CDS206_Lab1: Fault injection in Xilinx FPGAs

Introduction

In this lab, we will explore how to implement a Design Under Test (DUT) on an FPGA and perform fault injection to evaluate its reliability.

Configuration memory fault injection is a technique used to test the fault tolerance of an FPGA design by injecting faults into the configuration memory of the FPGA. This can be achieved using specialized tools and techniques, such as laser fault injection or manipulating the bitstream of the FPGA.

Bitstream manipulation

In the old days, configuration memory fault injection was performed using a technique called "bitstream manipulation." Bitstream manipulation involves modifying the bitstream file that is used to configure the FPGA, to insert faults into the configuration memory. The modified bitstream file was then loaded onto the FPGA to simulate the effect of a fault in the configuration memory.

Here are the steps to perform configuration memory fault injection using bitstream manipulation in a Xilinx FPGA design under test:

1. Generate a golden bitstream file for the design that you want to test.
2. Modify the golden bitstream file to insert faults into the configuration memory. This can be done by flipping bits.
3. Load the modified bitstream file onto the FPGA.
4. Run the test cases and observe the behavior of the design. If the design is fault-tolerant, it should be able to detect and correct the injected faults and produce correct results.

It is important to note that configuration memory fault injection should be performed with caution, as it can potentially damage the FPGA and may violate the manufacturer's warranty. Experienced professionals should only perform it using specialized tools and equipment in a controlled and secure environment.

Dynamic Partial Reconfiguration (DPR)

Start-of-the-art Xilinx FPGAs support Dynamic Partial Reconfiguration (DPR) that allows a portion of the FPGA to be reconfigured at run-time while the rest of the FPGA remains operational. This feature can be used to perform fault injection of the DUT at runtime.

Fault injection using DPR involves intentionally inserting faults into a module of the FPGA DUT in order to test the system's fault tolerance. Here are the steps involved in creating fault injection using DPR:

1. Establish a JTAG connection to the FPGA to access the configuration memory.
2. Read the configuration frame corresponding to the module or circuit you want to test.
3. Modify the configuration frame by flipping a bit.
4. Write the modified configuration frame back to the configuration memory.
5. Observe the effect of the introduced faults on the DUT's operation.
6. If the DUT can tolerate the fault and continue operating correctly, the fault is considered tolerable. Otherwise, it is considered critical.

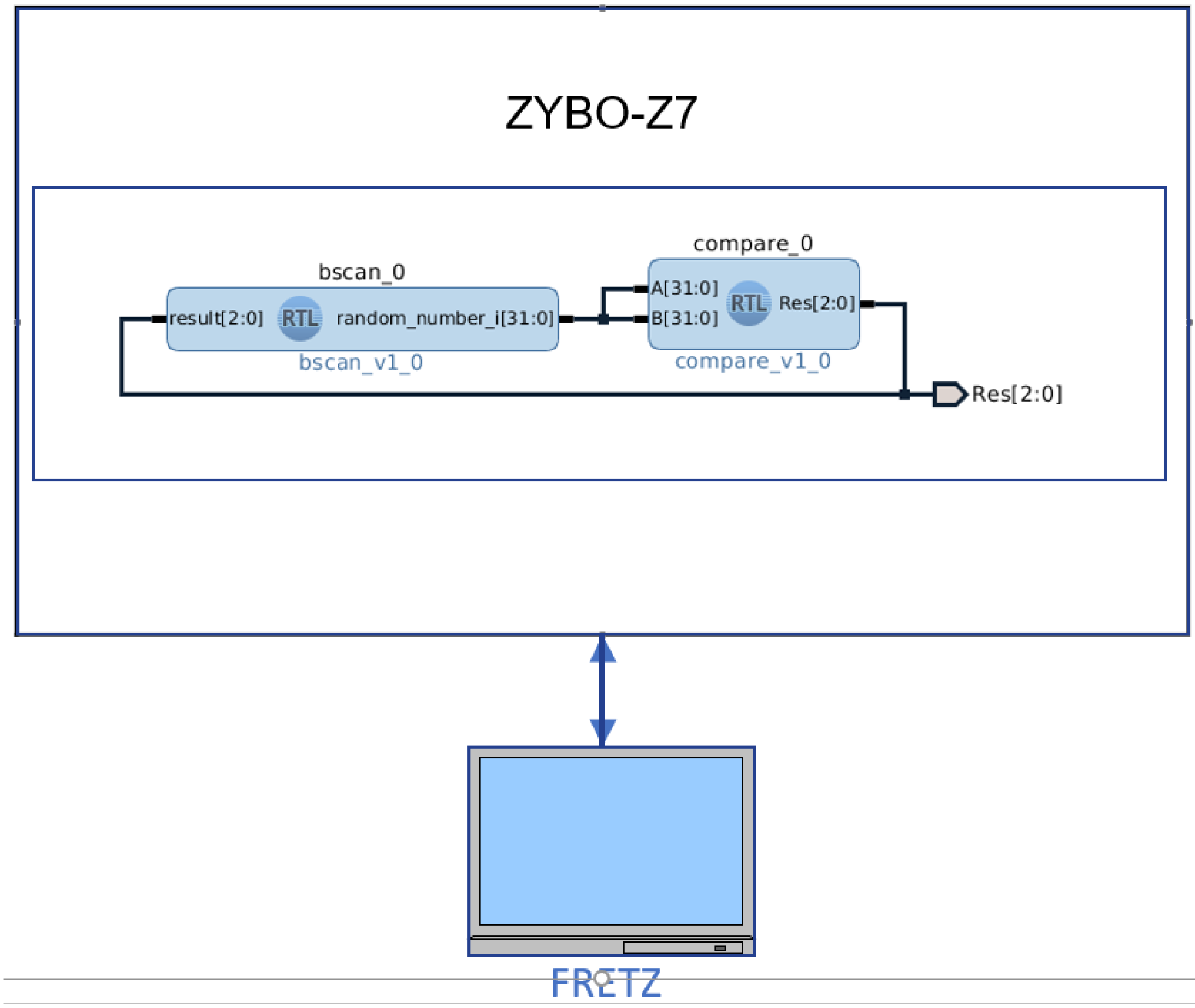
In this lab we insert faults via DPR.

Section 2: Impementation of the DUT

As a starting point, we will implement a 32-bit comparator and send/receive data via JTAG/BSCAN from

https://github.com/unipieslab/FREtZ, while injecting faults. Each time we inject a fault, and the DUT gives an erroneous result, we increase an error_counter. At the end of the experiment, we will count how many injected faults caused an error. This is called The Architectural Vulnerability Factor (AVF) of the DUT. AVF takes values in [0, 1]. The higher the AVF, the more vulnerable it is to SEUs.

The following figure shows the block design diagram of the DUT we will implement.



We will provide input data to the DUT and get the result via BSCAN. Xilinx BSCAN is a feature in Xilinx FPGAs that provides boundary scan testing capabilities. Boundary Scan or JTAG (Joint Test Action Group) is a standard for testing and debugging digital circuits that enables testing of individual pins or nets of a complex circuit board or device.

BSCAN stands for Boundary Scan Chain, which is a series of boundary scan cells that can be used to test and debug the FPGA. These cells allow engineers to perform non-intrusive testing, which means they can test the FPGA without altering its normal operation. BSCAN cells can also be used for device programming and for in-system programming of the FPGA.

Xilinx BSCAN provides a standardized interface for accessing the boundary scan cells in the FPGA, which enables compatibility with industry-standard boundary scan tools. This feature is particularly useful for testing and debugging complex designs with a large number of pins and for verifying connections between the FPGA and other components on the board.

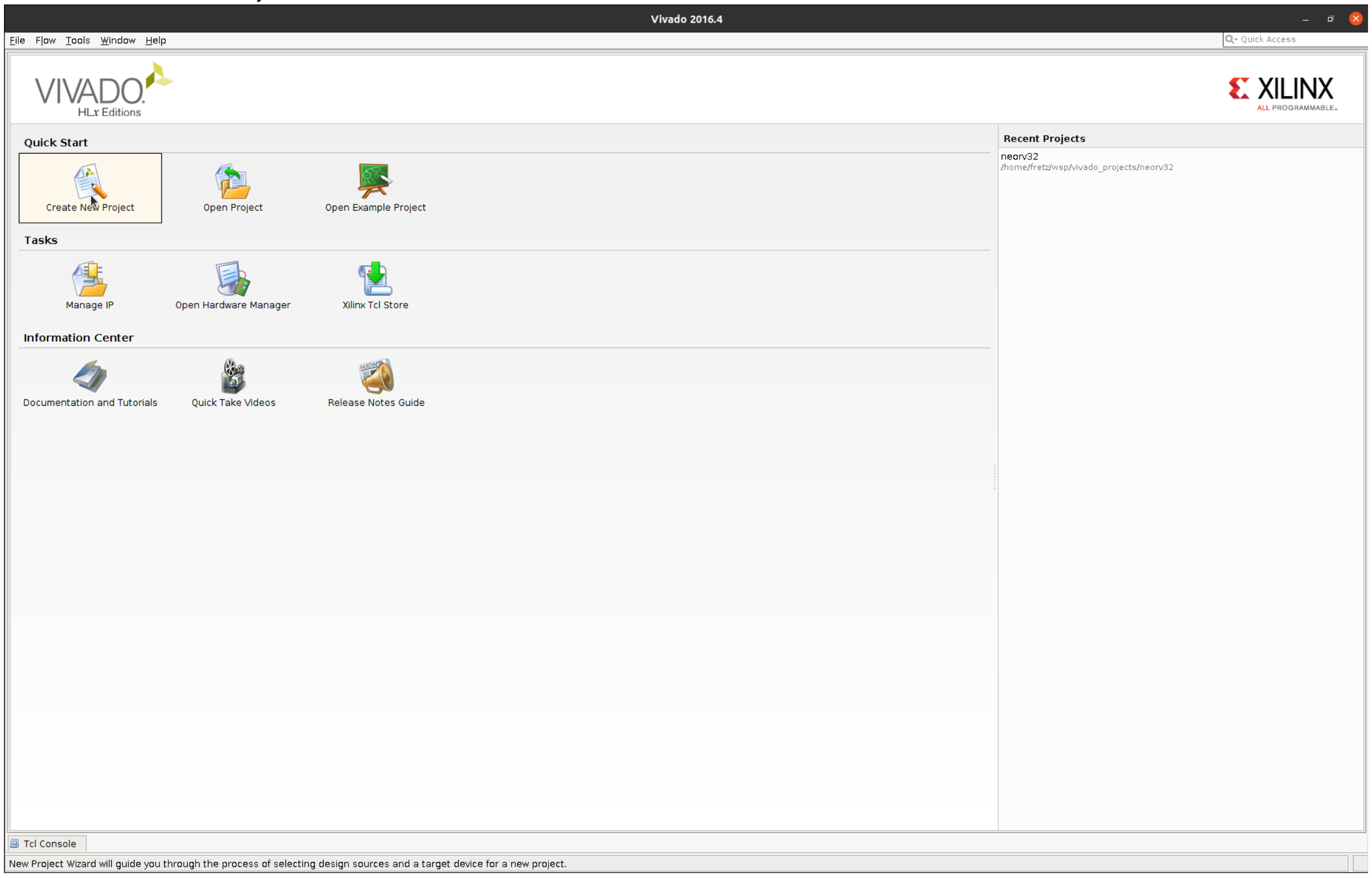
Using boundary scan testing, engineers can apply test patterns to the circuit's inputs and observe the outputs to verify that the circuit is functioning correctly. However, this testing typically requires specialized tools and software and is not intended to be used as a general-purpose input-output interface for the circuit.

Let’s start implementing the design.

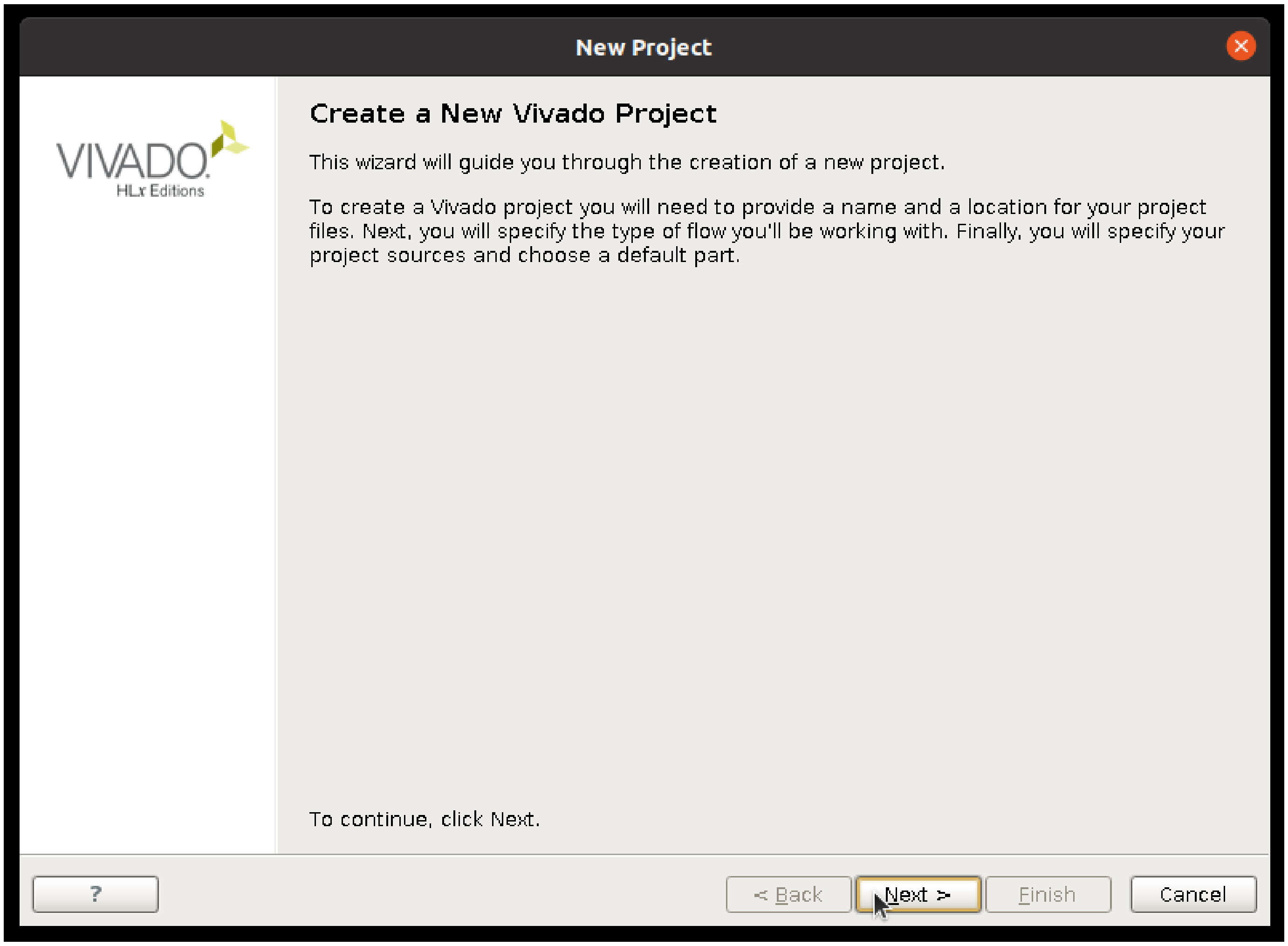
Open a terminal

```
source /opt/Xilinx/Vivado/2016.4/settings64.sh
vivado &
```

Choose -->Create New Project



New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.



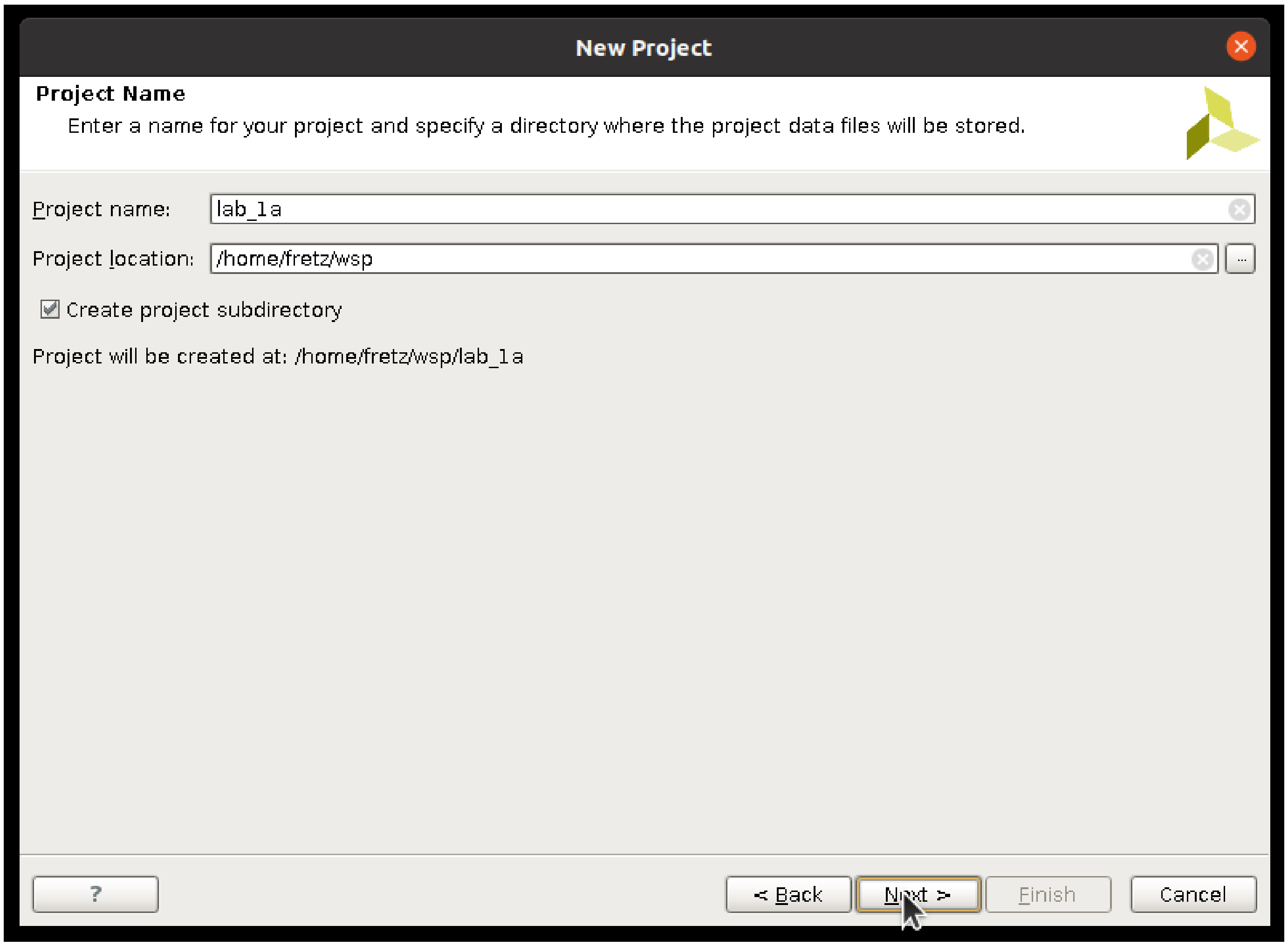
Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

Give the following project name and location names



New Project

Project Name

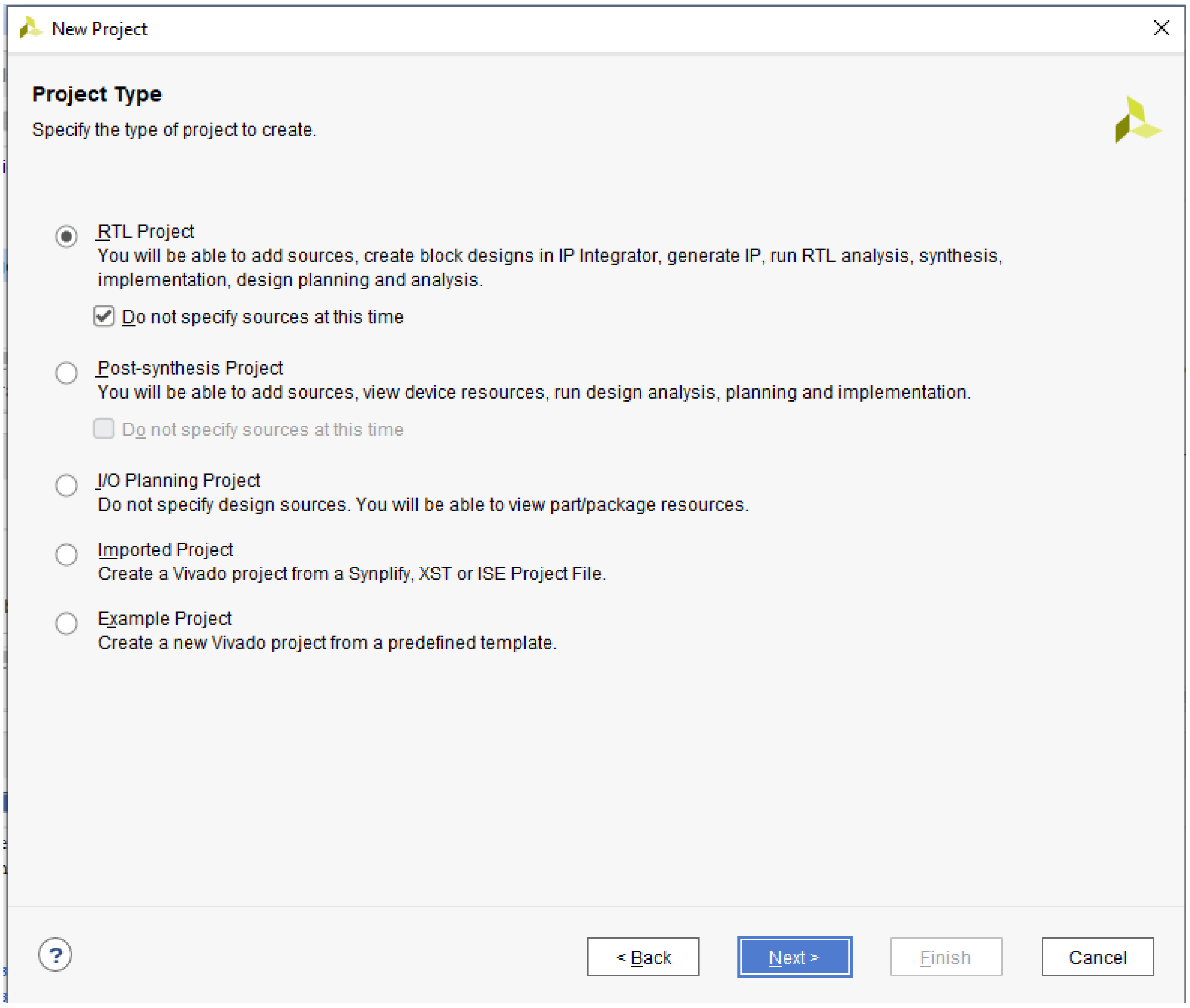
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: lab_1a

Project location: /home/fretz/wsp

☑ Create project subdirectory

Project will be created at: /home/fretz/wsp/lab_1a


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.

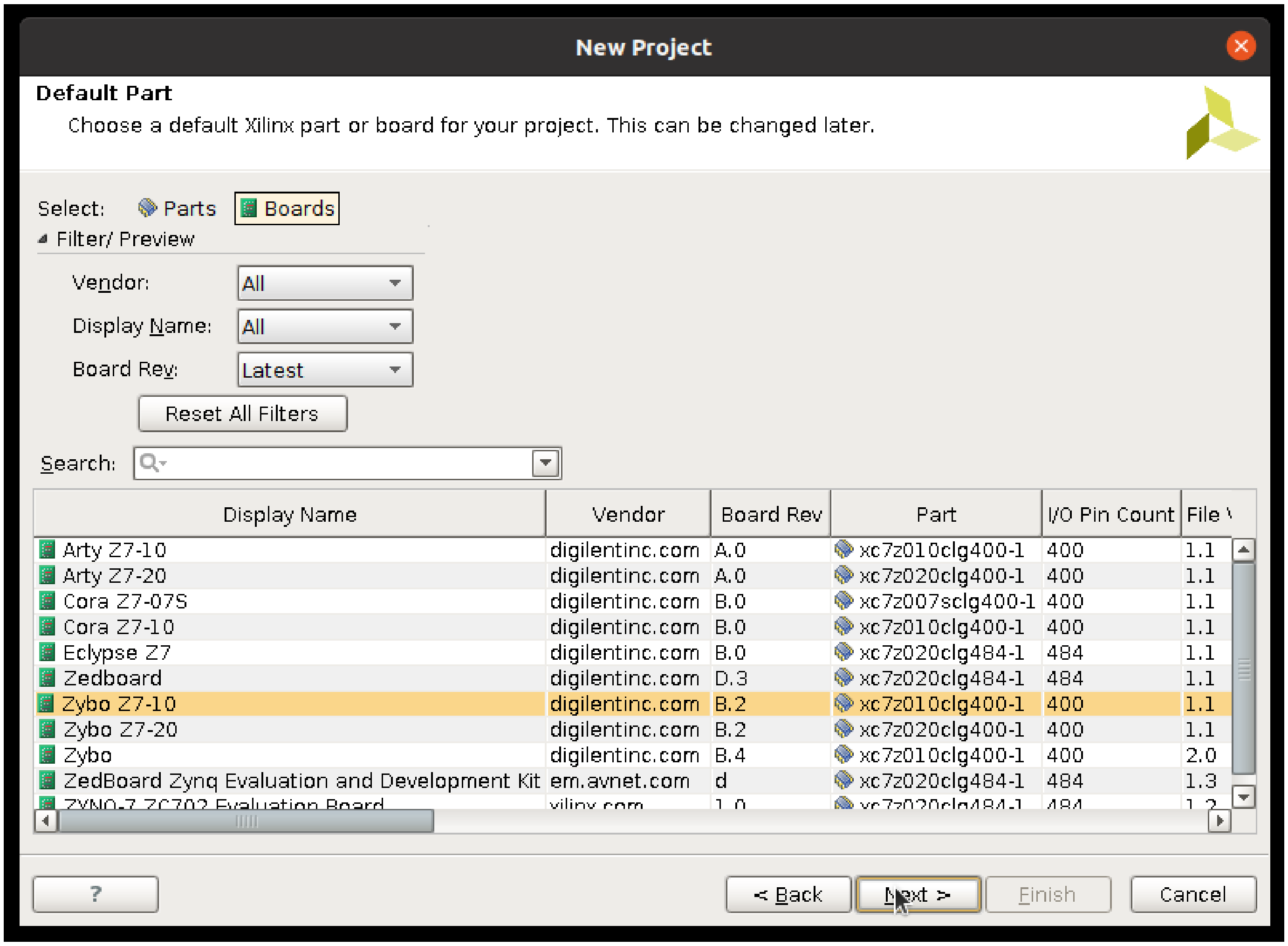
[ ] Do not specify sources at this time

[ ] 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.

Choose the Zybo Z7-10 board.



