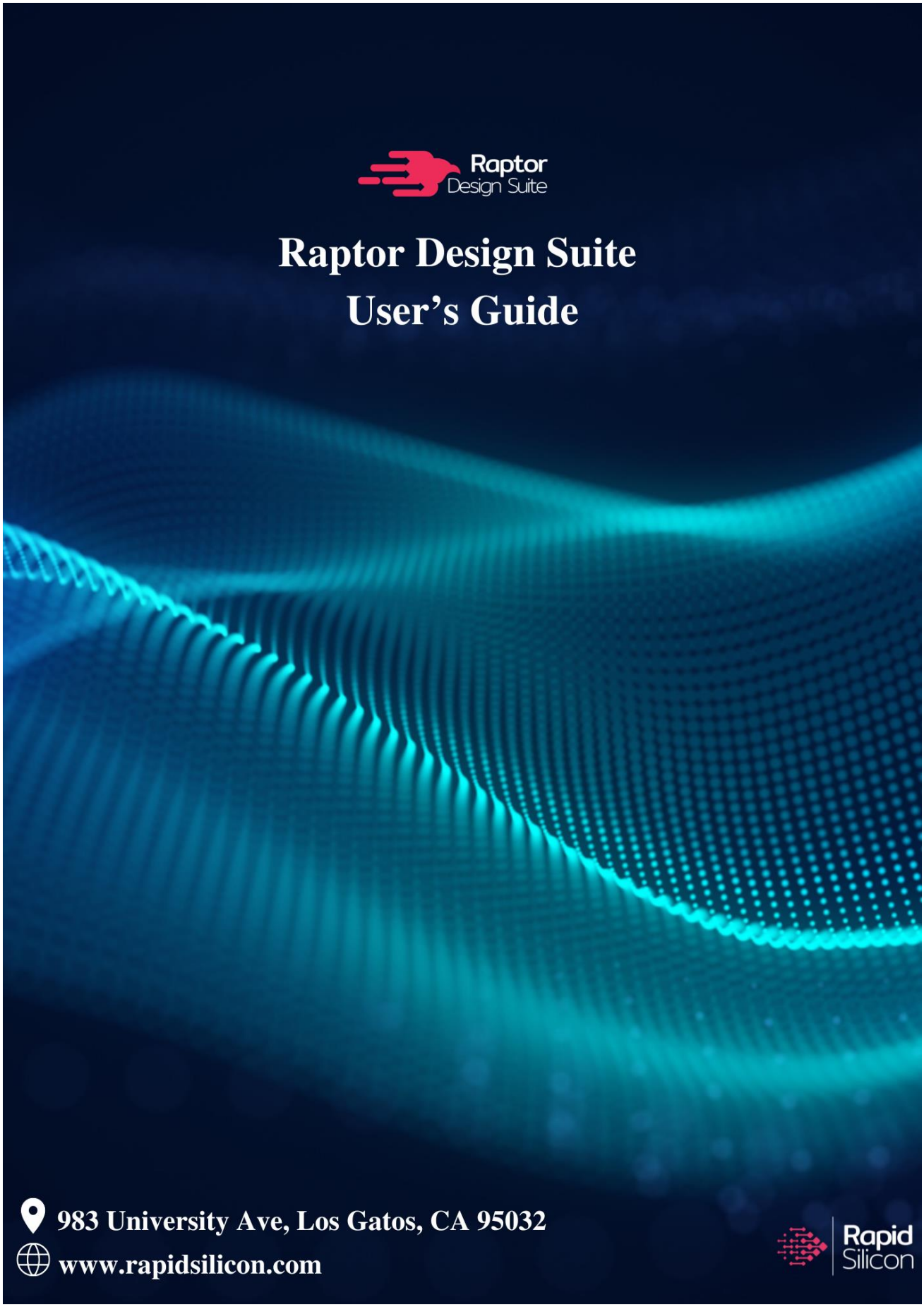

Raptor
Design Suite
Raptor Design Suite
User's Guide
983 University Ave, Los Gatos, CA 95032
www.rapidsilicon.com
Rapid
Silicon

Legal Disclaimer

© Rapid Silicon 2023

The information contained herein is for informational and promotional purposes only. No claims, representations or warranties or guarantees, whether express or implied, are made by Rapid Silicon as to the suitability, safety, reliability, durability, and performance of any of our companies' products or services for any particular purpose. Rapid Silicon does not assume any liability whatsoever arising out of the application or use of any product or circuit.

Rapid Silicon accepts no liability whatsoever regarding the suitability, safety, reliability, durability, and performance of any of our companies' products or services. Any specifications and information provided are indicative only. Potential buyers shall not rely on any information provided herein and shall not rely on data and performance specifications or parameters provided by Rapid Silicon.

