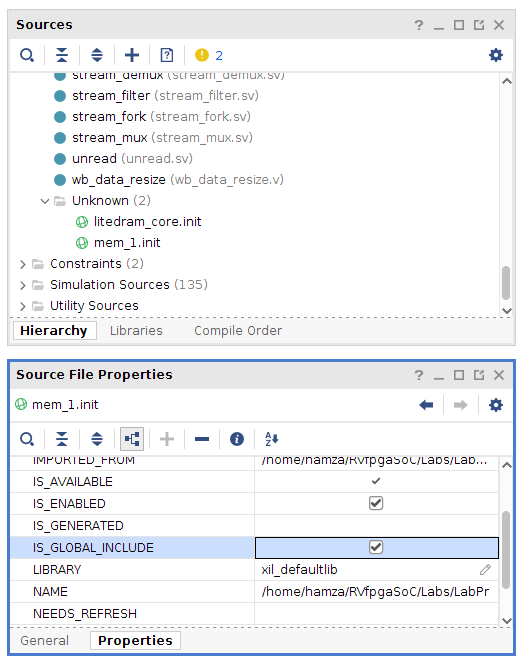
**Figure 19. Set common\_defines.vh as a Global include file**

Similarly, set the **“assign.svh”**, **“registers.svh”**, and **“typedef.svh”** SystemVerilogHeader filesas 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).

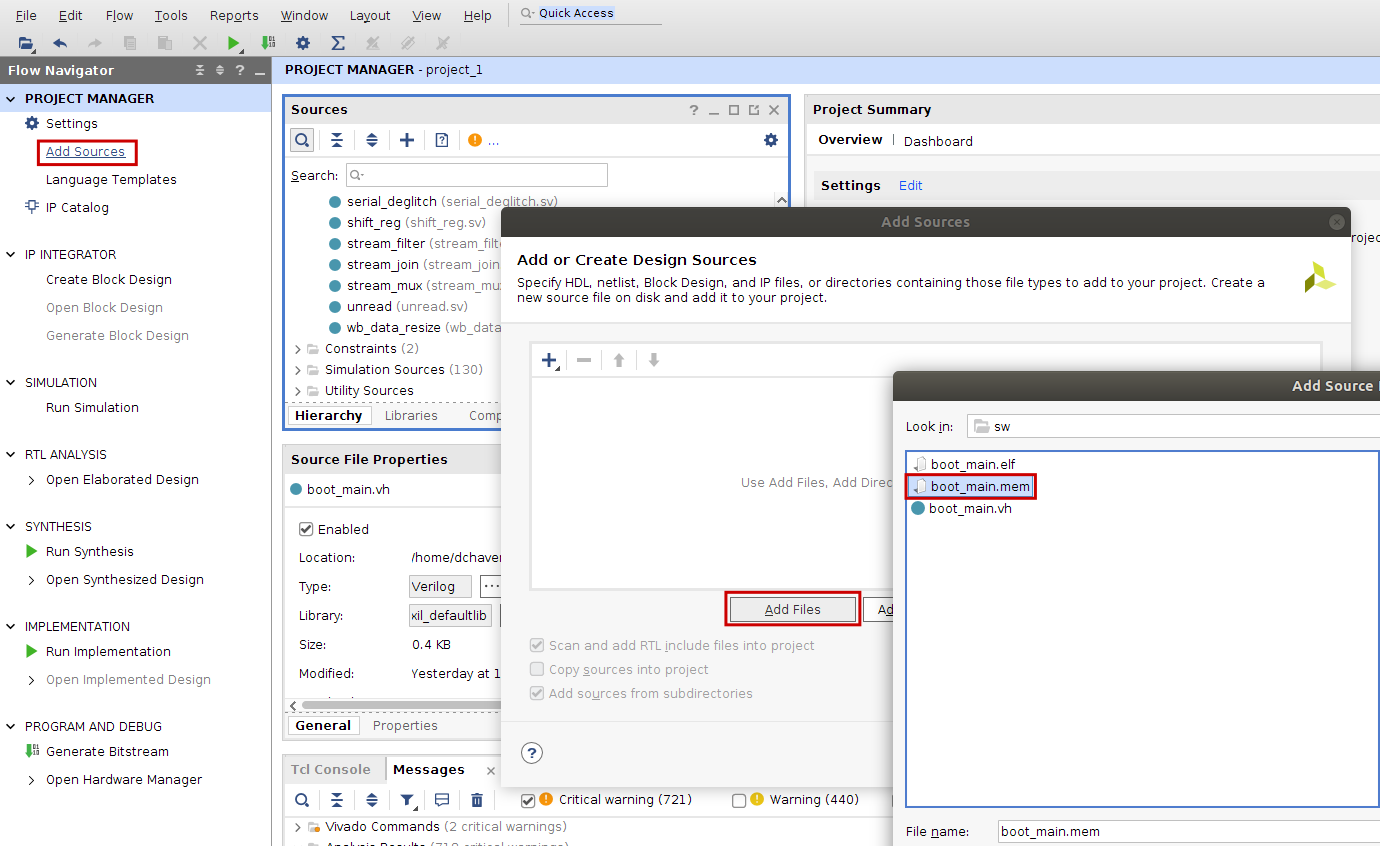


**Figure 21. Set litdram\_core.init as a Global include file**

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 6. 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 Design Sources/Memory File.



**Figure 22. Add Memory File boot\_main.mem**

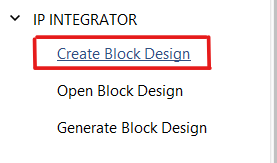
The design source files have now been added and now we can go ahead and start creating the block design.

# 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**

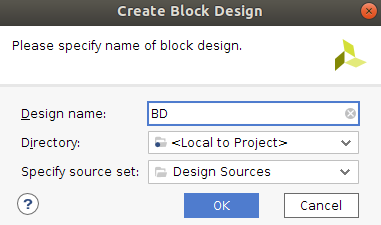
Create a new block design in the Flow Navigator by clicking on Create Block Design under the IP Integrator heading (See Figure 23).



**Figure 23. Create Block Design**

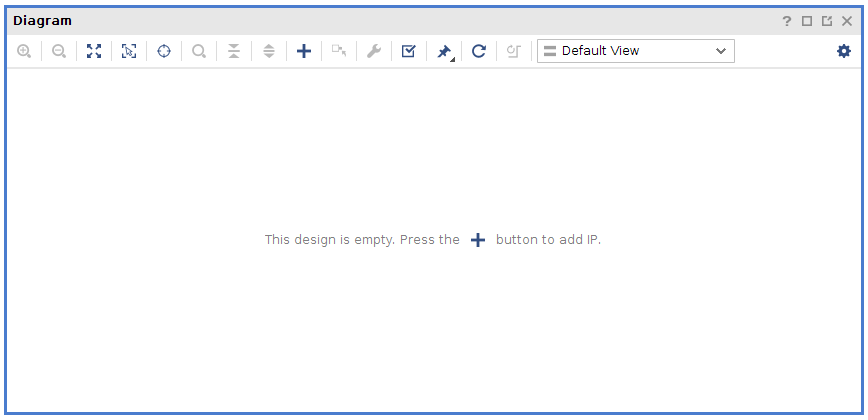
**Step 2. Select Block Design’s Name**

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**

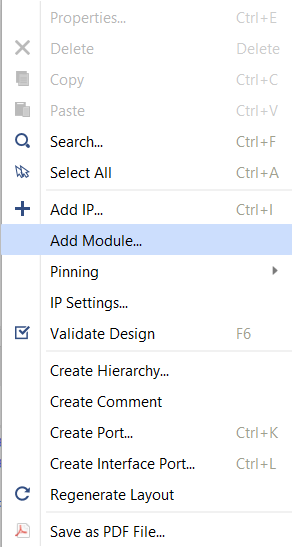
Now you will see a blank block design Diagram panel. (See Figure 25)



**Figure 25. Blank Block Design**

**Step 3. Add Modules to the Block Design**

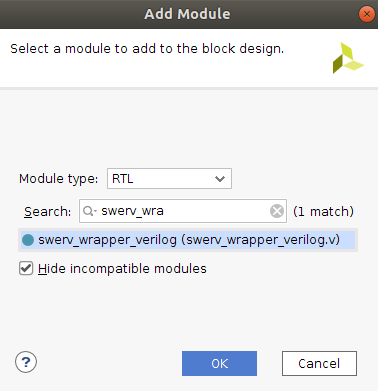
Now we can start adding modules to our Block Design. We can do that by right-clicking on the blank Block design and select the “**Add Module**” option (See Figure 26).



**Figure 26. Add Module**

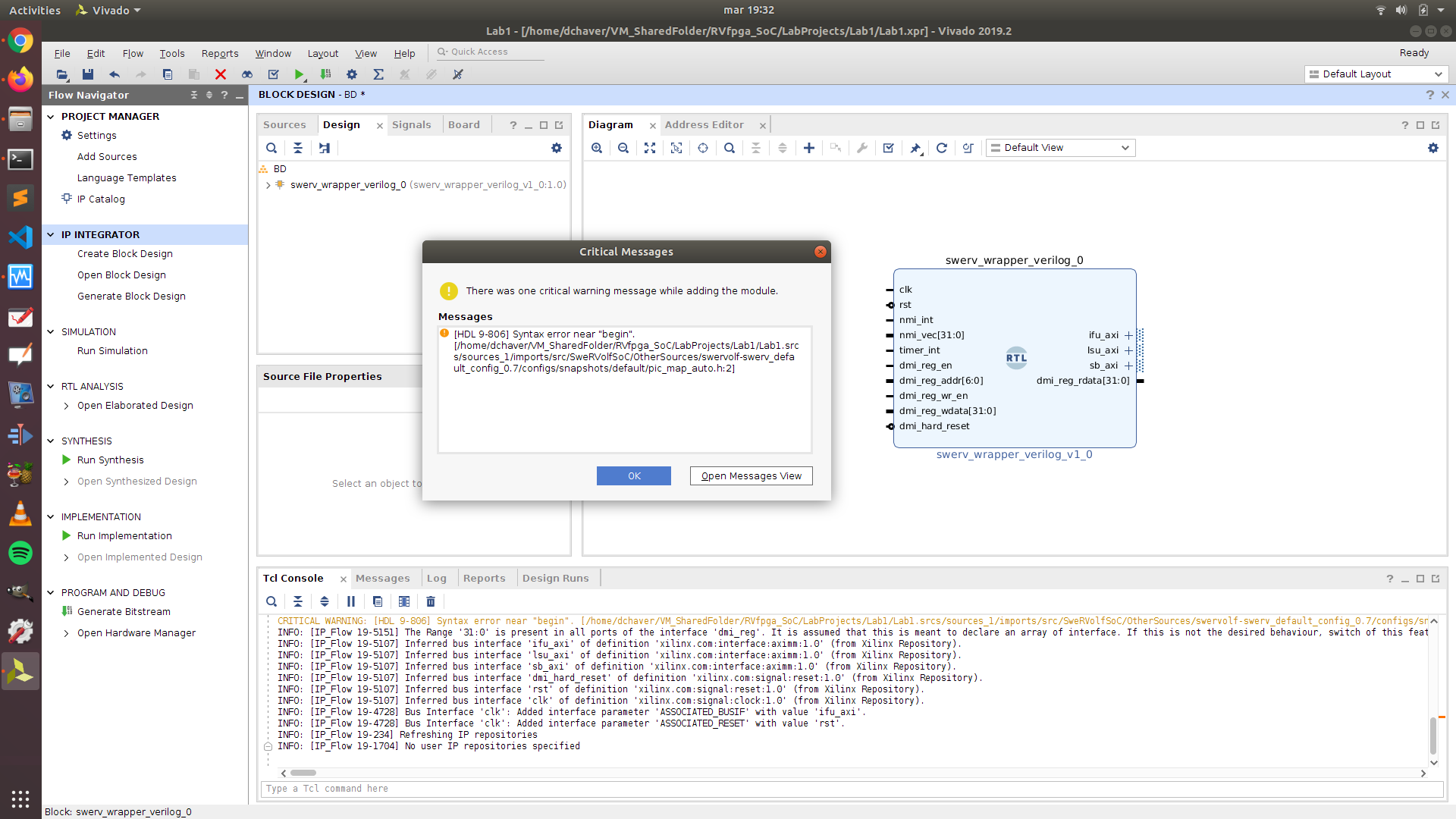
A dialogue box will appear; you can either scroll down or type in the search box the name of the required modules you would like to add. We will start by adding the SweRV EH1 Core Complex.