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts [✓ Boards]

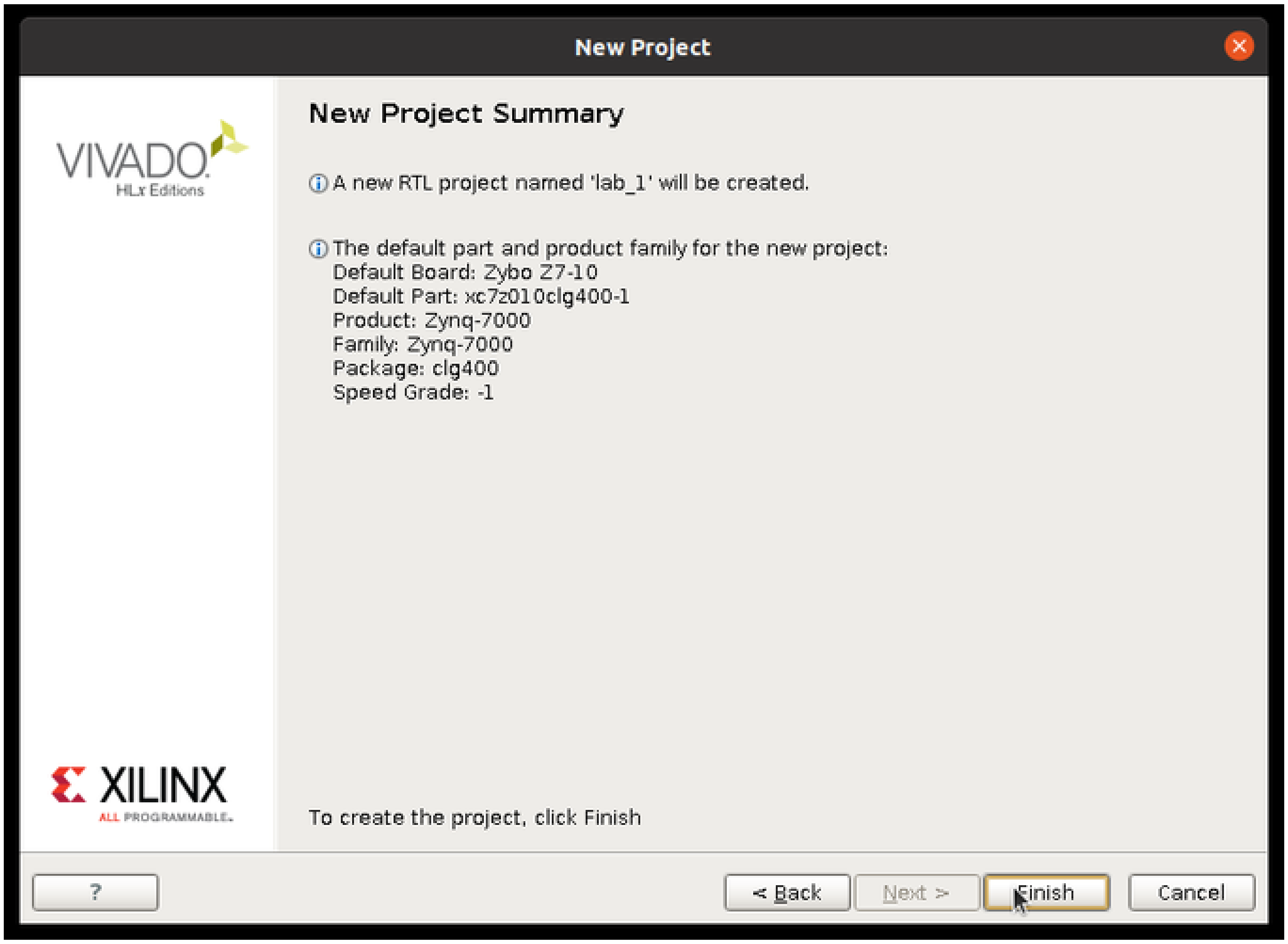
Filter/ Preview

Vendor:	All
Display Name:	All
Board Rev:	Latest

Reset All Filters

Search:

Display Name	Vendor	Board Rev	Part	I/O Pin Count	File \
Arty Z7-10	digilentinc.com	A.0	xc7z010clg400-1	400	1.1
Arty Z7-20	digilentinc.com	A.0	xc7z020clg400-1	400	1.1
Cora Z7-07S	digilentinc.com	B.0	xc7z007sclg400-1	400	1.1
Cora Z7-10	digilentinc.com	B.0	xc7z010clg400-1	400	1.1
Eclypse Z7	digilentinc.com	B.0	xc7z020clg484-1	484	1.1
Zedboard	digilentinc.com	D.3	xc7z020clg484-1	484	1.1
Zybo Z7-10	digilentinc.com	B.2	xc7z010clg400-1	400	1.1
Zybo Z7-20	digilentinc.com	B.2	xc7z020clg400-1	400	1.1
Zybo	digilentinc.com	B.4	xc7z010clg400-1	400	2.0
ZedBoard Zynq Evaluation and Development Kit	em.avnet.com	d	xc7z020clg484-1	484	1.3
ZYNQ-7 ZC702 Evaluation Board	xilinx.com	1.0	xc7z020clg484-1	484	1.2



New Project

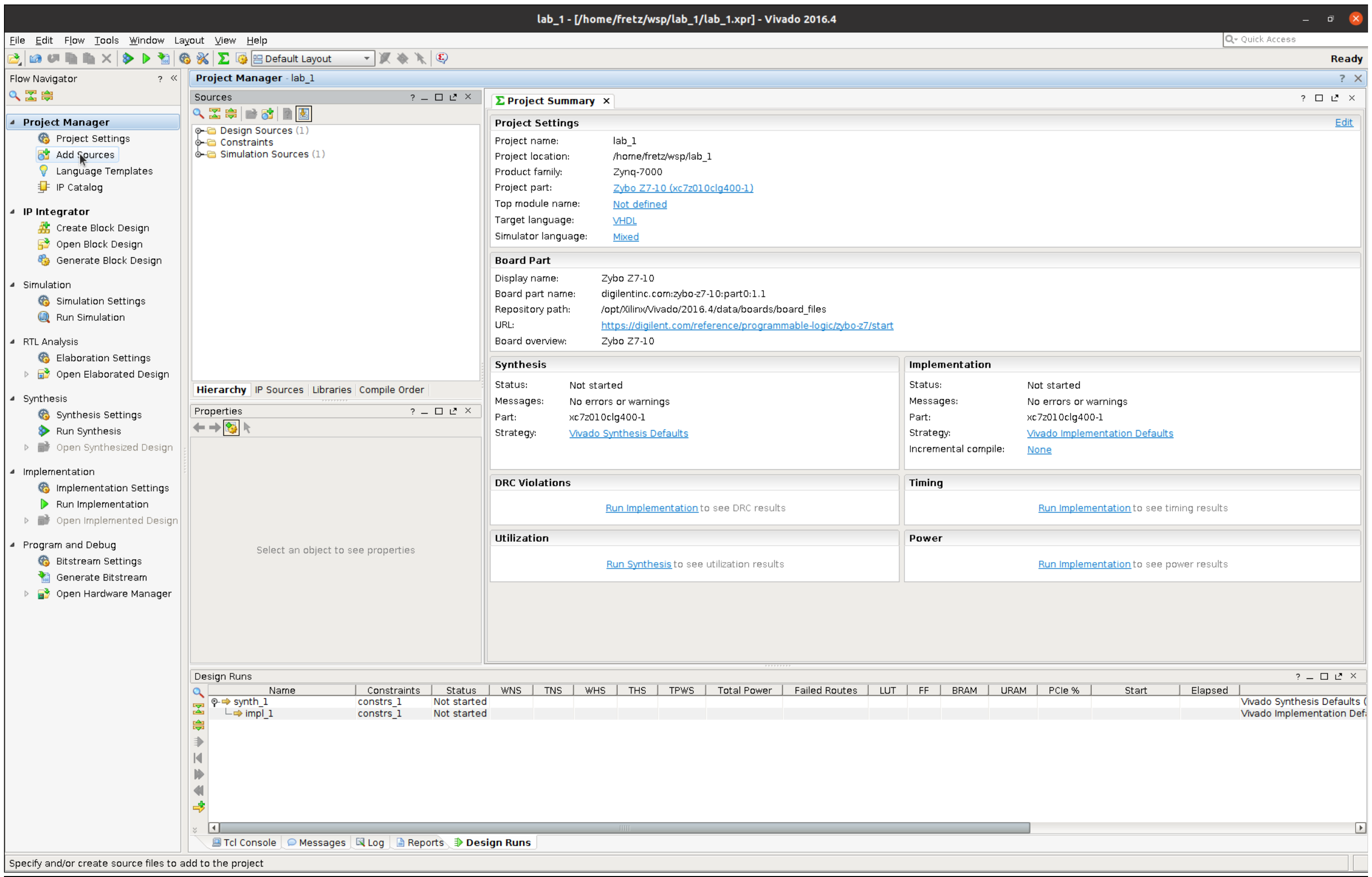
New Project Summary

A new RTL project named 'lab_1' will be created.

The default part and product family for the new project:
Default Board: Zybo Z7-10
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

To create the project, click Finish

Add sources the sources of the comparator



Project Settings Edit

Project name:	lab_1
Project location:	/home/fretz/wsp/lab_1
Product family:	Zynq-7000
Project part:	Zybo Z7-10 (xc7z010clg400-1)
Top module name:	Not defined
Target language:	VHDL
Simulator language:	Mixed

Board Part

Display name:	Zybo Z7-10
Board part name:	digilentinc.com:zybo-z7-10:part0:1.1
Repository path:	/opt/Xilinx/Vivado/2016.4/data/boards/board_files
URL:	https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview:	Zybo Z7-10

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

DRC Violations
Run Implementation to see DRC results

Timing
Run Implementation to see timing results

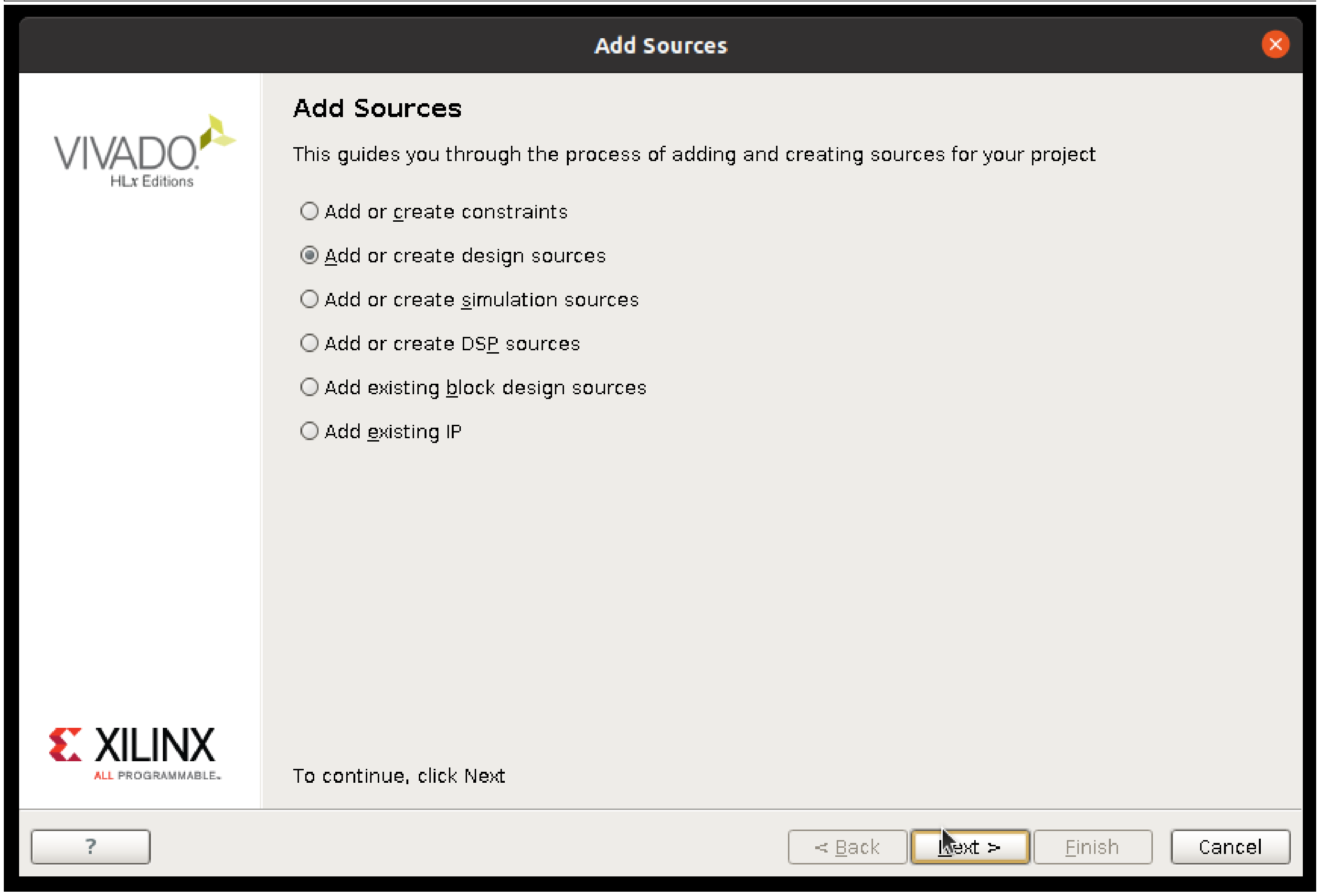
Utilization
Run Synthesis to see utilization results

Power
Run Implementation to see power results

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIe %	Start	Elapsed	
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (
impl_1	constrs_1	Not started															Vivado Implementation Defa

Specify and/or create source files to add to the project

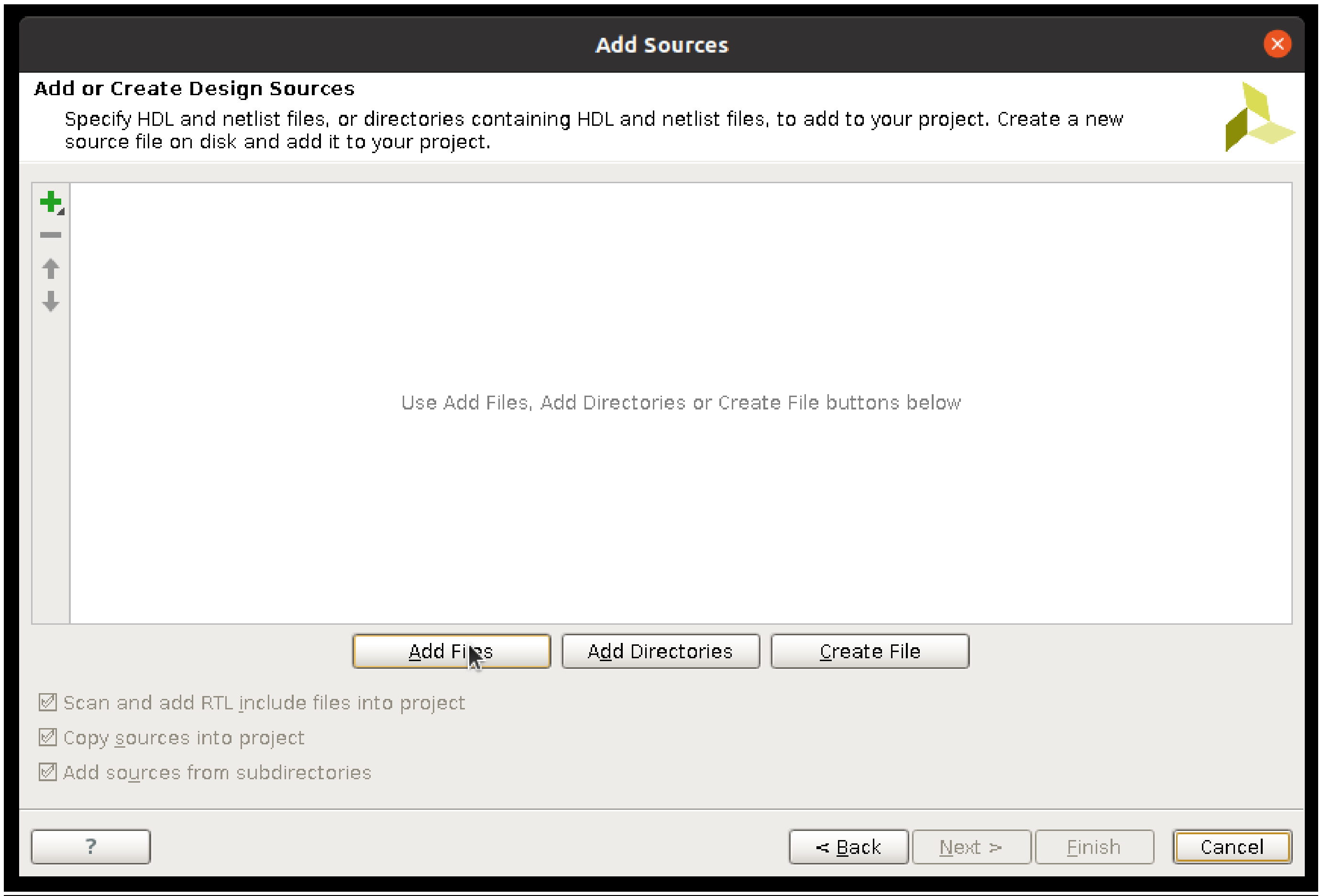


Add Sources

This guides you through the process of adding and creating sources for your project

- Add or create constraints
- [✓ Add or create design sources]
- Add or create simulation sources
- Add or create DSP sources
- Add existing block design sources
- Add existing IP

To continue, click Next

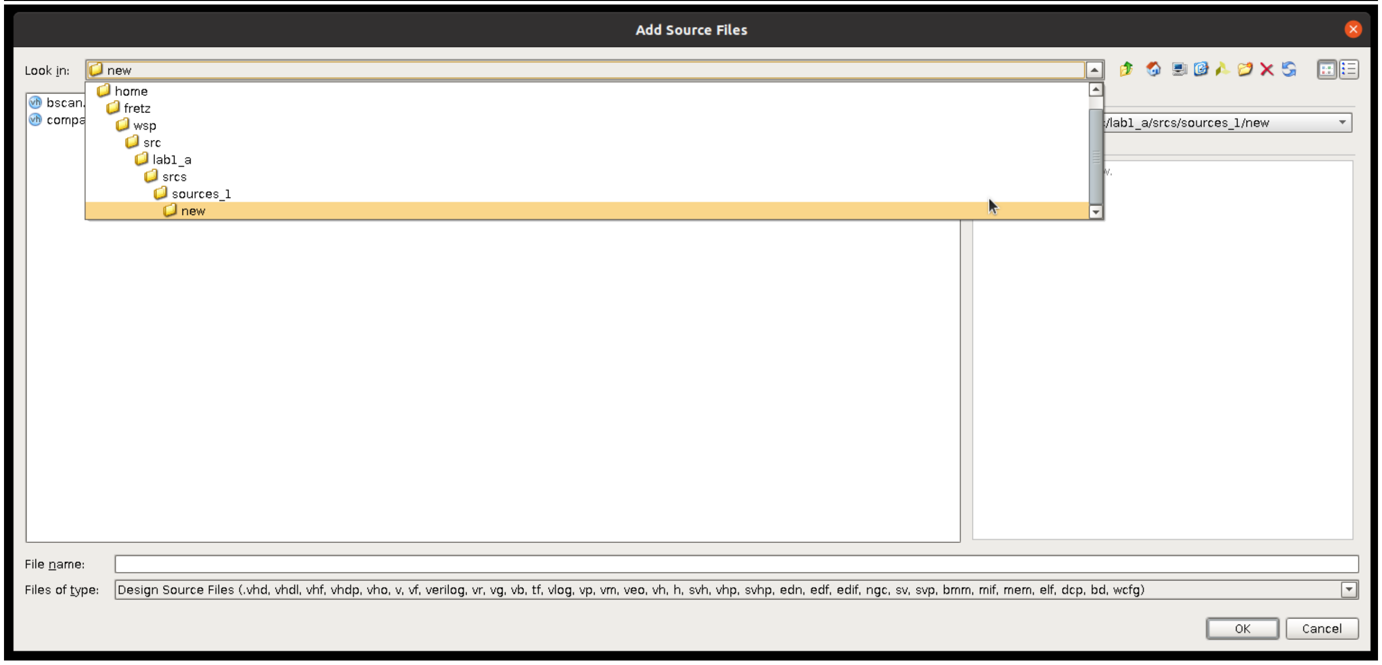


Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Use Add Files, Add Directories or Create File buttons below

☑ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories



Look in: new

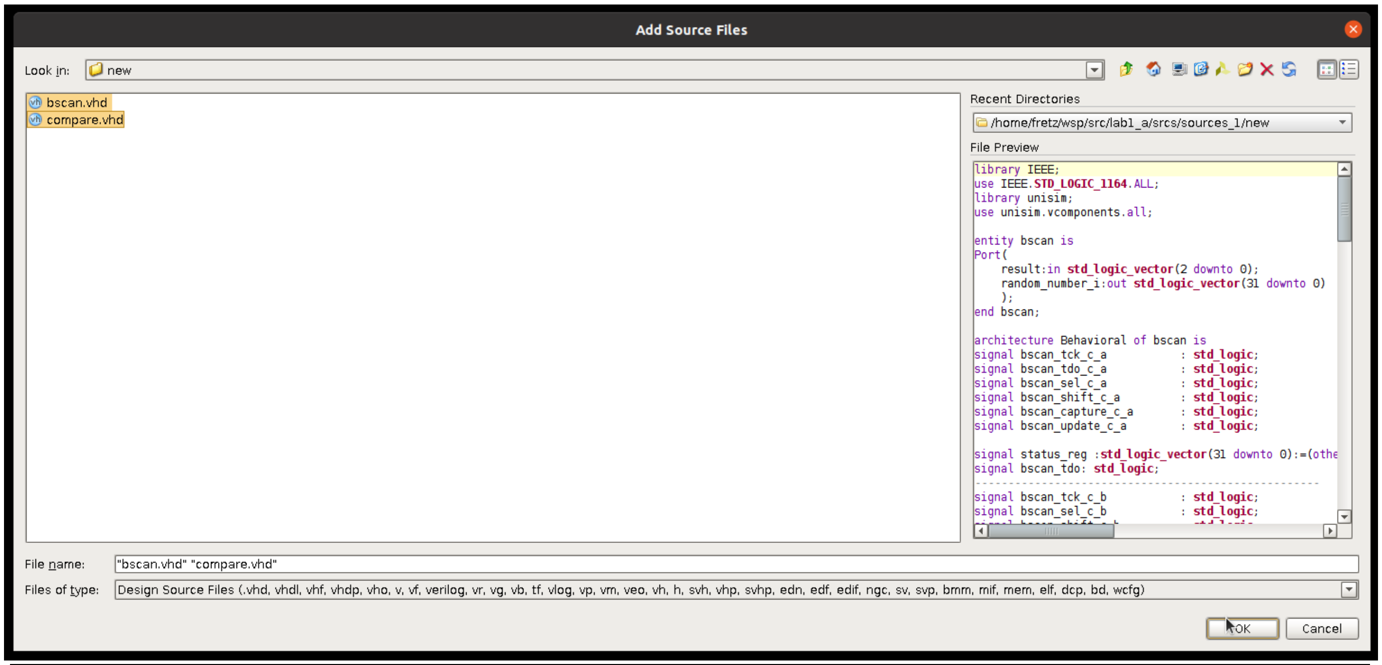
home
fretz
wsp
src
lab1_a
srcs
sources_1
new

bscan
compa

/lab1_a/srcs/sources_1/new

File name:

Files of type: Design Source Files (.vhd, vhdl, vhf, vhdp, vho, v, vf, verilog, vr, vg, vb, tf, vlog, vp, vm, veo, vh, h, svh, vhp, svhp, edn, edf, edif, ngc, sv, svp, bmm, mif, mem, elf, dcp, bd, wcfg)



Add Source Files

Look in: new

bscan.vhd
compare.vhd

Recent Directories

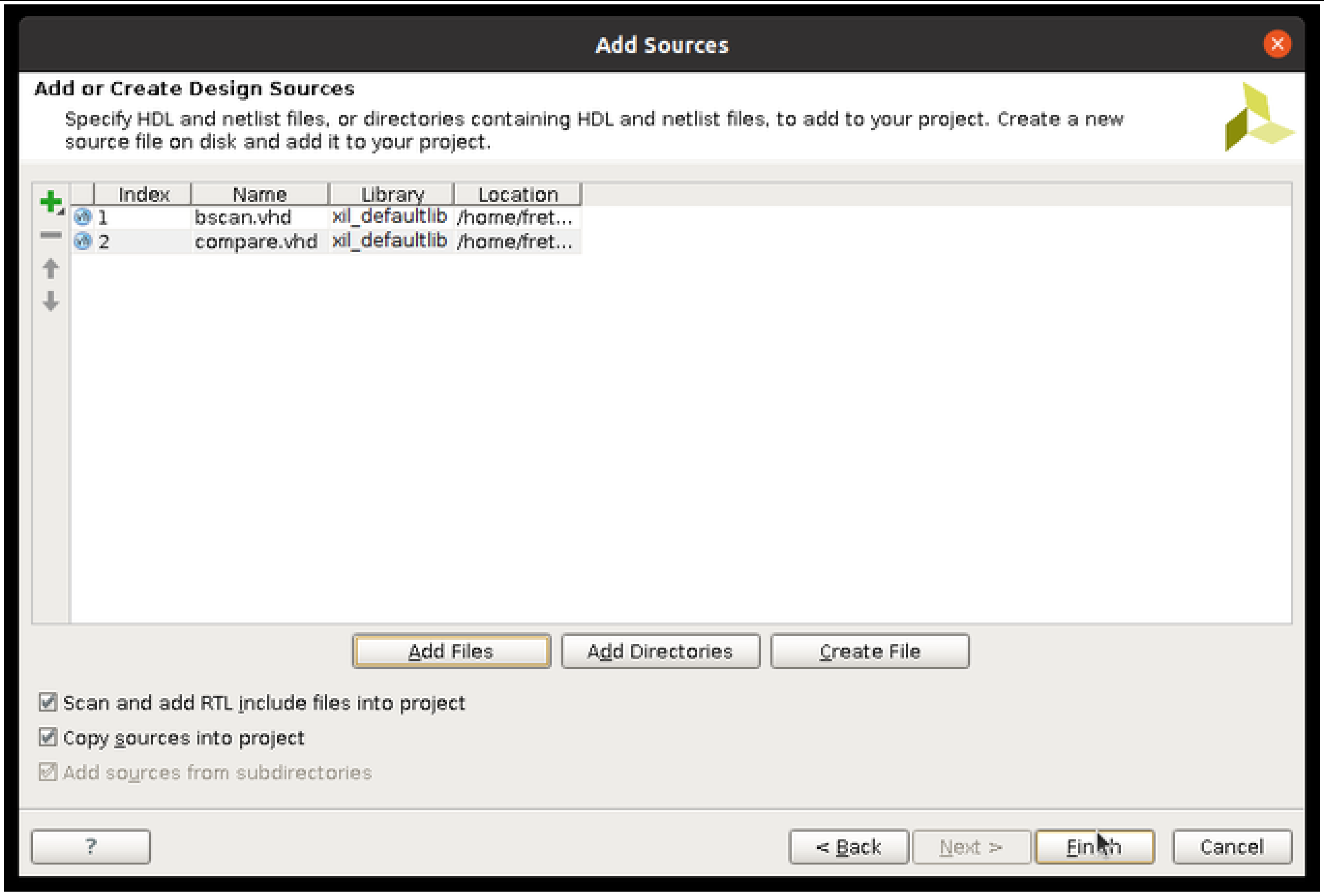
/home/fretz/wsp/src/lab1_a/srcs/sources_1/new

File Preview

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library unisim;
use unisim.vcomponents.all;

entity bscan is
Port(
    result:in std_logic_vector(2 downto 0);
    random_number_i:out std_logic_vector(31 downto 0)
    );
end bscan;

architecture Behavioral of bscan is
signal bscan_tck_c_a            : std_logic;
signal bscan_tdo_c_a            : std_logic;
signal bscan_sel_c_a            : std_logic;
signal bscan_shift_c_a          : std_logic;
signal bscan_capture_c_a        : std_logic;
signal bscan_update_c_a         : std_logic;