Rapid Silicon does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Rapid Silicon, and Rapid Silicon reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

Table of Contents

1 Using the Raptor Design Suite

The Raptor Design Suite is an integrated user design environment that provides an end-to-end system-on-chip design environment based on open-source tooling, allowing users to review the code used to implement their design. Further, users can suggest optimizations and new features within the open-source community.

For automation and ease-of-use, the Raptor design suite provides both a project-based GUI-based design cockpit, as well as a robust Tcl scripting environment. Users can switch between the GUI environment and the Tcl scripting environment at will and even automate the GUI experience using Tcl scripts.

1.1 Raptor Design Suite Features

The Raptor design suite is accessible through Tcl, allowing design changes to elements such as configuration, constraints, or settings to be performed in real time.

The Raptor design suite includes the following features:

- Register transfer level (RTL) design in VHDL, Verilog, and SystemVerilog
- Synthesis
- Easy integration of Intellectual Property (IP) for third-party cores
- Place and route*
- Static timing analysis (STA)*
- Performs Behavioral, functional, and timing simulation*

* The Gemini device (1GE100-ES1 and timing model currently available in the Raptor design suite version 0.5 is a placeholder to evaluate the software flow only. This version is not intended for design purposes. However, it can be used to evaluate the software flow and usability.

1.2 Raptor Design Suite Tools

The Raptor design suite is based on the following tools.

Table 1. Raptor Design Suite Tools

Tool	Name	Source Location
FOEDAG	GUI Toolkit	https://github.com/os-fpga/FOEDAG
LiteX	IP Integration	https://github.com/enjoy-digital/litex
Yosys	Synthesis	https://github.com/YosysHQ/yosys
ABC	Sequential Synthesis	https://github.com/berkeley-abc/abc
VPR	Place & Route	https://github.com/verilog-to-routing
OpenFPGA	Bitstream Generation	https://github.com/lnis-uofu/OpenFPGA
OpenOCD	Device Programmer	https://github.com/openocd-org/openocd

1.2.1 Project and Tcl Modes

There are two design flow modes available in the Raptor Design Suite:

- Project mode
- Tcl mode

In Project mode, a project is created within the Raptor design suite using GUI. This approach automatically saves the design at regular intervals. It also automatically manages the source files and generates reports as necessary.

Tcl mode is executed using Tcl commands or scripts. In Tcl mode, the user still has full control of the design flow as with Project mode, but the Raptor design suite does not automatically manage source files or report the design state. Rather, the Raptor design suite can be opened at each stage of the design in order to manage the source files and perform design analysis. In Tcl mode the active design has an in-memory presentation that is "session persistent", so any changes are automatically passed forward in the flow. It also allows the user to update to constraint files or design checkpoints.

In both modes, the Raptor Design Suite supports the following features:

- Design source file list definition (RTL files, include paths, libraries...)
- Status reports on running tasks
- Automatically generated standard reports
- Persistent project settings and design configuration
- Experimentation with synthesis runs
- Definition, usage, and management of constraint sets
- Creation of report (rpt) files generated for each compilation step
- Easy integration and configuration of third-party IP

1.2.2 Launching the Raptor Design Suite

The Raptor Design Suite currently supports Linux and can be invoked using one of the following methods:

- Project mode
- Tcl mode

1.2.2.1 Working with the Raptor Design Suite

Setting up the Raptor Design Suite Environment

Prior to launching the Raptor design suite, the appropriate environment must be set up. This process has been simplified with a script provided in the Raptor design suite install directory. Users can source the Raptor OS environment setup script using the following command

```
$ source <install_path>/raptorenv_lin64.sh
```

Launching the Raptor Design Suite from the Command Line on Linux

Enter the following command at the command prompt:

```
raptor (GUI mode)
raptor --batch (Batch mode
```

1.2.2.2 Working with TCL

When using the Raptor design suite in Tcl mode, the Tcl commands can be executed using either of the following methods:

- Enter individual Tcl commands from a Tcl shell outside of the Raptor design suite.
- Run Tcl scripts from the Raptor Design Suite Tcl console within the Raptor GUI (see Figure 6).

Launching the Raptor Design Suite Tcl Shell

The following commands can be used to invoke the Raptor Design Suite Tcl shell at the Linux command prompt:

```
raptor --batch
```

This command enters the Tcl prompt

#

Launching the Raptor Design Suite Using a Batch Tcl Script

The Raptor design suite can be invoked in batch mode by supplying a Tcl script. Use the following command at the Linux command prompt.