Select **“swerv\_wrapper\_verilog”** and click OK.



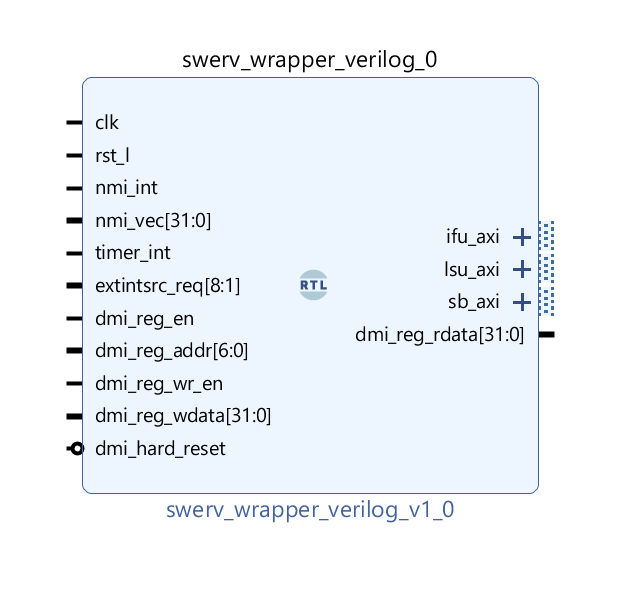
**Figure 27. Add swerv\_wrapper\_verilog**

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)



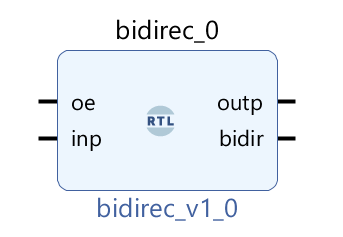
* **“syscon\_wrapper”** (System Controller Module)



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)

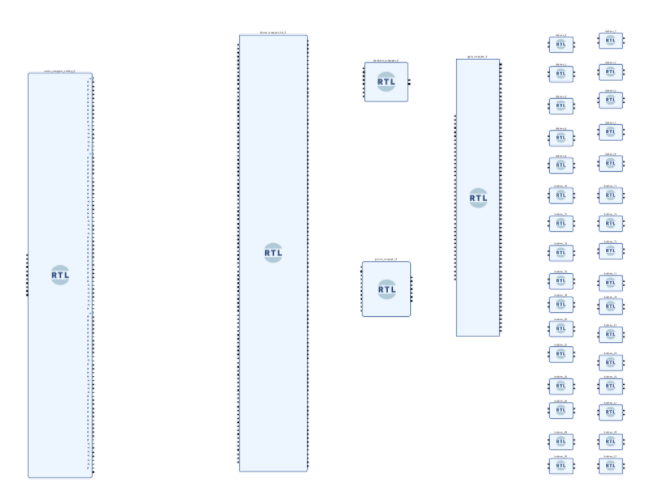




**Figure 30. bidir GPIO module**

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 32 for the following instructions. 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\_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)

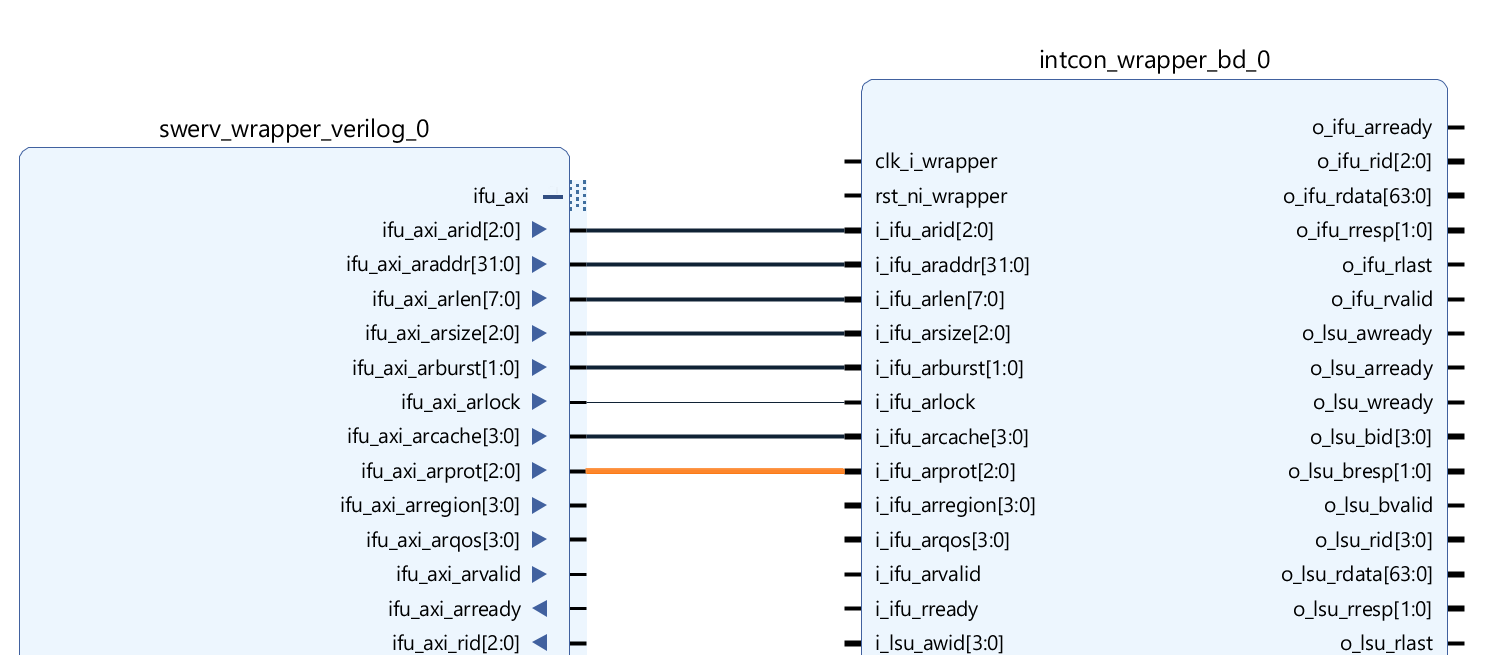
We will first start with connecting the pins related to the IFU. Connect pin “ *ifu\_axi\_arid[2:0]* ” of “**swerv\_wrapper\_verilog**” module to the “*i\_ifu\_arid[2:0]* ” pin of the “**intcon\_wrapper\_bd**”.

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.



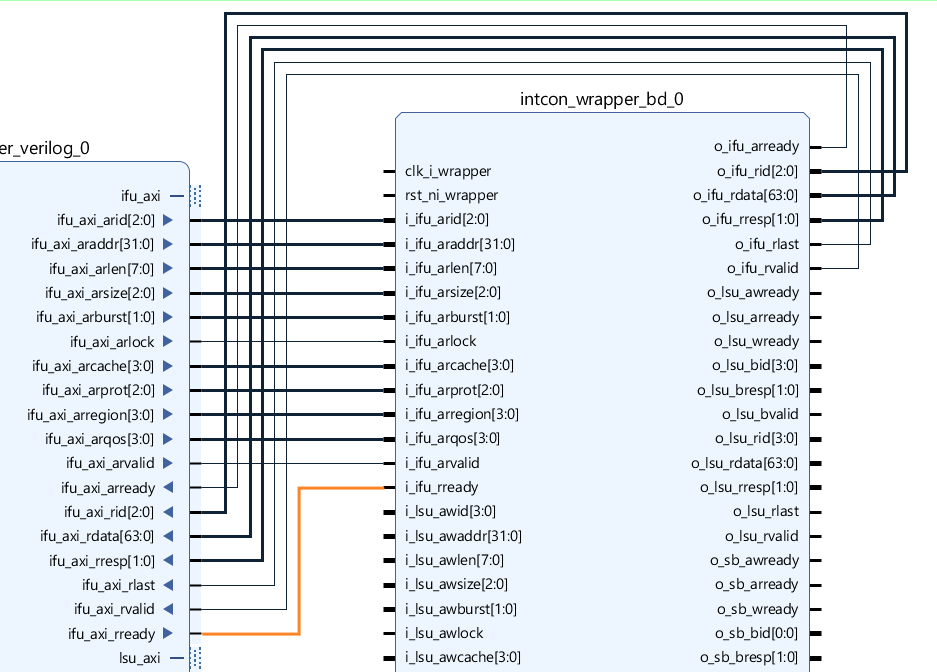
**Figure 32. Connect the relevant pin**

Similarly, we will connect all the **IFU** **(Instruction Fetch Unit)** pins of “**swerv\_wrapper\_verilog**” with the **IFU’s** pins of “**intcon\_wrapper\_bd**”.