signal status_reg :std_logic_vector(31 downto 0):=(othe
signal bscan_tdo: std_logic;
---------------------------------------------------
signal bscan_tck_c_b            : std_logic;
signal bscan_sel_c_b            : std_logic;
```

File name: "bscan.vhd" "compare.vhd"

Files of type: Design Source Files (.vhd, vhdl, vhf, vhdp, vho, v, vf, verilog, vr, vg, vb, tf, vlog, vp, vm, veo, vh, h, svh, vhp, svhp, edn, edf, edif, ngc, sv, svp, bmm, mif, mem, elf, dcp, bd, wcfg)



Add Sources

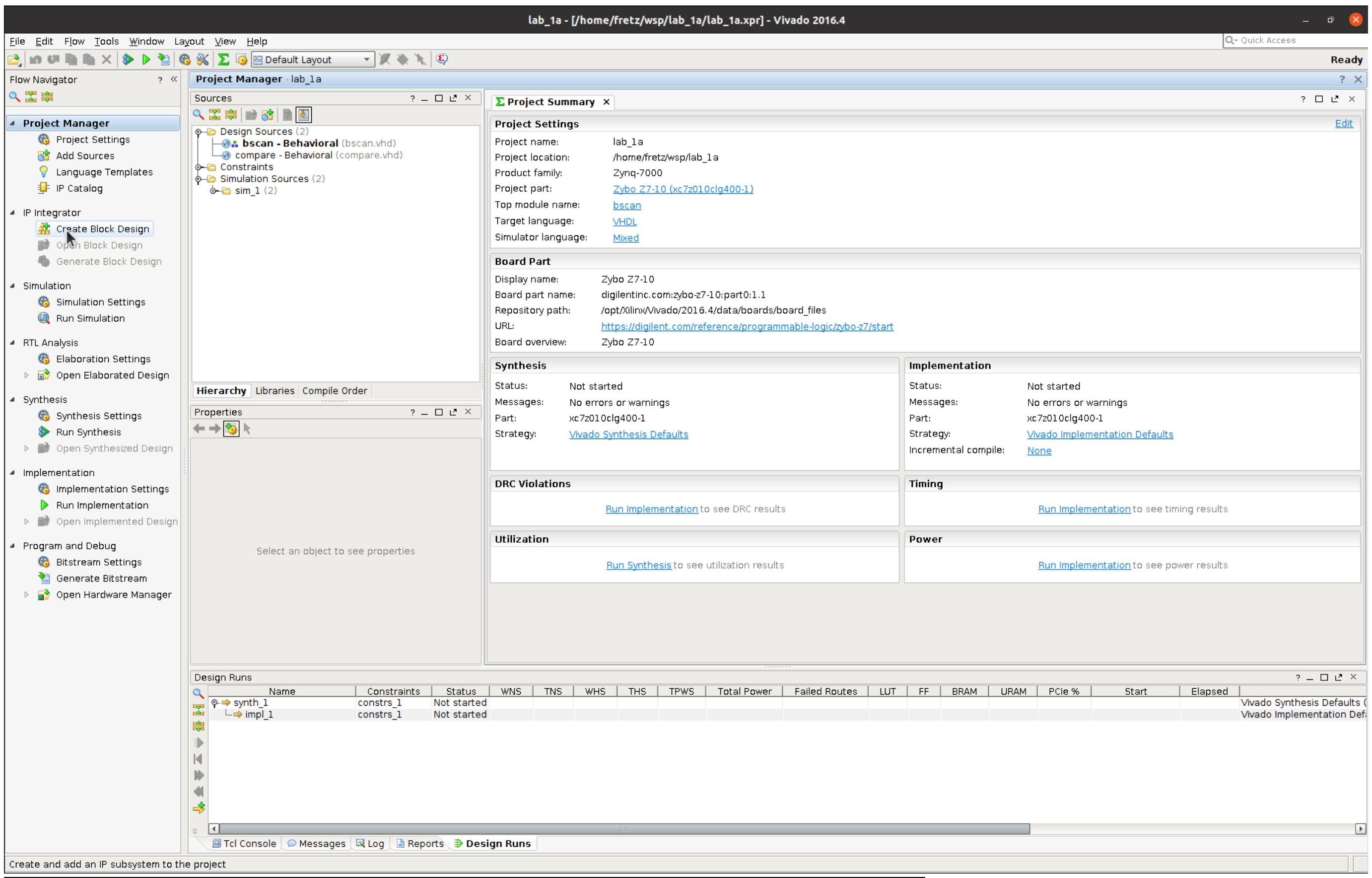
Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Index	Name	Library	Location
1	bscan.vhd	xil_defaultlib	/home/fret...
2	compare.vhd	xil_defaultlib	/home/fret...

☑ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Create a block design (choose Create Block Design from the left side window of Vivado)



Design Sources (2)
- bscan - Behavioral (bscan.vhd)
  - compare - Behavioral (compare.vhd)

Constraints

Simulation Sources (2)
- sim_1 (2)

Select an object to see properties

Project Settings Edit

Project name:	lab_1a
Project location:	/home/fretz/wsp/lab_1a
Product family:	Zynq-7000
Project part:	Zybo Z7-10 (xc7z010clg400-1)
Top module name:	bscan
Target language:	VHDL
Simulator language:	Mixed

Board Part

Display name:	Zybo Z7-10
Board part name:	digilentinc.com:zybo-z7-10:part0:1.1
Repository path:	/opt/Xilinx/Vivado/2016.4/data/boards/board_files
URL:	https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview:	Zybo Z7-10

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z010clg400-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization

Run Synthesis to see utilization results

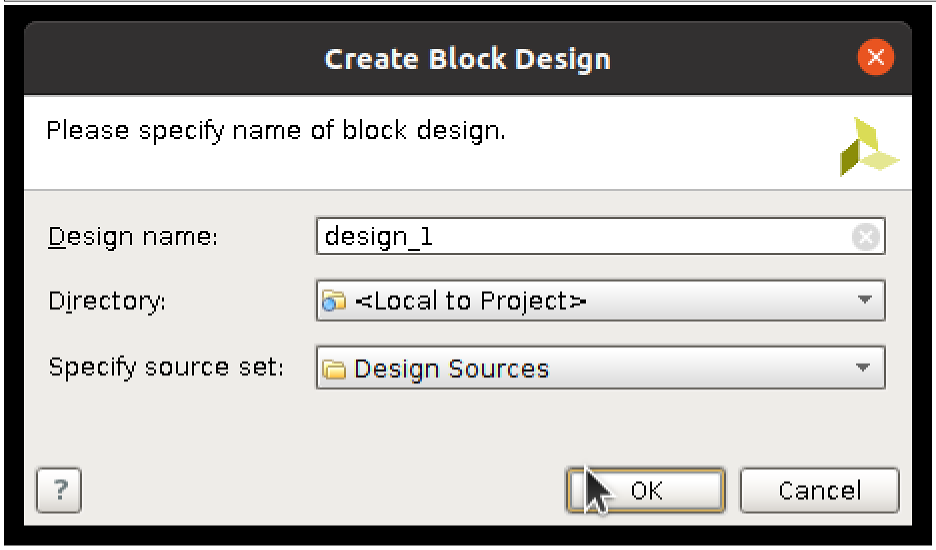
Power

Run Implementation to see power results

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIe %	Start	Elapsed	
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (
impl_1	constrs_1	Not started															Vivado Implementation Def

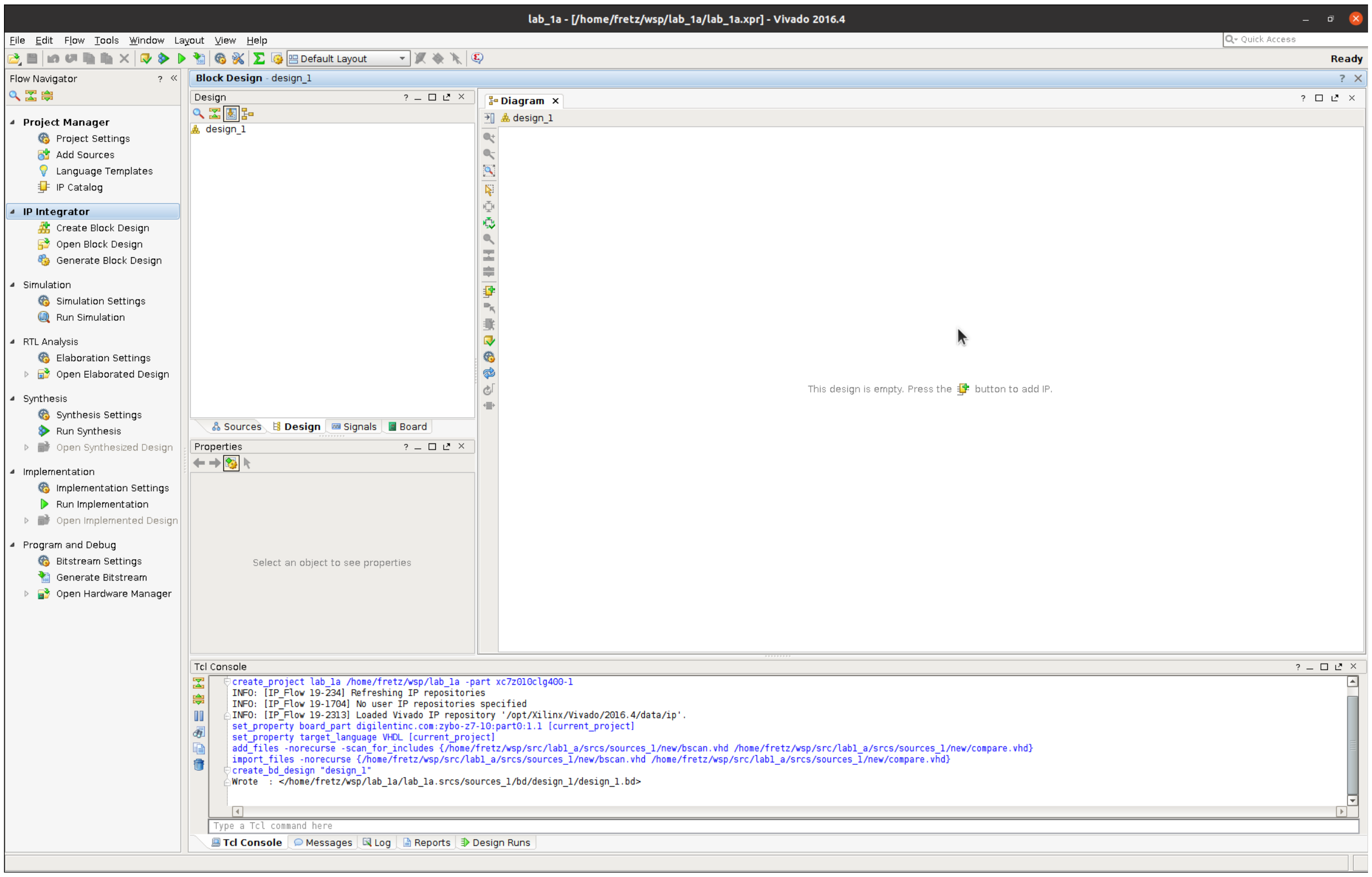
Create and add an IP subsystem to the project



Please specify name of block design.

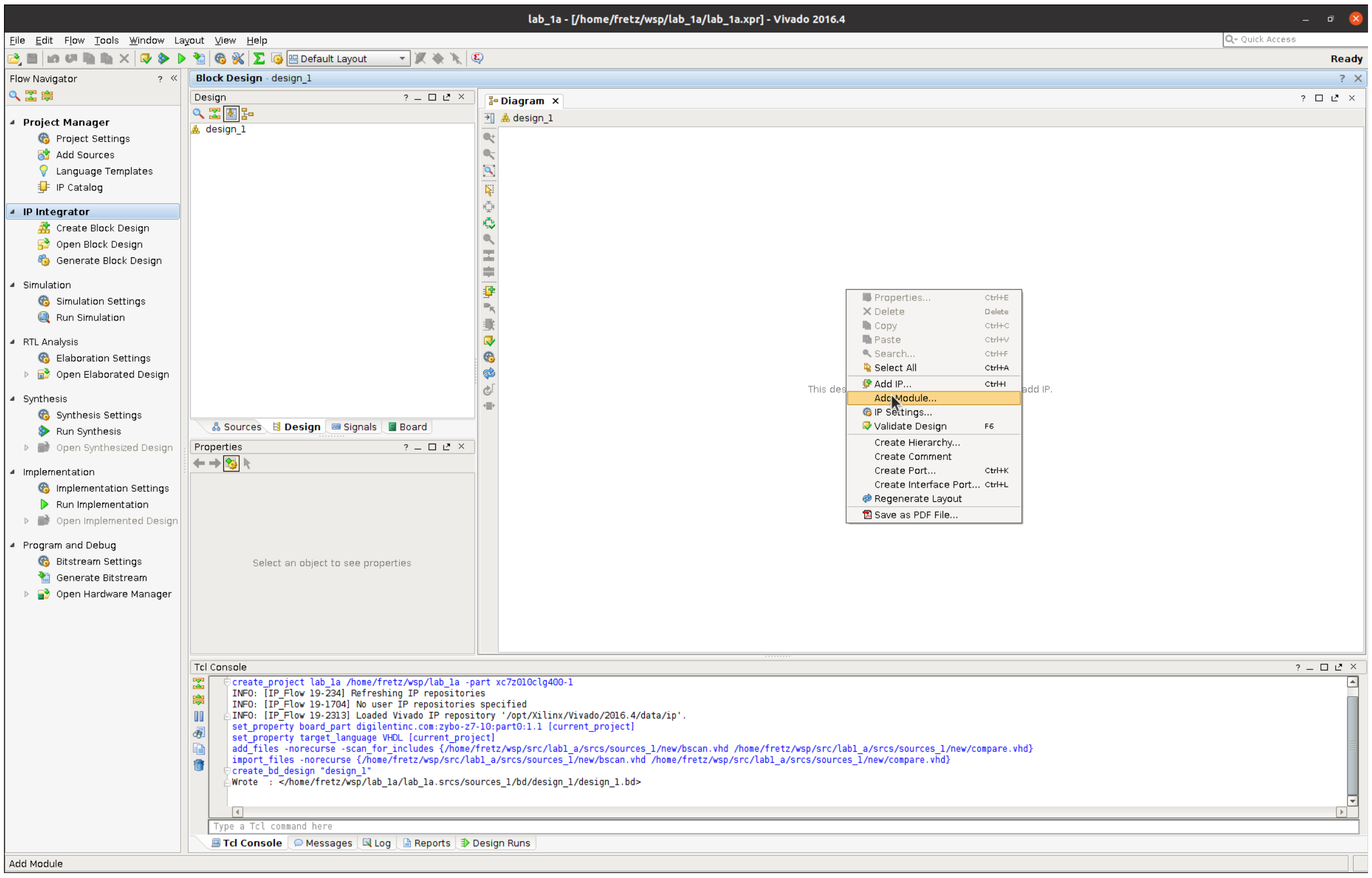
Design name:	design_1
Directory:	<Local to Project>
Specify source set:	Design Sources

Press right-click on the empty block design


lab_1a - [/home/fretz/wsp/lab_1a/lab_1a.xpr] - Vivado 2016.4
Flow Navigator
Project Manager
IP Integrator
Create Block Design
Block Design - design_1
Diagram
This design is empty. Press the button to add IP.
Select an object to see properties

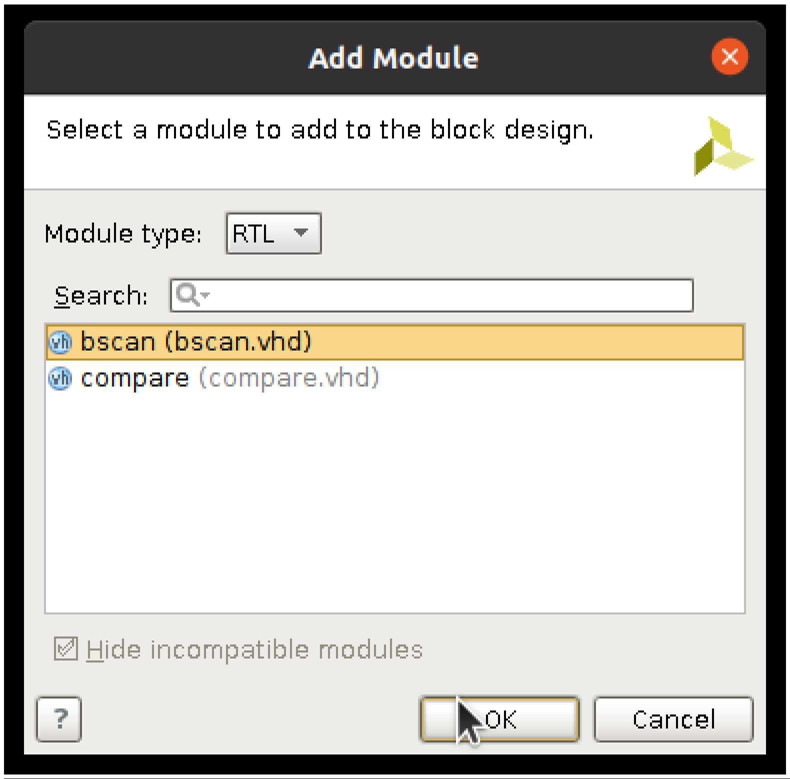
```
create_project lab_1a /home/fretz/wsp/lab_1a -part xc7z010clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2016.4/data/ip'.
set_property board_part digilentinc.com:zybo-z7-10:part0:1.1 [current_project]
set_property target_language VHDL [current_project]
add_files -norecurse -scan_for_includes {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
import_files -norecurse {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
create_bd_design "design_1"
Wrote  : </home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd>
```

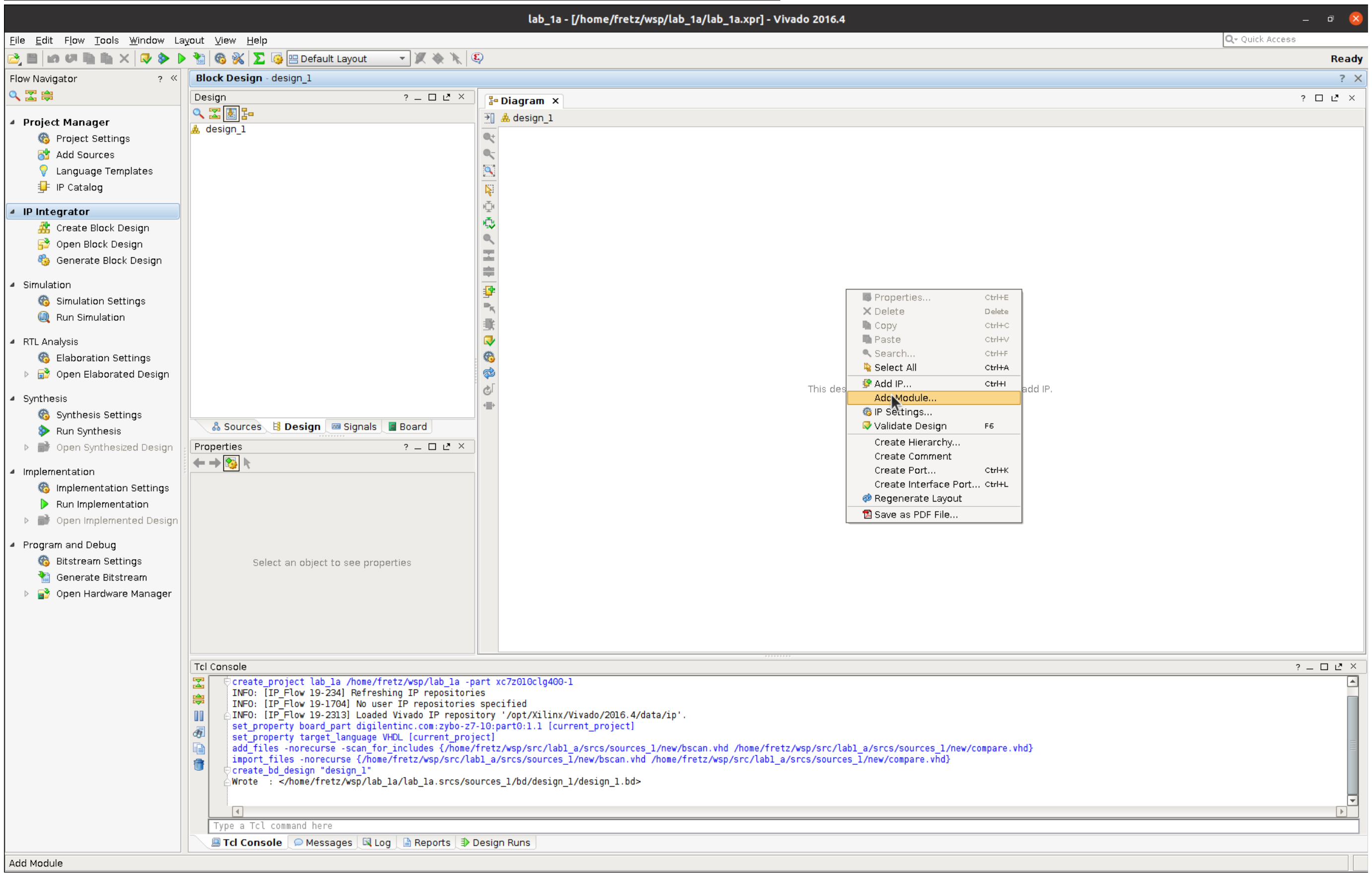
Choose the Add Module



```
create_project lab_1a /home/fretz/wsp/lab_1a -part xc7z010clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2016.4/data/ip'.
set_property board_part digilentinc.com:zybo-z7-10:part0:1.1 [current_project]
set_property target_language VHDL [current_project]
add_files -norecurse -scan_for_includes {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
import_files -norecurse {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
create_bd_design "design_1"
Wrote  : </home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd>
```

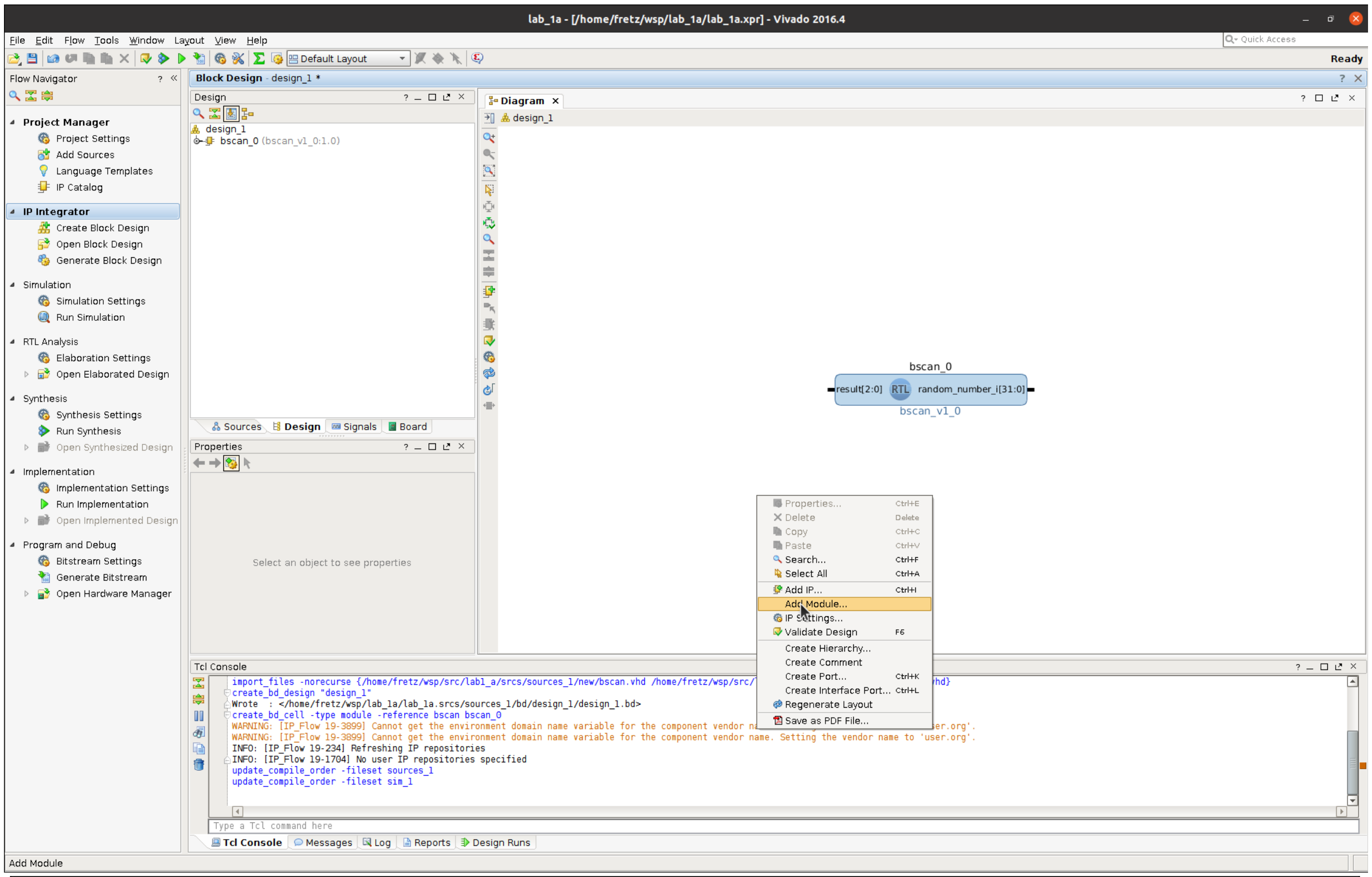
Add the sources of bscan module



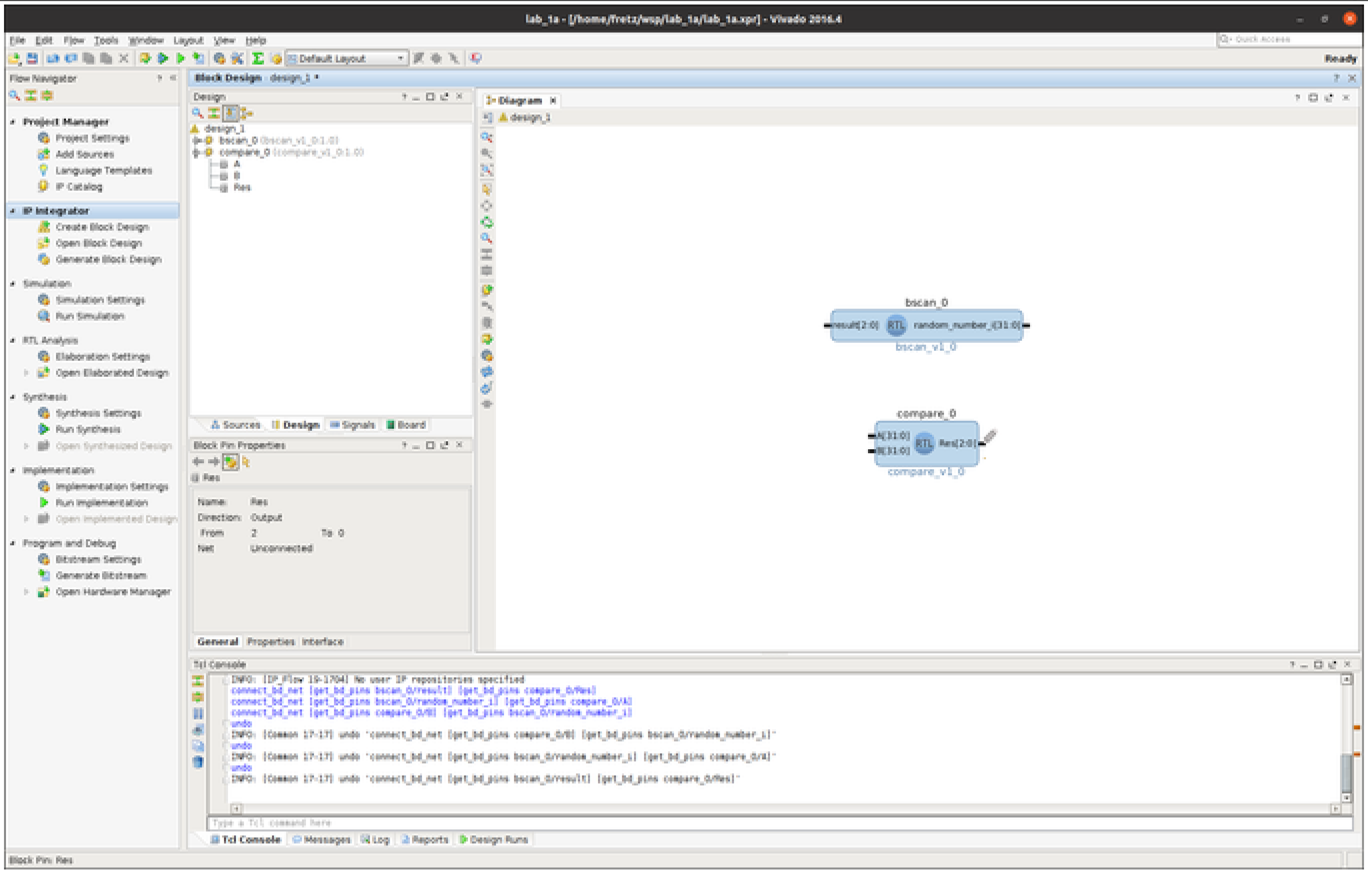


```
create_project lab_1a /home/fretz/wsp/lab_1a -part xc7z010clg400-1
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2016.4/data/ip'.
set_property board_part digilentinc.com:zybo-z7-10:part0:1.1 [current_project]
set_property target_language VHDL [current_project]
add_files -norecurse -scan_for_includes {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
import_files -norecurse {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/lab1_a/srcs/sources_1/new/compare.vhd}
create_bd_design "design_1"
Wrote  : </home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd>
```

Add the sources of the comparator module



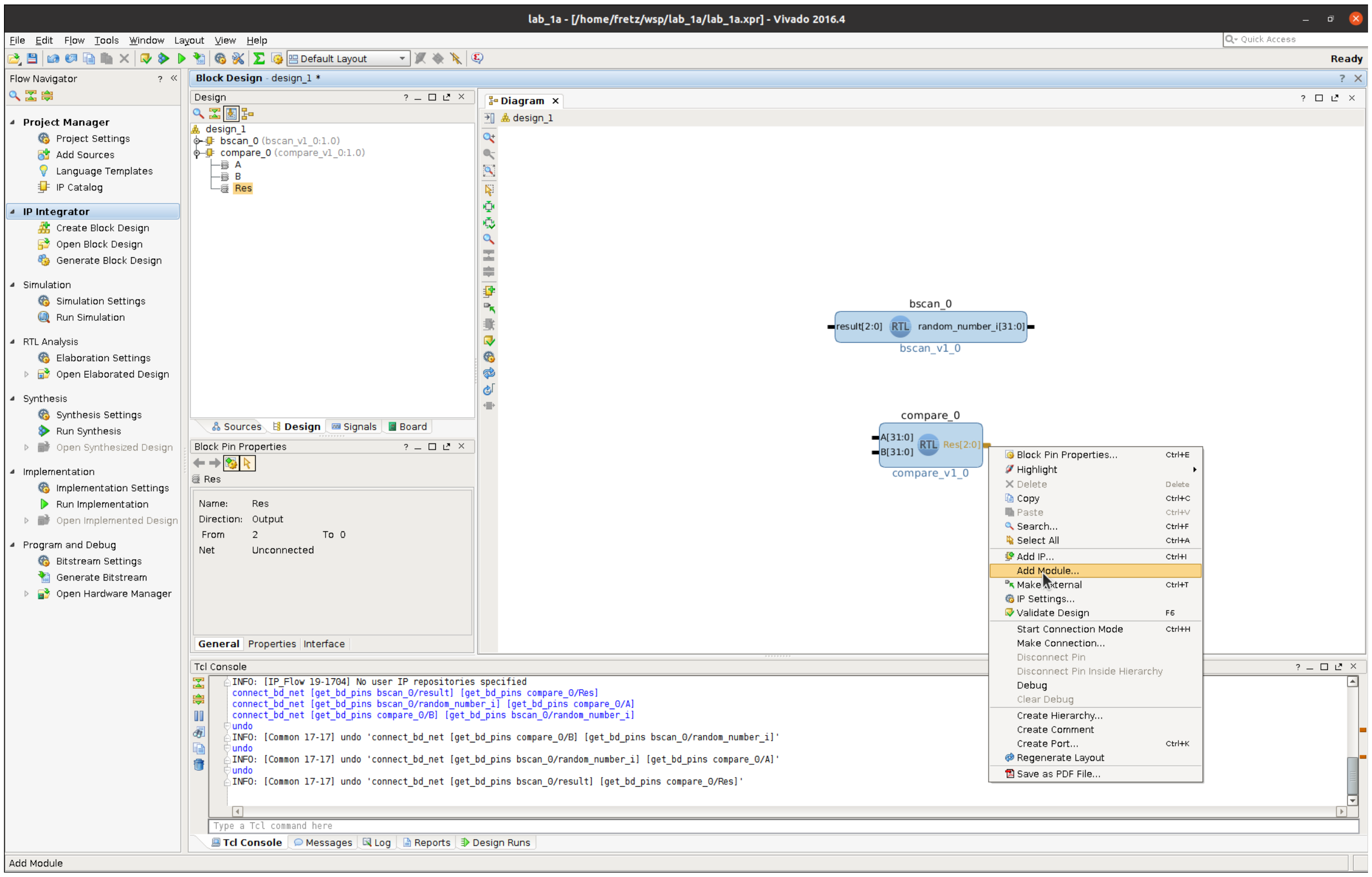
```
import_files -norecurse {/home/fretz/wsp/src/lab1_a/srcs/sources_1/new/bscan.vhd /home/fretz/wsp/src/                    hd}
create_bd_design "design_1"
Wrote  : </home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd>
create_bd_cell -type module -reference bscan bscan_0
WARNING: [IP_Flow 19-3899] Cannot get the environment domain name variable for the component vendor na                    er.org'.
WARNING: [IP_Flow 19-3899] Cannot get the environment domain name variable for the component vendor name. Setting the vendor name to 'user.org'.
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
```



Name:	Res
Direction:	Output
From	2 To 0
Net	Unconnected