To launch the GUI driven by script:

```
raptor --script <path-to-tcl-script>
```

To launch Raptor for regressions (where the script will run and exit upon completion), i.e., batch driven by script:

```
raptor --batch -script <path-to-tcl-script>
```

1.2.3 Welcome Page

When you open the Raptor design suite, the Welcome page appears as shown in Figure 1. The Raptor Design Suite Welcome page assists in creating and opening projects or example designs.

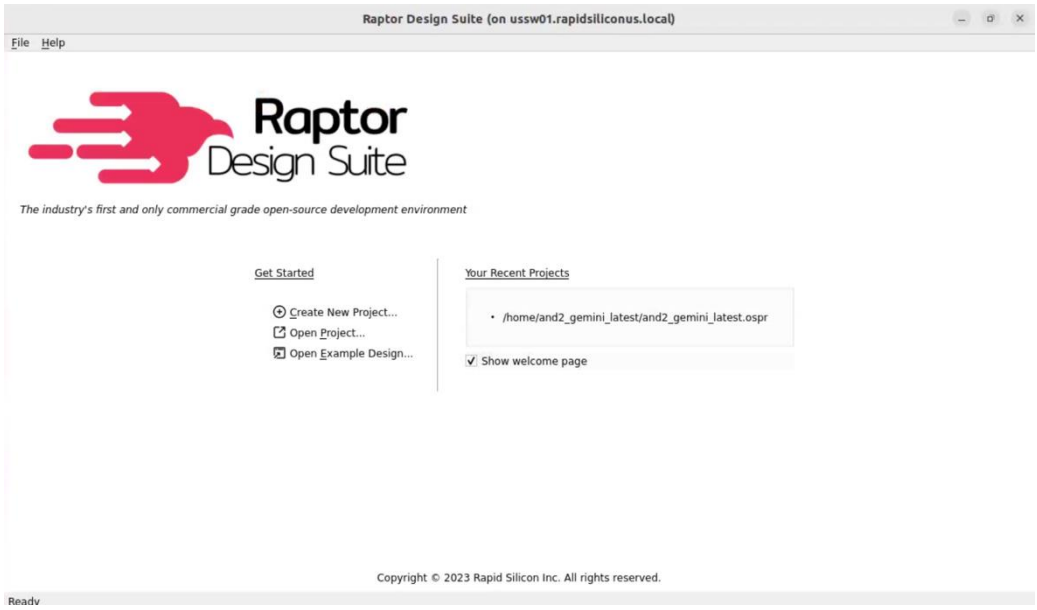


Figure 1. Raptor Design Suite Welcome Page

1.3 Using the Raptor Design Suite Viewing Environment

This section contains general information on the terminology, layout, and project features of the Raptor design suite in project mode using the GUI

1.3.1 Menu Bar

The main menu bar in Figure 2 provides access to Raptor design suite commands.

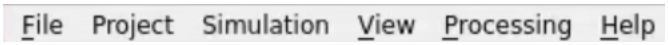


Figure 2. Raptor Design Suite Menu Bar

1.3.2 Main Tool Bar

The main tool bar in Figure 3 provides one-click access to the most used commands in the Raptor design suite such as save and stop.

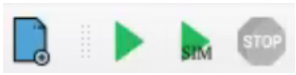


Figure 3. Raptor Design Suite Main Tool Bar

1.3.3 Task Manager

The Task Manager provides access to commands as shown in Figure 4.

Figure 4. Raptor Design Suite Task Manager

1.3.4 Layout

The Raptor design suite provides predefined window layouts to facilitate various tasks in the design process. The user can move the windows to a desired location such as moving task window to the right of the design window. An empty project view is shown in Figure 5.

Figure 5. Raptor Design Suite Layout Selector

Figure 6 shows an example project (and2_gemini) with all main views highlighted. Each number in the figure is described below.