**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



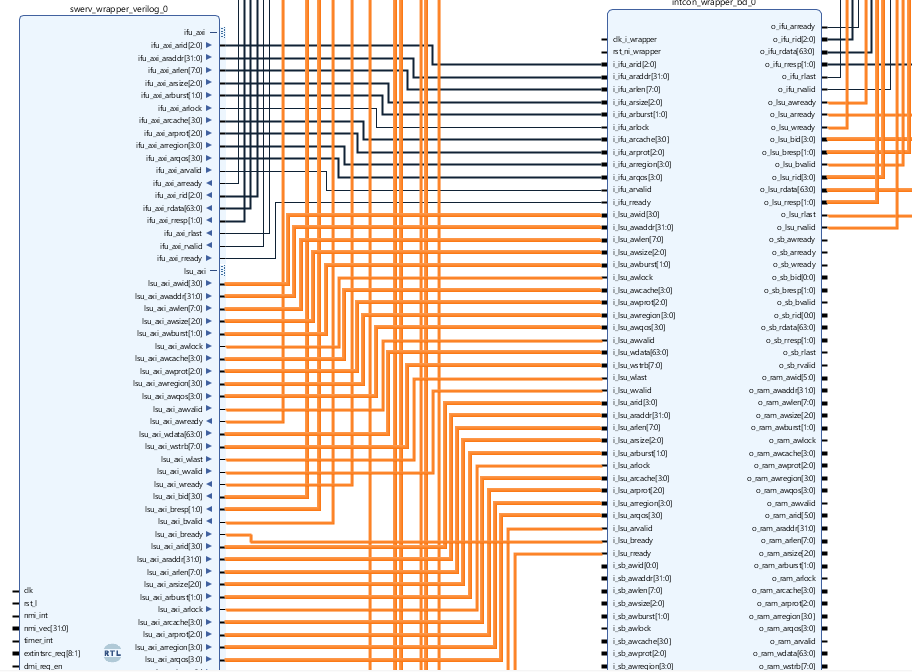
**Figure 33. Connect all the IFU pins**

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.

**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



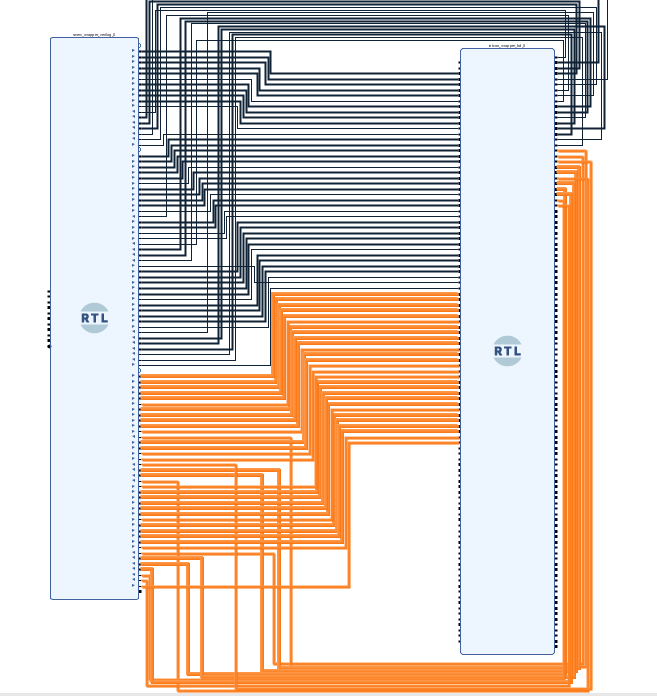
**Figure 34. Connect all the LSU pins**

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**”.

**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/3\_SwervW\_IntconW\_SB.pdf



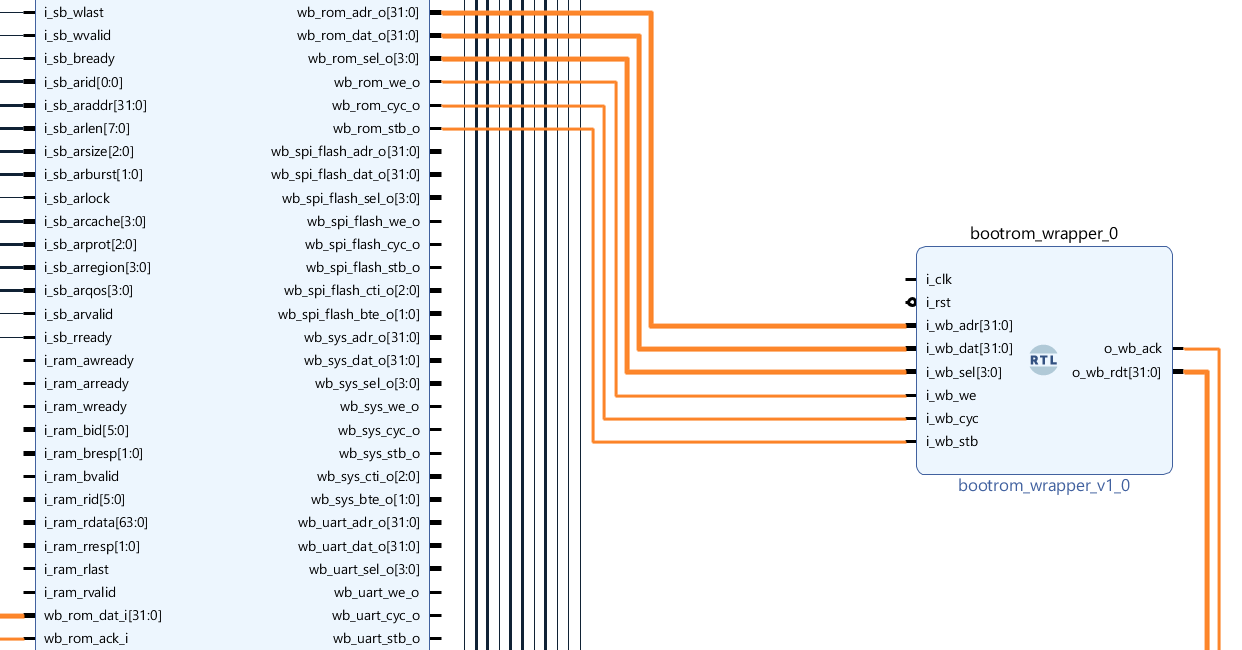
**Figure 35. Connect all the SB pins**

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**”.

**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



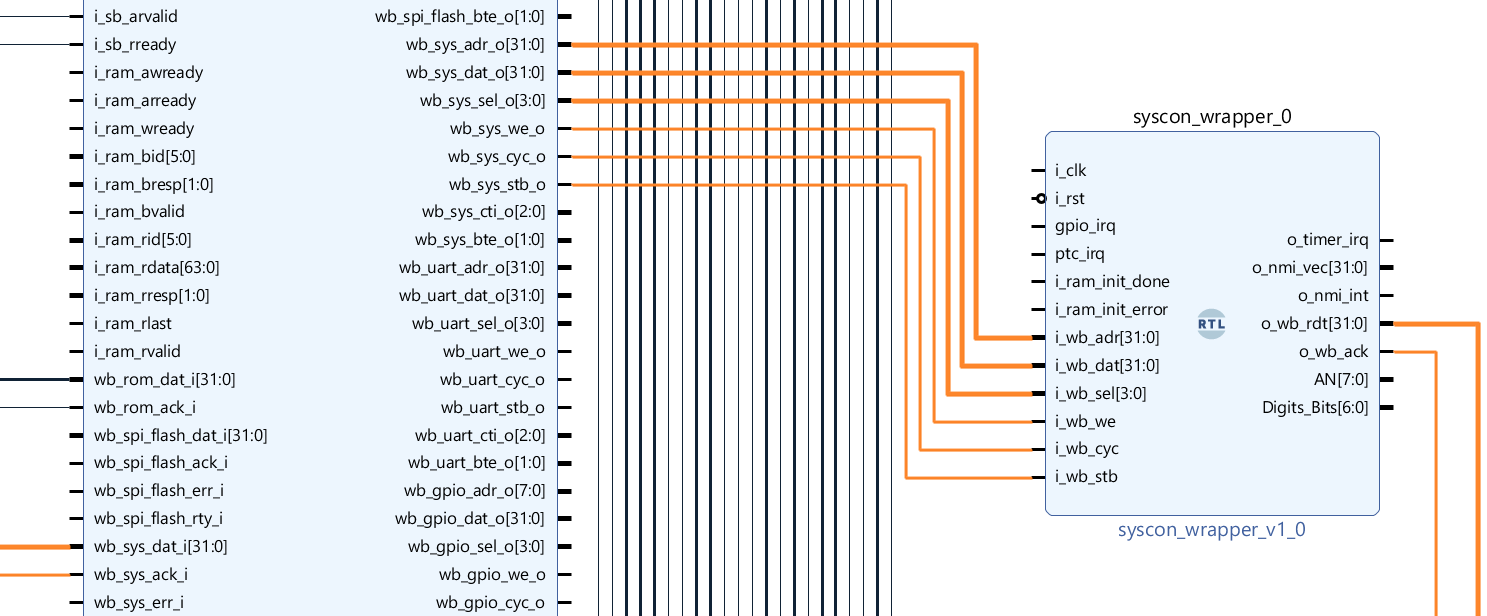
**Figure 36. Connect the BootROM module with the Interconnect Wrapper module**

Now we will connect the “**syscon\_wrapper**” module with the “**Intcon \_wrapper\_bd**” module.

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

[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

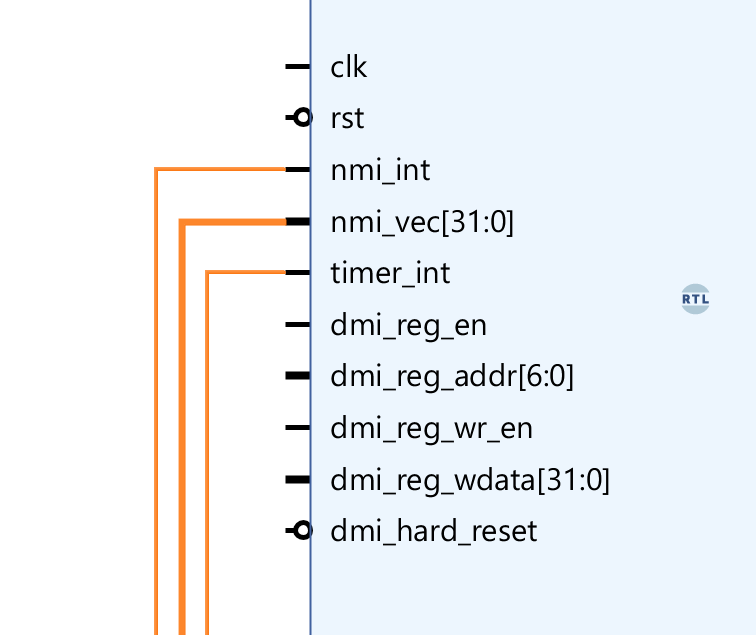
/InternalConnections/5\_SysconW\_IntconW.pdf