```
INFO: [IP_Flow 19-1704] No user IP repositories specified
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]'
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]'
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]'
```

Choose the output wires of the comparator and press right click Make external.

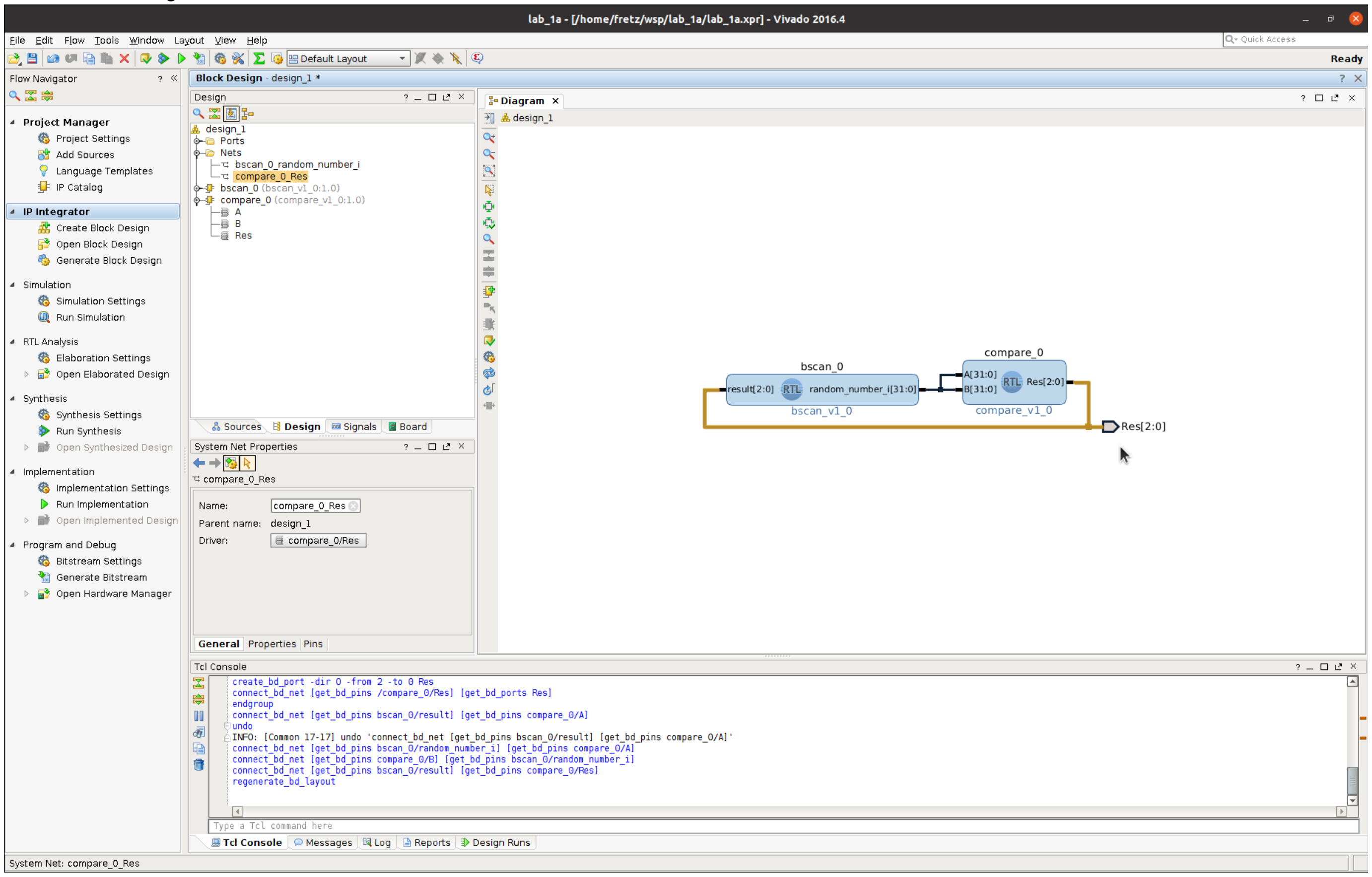


Block Pin Properties: Res

Name: Res
Direction: Output
From 2 To 0
Net Unconnected

```
INFO: [IP_Flow 19-1704] No user IP repositories specified
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]'
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]'
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]'
```

Make the following connections

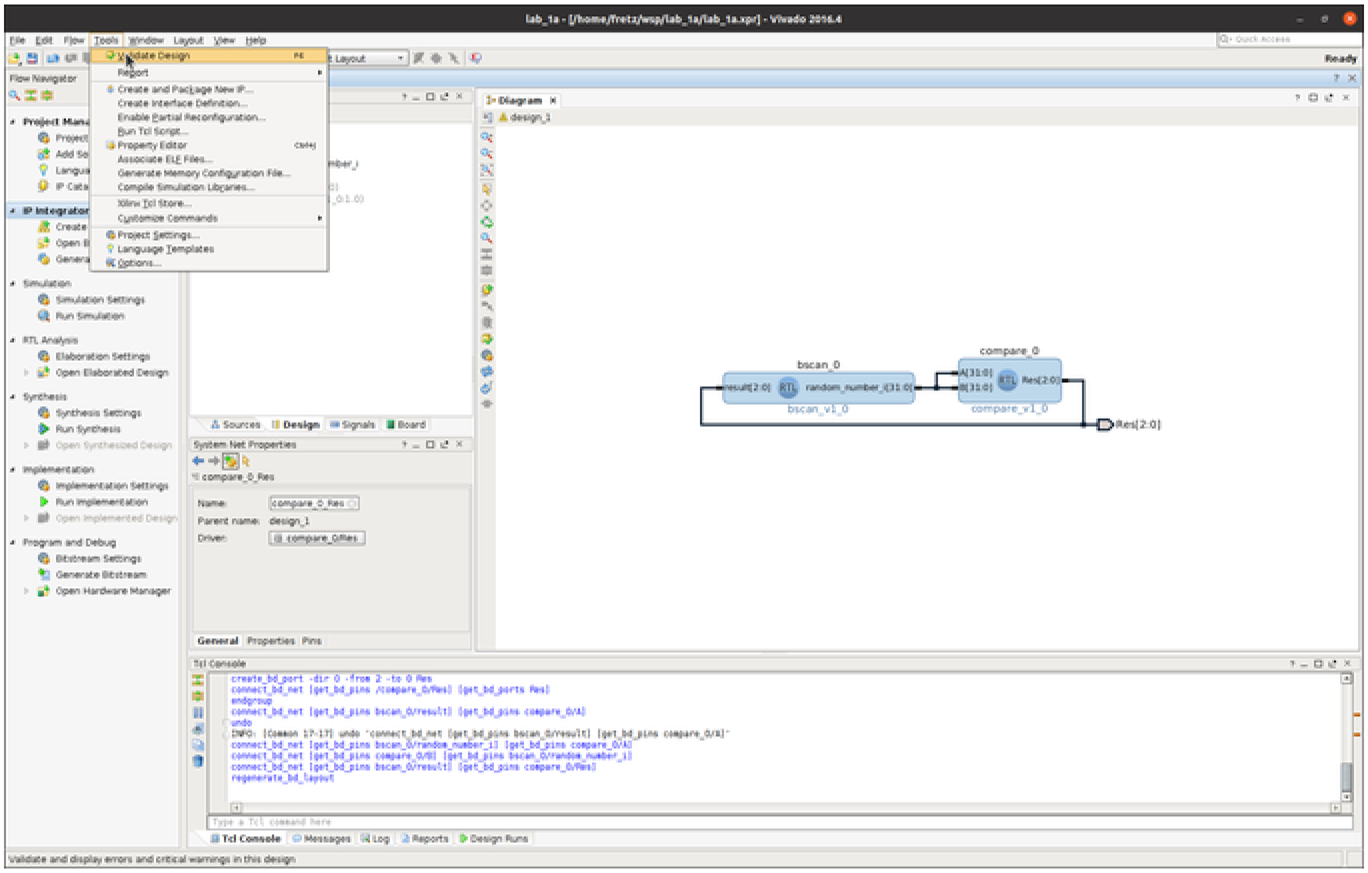

lab_1a - [/home/fretz/wsp/lab_1a/lab_1a.xpr] - Vivado 2016.4
Block Design - design_1 *
Diagram
bscan_0
result[2:0]
random_number_i[31:0]
bscan_v1_0
compare_0
A[31:0]
B[31:0]
Res[2:0]
compare_v1_0
Res[2:0]
System Net: compare_0_Res

System Net Properties: compare_0_Res

Name: compare_0_Res
Parent name: design_1
Driver: compare_0/Res

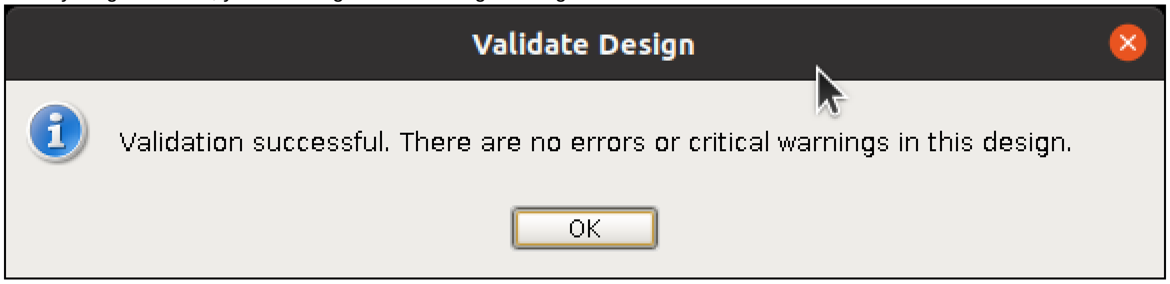
```
create_bd_port -dir O -from 2 -to 0 Res
connect_bd_net [get_bd_pins /compare_0/Res] [get_bd_ports Res]
endgroup
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]'
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
regenerate_bd_layout
```

Validate the correctness of your block design



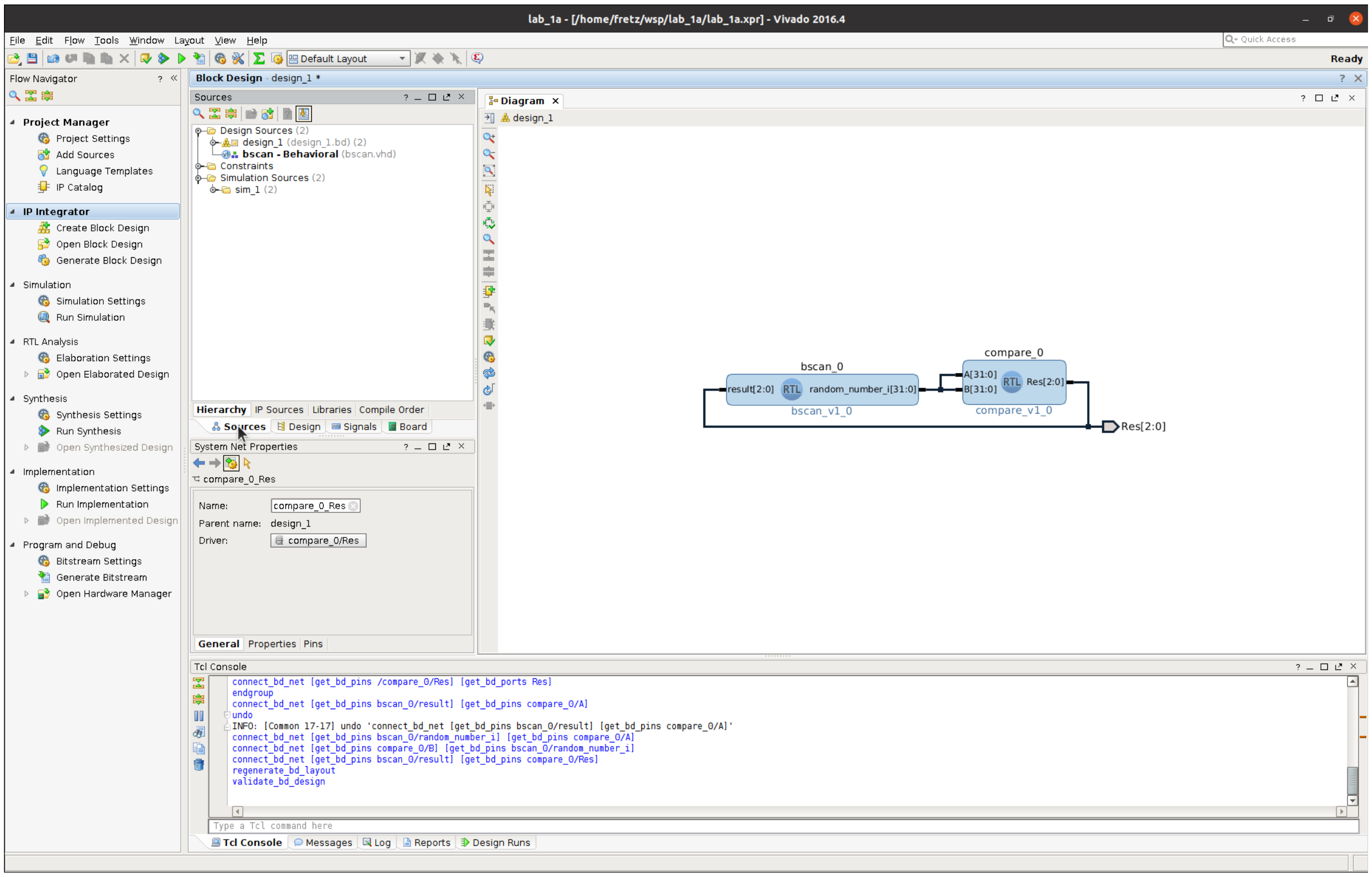
```
create_bd_port -dir O -from 2 -to 0 Res
connect_bd_net [get_bd_pins /compare_0/Res] [get_bd_ports Res]
endgroup
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]'
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
regenerate_bd_layout
```

If everything went well, you should get the following message



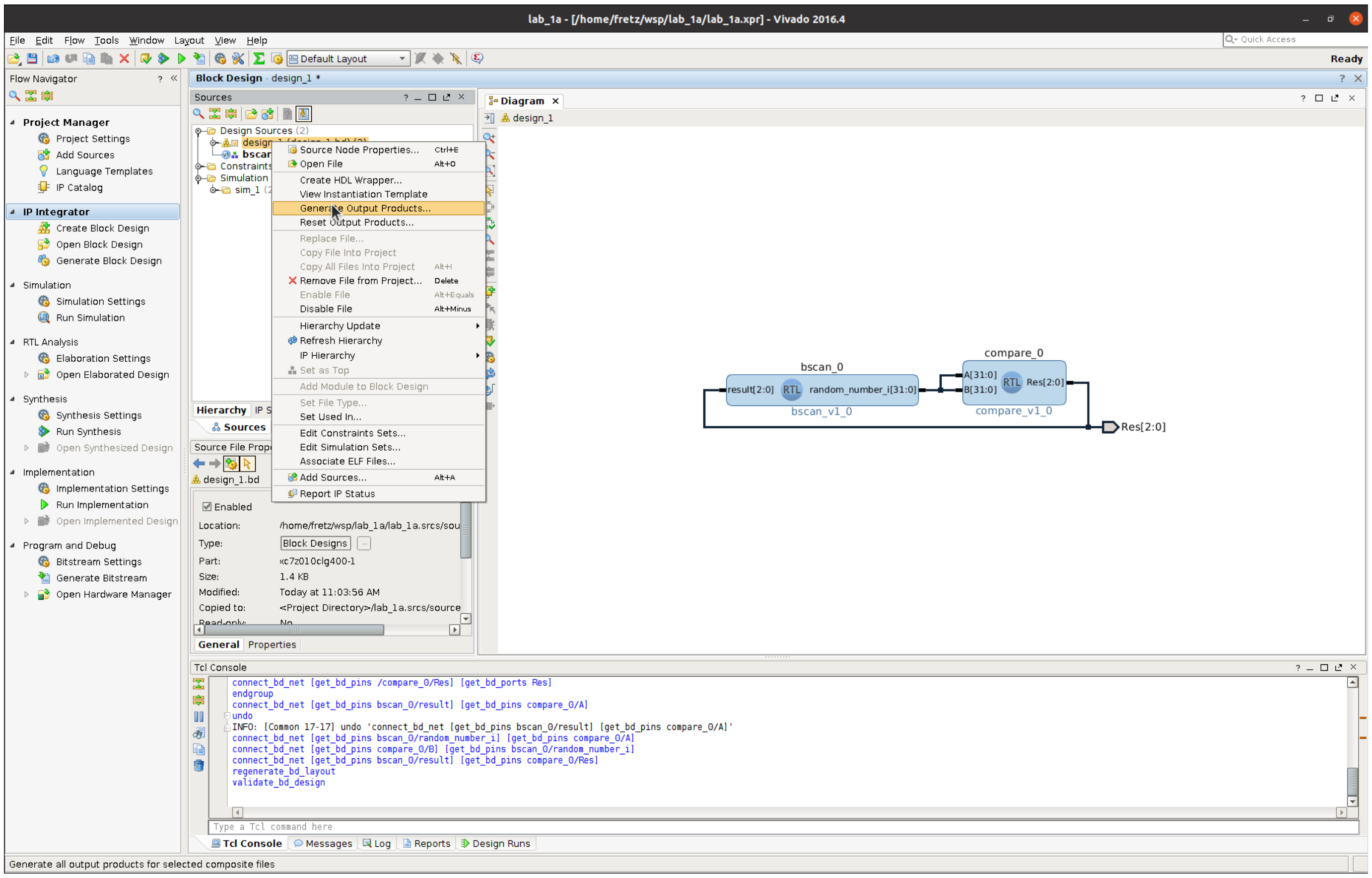
Validation successful. There are no errors or critical warnings in this design.

Now let's make a `top wrapper` for the block design. Choose the Sources -> See carefully the mouse cursor

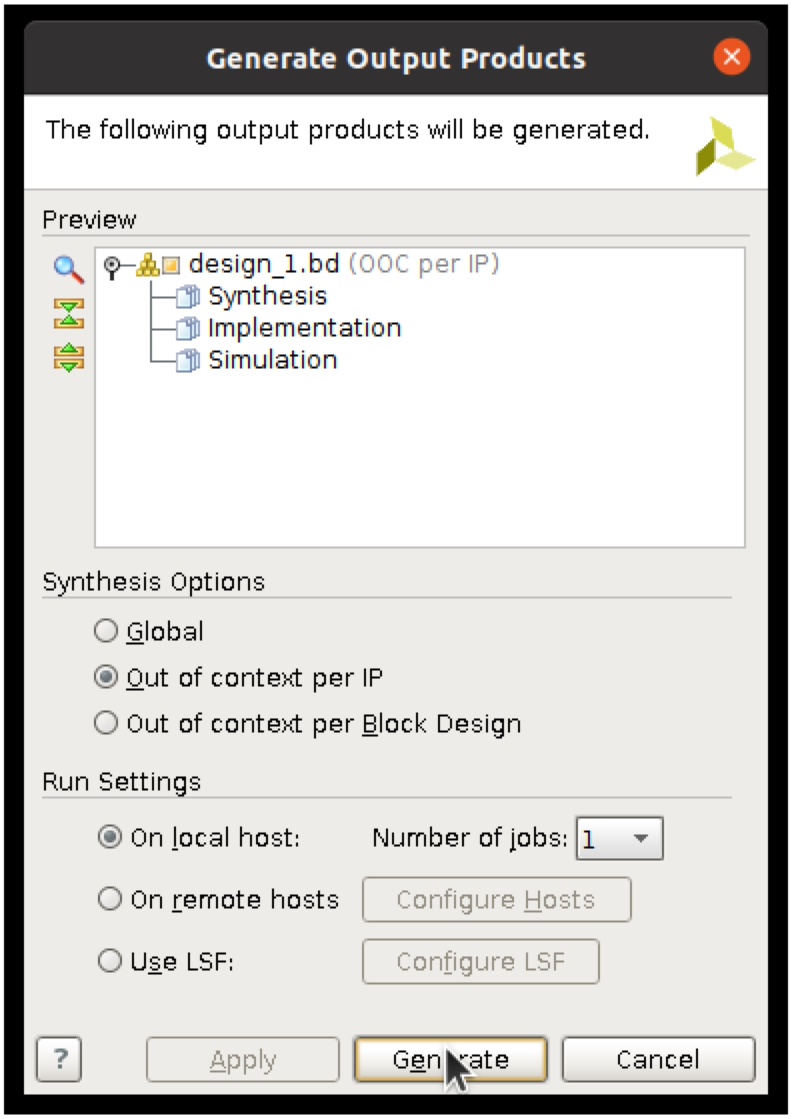


```
connect_bd_net [get_bd_pins /compare_0/Res] [get_bd_ports Res]
endgroup
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]'
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
regenerate_bd_layout
validate_bd_design
```

Press right-click on the design_1.bd file and press `Generate Output Products`


lab_1a - [/home/fretz/wsp/lab_1a/lab_1a.xpr] - Vivado 2016.4
Source Node Properties... Ctrl+E
Open File Alt+O
Create HDL Wrapper...
View Instantiation Template
Generate Output Products...
Reset Output Products...
Replace File...
Copy File Into Project
Copy All Files Into Project Alt+I
Remove File from Project... Delete
Enable File Alt+Equals
Disable File Alt+Minus
Hierarchy Update
Refresh Hierarchy
IP Hierarchy
Set as Top
Add Module to Block Design
Set File Type...
Set Used In...
Edit Constraints Sets...
Edit Simulation Sets...
Associate ELF Files...
Add Sources... Alt+A
Report IP Status
Source File Properties
design_1.bd
Enabled
Location: /home/fretz/wsp/lab_1a/lab_1a.srcs/sou
Type: Block Designs
Part: xc7z010clg400-1
Size: 1.4 KB
Modified: Today at 11:03:56 AM
Copied to: <Project Directory>/lab_1a.srcs/source
Generate all output products for selected composite files