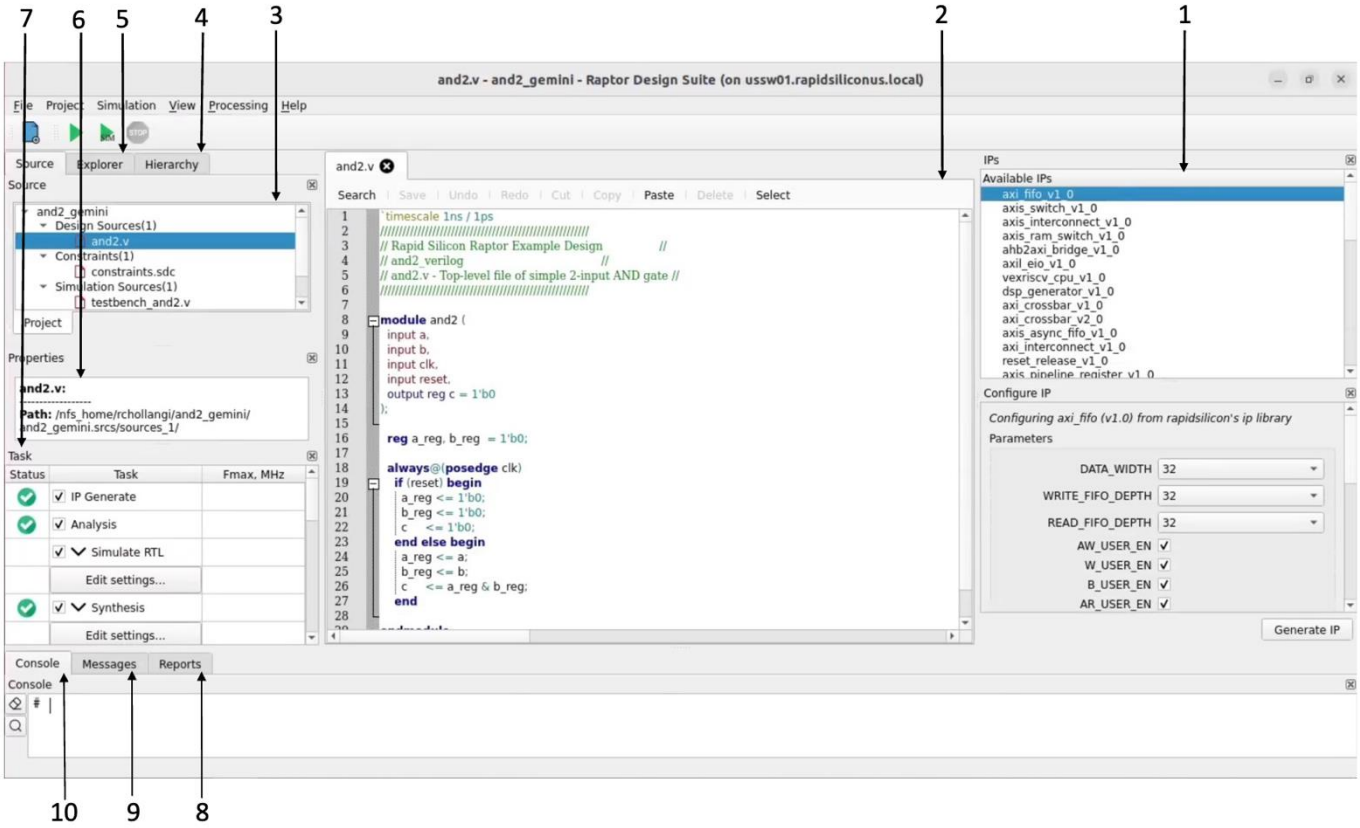


Figure 6. Example Project View

1. IP Configurator. Configure and generate IPs for project. Refer Section 1.3.8 "IP Configurator" for details.
2. Editor. Text editor to view and edit text-based source files.
3. Design file browser. Once design files are added while creating a project or later using the project settings, they can be viewed here. Right click on each will open provide users various ways to modify the design.

   Hierarchical viewer. View instances and module names defined in the project. Figure X shows an example and right-clicking allows user to go to the definition/instantiation of the instance.<top>

   <instance name 1>: <module name>
   <instance name 2>: <module name>
   <instance name 1>: <module name>
   <instance name 2>: <module name>
   <instance name 3>: <module name>

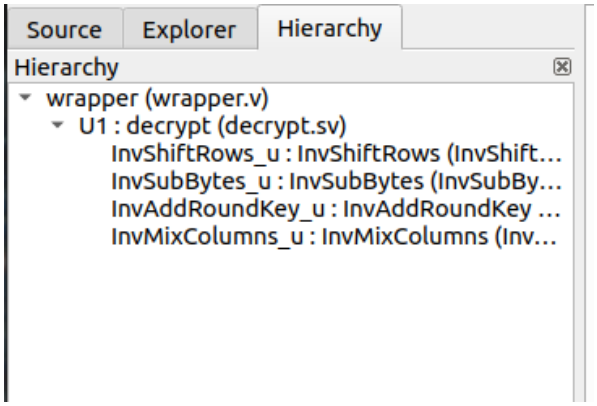


Figure 7. Hierarchical Viewer

4. Note: Known issue with "Go to definition" context menu such that it takes user to the end of module definition. Explorer. Filesystem explorer for navigating the project folder (default) and other locations as needed.
5. Properties. File location information about the open file that one is editing, including line numbers will be displayed in the editor window.
6. Task. Displays sequential tasks in the compilation of the design intended from source files displayed.
7. Reports for tasks as applicable.
8. Messages for tasks as applicable.
9. Tcl Console. Tcl interpreter built-into the Raptor design suite. Refer to the Tcl section for detailed commands available

1.3.5 Project Status Column in Task Window and Progress Bar

The project status column displays the current status of the active design compilation in the Task window as shown in Figure 8. Each number in the figure is further described below.