****

**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**”.

* o\_timer\_irq
* o\_nmi\_vec[31:0]
* o\_nmi\_int

****

**Figure 38. Connect the syscon\_wrapper with the swerv\_wrapper\_verilog pins**

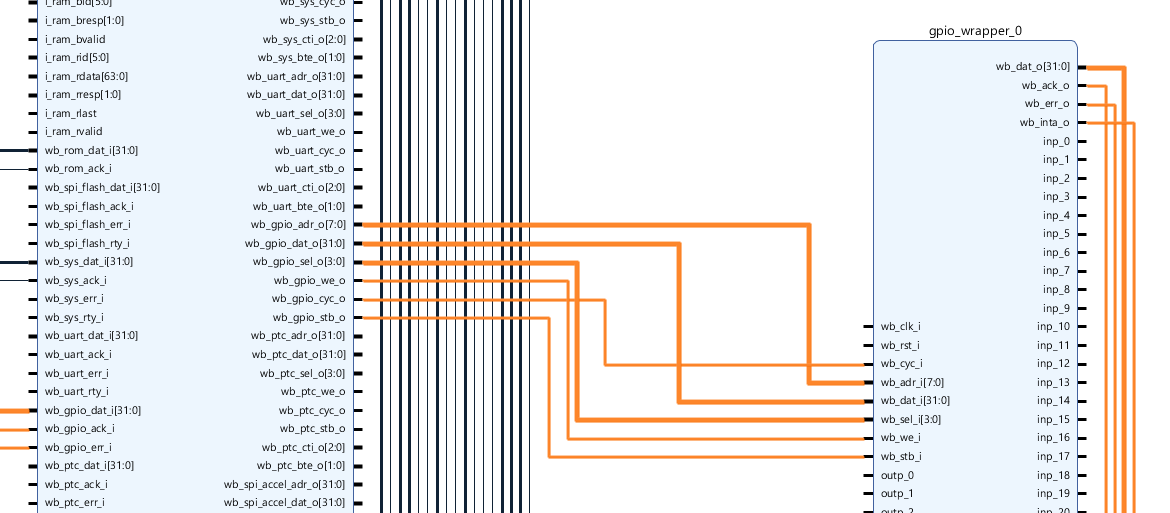
Now we will connect the “**gpio\_wrapper**” module with the “**intcon\_wrapper\_bd**”. Connect the “wb\_gpio\_xxx\_x” pins of the “**intcon\_wrapper\_bd**” module with the “**gpio\_wrapper**” module pins.

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,

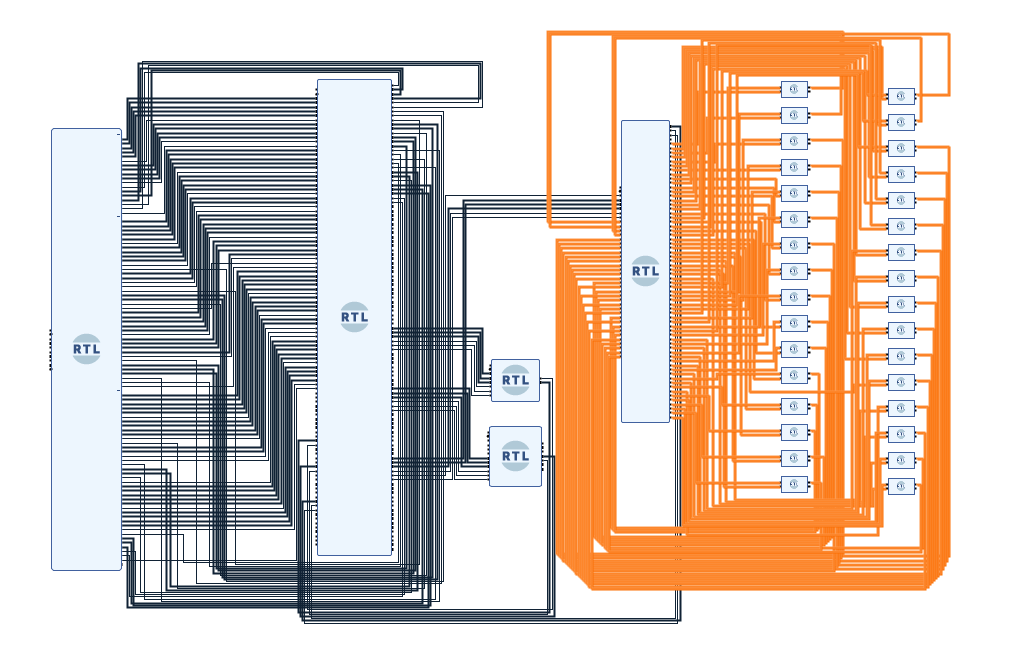
*“****outp\_0****”* pin of “**gpio\_wrapper\_0”** will be connected to **“*outp”***of “**bidirec\_0**”,

*“****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

****

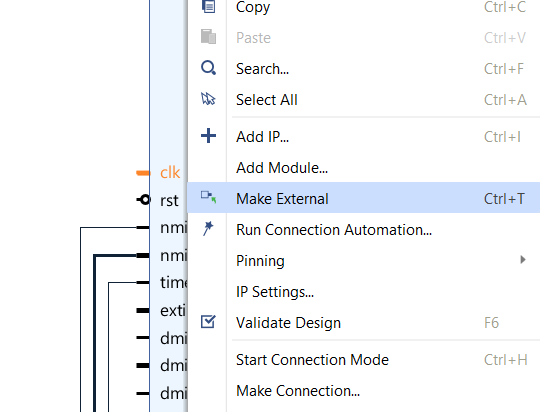
**Figure 40. All GPIO Bidirec Modules connected to gpio\_wrapper module**

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/OPins**

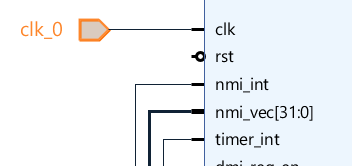
Now it is time to connect the pins coming into our block design as an Input or going out of our block design as an Output. We will connect these pins as the external Pins/Ports. These external pins include the pins of **RAM** (DDR), **CLK** (Clock), **RST** (Reset), and **DMI** (Debug Module interface).

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).

****

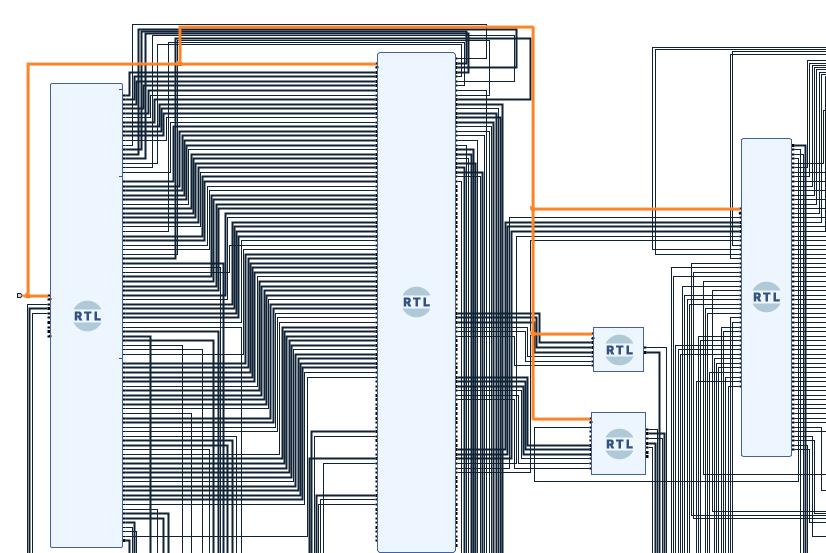
**Figure 42. “clk” becomes an external connection**

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

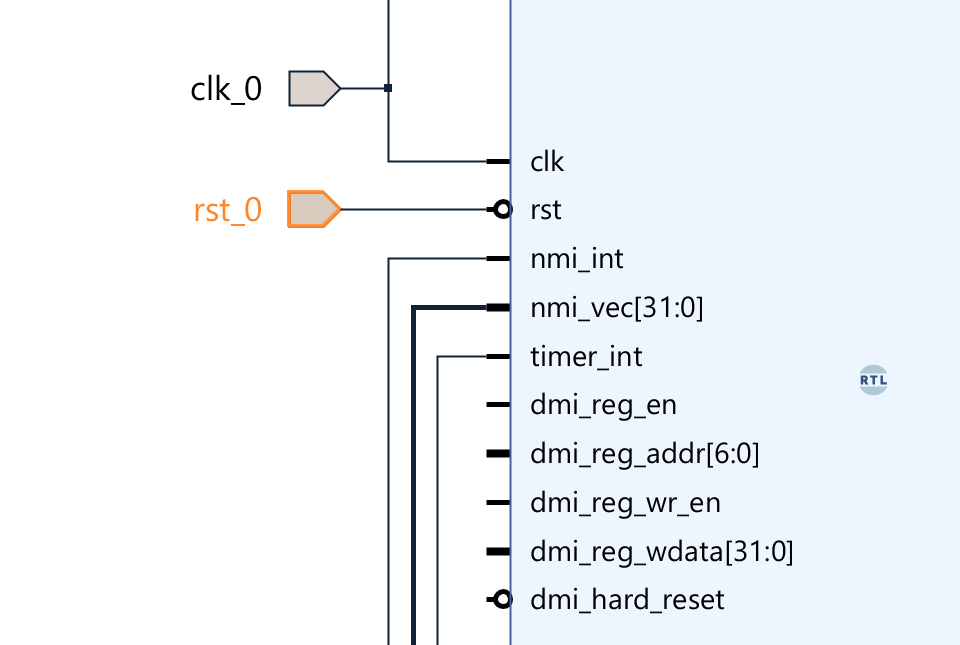
/ExternalConnections/1\_Clock.pdf



**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 for “**rst**”. Now again, go to the “**swerv\_wrapper\_verilog**” module and right-click on the “**rst**” pin, and make it external.

****

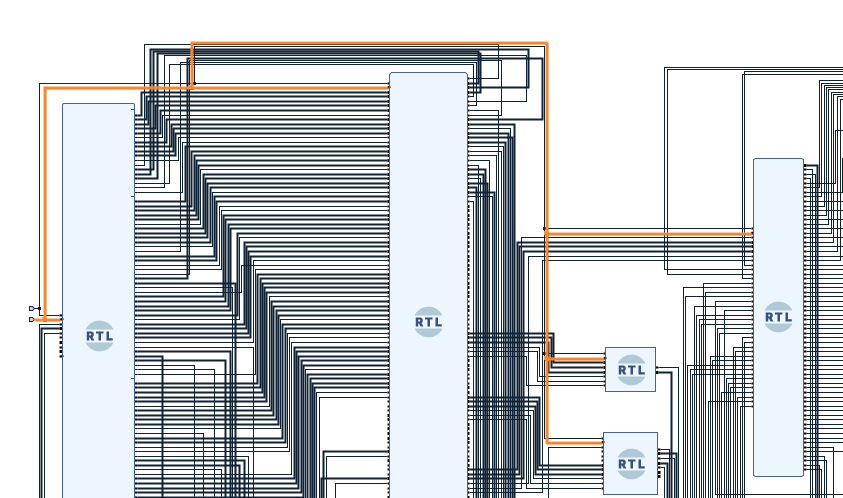
**Figure 44. Make rst\_l as an external pin**

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:

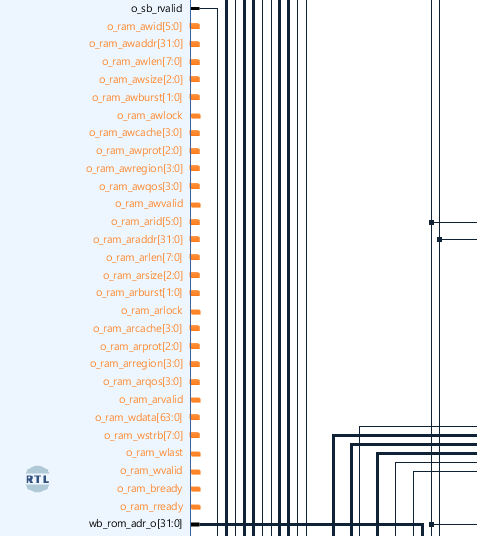
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/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.