```
connect_bd_net [get_bd_pins /compare_0/Res] [get_bd_ports Res]
endgroup
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]
undo
INFO: [Common 17-17] undo 'connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/A]'
connect_bd_net [get_bd_pins bscan_0/random_number_i] [get_bd_pins compare_0/A]
connect_bd_net [get_bd_pins compare_0/B] [get_bd_pins bscan_0/random_number_i]
connect_bd_net [get_bd_pins bscan_0/result] [get_bd_pins compare_0/Res]
regenerate_bd_layout
validate_bd_design
```



The following output products will be generated.

Preview

- design_1.bd (OOC per IP)
  - Synthesis
  - Implementation
  - Simulation

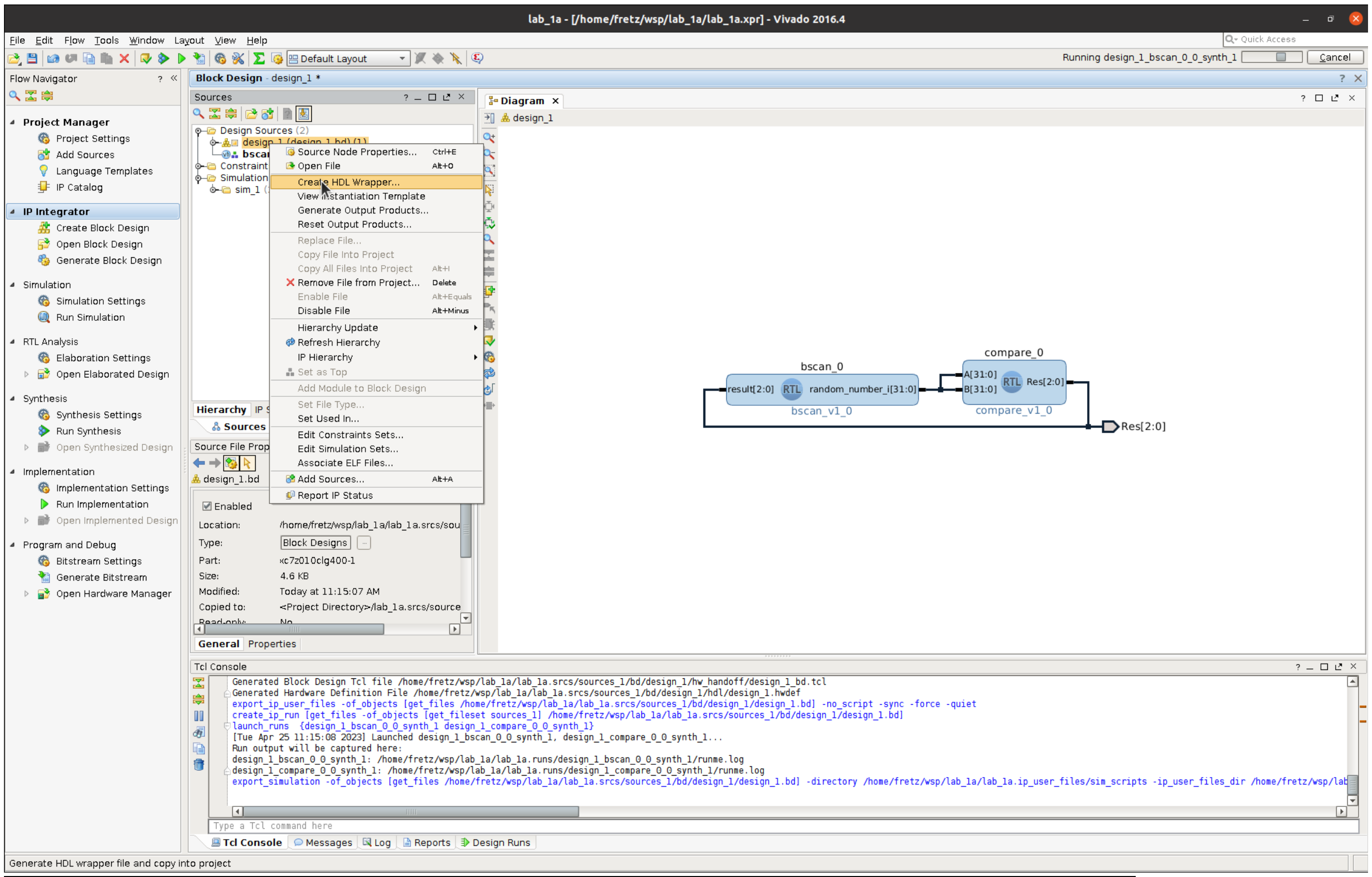
Synthesis Options

- ( ) Global
- (●) Out of context per IP
- ( ) Out of context per Block Design

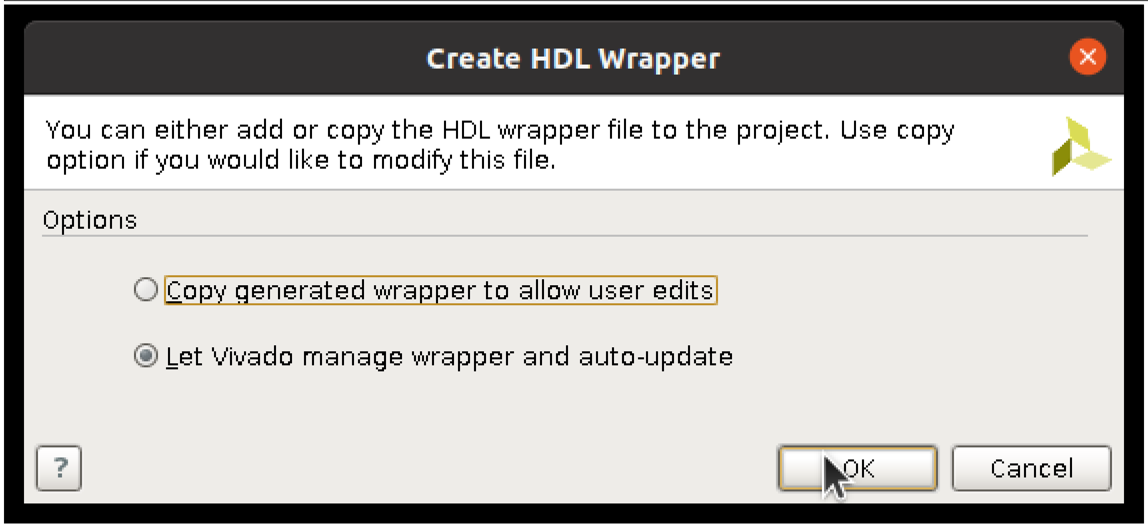
Run Settings

- (●) On local host: Number of jobs: 1
- ( ) On remote hosts
- ( ) Use LSF:

Press right-click on the design_1.bd file and press `Create HDL Wrapper...`



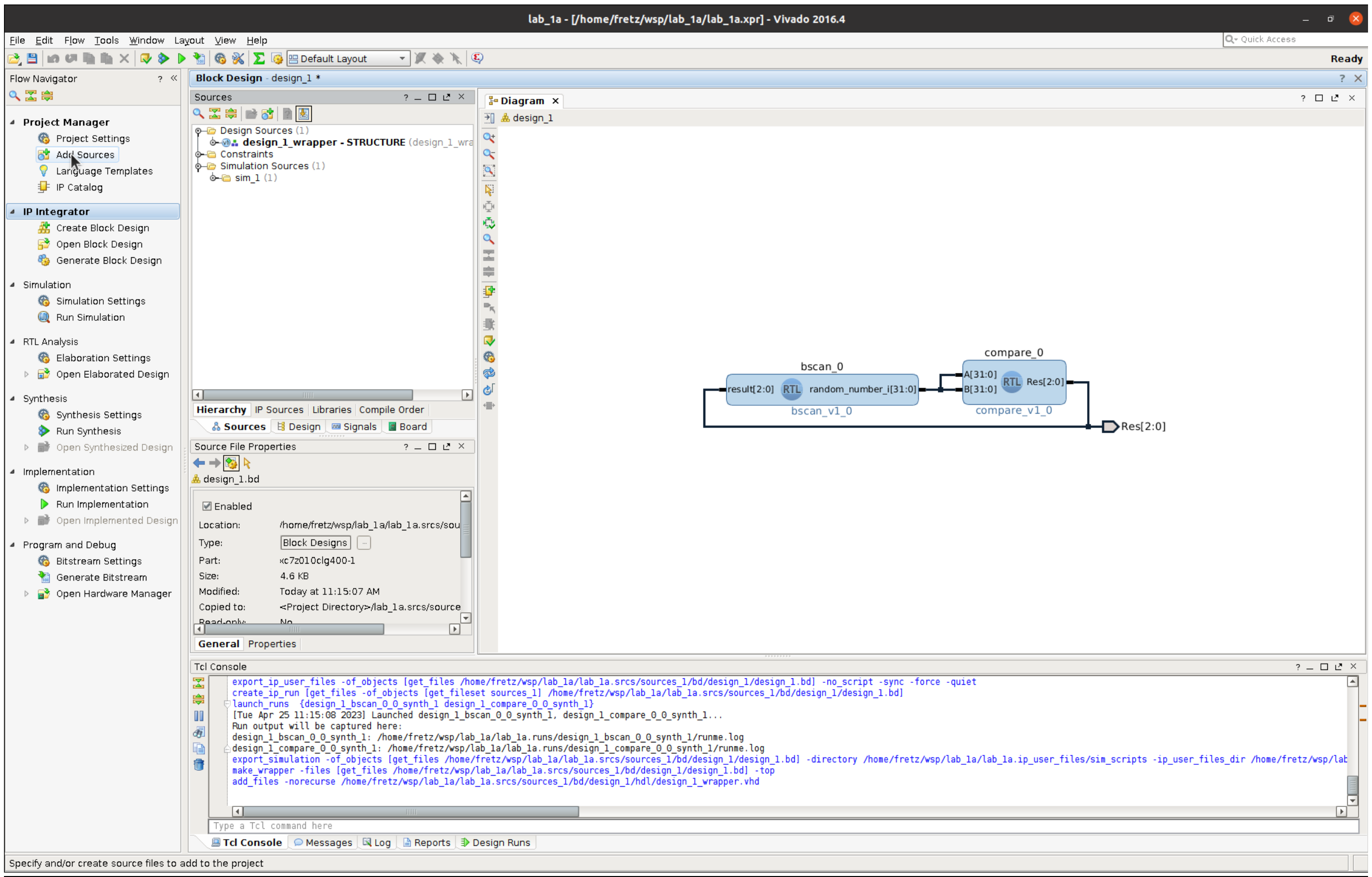
```
Generated Block Design Tcl file /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/hw_handoff/design_1_bd.tcl
Generated Hardware Definition File /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/hdl/design_1.hwdef
export_ip_user_files -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -no_script -sync -force -quiet
create_ip_run [get_files -of_objects [get_fileset sources_1] /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd]
launch_runs  {design_1_bscan_0_0_synth_1 design_1_compare_0_0_synth_1}
[Tue Apr 25 11:15:08 2023] Launched design_1_bscan_0_0_synth_1, design_1_compare_0_0_synth_1...
Run output will be captured here:
design_1_bscan_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_bscan_0_0_synth_1/runme.log
design_1_compare_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_compare_0_0_synth_1/runme.log
export_simulation -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -directory /home/fretz/wsp/lab_1a/lab_1a.ip_user_files/sim_scripts -ip_user_files_dir /home/fretz/wsp/lab
```



You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

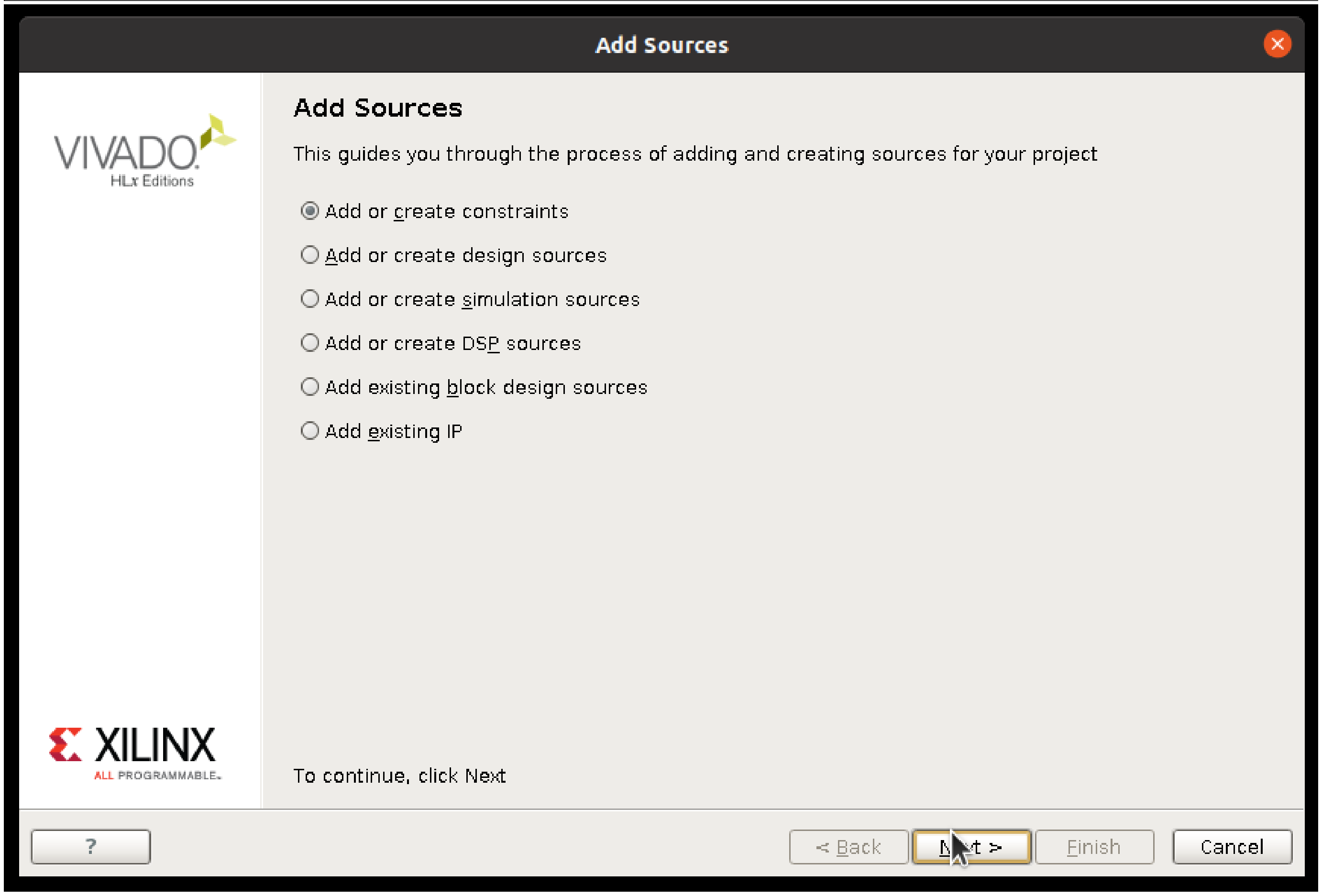
( ) Copy generated wrapper to allow user edits
[✓ Let Vivado manage wrapper and auto-update]

Now let's add the constraints file (the file which specifies which I/O ports of the Z7 board the DUT will use) to the project sources. Click the Add `Sources` button on Vivado's left menu.



Enabled
Location: /home/fretz/wsp/lab_1a/lab_1a.srcs/sou
Type: Block Designs
Part: xc7z010clg400-1
Size: 4.6 KB
Modified: Today at 11:15:07 AM
Copied to: <Project Directory>/lab_1a.srcs/source
Read-only: No

```
export_ip_user_files -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -no_script -sync -force -quiet
create_ip_run [get_files -of_objects [get_fileset sources_1] /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd]
launch_runs  {design_1_bscan_0_0_synth_1 design_1_compare_0_0_synth_1}
[Tue Apr 25 11:15:08 2023] Launched design_1_bscan_0_0_synth_1, design_1_compare_0_0_synth_1...
Run output will be captured here:
design_1_bscan_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_bscan_0_0_synth_1/runme.log
design_1_compare_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_compare_0_0_synth_1/runme.log
export_simulation -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -directory /home/fretz/wsp/lab_1a/lab_1a.ip_user_files/sim_scripts -ip_user_files_dir /home/fretz/wsp/lab
make_wrapper -files [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -top
add_files -norecurse /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/hdl/design_1_wrapper.vhd
```

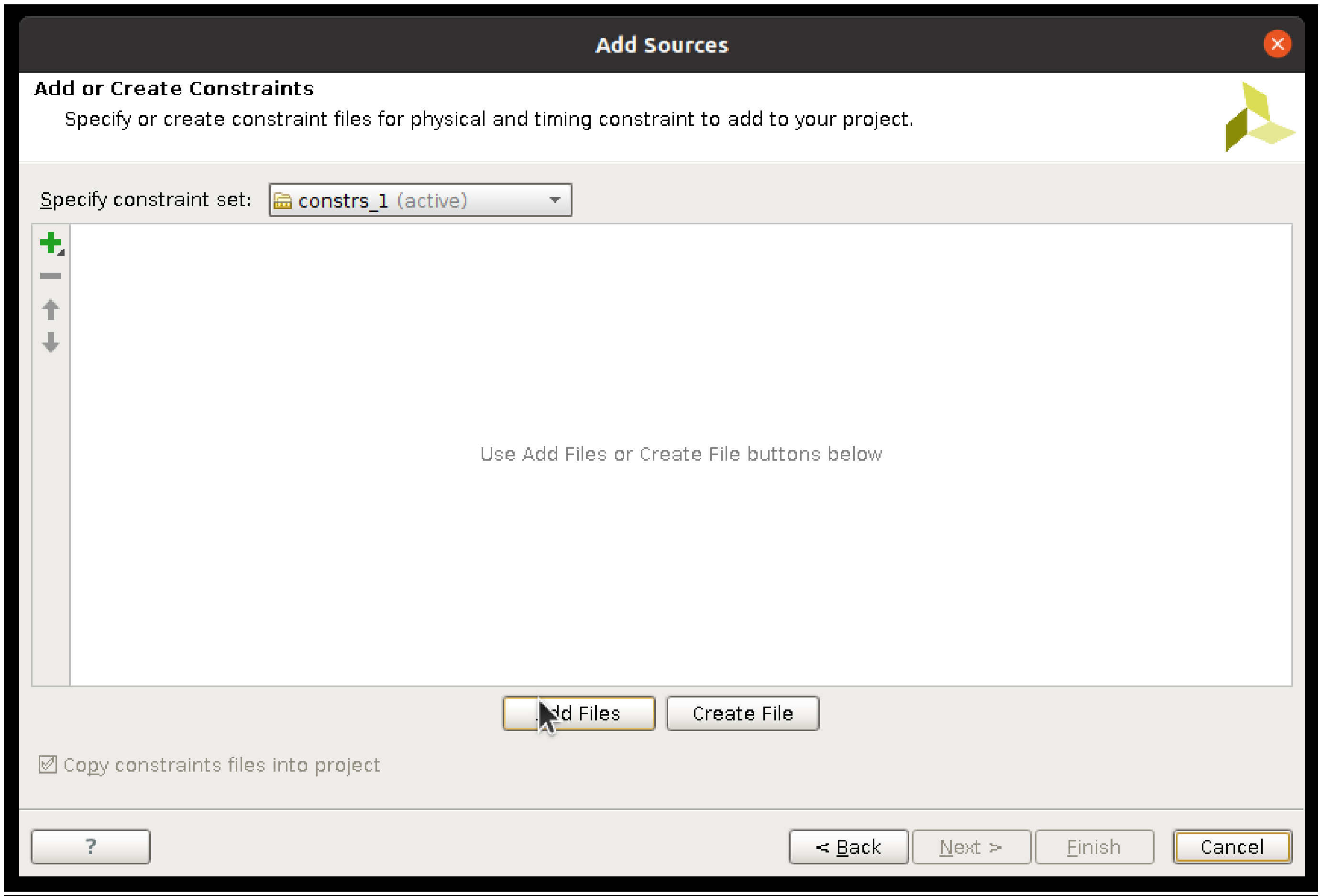


Add Sources

This guides you through the process of adding and creating sources for your project

[✓ Add or create constraints]
Add or create design sources
Add or create simulation sources
Add or create DSP sources
Add existing block design sources
Add existing IP

To continue, click Next



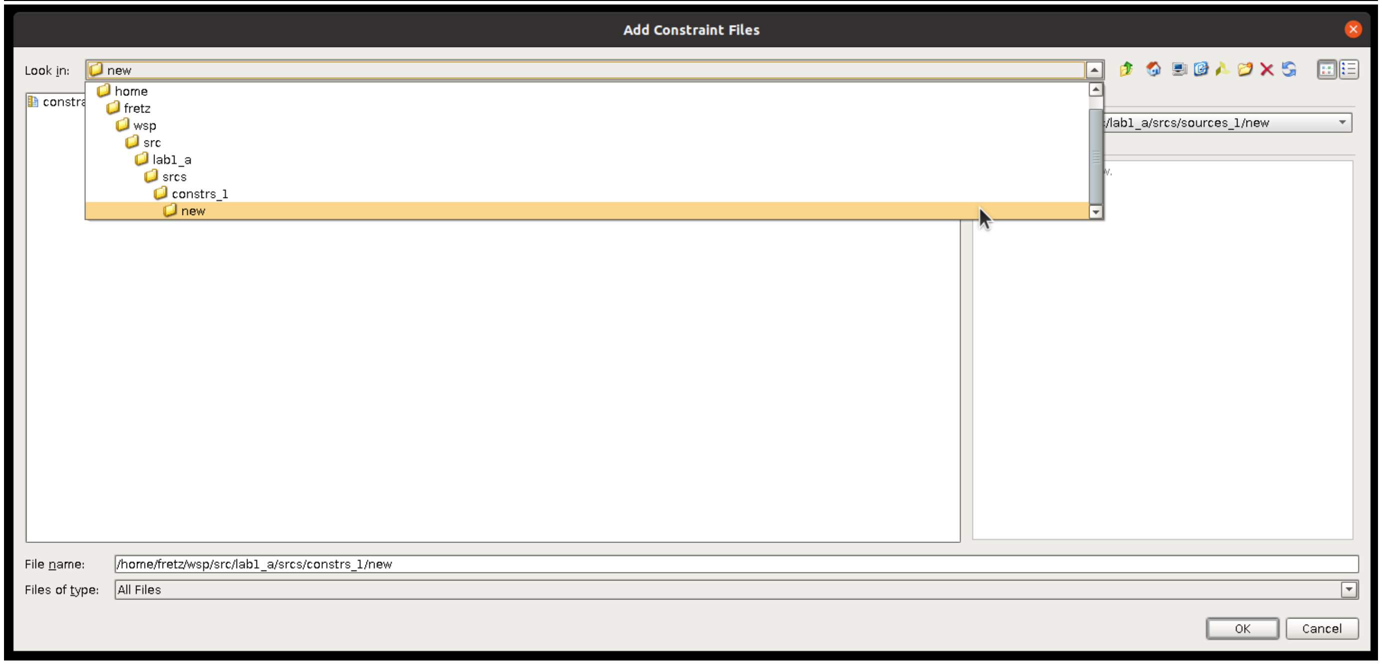
Add Sources

Add or Create Constraints

Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Use Add Files or Create File buttons below



Add Constraint Files

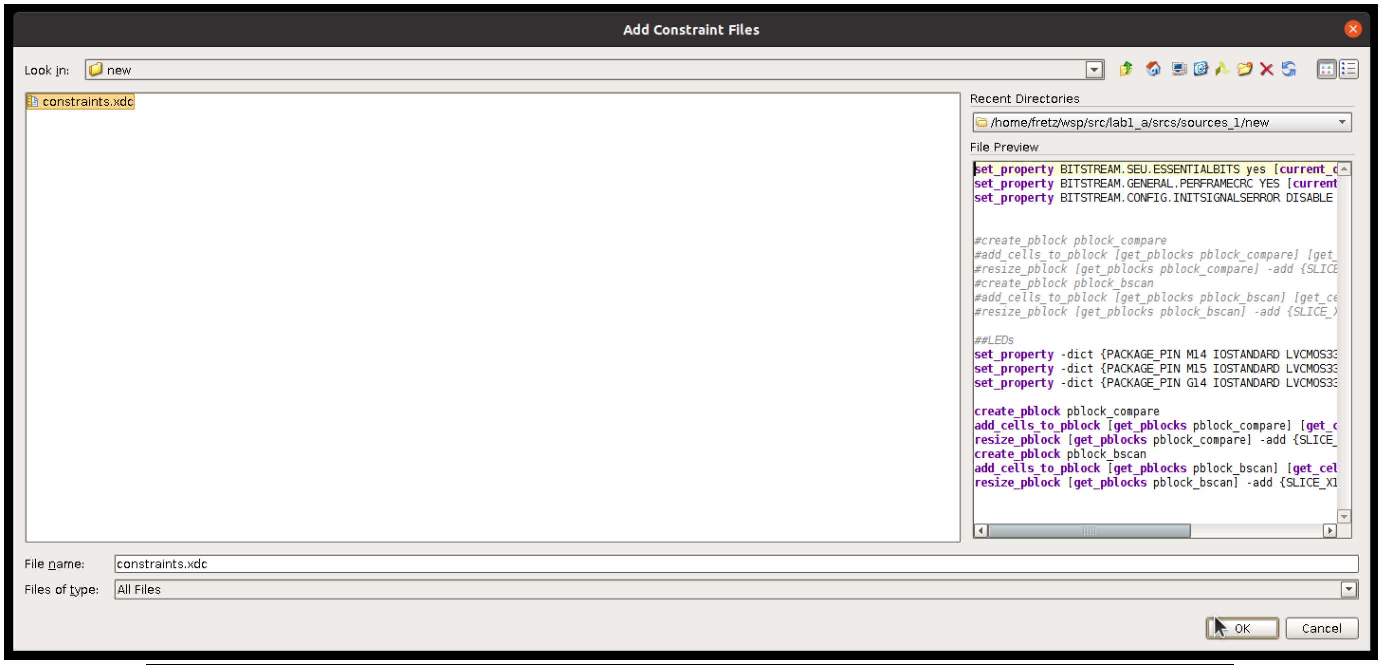
Look in: new

```
home
 fretz
  wsp
   src
    lab1_a
     srcs
      constrs_1
       new
```

/lab1_a/srcs/sources_1/new

File name: /home/fretz/wsp/src/lab1_a/srcs/constrs_1/new

Files of type: All Files



constraints.xdc

Recent Directories

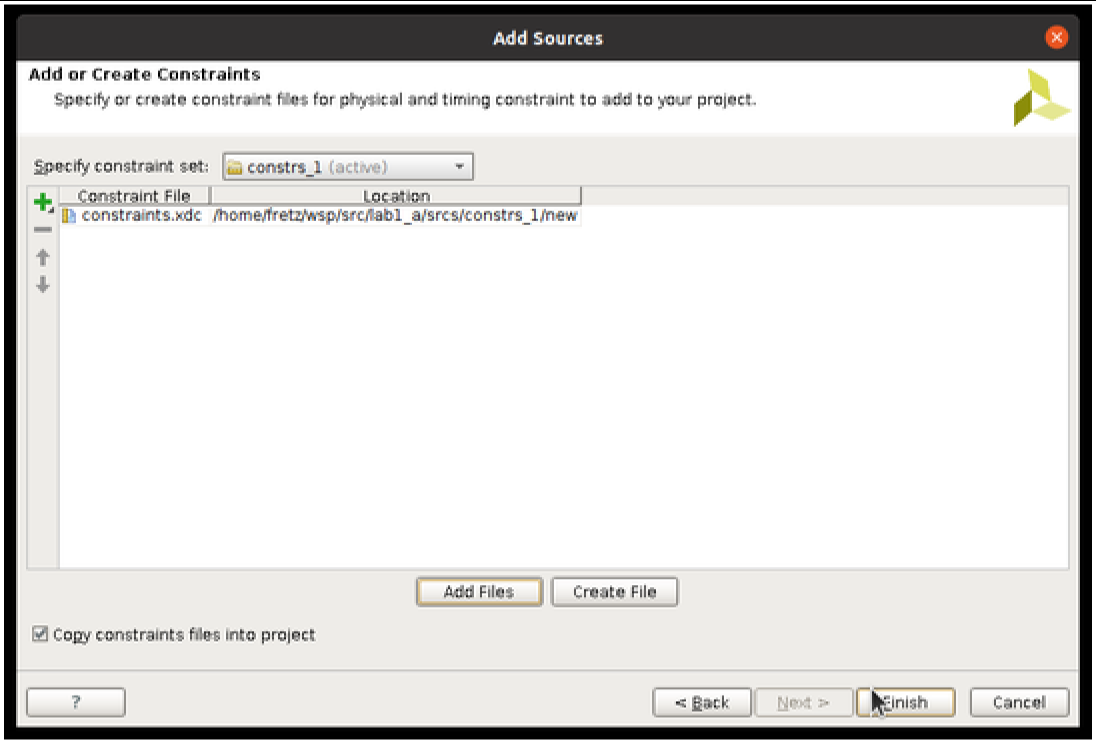
/home/fretz/wsp/src/lab1_a/srcs/sources_1/new

File Preview

```
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_d
set_property BITSTREAM.GENERAL.PERFRAMECRC YES [current
set_property BITSTREAM.CONFIG.INITSIGNALSERROR DISABLE


#create_pblock pblock_compare
#add_cells_to_pblock [get_pblocks pblock_compare] [get_
#resize_pblock [get_pblocks pblock_compare] -add {SLICE
#create_pblock pblock_bscan
#add_cells_to_pblock [get_pblocks pblock_bscan] [get_ce
#resize_pblock [get_pblocks pblock_bscan] -add {SLICE_X

##LEDs
set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33
set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33
set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33

create_pblock pblock_compare
add_cells_to_pblock [get_pblocks pblock_compare] [get_c
resize_pblock [get_pblocks pblock_compare] -add {SLICE_
create_pblock pblock_bscan
add_cells_to_pblock [get_pblocks pblock_bscan] [get_cel
resize_pblock [get_pblocks pblock_bscan] -add {SLICE_X1
```

File name: constraints.xdc

Files of type: All Files



Add or Create Constraints

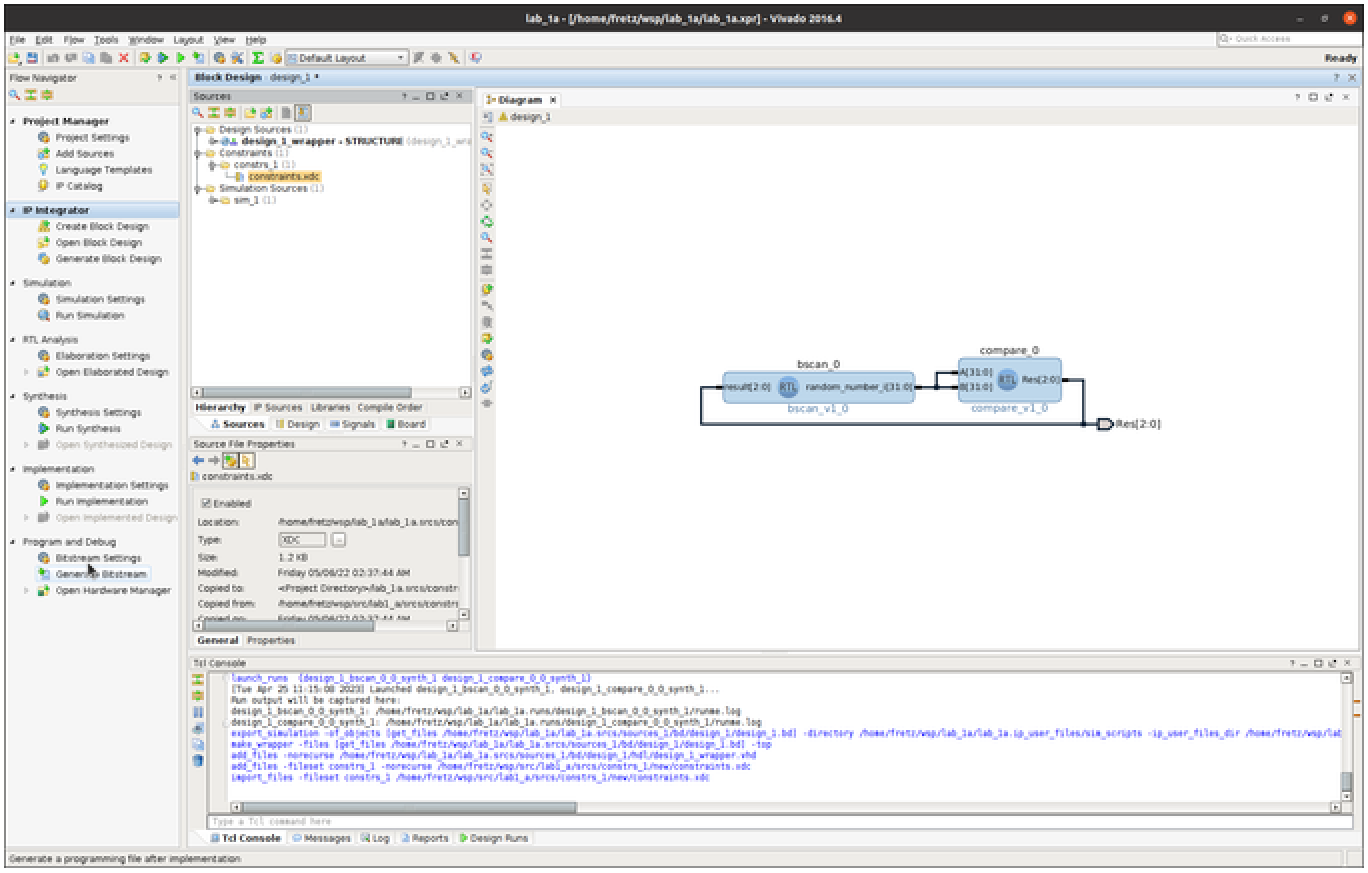
Specify or create constraint files for physical and timing constraint to add to your project.

Specify constraint set: constrs_1 (active)

Constraint File	Location
constraints.xdc	/home/fretz/wsp/src/lab1_a/srcs/constrs_1/new

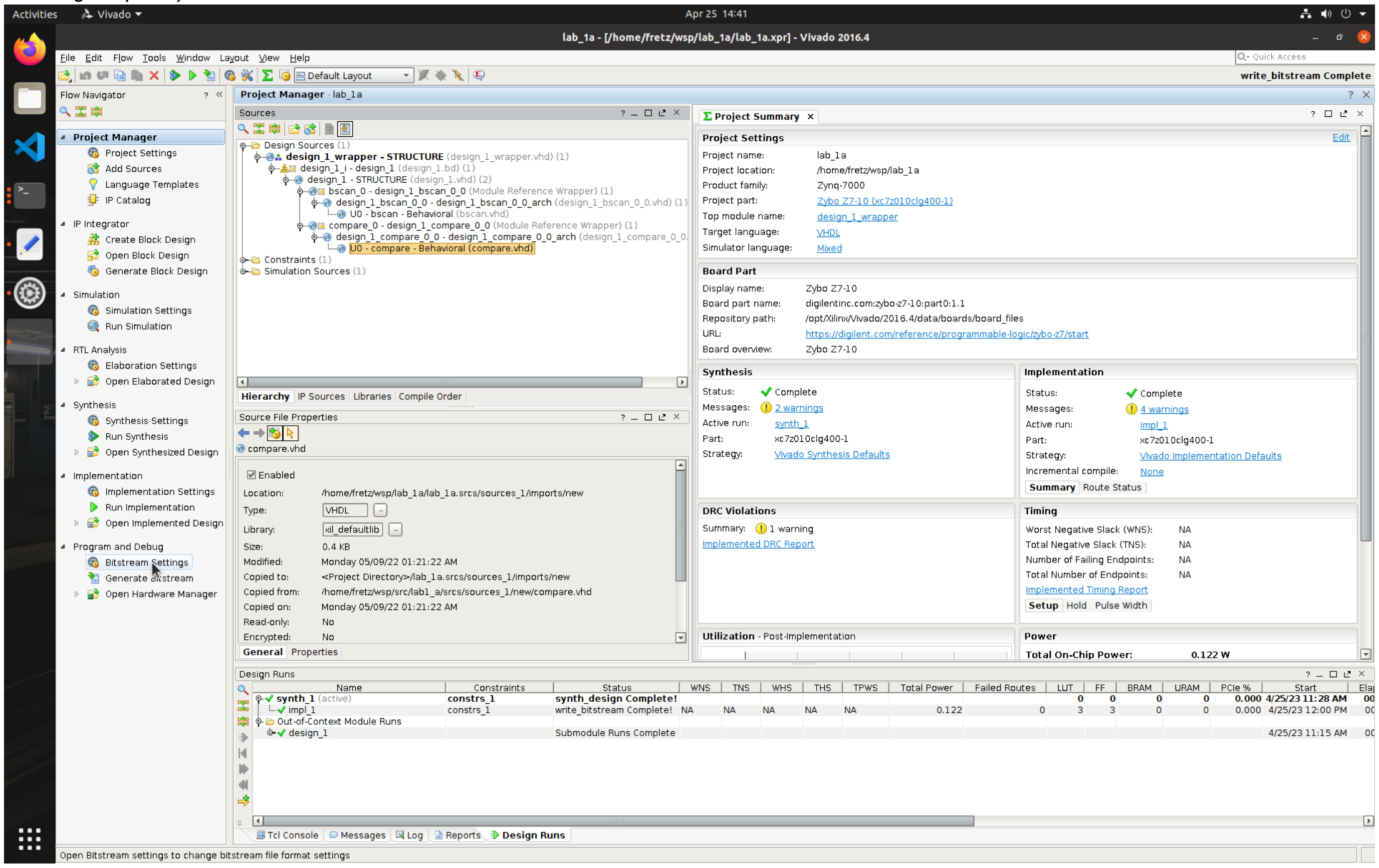
☑ Copy constraints files into project

After clicking the `Finish` button, you will see the `constraints.xdc` added to your source files.