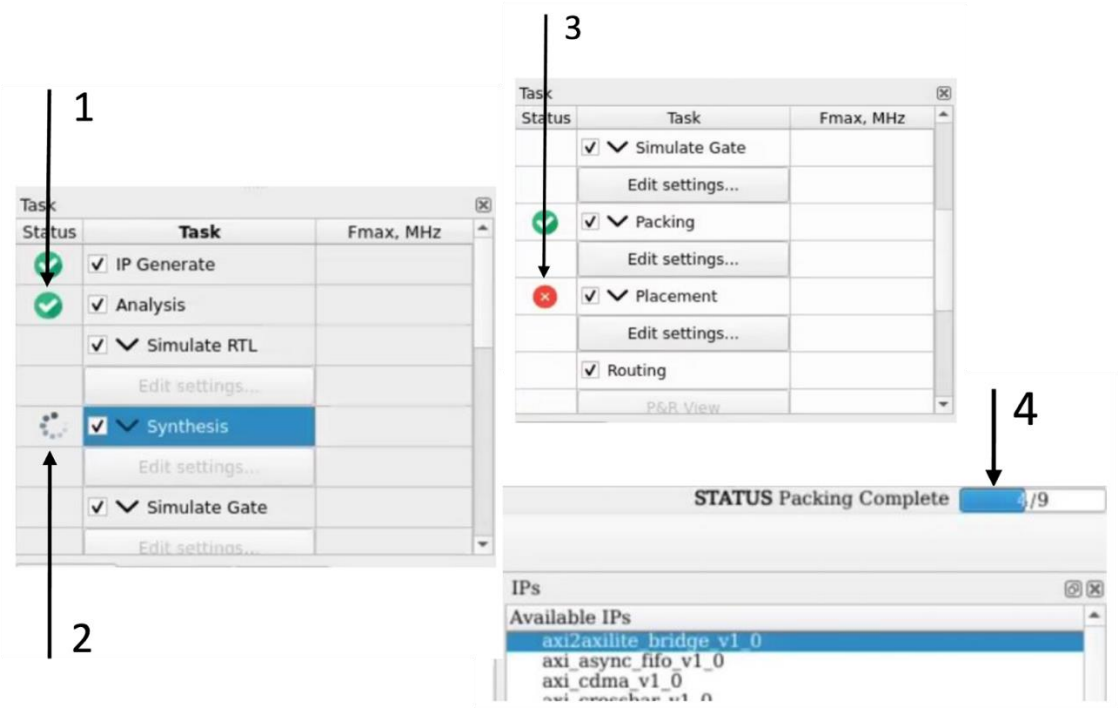


Figure 8. Project Status Column in Task Window and Progress Bar

1. Green check mark for a successful compilation step.
2. Loading symbol for an active compilation step.
3. Red X mark for a failed compilation step.

1.3.6 Creating Projects

The Raptor design suite contains a 'New Project' option to easily create different types of projects. To open this wizard, select File → Project → New, then specify the project location and name to create the project.

1.3.6.1 Creating Projects Using TCL Commands

In addition to using the wizard described above, you can create a project using Tcl commands. Commands can be:

- Entered in the Tcl Console of the Raptor design suite
- Sourced directly from a Tcl file

Passed in a tcl file to raptor invocation as a CLI option raptor --script <tcl file> An example script is shown below.

```
#
D
e
s
i
g
n
 create_design and2_gemini
 add_design_file -
 V_2001 ./rtl/and2.v
 set_top_module and2
 add_constraint_file
 constraints.sdc
 # Device target

 target_device 1GE100-ES1

 # Compilation
 analyze
 synthesize delay
 packing
 place
 route
 sta
 bitstream
```

1.3.6.2 Creating an RTL Project Using the Raptor Design Suite

Follow the steps below to create a project using the Raptor design suite.

A new project can be created using the menu shown below, or via the Welcome page shown in Figure 1.

Figure 9. Create a New Project

The project wizard will guide the user through each step shown below. The following steps show how to create a project using the example design "`and2_gemini`" bundled with Raptor.