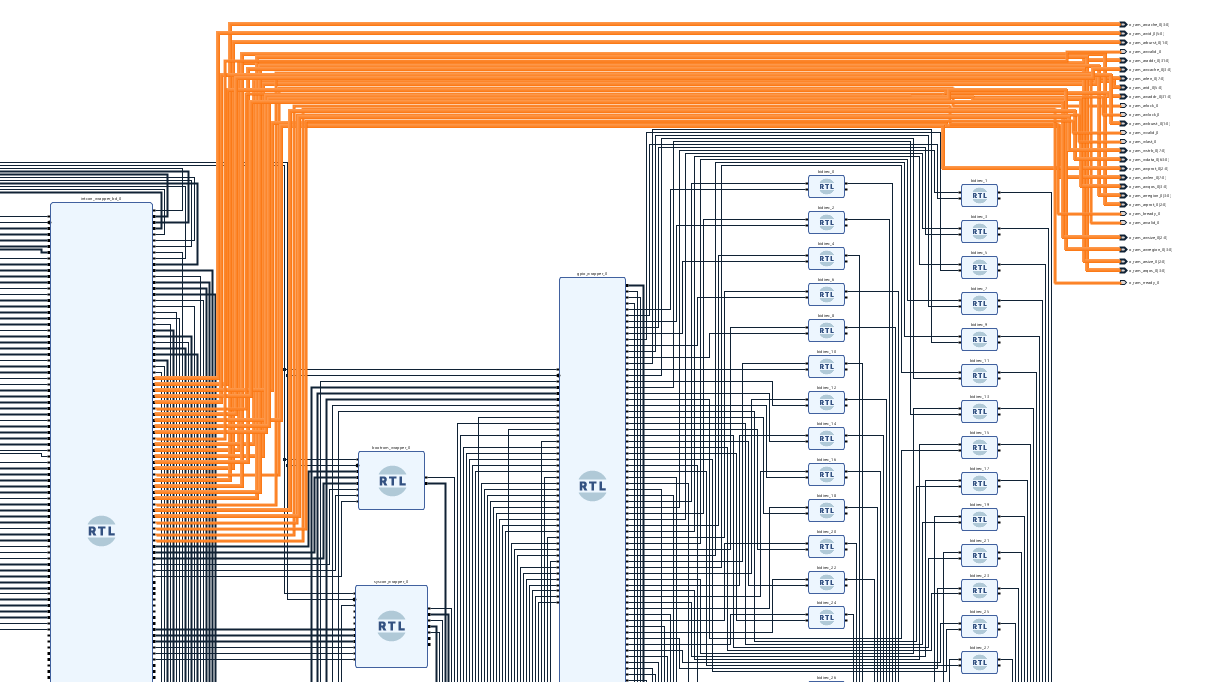
**Figure 46. Intcon wrapper right-hand side RAM pins**

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:

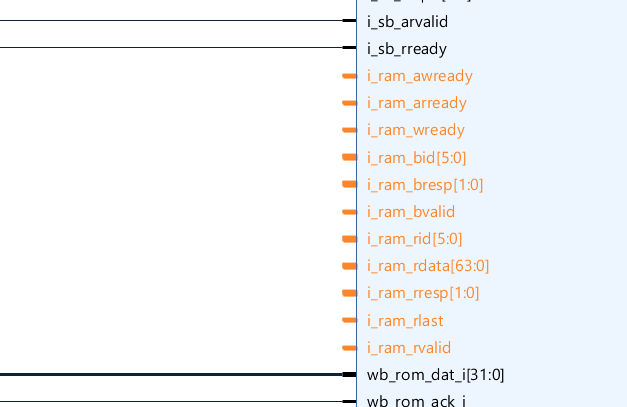
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/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.



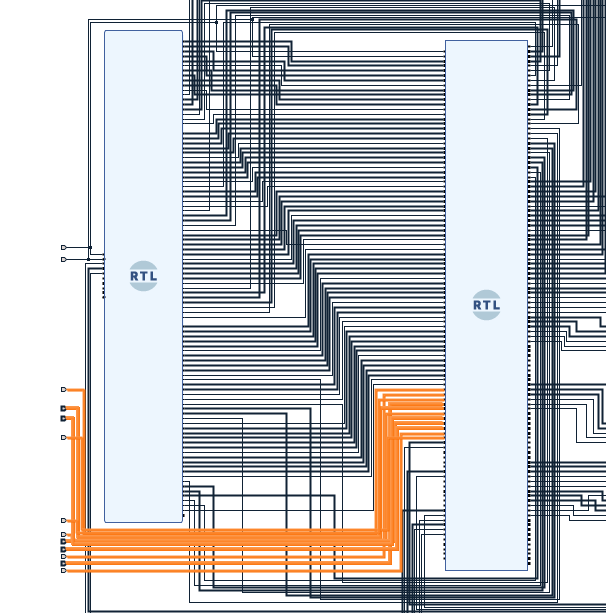
**Figure 48. Left Side RAM Pins of Interconnect Wrapper**

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:

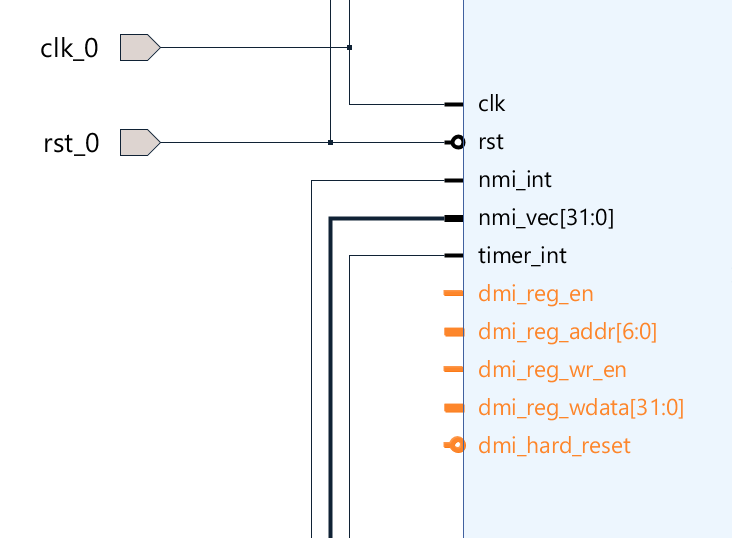
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/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.

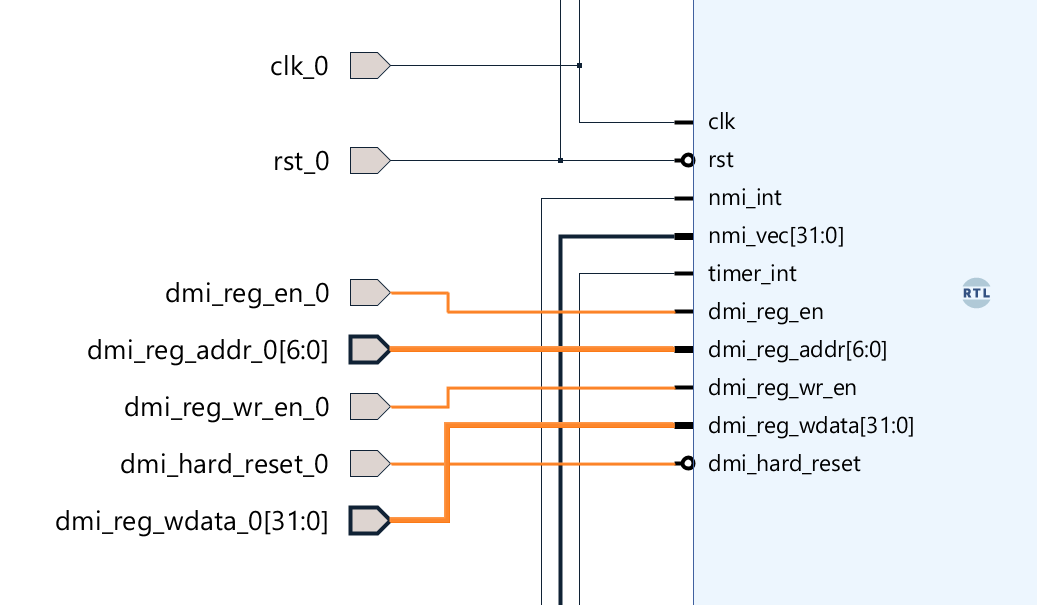
****

**Figure 50. dmi pins op swerv\_wrapper\_verilog (Left Side)**

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

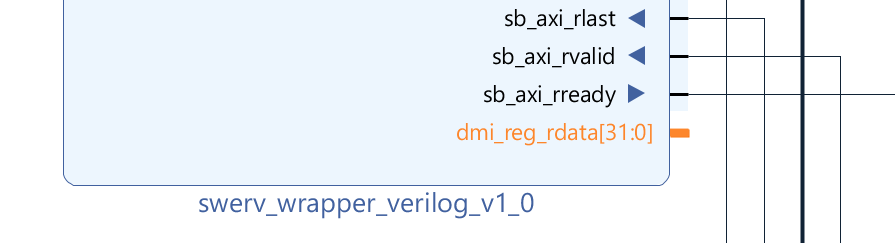
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs

/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**

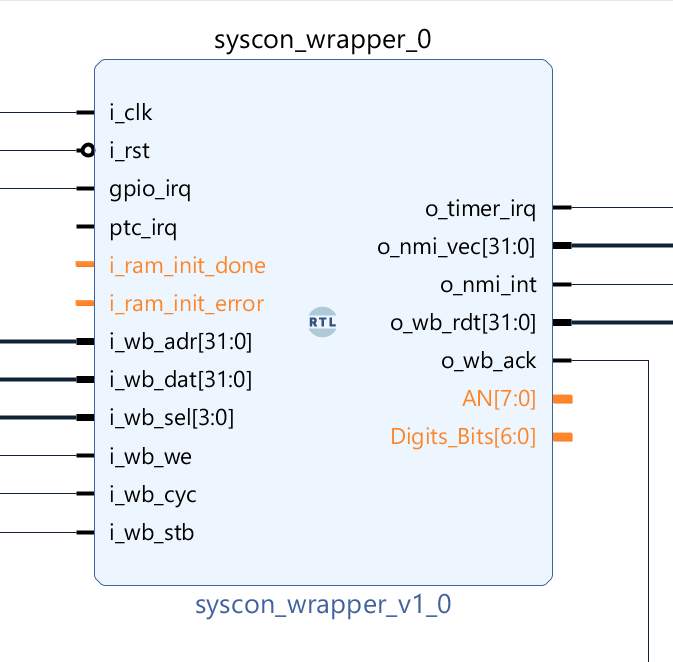
We will now make the following pins of the “**syscon\_wrapper**” module as external pins.