```
launch_runs  {design_1_bscan_0_0_synth_1 design_1_compare_0_0_synth_1}
[Tue Apr 25 11:15:08 2023] Launched design_1_bscan_0_0_synth_1, design_1_compare_0_0_synth_1...
Run output will be captured here:
design_1_bscan_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_bscan_0_0_synth_1/runme.log
design_1_compare_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_compare_0_0_synth_1/runme.log
export_simulation -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -directory /home/fretz/wsp/lab_1a/lab_1a.ip_user_files/sim_scripts -ip_user_files_dir /home/fretz/wsp/lab
make_wrapper -files [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -top
add_files -norecurse /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/hdl/design_1_wrapper.vhd
add_files -fileset constrs_1 -norecurse /home/fretz/wsp/src/lab1_a/srcs/constrs_1/new/constraints.xdc
import_files -fileset constrs_1 /home/fretz/wsp/src/lab1_a/srcs/constrs_1/new/constraints.xdc
```

In order to perform fault injection, we need to make a few changes in the bitstream settings. Click on the bitstream settings (see the cursor in the following snapshot)



Project Settings
Project name: lab_1a
Project location: /home/fretz/wsp/lab_1a
Product family: Zynq-7000
Project part: Zybo Z7-10 (xc7z010clg400-1)
Top module name: design_1_wrapper
Target language: VHDL
Simulator language: Mixed

Board Part
Display name: Zybo Z7-10
Board part name: digilentinc.com:zybo-z7-10:part0:1.1
Repository path: /opt/Xilinx/Vivado/2016.4/data/boards/board_files
URL: https://digilent.com/reference/programmable-logic/zybo-z7/start
Board overview: Zybo Z7-10

Synthesis
Status: Complete
Messages: 2 warnings
Active run: synth_1
Part: xc7z010clg400-1
Strategy: Vivado Synthesis Defaults

Implementation
Status: Complete
Messages: 4 warnings
Active run: impl_1
Part: xc7z010clg400-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations
Summary: 1 warning

Timing
Worst Negative Slack (WNS): NA
Total Negative Slack (TNS): NA
Number of Failing Endpoints: NA
Total Number of Endpoints: NA

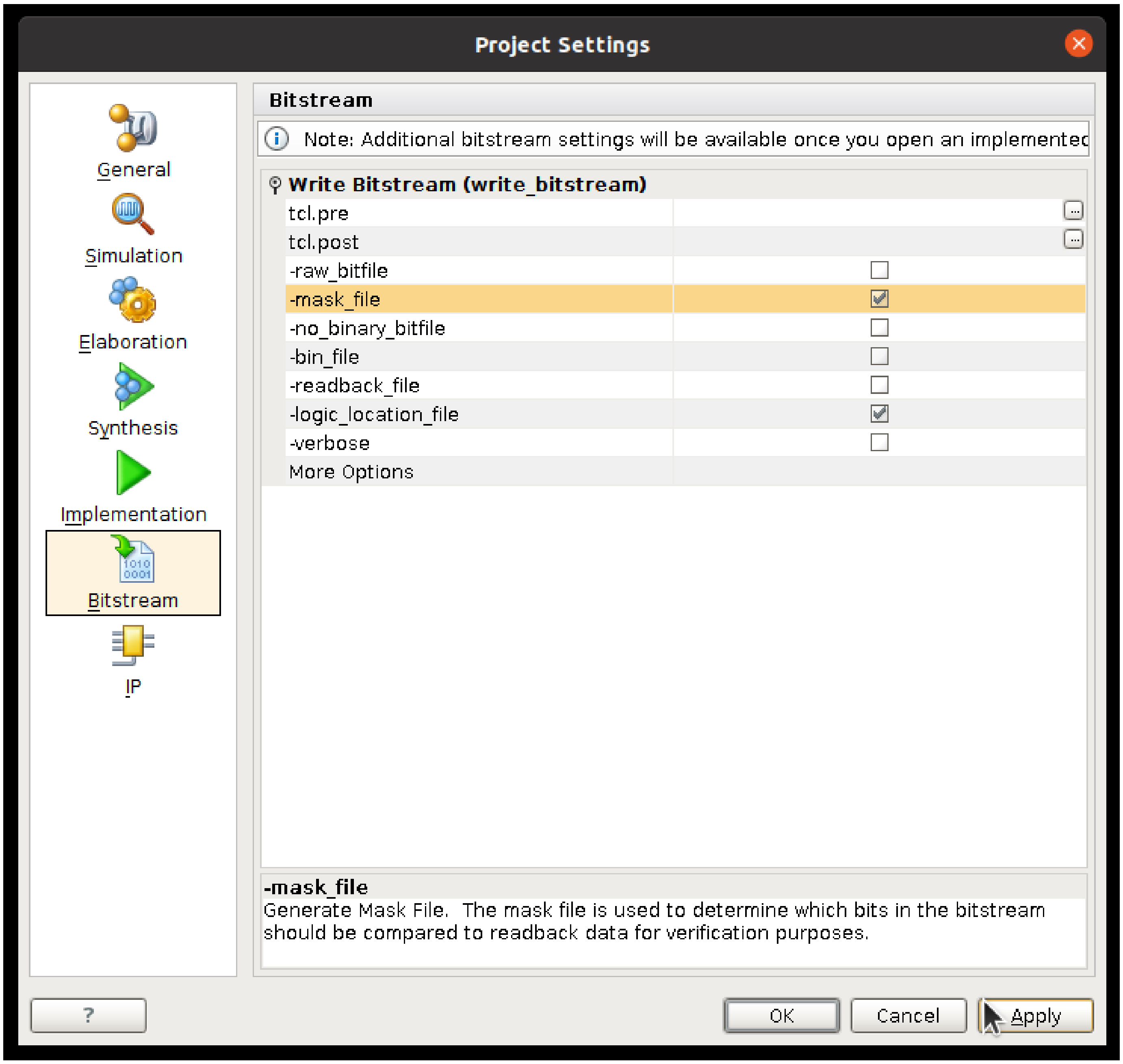
Power
Total On-Chip Power: 0.122 W

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIe %	Start
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0	0	0.000	4/25/23 11:28 AM
impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA	NA	0.122	0	3	3	0	0	0.000	4/25/23 12:00 PM
Out-of-Context Module Runs															
design_1		Submodule Runs Complete													4/25/23 11:15 AM

Open Bitstream settings to change bitstream file format settings

Tick the `mask file` and `logic_location_file`flags and press apply



Bitstream

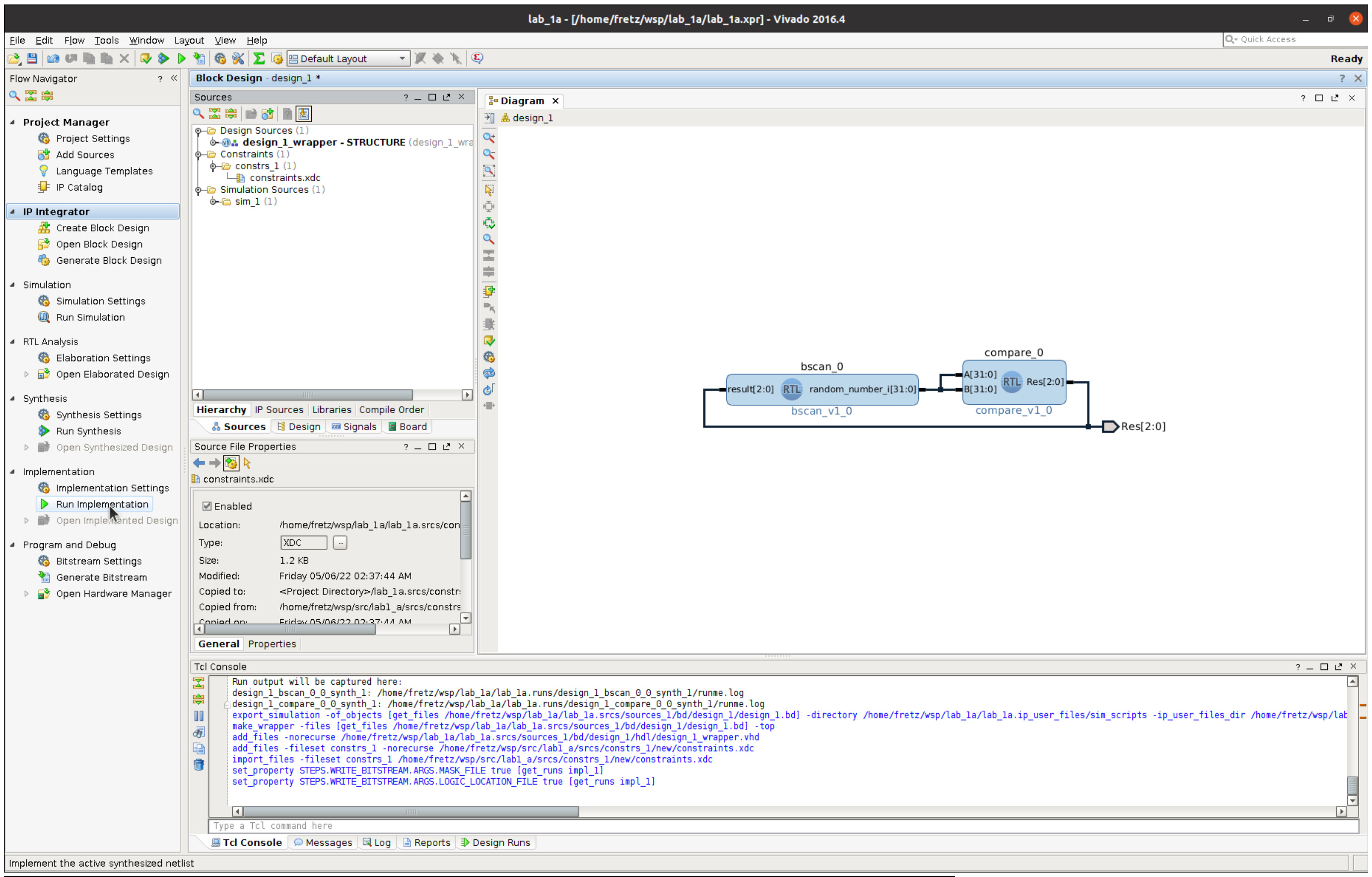
Note: Additional bitstream settings will be available once you open an implementec

Write Bitstream (write_bitstream)

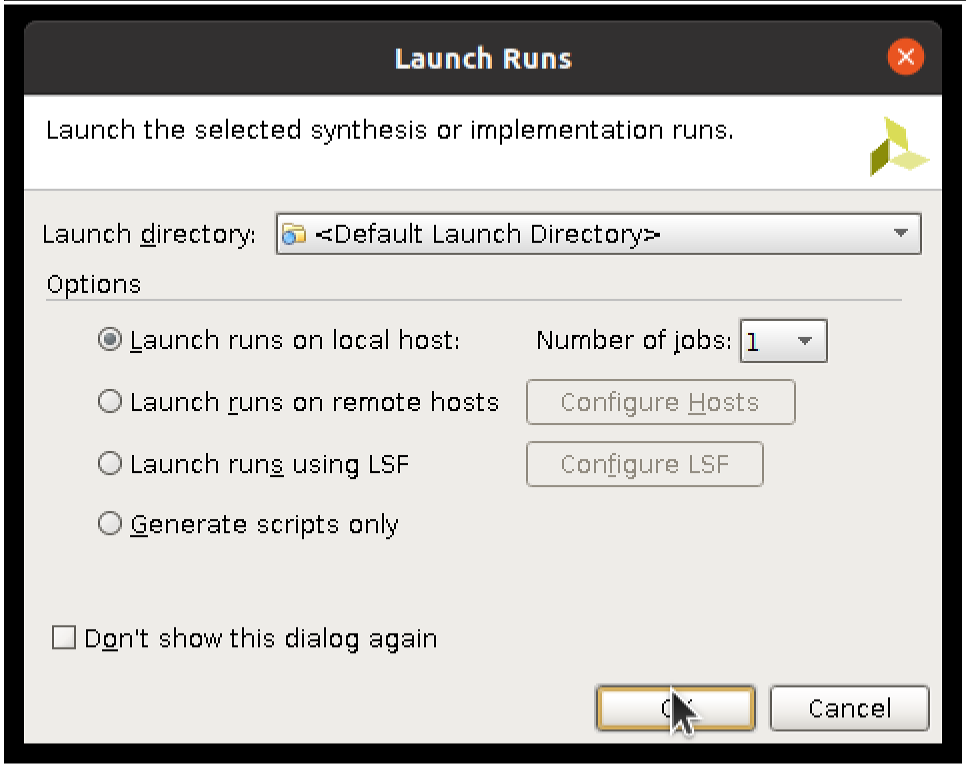
tcl.pre	
tcl.post	
-raw_bitfile	☐
-mask_file	✓
-no_binary_bitfile	☐
-bin_file	☐
-readback_file	☐
-logic_location_file	✓
-verbose	☐
More Options	

-mask_file
Generate Mask File. The mask file is used to determine which bits in the bitstream should be compared to readback data for verification purposes.

We are ready to implement the design! Click the `Run implementation` button



```
Run output will be captured here:
design_1_bscan_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_bscan_0_0_synth_1/runme.log
design_1_compare_0_0_synth_1: /home/fretz/wsp/lab_1a/lab_1a.runs/design_1_compare_0_0_synth_1/runme.log
export_simulation -of_objects [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -directory /home/fretz/wsp/lab_1a/lab_1a.ip_user_files/sim_scripts -ip_user_files_dir /home/fretz/wsp/lab
make_wrapper -files [get_files /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/design_1.bd] -top
add_files -norecurse /home/fretz/wsp/lab_1a/lab_1a.srcs/sources_1/bd/design_1/hdl/design_1_wrapper.vhd
add_files -fileset constrs_1 -norecurse /home/fretz/wsp/src/lab1_a/srcs/constrs_1/new/constraints.xdc
import_files -fileset constrs_1 /home/fretz/wsp/src/lab1_a/srcs/constrs_1/new/constraints.xdc
set_property STEPS.WRITE_BITSTREAM.ARGS.MASK_FILE true [get_runs impl_1]
set_property STEPS.WRITE_BITSTREAM.ARGS.LOGIC_LOCATION_FILE true [get_runs impl_1]
```


Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 1

Launch runs on remote hosts — Configure Hosts

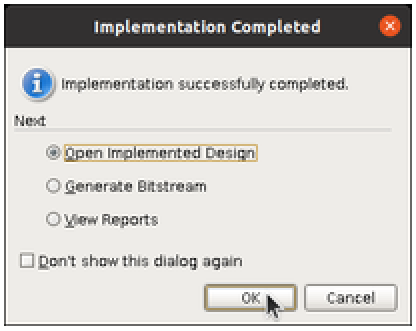
Launch runs using LSF — Configure LSF

Generate scripts only

☐ Don't show this dialog again

OK Cancel

When the design is implemented, open it to see where the DUT is placed in the FPGA



Implementation Completed

Implementation successfully completed.

Next

[✓ Open Implemented Design]

Generate Bitstream

View Reports

Don't show this dialog again

This is achieved via floorplanning. Floorplanning is a stage in the physical design process of an integrated circuit (IC) or Field-Programmable Gate Array (FPGA) that involves assigning and positioning the various functional blocks or components of the design on the chip or FPGA die.

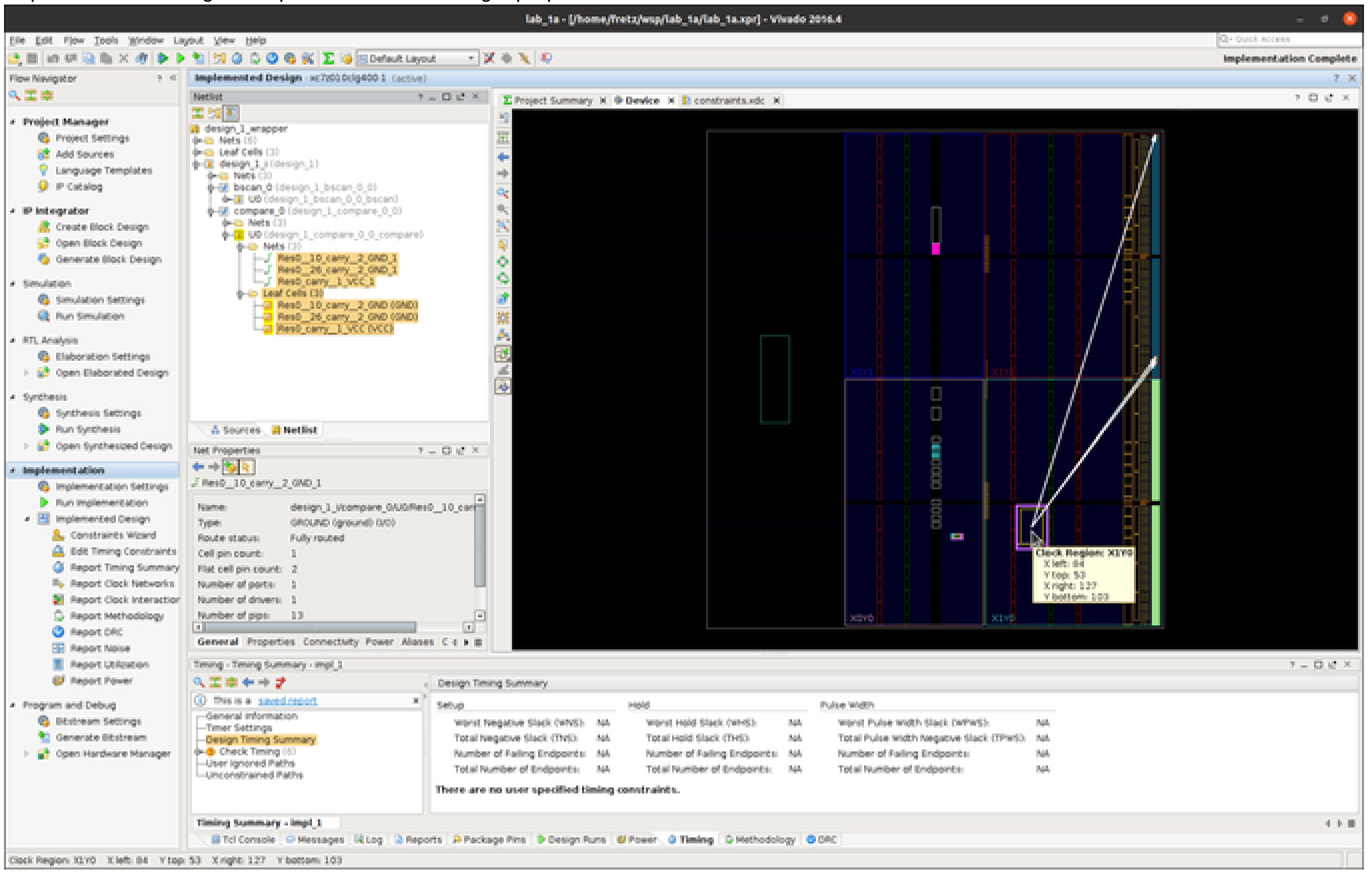
Xilinx floorplanning is a feature in the Xilinx ISE or Vivado design tools that allows designers to define the physical layout of the FPGA design, including placement of components, routing of signals, and optimization of timing and power.

In Xilinx floorplanning, designers use a graphical interface to specify the location and placement of the various components and sub-blocks within the FPGA. They can also use floorplanning tools to optimize the physical design for performance, power consumption, and area usage.

Floorplanning can significantly impact the overall performance and efficiency of an FPGA design. By carefully arranging the components and sub-blocks on the FPGA die, designers can reduce signal delays, minimize power consumption, and improve overall system performance.

Xilinx floorplanning also allows designers to perform advanced functions such as pin placement, placement constraints, physical design rule checking, and power optimization. These features enable designers to optimize their FPGA designs for specific applications and requirements.

See that the COMPARATOR has been implemented in the pink rectangular area shown below. There is also a small ping rectangular area for BSCAN module. We do this in order to inject faults only in the DUT, which is the COMPARATOR. The BSCAN is our test vehicle, and we need to keep it fault-free during our experiments in order to get proper results.



Name:	design_1_i/compare_0/U0/Res0__10_carr
Type:	GROUND (ground) (I/O)
Route status:	Fully routed
Cell pin count:	1
Flat cell pin count:	2
Number of ports:	1
Number of drivers:	1
Number of pips:	13

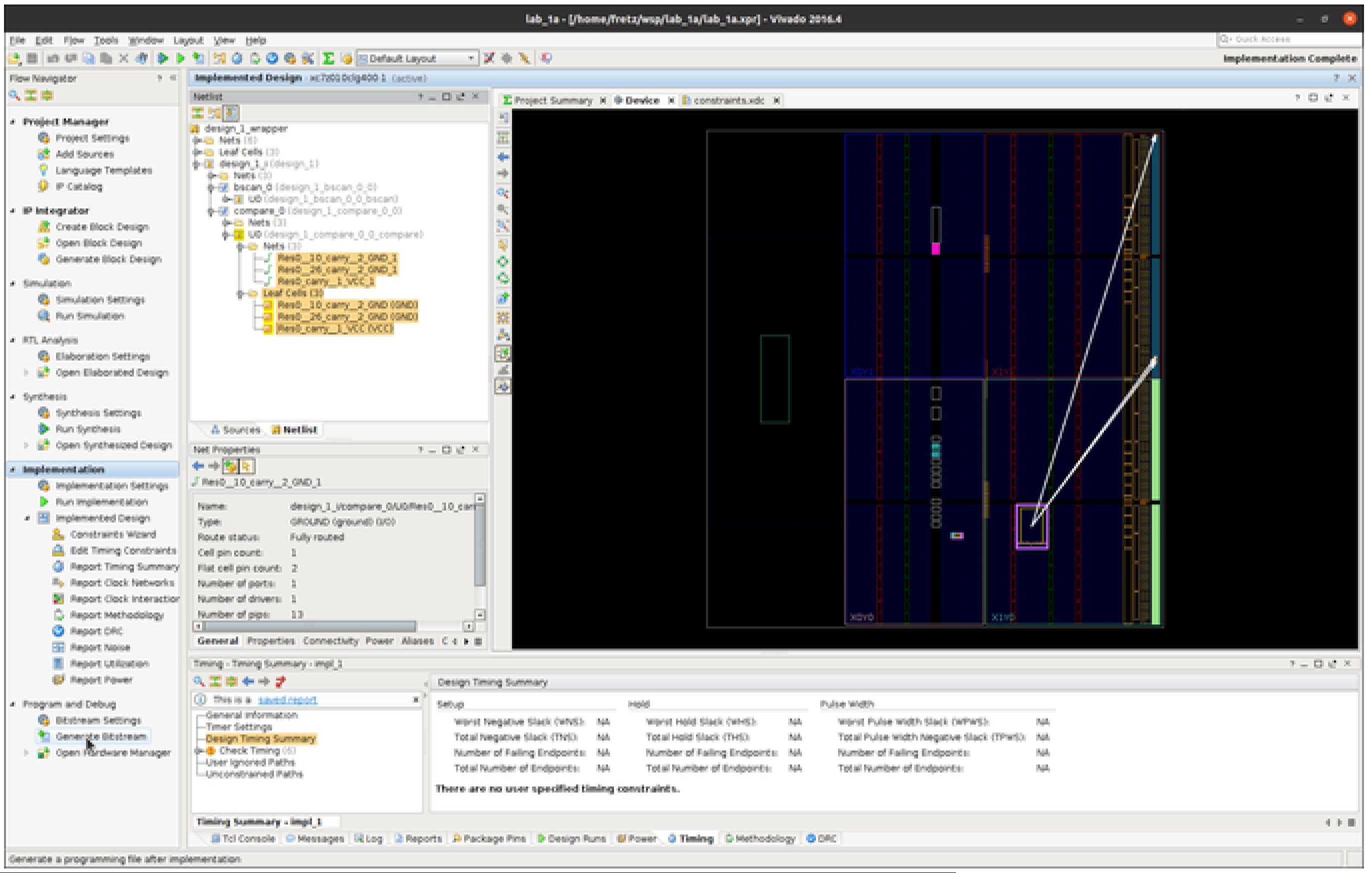
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	NA	Worst Hold Slack (WHS):	NA	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	NA	Total Hold Slack (THS):	NA	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	NA	Number of Failing Endpoints:	NA	Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA	Total Number of Endpoints:	NA	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Clock Region: X1Y0 X left: 84 Y top: 53 X right: 127 Y bottom: 103

Now we are ready to generate the bitstream of the design. Click the `Generate Bitstream`button


lab_1a - [/home/fretz/wsp/lab_1a/lab_1a.xpr] - Vivado 2016.4

Net Properties: Res0_10_carry_2_GND_1

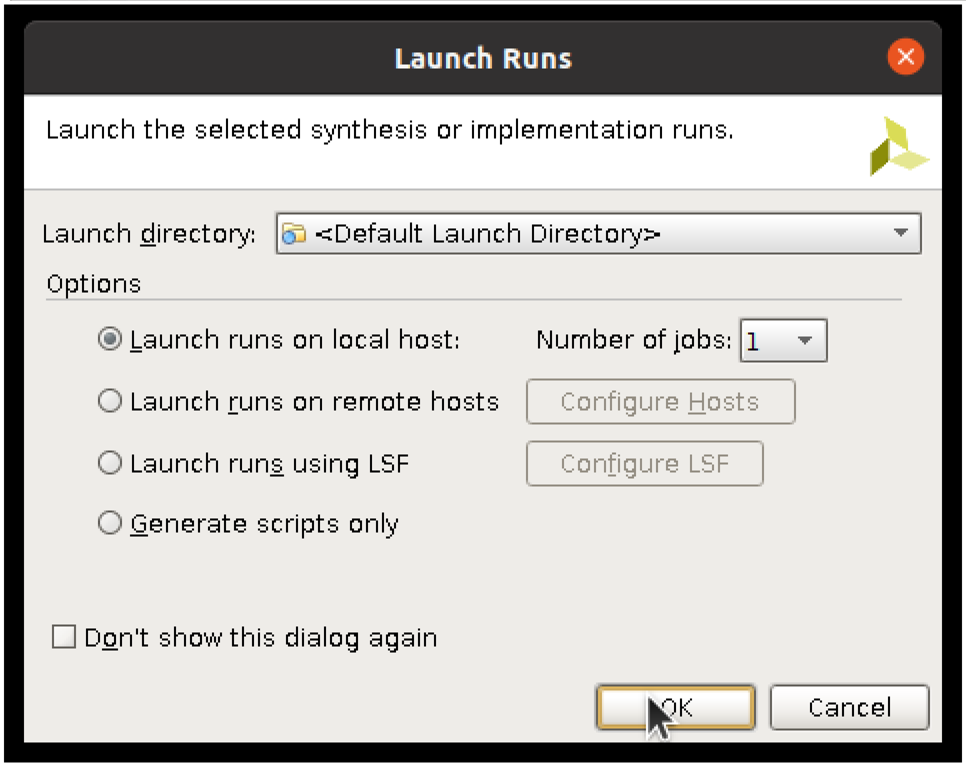
Name:	design_1_i/compare_0/U0/Res0_10_carr
Type:	GROUND (ground) (1/0)
Route status:	Fully routed
Cell pin count:	1
Flat cell pin count:	2
Number of ports:	1
Number of drivers:	1
Number of pips:	13

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	NA	Worst Hold Slack (WHS):	NA	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	NA	Total Hold Slack (THS):	NA	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	NA	Number of Failing Endpoints:	NA	Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA	Total Number of Endpoints:	NA	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Generate a programming file after implementation


Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 1

Launch runs on remote hosts — Configure Hosts

Launch runs using LSF — Configure LSF

Generate scripts only

Don't show this dialog again

OK Cancel

Section 3: Developing the algorithm to inject faults into the DUT with FRETZ

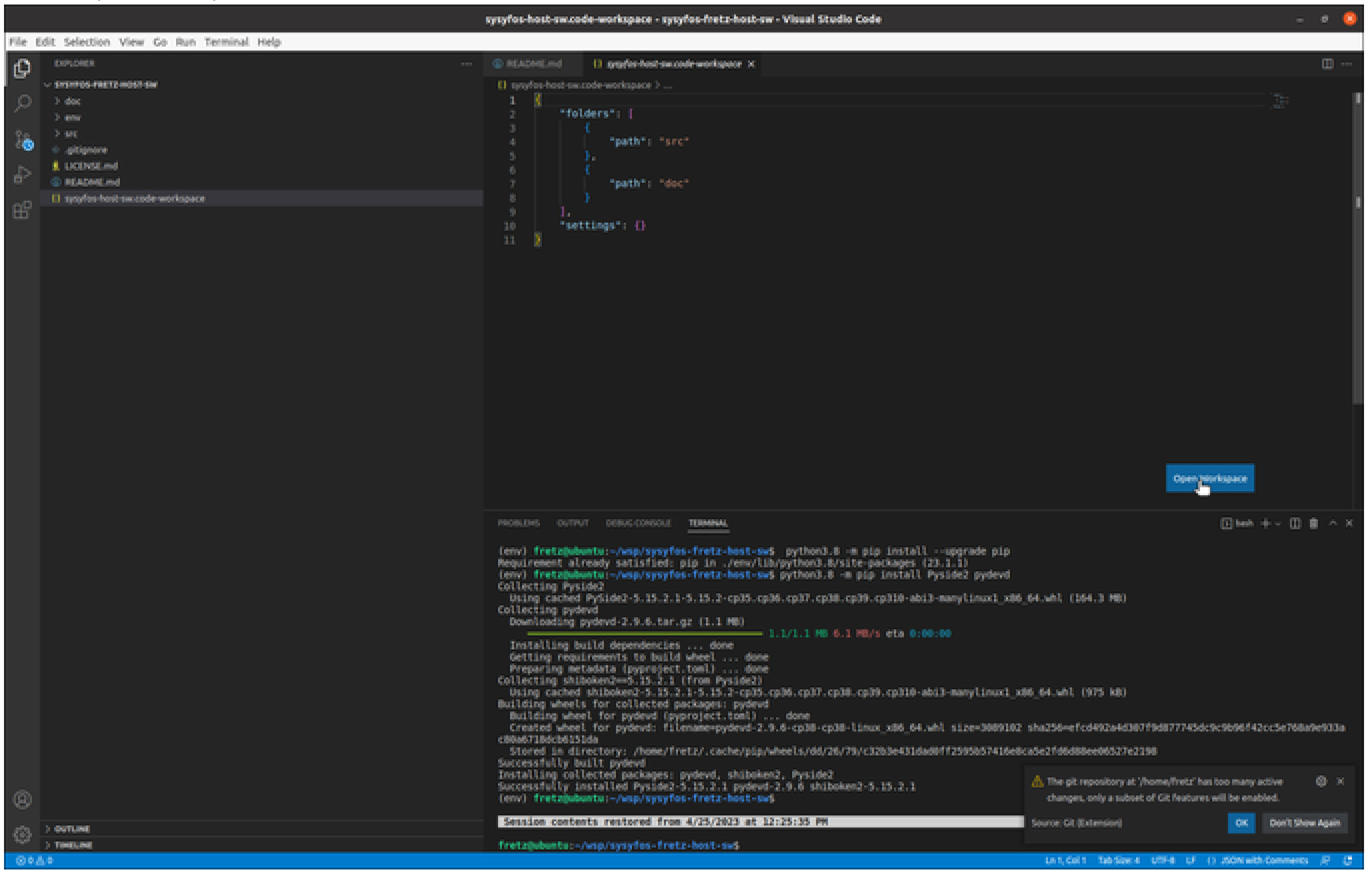
We will use https://github.com/unipieslab/FREtZ (https://osda.gitlab.io/19/3.2.pdf) to inject faults into the FPGA.

FREtZ provides a rich set of high-level Python APIs and application examples to readback, verify and manipulate the bitstream and the device state of all AMD 7-series and UltraScale/UltraScale+ MPSoC/FPGAs. Specifically, FREtZ increases the productivity of performing fault-injection and radiation experiments by hiding low-level Vivado TCL/JTAG commands that are executed behind the scenes to access the PS and PL memories of the target device.

1) Open a terminal