1. Project Directory: Set a project name and location as shown in Figure 10.

NOTE: The user must have read and write permissions to the project directory location

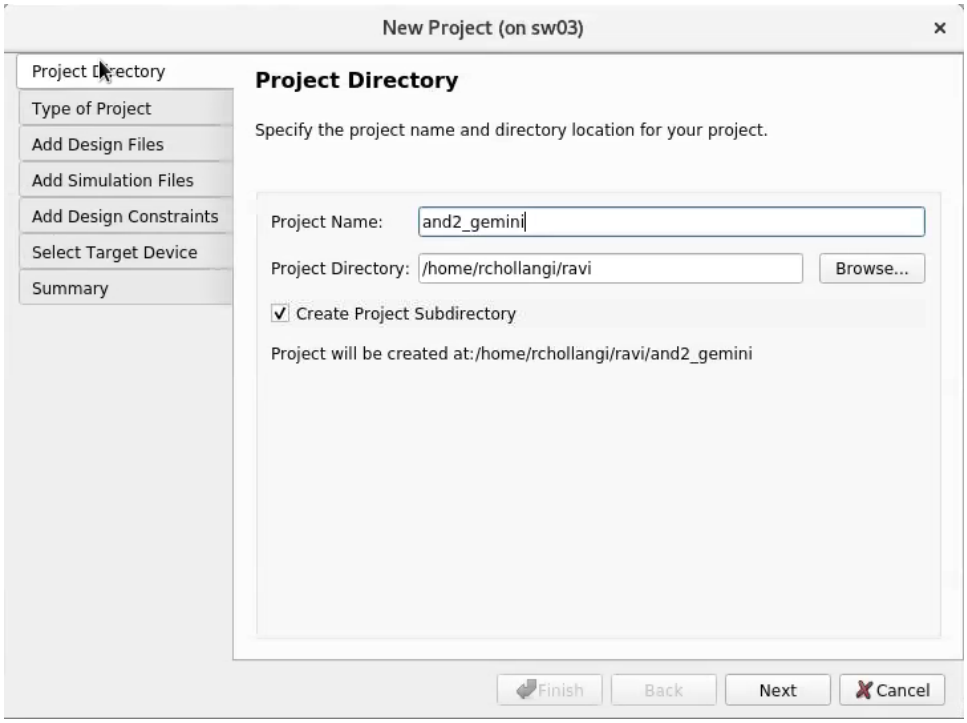


Figure 10. Project Directory Page

2. Type of Project: User can skip the design sources and add them later using the source view. Refer to Figure 11

   Note: Default type of project is RTL