* 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

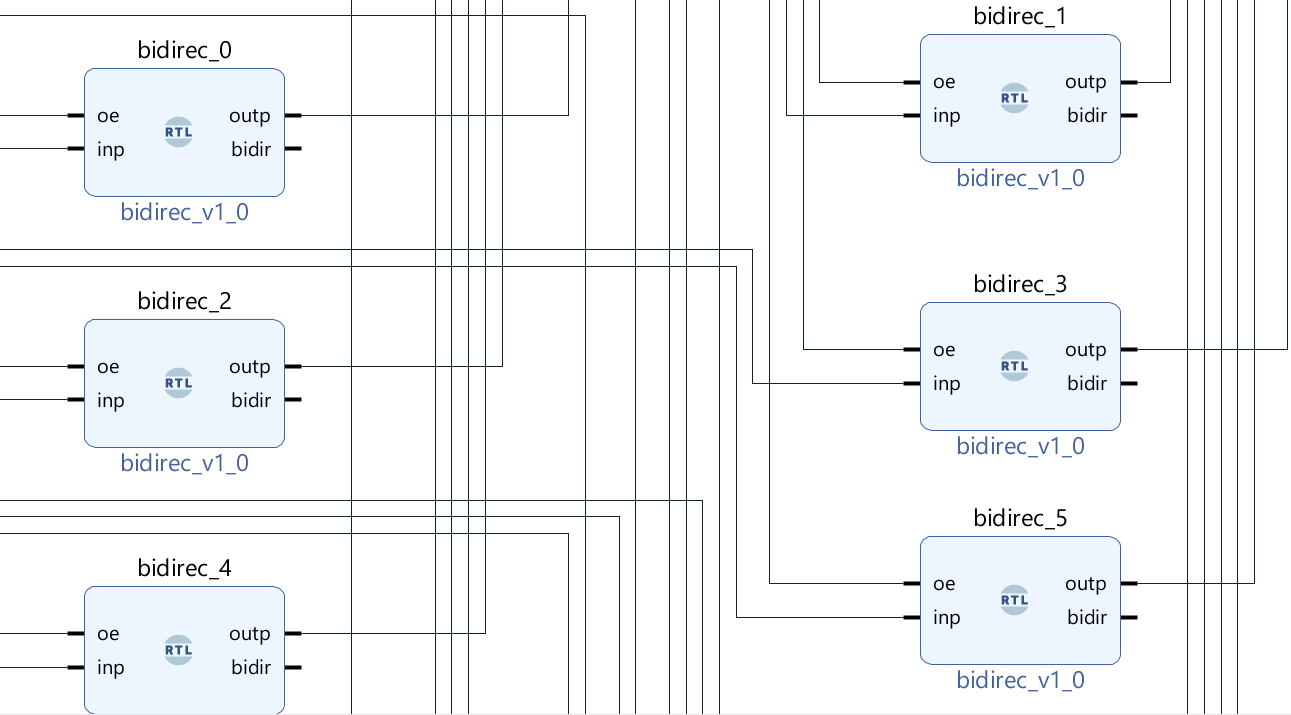
/ExternalConnections/6\_SysconW\_External.pdf



**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. |

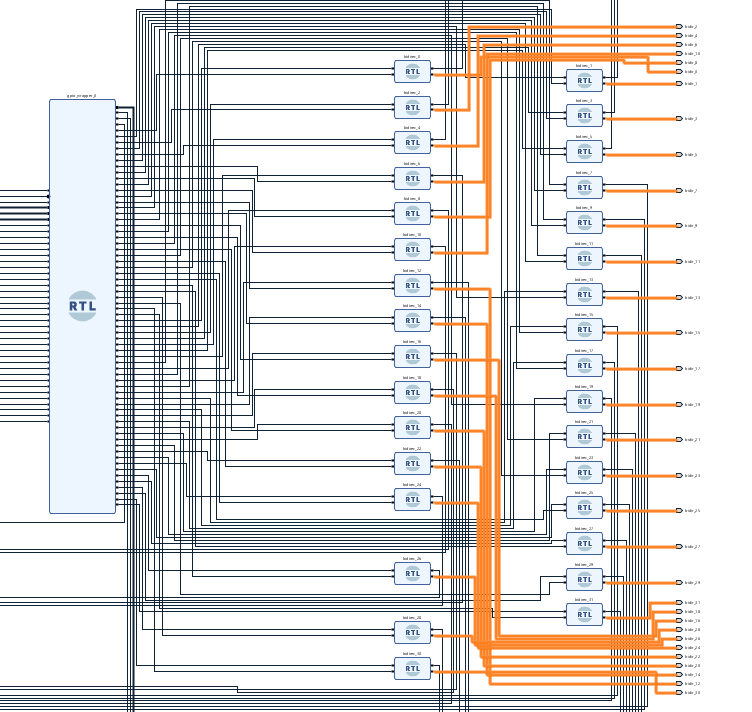


**Figure 55. Make “bidir” Pin of our GPIO Bidirec Modules as External Pins**

**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

****

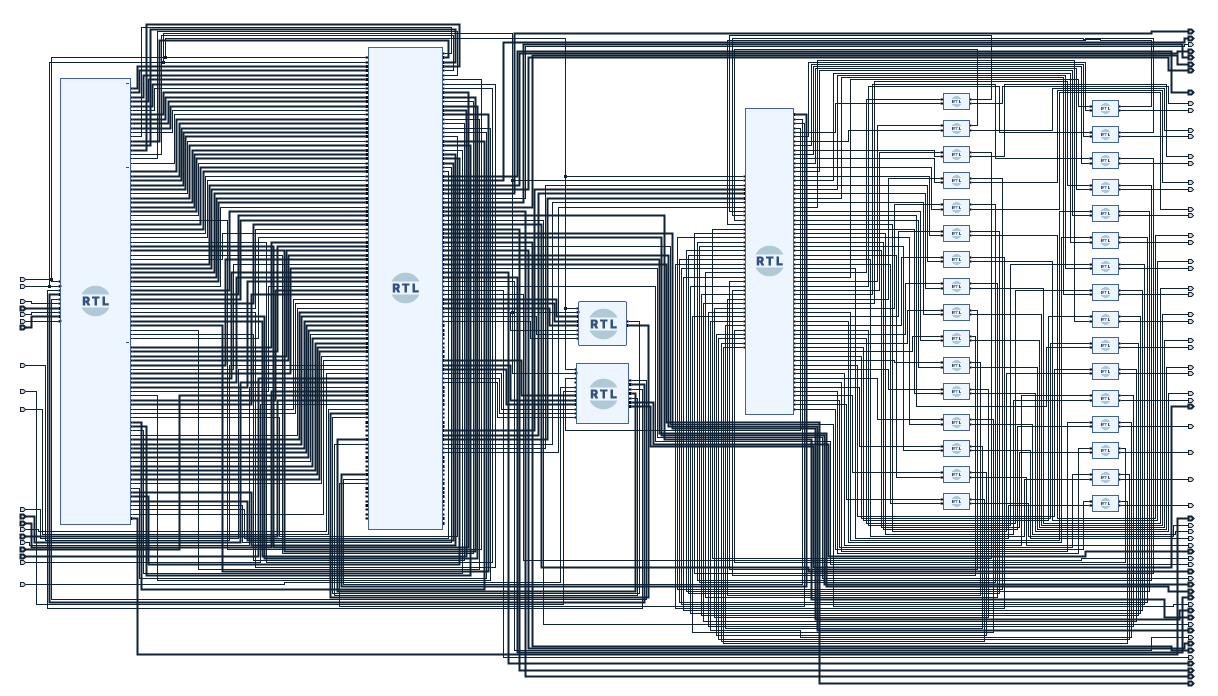
**Figure 56. Make “bidir” External connections**

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 SweRVolfX 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)

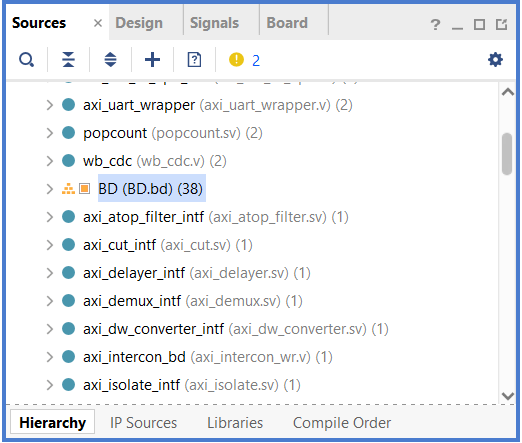
See Figure 57.

**Figure 57. Block design SoC completed**

# Generating The Block Design Module Verilog File

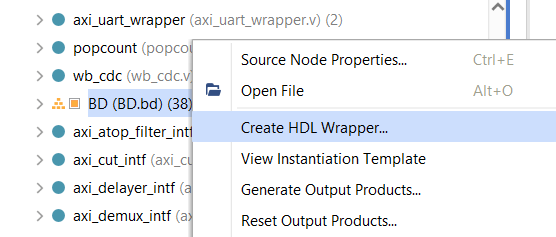
We will now generate the Verilog module file of the block design that we have created.

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

****

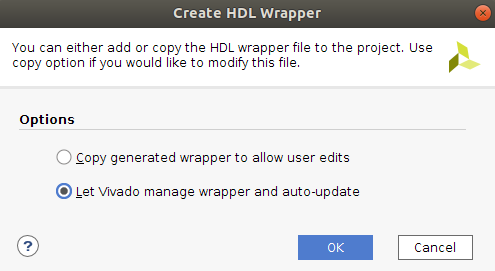
**Figure 58. Find “BD” in Sources**

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

****

**Figure 59. Create HDL Wrapper**

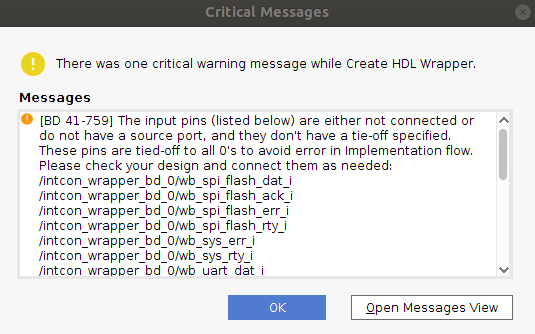
Select the “**Let Vivado manage wrapper and auto-update**” option and click OK to proceed.