```
cd wsp/sysyfos-fretz-host-sw/
source env/bin/activate
code .
```

Click the `Open Workspace`



```
{
    "folders": [
        {
            "path": "src"
        },
        {
            "path": "doc"
        }
    ],
    "settings": {}
}
```

```
(env) fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$  python3.8 -m pip install --upgrade pip
Requirement already satisfied: pip in ./env/lib/python3.8/site-packages (23.1.1)
(env) fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ python3.8 -m pip install Pyside2 pydevd
Collecting Pyside2
  Using cached PySide2-5.15.2.1-5.15.2-cp35.cp36.cp37.cp38.cp39.cp310-abi3-manylinux1_x86_64.whl (164.3 MB)
Collecting pydevd
  Downloading pydevd-2.9.6.tar.gz (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 6.1 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting shiboken2==5.15.2.1 (from Pyside2)
  Using cached shiboken2-5.15.2.1-5.15.2-cp35.cp36.cp37.cp38.cp39.cp310-abi3-manylinux1_x86_64.whl (975 kB)
Building wheels for collected packages: pydevd
  Building wheel for pydevd (pyproject.toml) ... done
  Created wheel for pydevd: filename=pydevd-2.9.6-cp38-cp38-linux_x86_64.whl size=3089102 sha256=efcd492a4d307f9d877745dc9c9b96f42cc5e768a9e933a
c80a6718dcb6151da
  Stored in directory: /home/fretz/.cache/pip/wheels/dd/26/79/c32b3e431da80ff25958b57416e8ca5e2fd6d88ee06527e2198
Successfully built pydevd
Installing collected packages: pydevd, shiboken2, Pyside2
Successfully installed Pyside2-5.15.2.1 pydevd-2.9.6 shiboken2-5.15.2.1
(env) fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$

 Session contents restored from 4/25/2023 at 12:25:35 PM

fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$
```

The git repository at '/home/fretz' has too many active changes, only a subset of Git features will be enabled.

Source: Git (Extension)

FREtZ has many classes and methods, but a user must describe the fault injection procedure in the Python file `UI/UserApplication.py`. Feel free to check the current fault injection scenario described in the `UI/UserApplication.py`

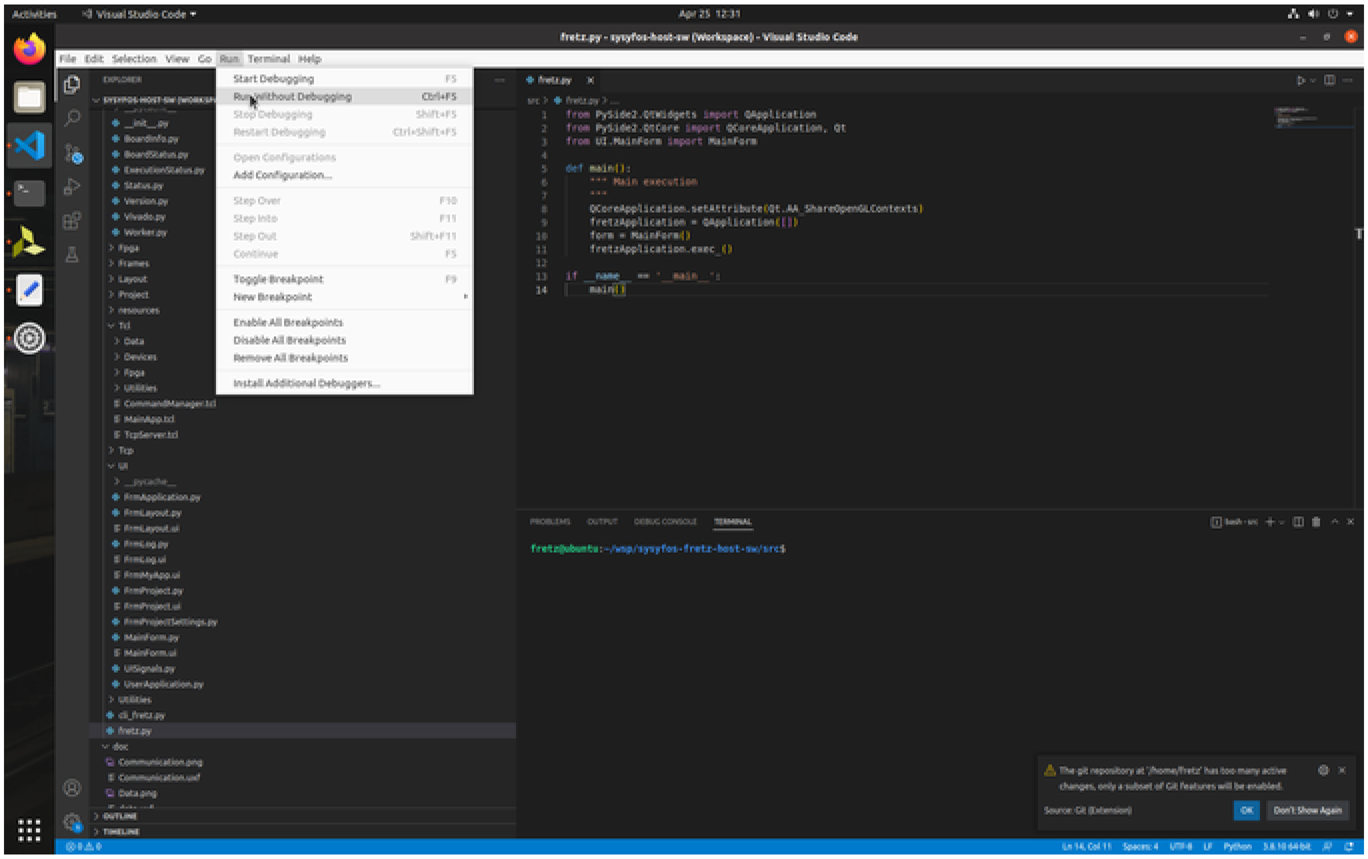
In the following, we provide basic functions to perform fault injection:

```
def ConfigureDevice(self, bitstreamFileName : str) -> ExecutionStatus:
        """Sends a command to configure the FPGA

        :param bitstreamFileName: The filename of the bitstream which
will be used for device configuration
        :type bitstreamFileName: str
        :return: The status of the execution process
        :rtype: ExecutionStatus
        """

@staticmethod
```

```
    def FindNonMaskedSensitiveBits(ebdFrames : List[EbdFrame],
mskFrames : List[Frame]) -> List[tuple]:
        """Finds all the sensitive bits which are non-masked

        :param ebdFrames: The design EBD frames
        :type ebdFrames: List[EbdFrame]
        :param mskFrames: The design mask frames
        :type mskFrames: List[Frame]
        :return: A list of tuples where each tuple consists of:
        +-------+-----------------------
+-------------------------------------------------------------------+
        | Index | Name                  |
Description                                                     |

+=======+=======================+======================================
===========================+
        | 0     | frameIndex            | The index in the provided
list where the item resides           |
        +-------+-----------------------
+-------------------------------------------------------------------+
        | 1     | bitIndex              | The bit index (which is
sensitive and non-masked) in the frame  |
        +-------+-----------------------
+-------------------------------------------------------------------+
        | 2     | frame address value    | The frame
address                                         |
        +-------+-----------------------
+-------------------------------------------------------------------+

        .. note:: The method could return an empty list
        :rtype: List[tuple]
        """

def ReadFrame(self, address : int, framesToRead : int) -> List[Frame]:
        """Reads frames from the remote device

        :param address: The starting address of the frame read process
        :type address: int
        :param framesToRead: The number of frames to read
        :type framesToRead: int
        :return: The frames read from the remote device.
        :rtype: List[Frame]
        """
def BitFlip(self, bitPosition : int, word = None):
        """Flips a bit in the frame content given the bit position and
the word

        :param bitPosition: The bit position which will be flipped
        :type bitPosition: int
        :param word: The word where the bit resides. If this parameter
```

```
is None then bitPosition is related to the length of the frame,
defaults to None
        :type word: int, optional

        Example 1: bitflip at bit position 2100  -> frame.BitFlip(2100)
        Example 2: bitflip at word 20 and bit 17 -> frame.BitFlip(17,
20)
        """

def WriteFrame(self, frames : List[Frame]) -> ExecutionStatus:
        """Writes a list of frames

        :param frames: The frames to be written
        :type frames: List[Frame]
        :return: The execution status of the command
        :rtype: ExecutionStatus
        """
def WriteBscanRegister(self, address : int, value : int) ->
ExecutionStatus:
    """Writes a BSCAN register

    :param address: The address of the BSCAN register
    :type address: int
    :param value: The value to be written
    :type value: int
    :return: The status of the execution process
    :rtyp

def ReadBscanRegister(self, address : int) -> int:
        """Reads a BSCAN register

        :param address: The address of the BSCAN register to read
        :type address: int
        :return: The value of the BSCAN register
        :rtype: int
        """
```

To run FREtZ, please select `fretz.`pythat is located in the root directory of FREtZ, and select `Run Without Debubbing`.

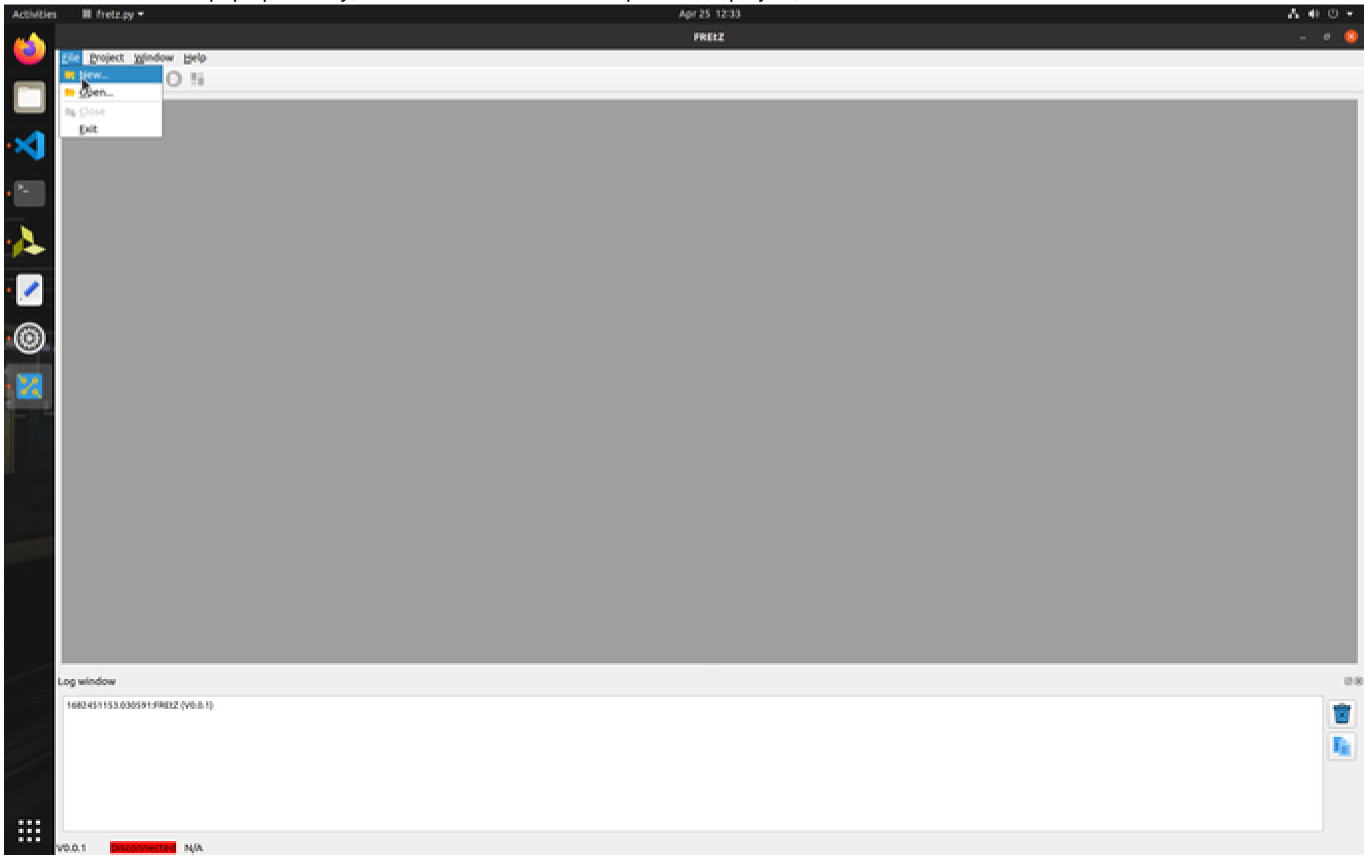


```
from PySide2.QtWidgets import QApplication
from PySide2.QtCore import QCoreApplication, Qt
from UI.MainForm import MainForm

def main():
    """ Main execution
    """
    QCoreApplication.setAttribute(Qt.AA_ShareOpenGLContexts)
    fretzApplication = QApplication([])
    form = MainForm()
    fretzApplication.exec_()

if __name__ == '__main__':
    main()
```

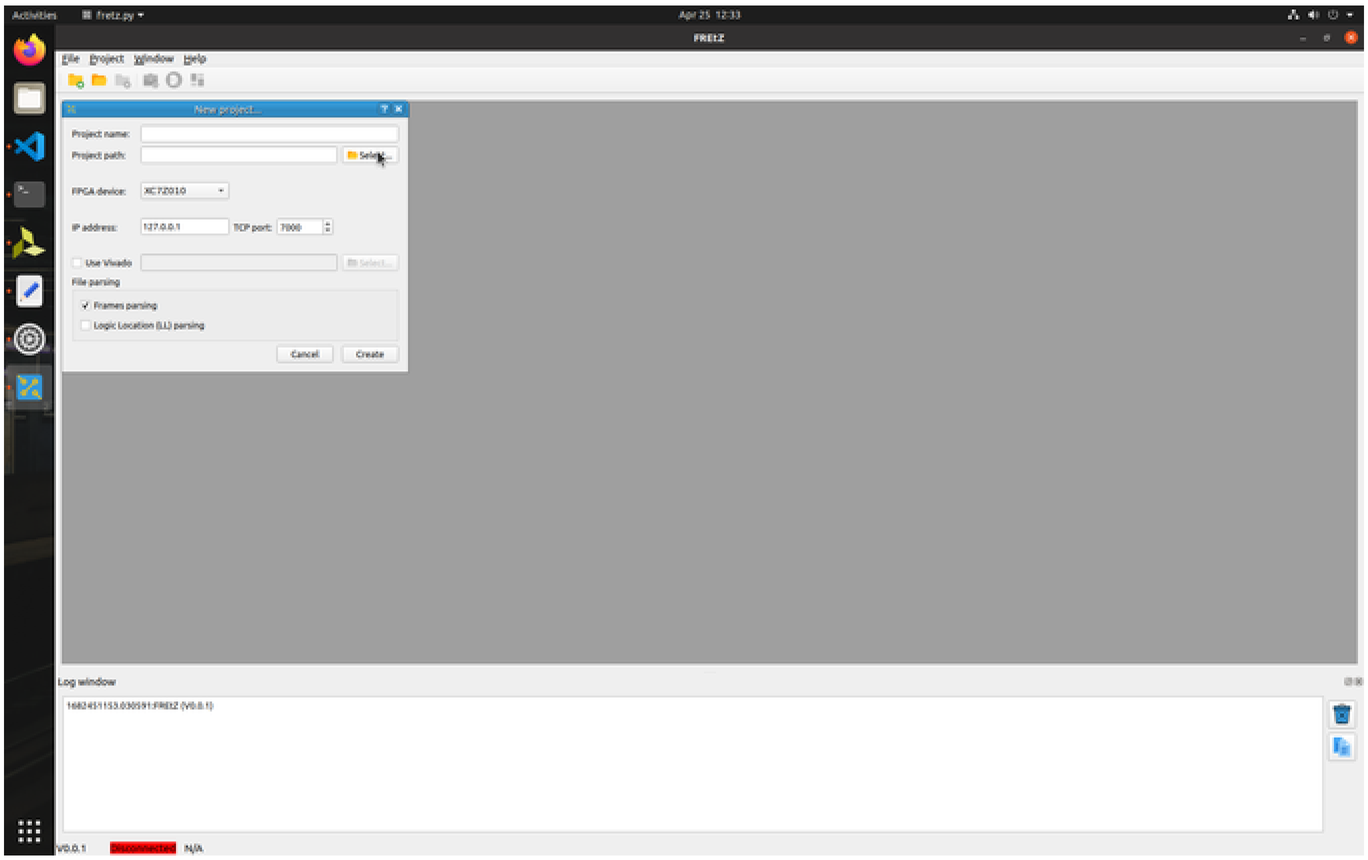
```
fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw/src$
```

The git repository at '/home/fretz' has too many active changes, only a subset of Git features will be enabled.

The GUI of FREtZ will pop up. Initially, we need to create and set up a FREtZ project, as shown below:

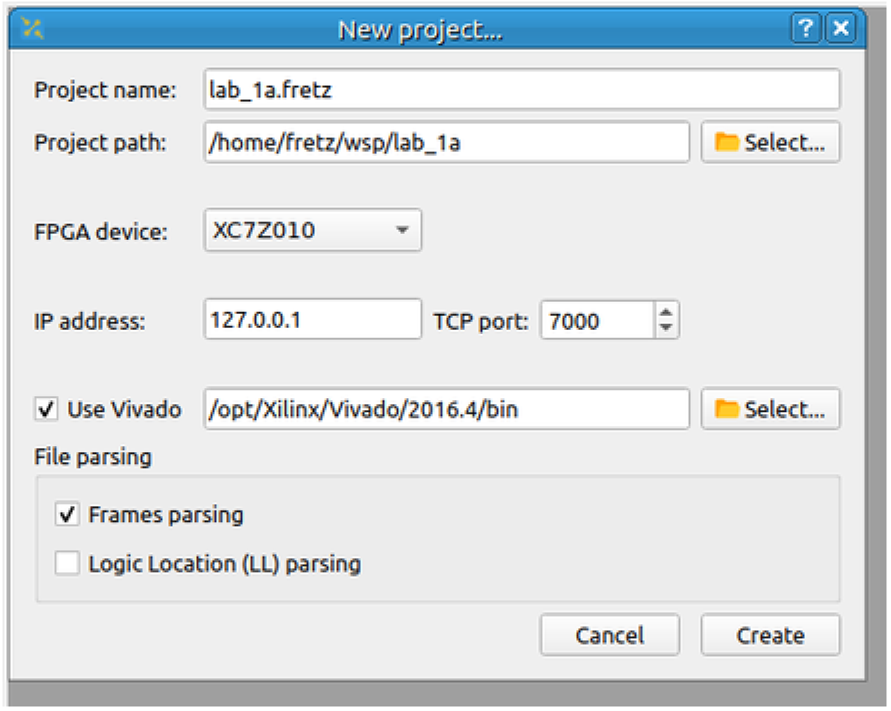


```
1682451153.030591:FREtZ (V0.0.1)
```



The project directory name that we specified is `lab_1a.fretz`.

When you specify the following settings please press the Create button



New project...

Project name: lab_1a.fretz

Project path: /home/fretz/wsp/lab_1a

FPGA device: XC7Z010

IP address: 127.0.0.1 TCP port: 7000

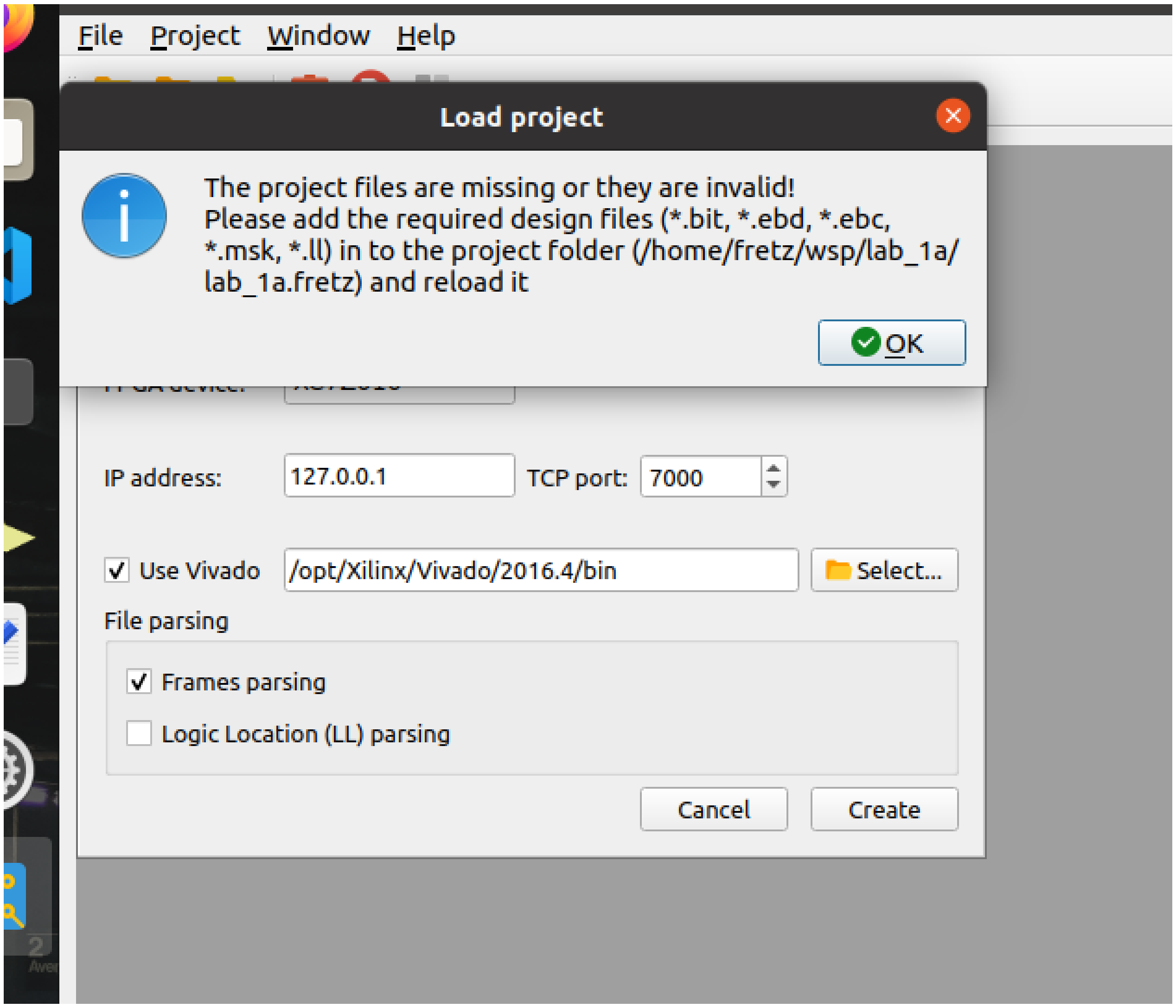
[✓] Use Vivado /opt/Xilinx/Vivado/2016.4/bin

File parsing

[✓] Frames parsing

[ ] Logic Location (LL) parsing

The following window will pop up.



The project files are missing or they are invalid!
Please add the required design files (*.bit, *.ebd, *.ebc, *.msk, *.ll) in to the project folder (/home/fretz/wsp/lab_1a/lab_1a.fretz) and reload it

IP address: 127.0.0.1 TCP port: 7000

☑ Use Vivado /opt/Xilinx/Vivado/2016.4/bin

File parsing

☑ Frames parsing

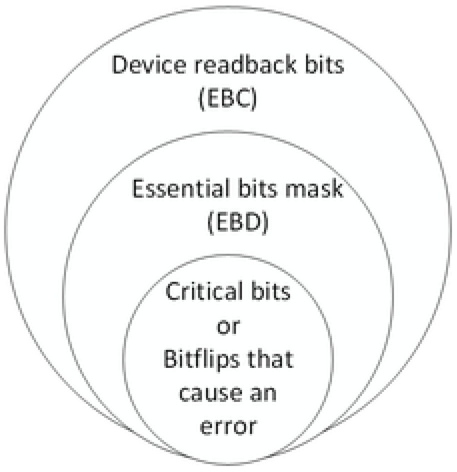
☐ Logic Location (LL) parsing

It instructs you to copy the following files:

- BITSTREAM (.bin): <vivado_project_name>.bin
- MASK FILE (.msk): <vivado_project_name>.msk
- DEVICE READBACK BITS (.ebc): <vivado_project_name>.ebc
- ESSENTIAL BITS FILE MASK (.ebd): <vivado_project_name>.ebd
- LOGIC ALLOCATION FILE(.ll): <vivado_project_name>.ll

Xilinx 4—7 Series devices allow users to read the configuration memory. There are two readback modes: Readback Verify (RbV) and Readback Capture (RbC). The RbV and RbC procedure outputs a readback configuration bit file of a device (.ebc) file.

The configuration bits of the device (.ebc) can be classified as essential (.ebd) and critical bits, as shown in the following figure.



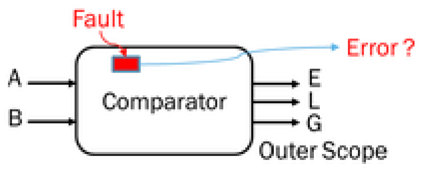
The essential bits can potentially cause an error on the DUT when corrupted. We try to identify which essential bits will cause the error with fault injection experiments. These are called critical bits.

As mentioned, the Architectural Vulnerability Factor (AVF) is a popular reliability metric that shows how sensitive a DUT is to soft errors. In other words, it shows the portion of faults that lead to an output error.

AVF=output errors/total injected faults.

The AVF of an FPGA circuit depends on many factors, such as the circuit's architecture, how the circuit is placed and routed onto the FPGA, and the architecture of the FPGA itself.

In this lab, we inject a fault into the comparator and check if the fault leads to an error, as shown below:



BIT file

A binary file that contains proprietary header information as well as configuration data.

MASK file

A mask of the bit file that indicates which bits are not dynamic, i.e., do not change during circuit operation

EBC file

The EBC file is a reference file containing the FPGA's memory cell content. This is the same content read back by the Vivado hardware manager. It is important to note that this file is not the same as the bitstream used to program the part.

EBD file

The EBD file is used to mask the EBC file meaning that a 1 in the EBD file corresponds to an essential bit in the EBC file. An EBC file bit of 1 or 0 can be essential or critical depending on if there is a corresponding 1 in the EBD file for this bit.

LL file

With RbC mode, one can check the state of registers in a circuit since RbC mode allows the state of the CLB configuration memory cells to be read. This can be done by issuing a GCAPTURE command to the configuration access port of the FPGA so as to sample all CLB register values into configuration memory cells. These values can then be read back along with the configuration frame containing the status of user memory elements (e.g., registers). However, designers must know the frame address and configuration bit offset of the SRAM cell corresponding to the desired register output of the DUT. These parameters are given in the logic allocation (*.ll) file, which is automatically generated by the Xilinx ISE /Vivado design tools. The logic allocation file includes four fields, namely a bit offset, a frame address, a frame offset, and information for each configured resource, as depicted in Fig. 5. In the following, we provide an example where the registers corresponding to the voter status of a TMR component are determined from the information fields that then allow the frame addresses and frame offsets to be extracted:

```
<bit offset> <frame addr> <offset> <Information>
Bit 19488835 0x0042021f 3107 Block=SLICE_X3Y48 Latch=AQ Net=voters[6]
/status_bits[1]
```

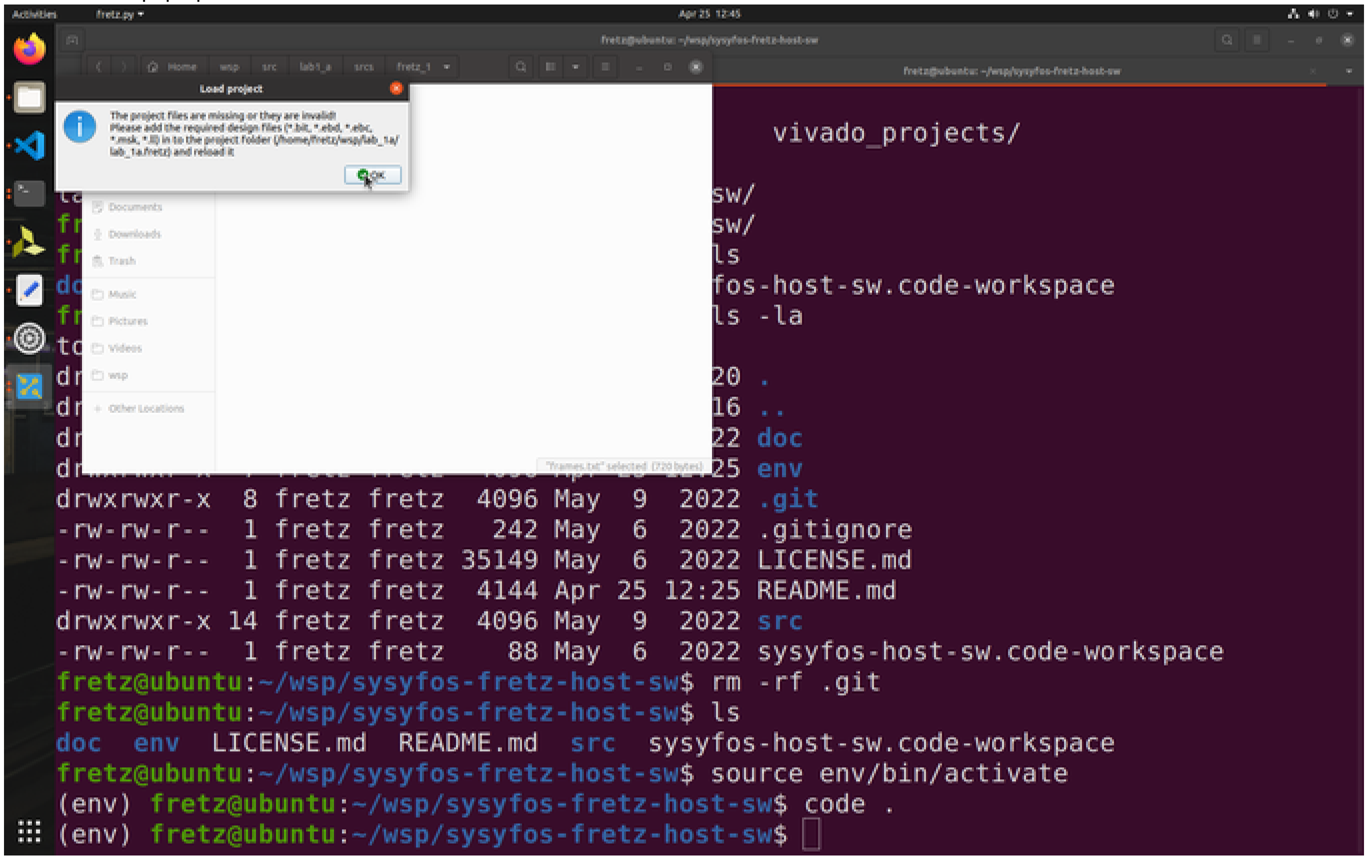
READMORE in https://support.xilinx.com/s/article/14468?language=en_US

OK, now that we know what all these files mean, let's copy them from the Vivado project to the Fretz project folder in order to perform the fault injection.

Open terminal

```
cd ~/wsp/lab_1a/lab_1a.fretz/
cp ../lab_1a.runs/impl_1/*.ll ../lab_1a.runs/impl_1/*.bit  ./
cp ../lab_1a.runs/impl_1/*.ebd ../lab_1a.runs/impl_1/*.ebc ./
cp ../lab_1a.runs/impl_1/*.msk ./
cp ../../src/lab1_a/srcs/fretz_1/frames.txt ./
```

Then close the pop-up window:

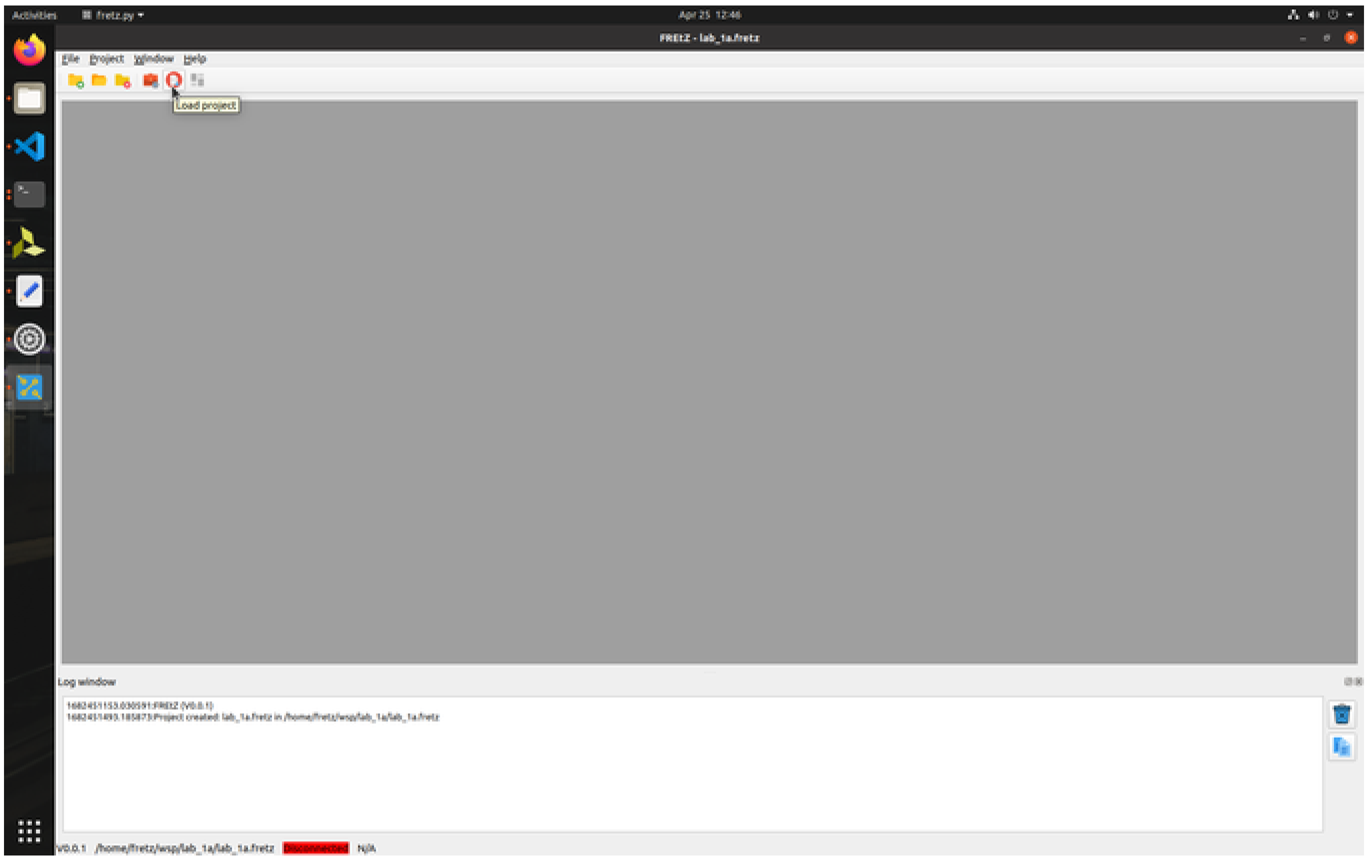


Load project

The project files are missing or they are invalid!
Please add the required design files (*.bit, *.ebd, *.ebc, *.msk, *.ll) in to the project folder (/home/fretz/wsp/lab_1a/lab_1a.fretz) and reload it

```
drwxrwxr-x  8 fretz fretz  4096 May  9  2022 .git
-rw-rw-r--  1 fretz fretz   242 May  6  2022 .gitignore
-rw-rw-r--  1 fretz fretz 35149 May  6  2022 LICENSE.md
-rw-rw-r--  1 fretz fretz  4144 Apr 25 12:25 README.md
drwxrwxr-x 14 fretz fretz  4096 May  9  2022 src
-rw-rw-r--  1 fretz fretz    88 May  6  2022 sysyfos-host-sw.code-workspace
fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ rm -rf .git
fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ ls
doc  env  LICENSE.md  README.md  src  sysyfos-host-sw.code-workspace
fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ source env/bin/activate
(env) fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ code .
(env) fretz@ubuntu:~/wsp/sysyfos-fretz-host-sw$ 
```

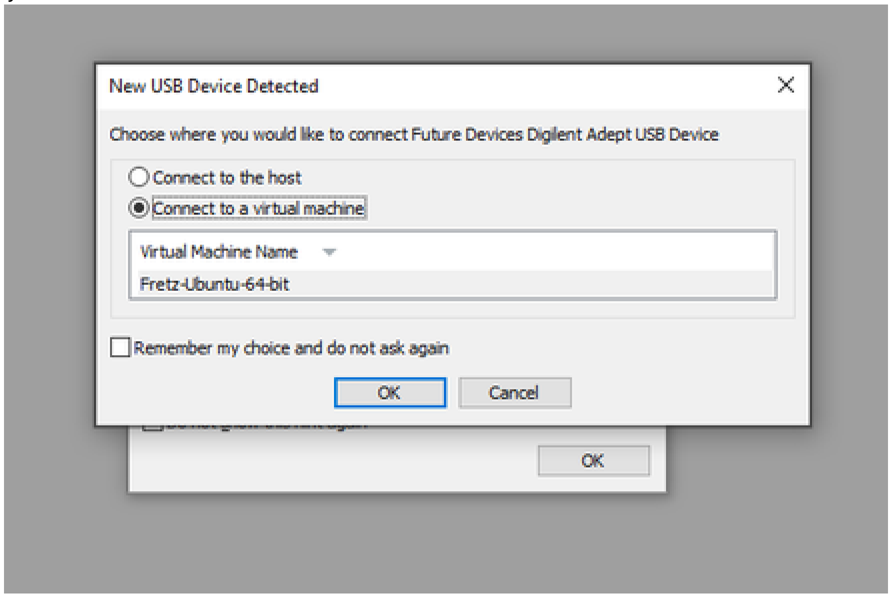
In FREtZ GUI press the `Load project` button.



```
1682451153.030591:FRETZ (V0.0.1)
1682451493.185873:Project created: lab_1a.fretz in /home/fretz/wsp/lab_1a/lab_1a.fretz
```

```
V0.0.1   /home/fretz/wsp/lab_1a/lab_1a.fretz   Disconnected   N/A
```

Turn on and connect the Zybo card to the Virtual machine.


New USB Device Detected

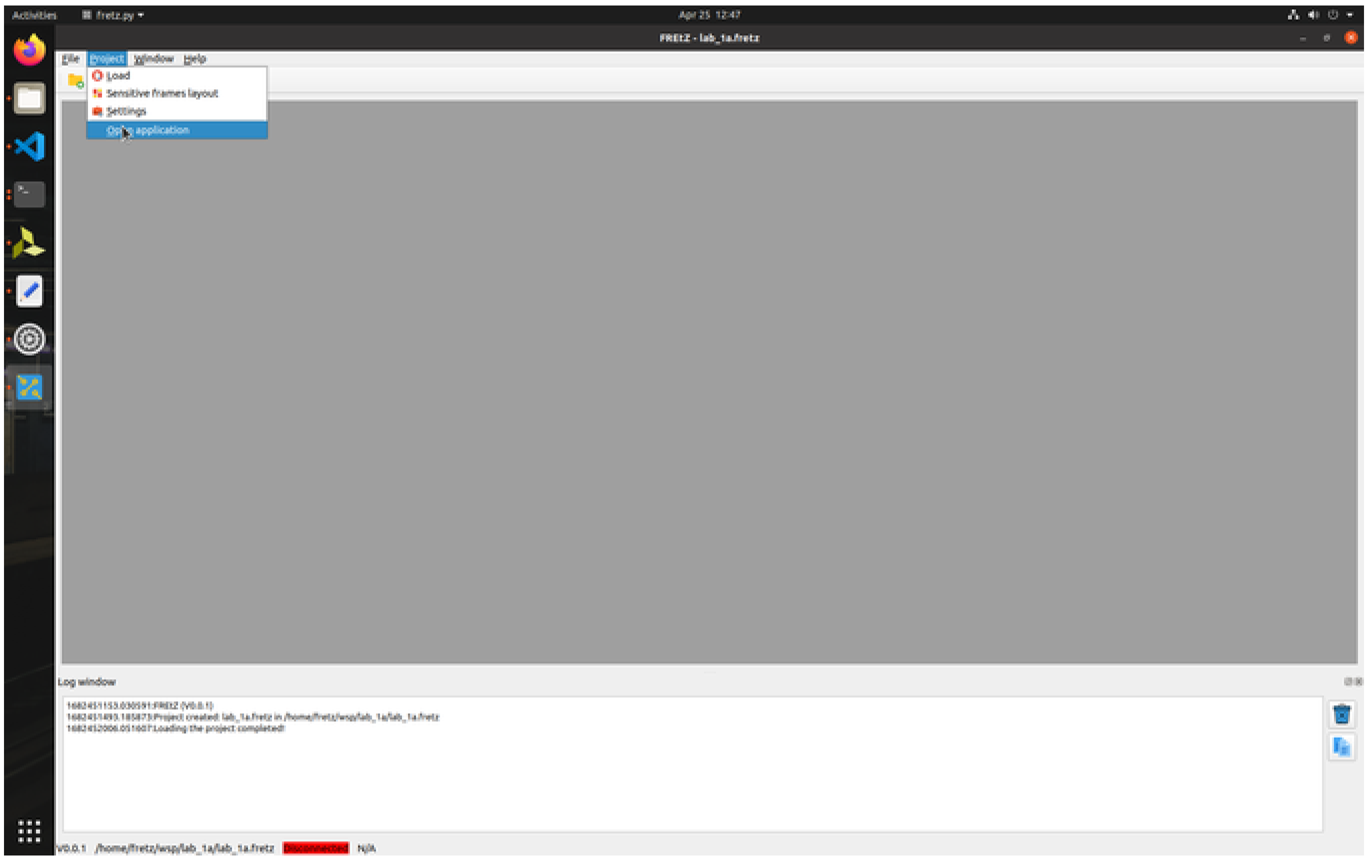
Choose where you would like to connect Future Devices Digilent Adept USB Device

( ) Connect to the host
(●) Connect to a virtual machine

Virtual Machine Name
Fretz-Ubuntu-64-bit

☐ Remember my choice and do not ask again

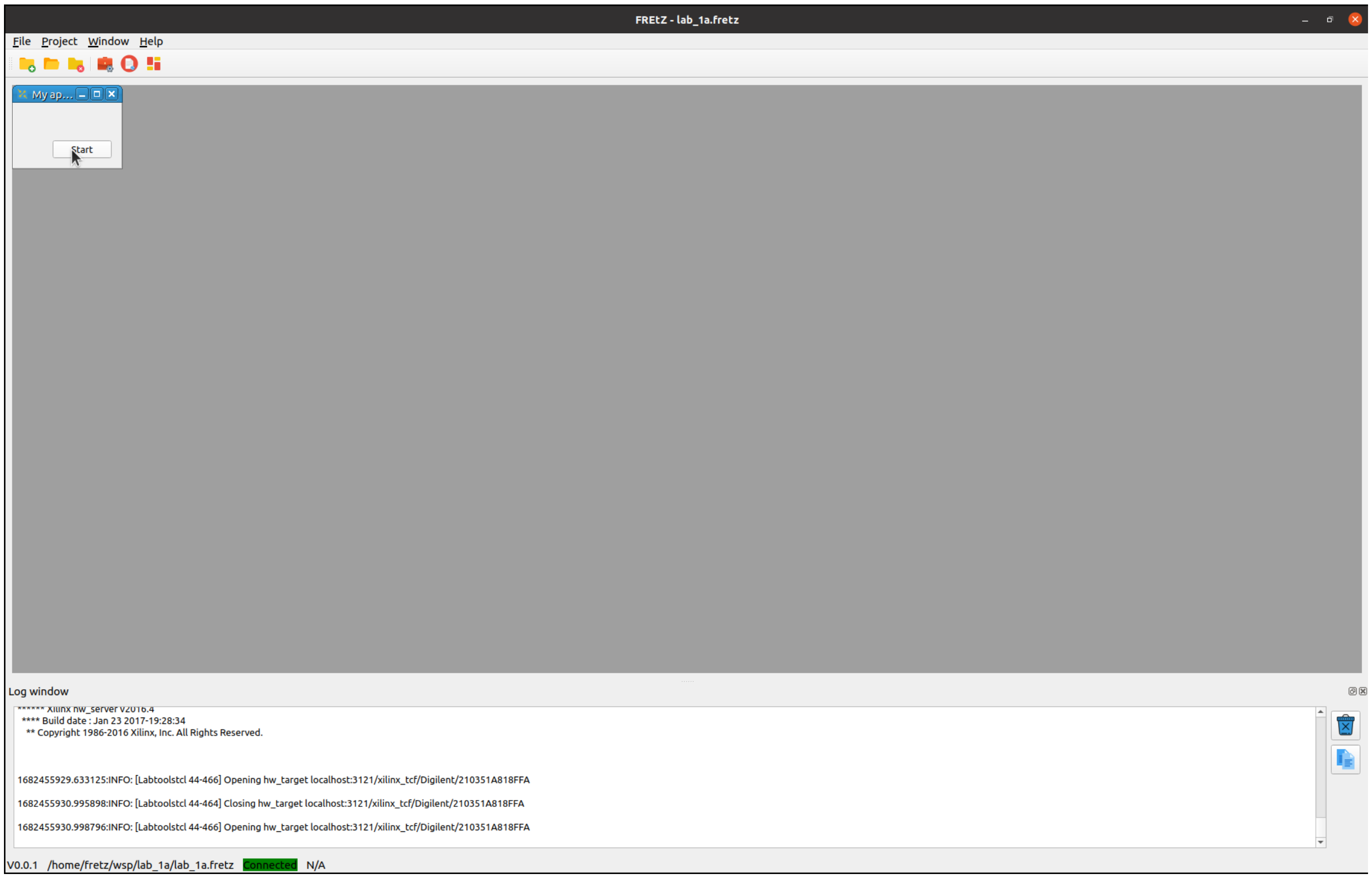
Click the `Open application` button.



```
1682451153.030591:FREtZ (V0.0.1)
1682451493.185873:Project created: lab_1a.fretz in /home/fretz/wsp/lab_1a/lab_1a.fretz
1682452006.051607:Loading the project completed!
```

```
V0.0.1  /home/fretz/wsp/lab_1a/lab_1a.fretz  Disconnected  N/A
```

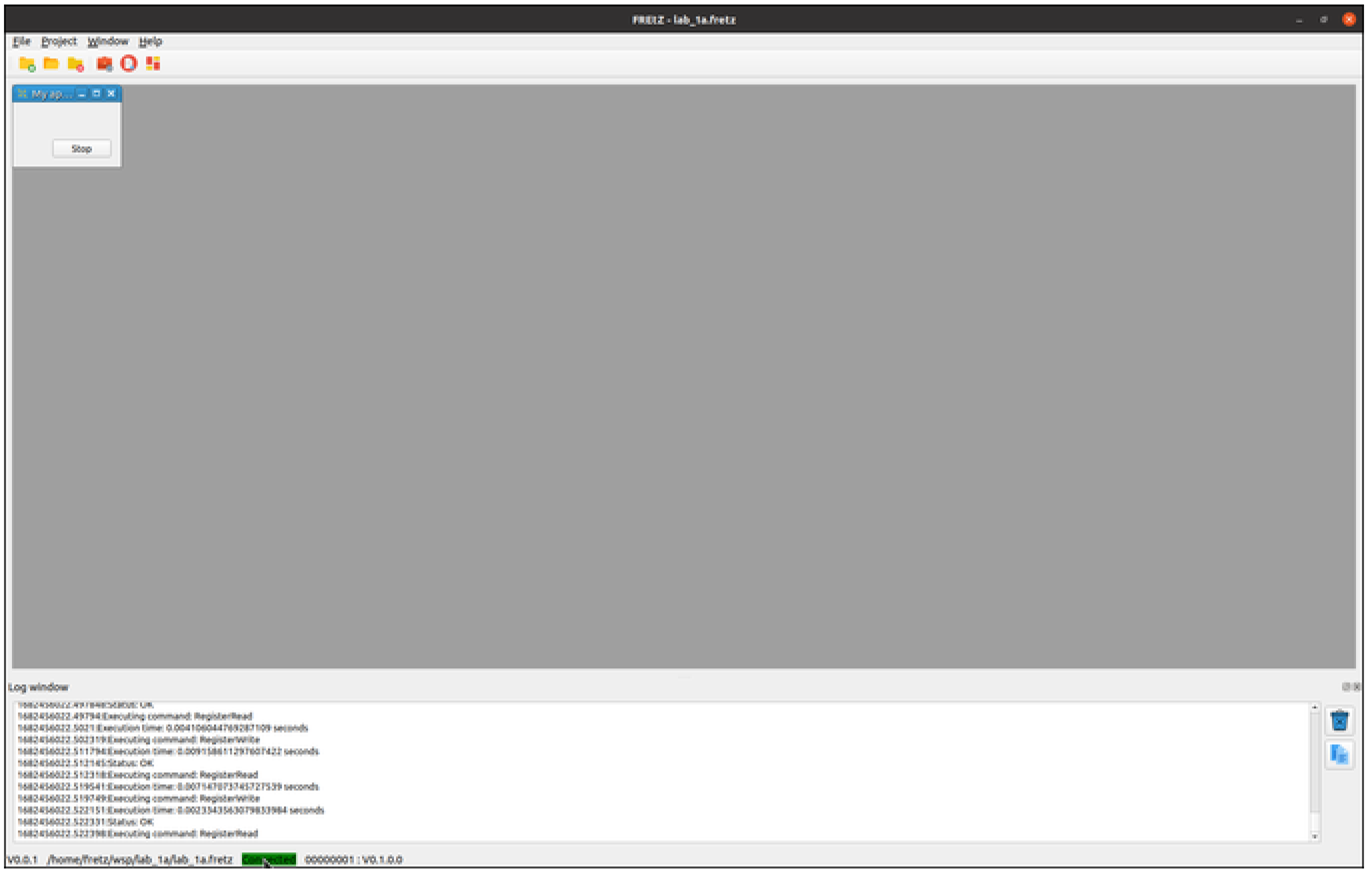
Click the `Start` button.



```
****** Xilinx hw_server v2016.4
  **** Build date : Jan 23 2017-19:28:34
    ** Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.


1682455929.633125:INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210351A818FFA

1682455930.995898:INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210351A818FFA

1682455930.998796:INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210351A818FFA
```

```
V0.0.1  /home/fretz/wsp/lab_1a/lab_1a.fretz  Connected  N/A
```

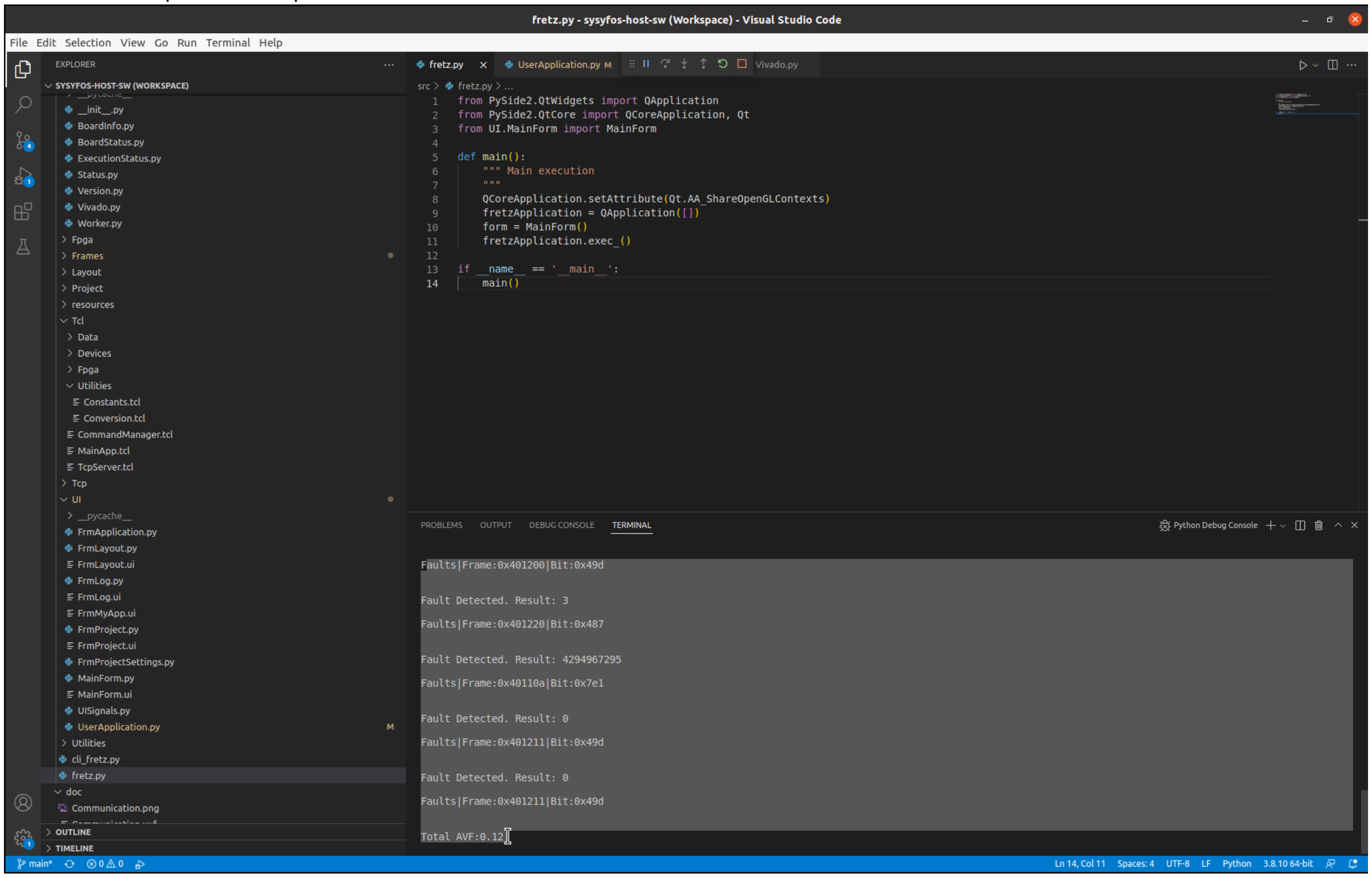
At the bottom of the FREtZ GUI, you will see that we have a connection with the board. Also, in the `Log window` you will see that FREtZ communicates with the Vivado Hardware Manager in order to instruct it to read and write frames during the fault injection procedure.



```
1682456022.49794:Executing command: RegisterRead
1682456022.5021:Execution time: 0.004106044769287109 seconds
1682456022.502319:Executing command: RegisterWrite
1682456022.511794:Execution time: 0.009158611297607422 seconds
1682456022.512145:Status: OK
1682456022.512318:Executing command: RegisterRead
1682456022.519541:Execution time: 0.007147073745727539 seconds
1682456022.519749:Executing command: RegisterWrite
1682456022.522151:Execution time: 0.0023543563079833984 seconds
1682456022.522331:Status: OK
1682456022.522398:Executing command: RegisterRead
```

V0.0.1 /home/fretz/wsp/lab_1a/lab_1a.fretz Connected 00000001 : V0.1.0.0

Now switch to the Microsoft Code editor to observe what is reported in the terminal. At the end of the fault injection experiment, you will see that the AVF of the experiment is reported.



```python
from PySide2.QtWidgets import QApplication
from PySide2.QtCore import QCoreApplication, Qt
from UI.MainForm import MainForm

def main():
    """ Main execution
    """
    QCoreApplication.setAttribute(Qt.AA_ShareOpenGLContexts)
    fretzApplication = QApplication([])
    form = MainForm()
    fretzApplication.exec_()

if __name__ == '__main__':
    main()
```

```
Faults|Frame:0x401200|Bit:0x49d

Fault Detected. Result: 3
Faults|Frame:0x401220|Bit:0x487

Fault Detected. Result: 4294967295
Faults|Frame:0x40110a|Bit:0x7e1

Fault Detected. Result: 0
Faults|Frame:0x401211|Bit:0x49d

Fault Detected. Result: 0
Faults|Frame:0x401211|Bit:0x49d

Total AVF:0.12
```

Now that you finished this design example, can you develop an experiment that performs fault injection in a 32-bit adder?

TIPS:

- Provide via BSCAN the same date to the adder's input
- Get the result of the adder's output via BSCAN
- Compare the result with a golden value
- Please uncomment the following lines (78-79) in the `UserApplication.py` if you want to debug

```
#pydevd.connected = True
#pydevd.settrace(suspend=False)
```