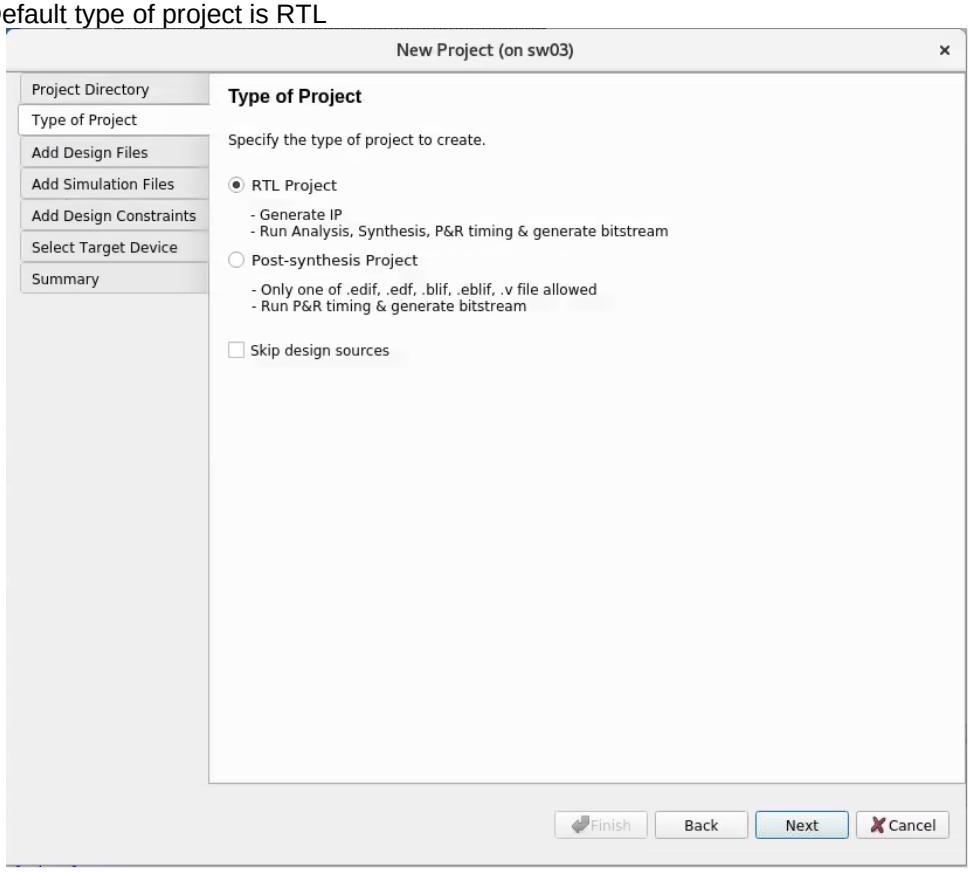


Figure 11. Type of Project

3. Add Design Files: Various methods to add design files are described in Figure 12

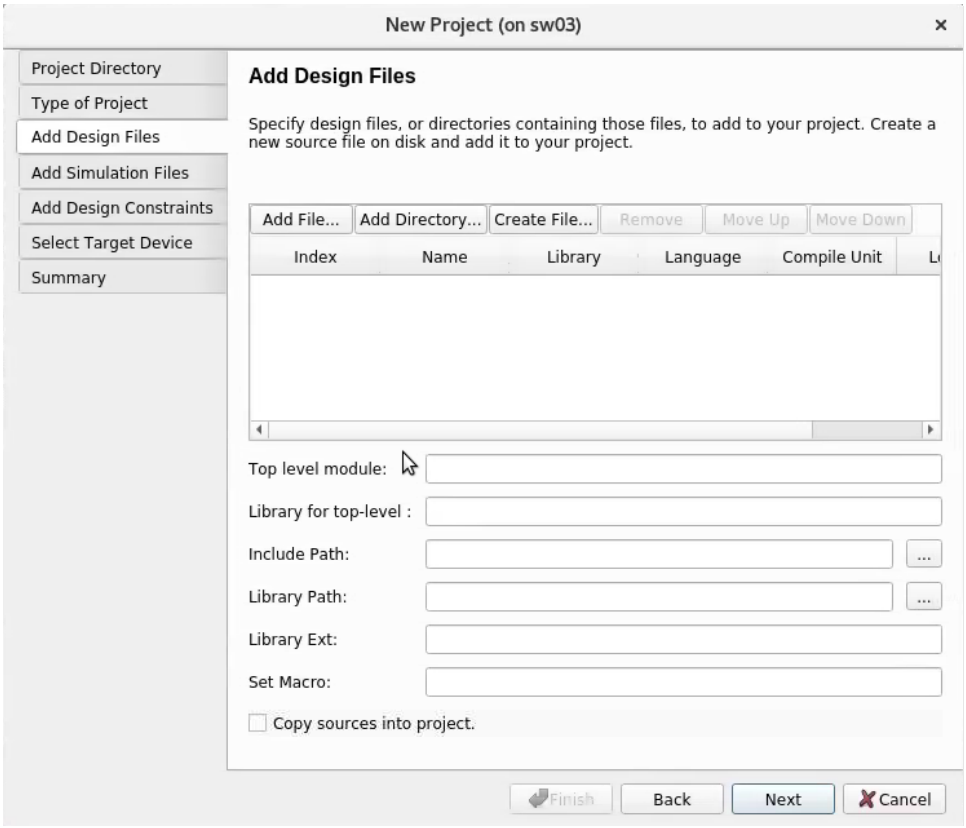


Figure 12. Add Design Files

4. Create a source file with supported file type as shown in Figure 13.

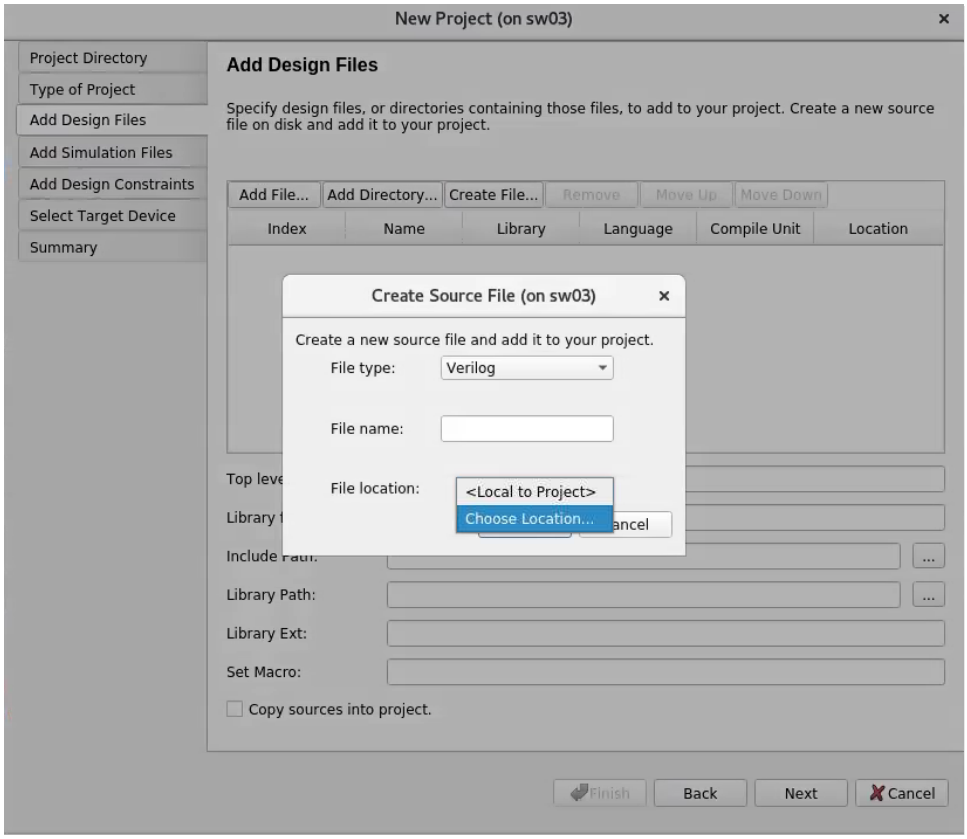