****

**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).

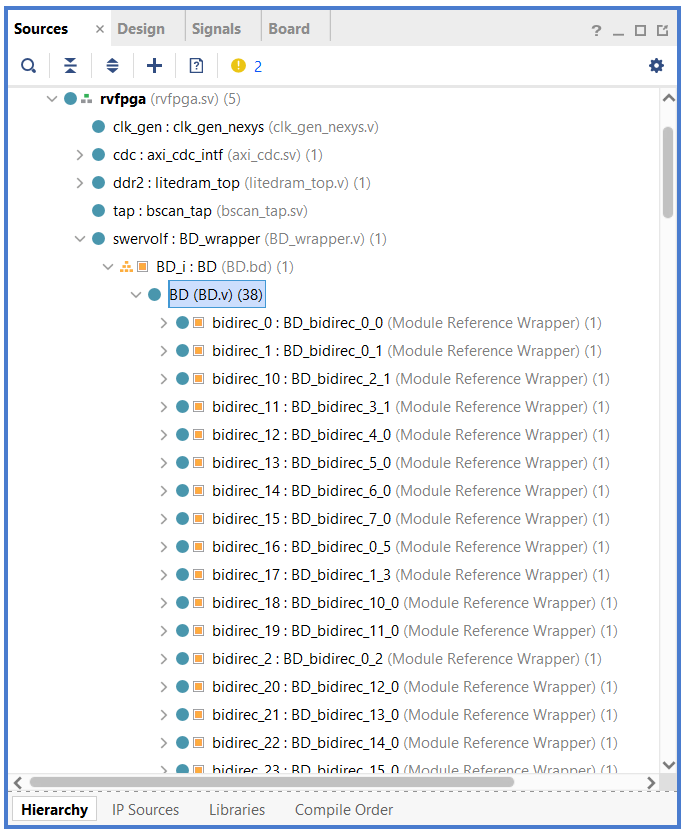
Click OK.



**Figure 61. Warning Pop-Up**

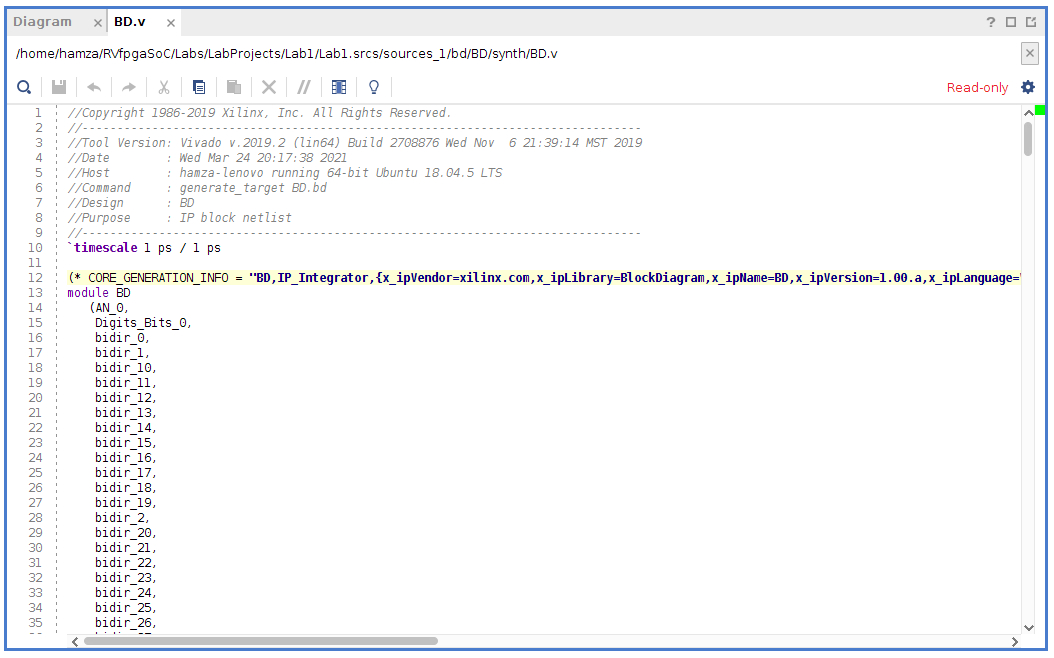
Now the Block design’s HDL Wrapper has been created. You can navigate to the Sources panel and scroll down until you see “**BD\_wrapper**”. Click on the dropdown icon next to it and then again for “**BD\_i**”.

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

****

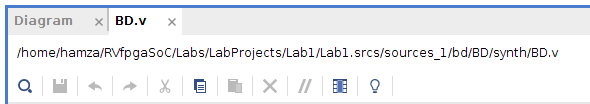
**Figure 62. Find “BD.v” in the sources panel**

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

****

**Figure 63. “BD.v” The Verilog file of Swervolf Core**

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.

****

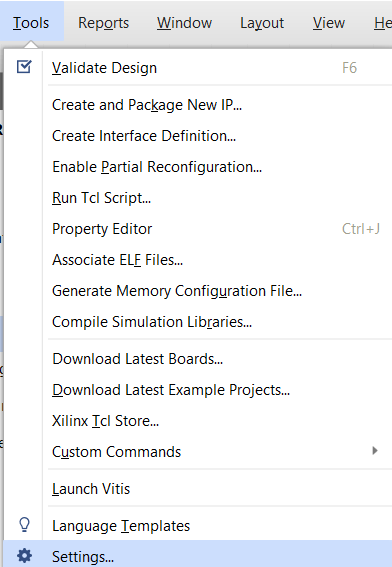
**Figure 64. Path of the “BD.v” file**

# 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.**

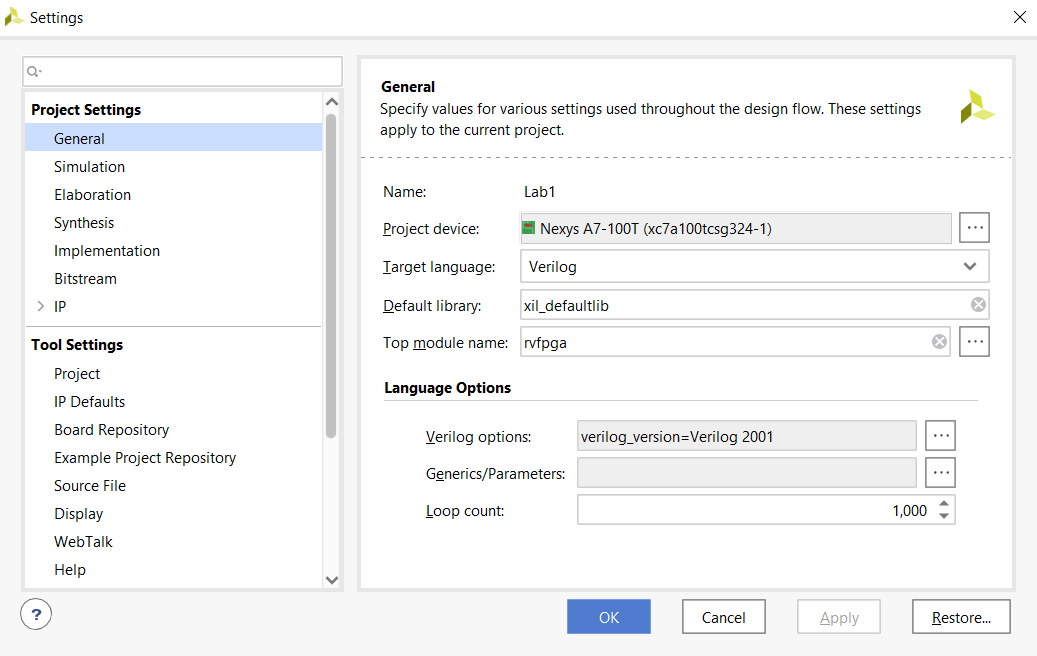
Go to “**Tools**” in the upper left side of the Navigation Bar of Vivado, then select “**Settings**” from the options.



**Figure 65. Go to Settings**

**Step 2**. **Navigate to the General tab**

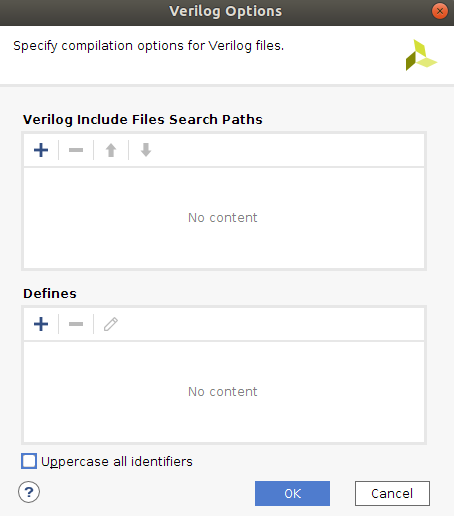
Go to the “**General**” tab, then select “**Verilog options**” from the language options section.

**** 

**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 four paths :

* [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs /sources\_1/imports/src/SweRVolfSoC/Interconnect

/AxiInterconnect/pulp-platform.org\_\_axi\_0.25.0/include

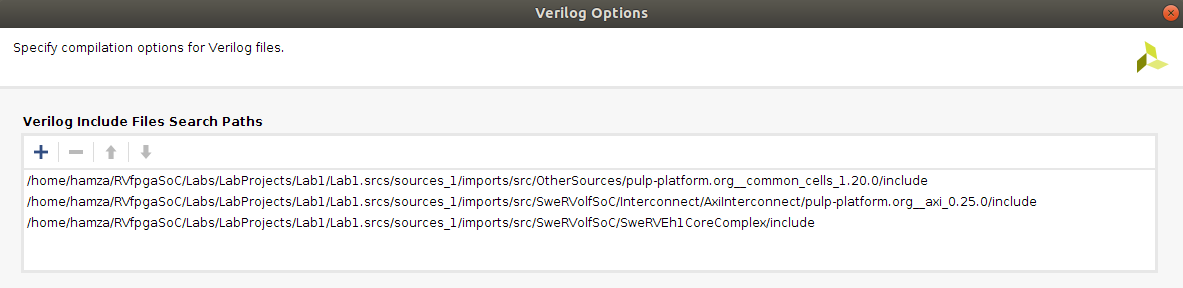
* [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs

/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

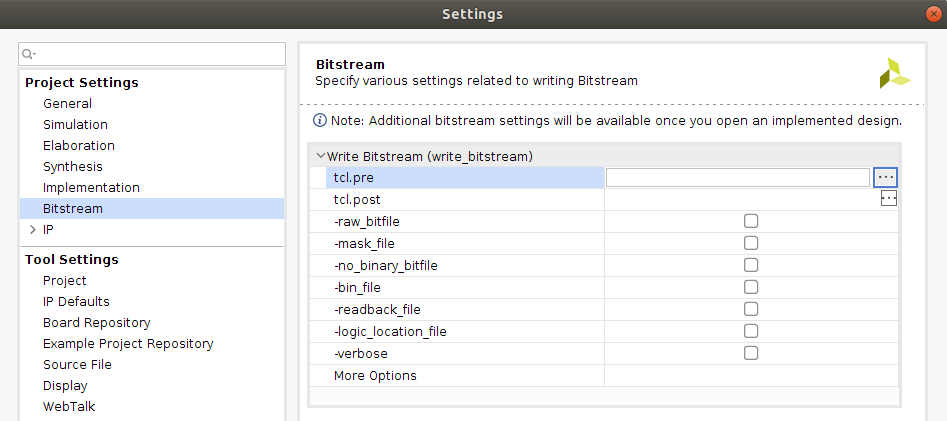
****

**Figure 68. Verilog Include Files Paths**

Click OK.

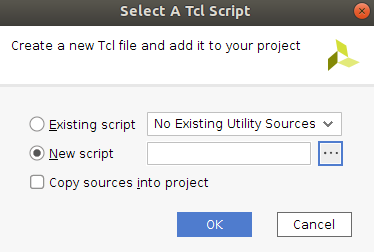
**Step 4**. **Navigate to the Bitstream tab**

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



**Figure 69. Bitstream setting**

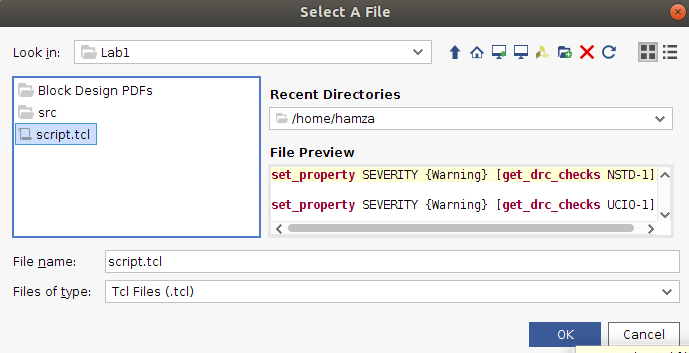
Select the “New script” option.



**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

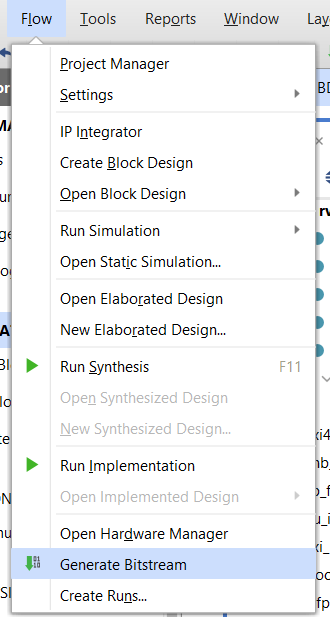


**Figure 72. import “script.tcl” file**

Click OK and apply the changes.

**Step 4**. **Generate Bitstream.**

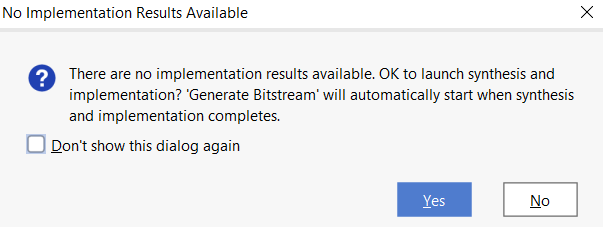
Now Click on Flow → Generate Bitstream, as shown in Figure 73.



**Figure 73. Generate Bitstream**

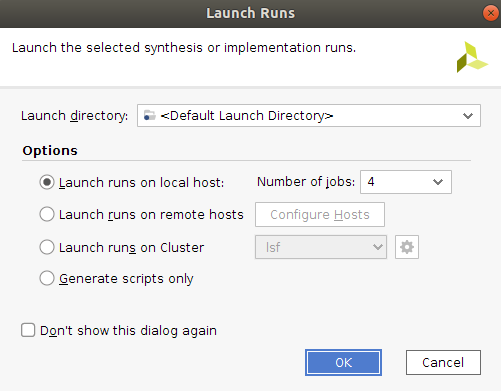
A window might pop up that says there are no implementation results available and ask to launch synthesis and implementation.

Click Yes (see Figure 74).



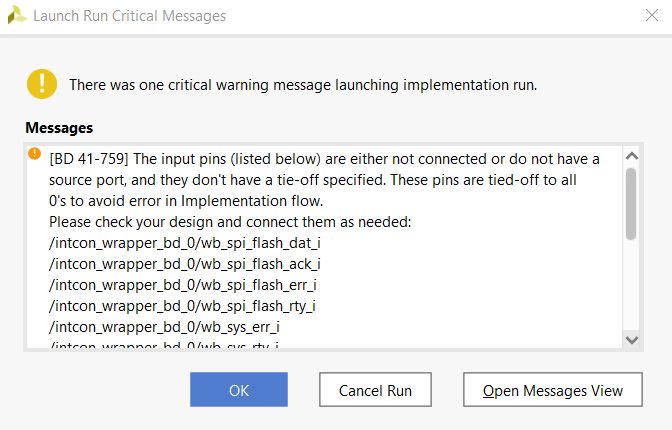
**Figure 74. Launch synthesis and implementation window**

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



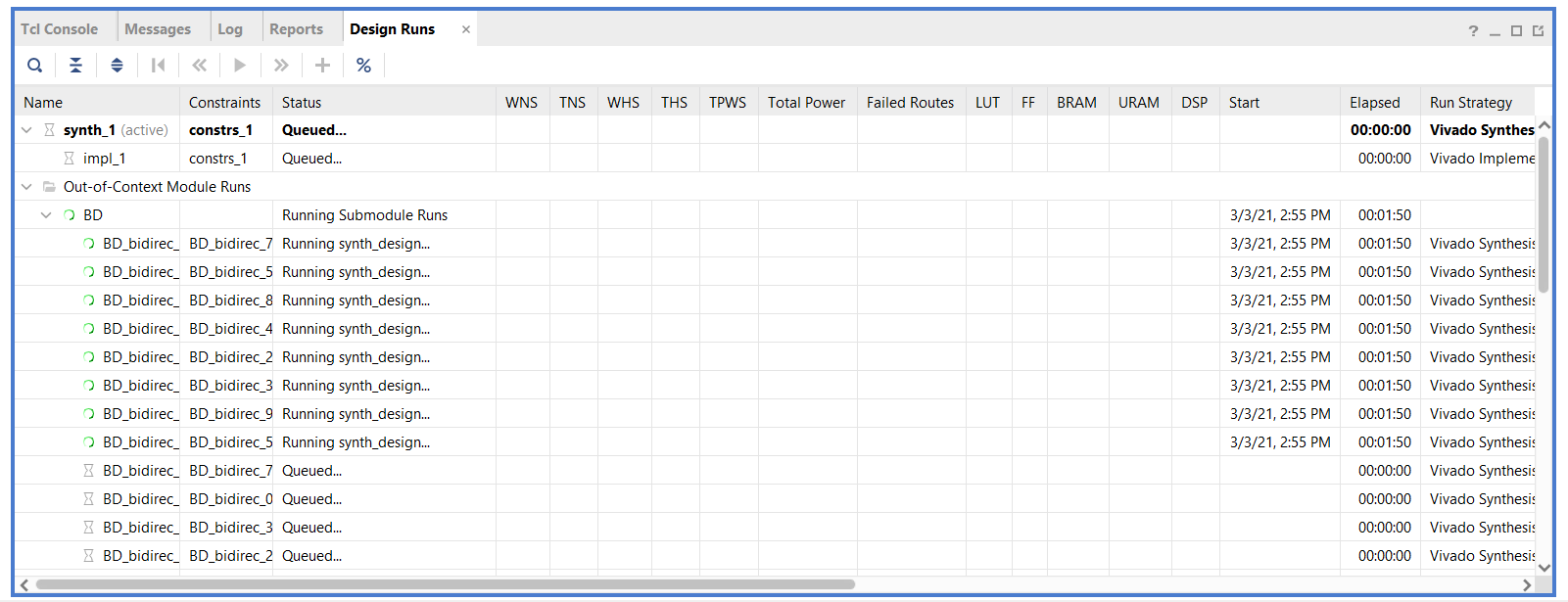
**Figure 75. Launch Runs**

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**

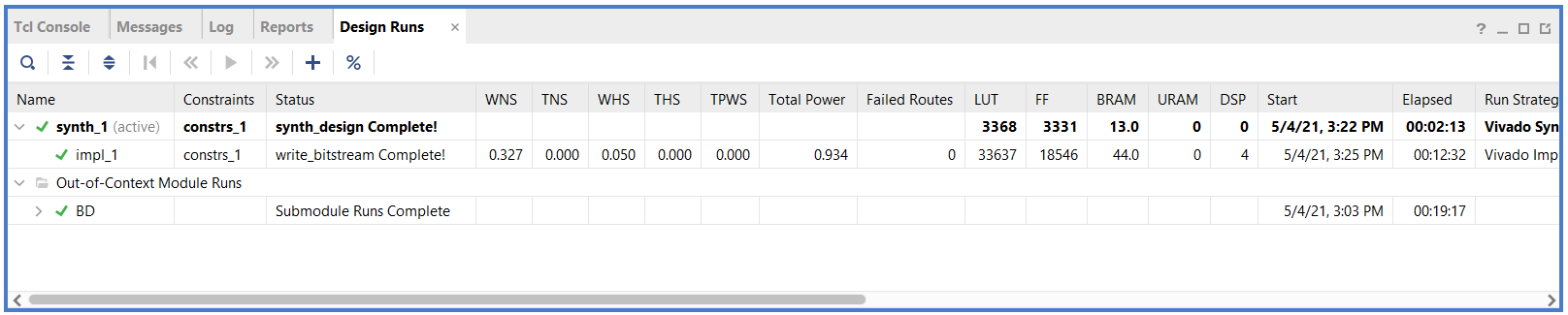
This step synthesizes **RVfpgaNexys** (as defined by the Verilog and SystemVerilog files in the project), maps it onto the FPGA, and creates the bitstream.



**Figure 77. Design Runs**

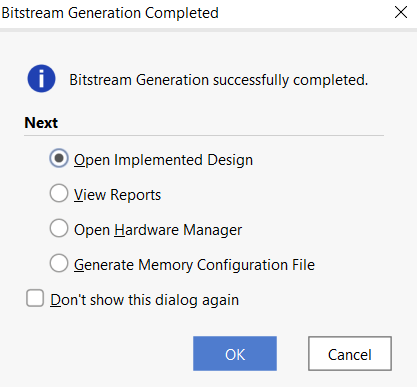
|  |
| --- |
| **Note:**  If you get an error like: Gtk-Message: Failed to load module "canberra-gtk-module"  Install a package by the following command to solve the issue.    sudo apt install libcanberra-gtk-module libcanberra-gtk3-module |

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



**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.