Figure 13. Create a Source File

5. The user can select a directory to scan for supported source files. Raptor will recursively search through the folder and add the supported source files to the project.

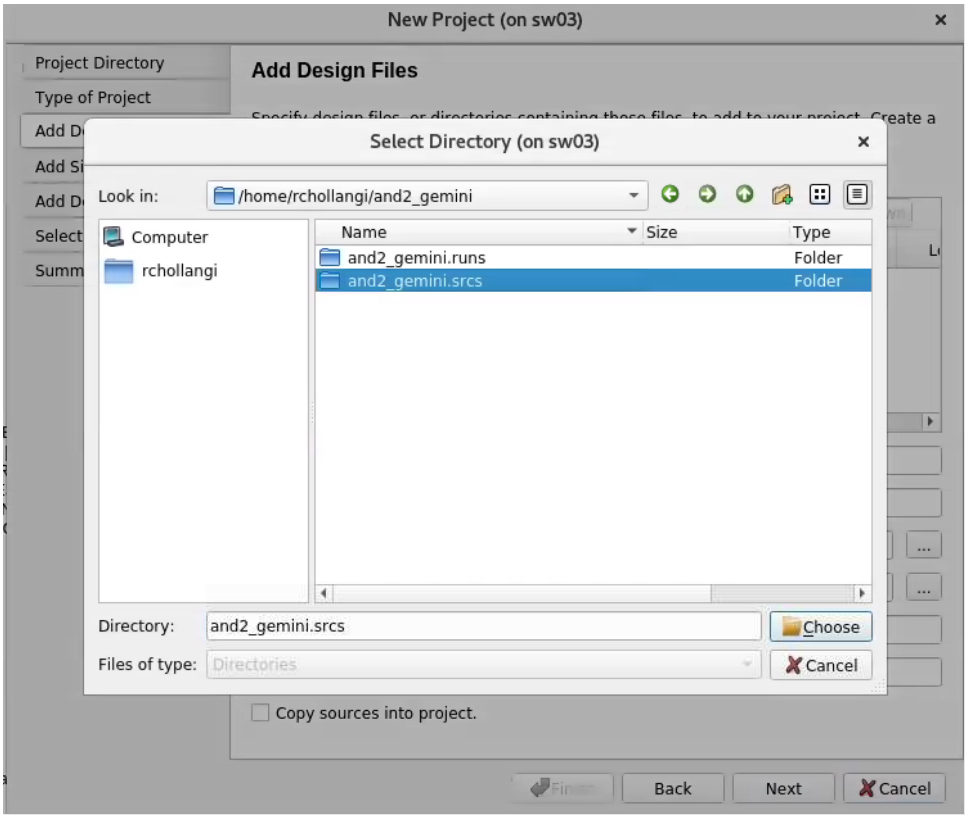


Figure 14. Scan for Supported Source Files

6. The user can also add individual source files using the "Add File" option as shown in Figure 15.

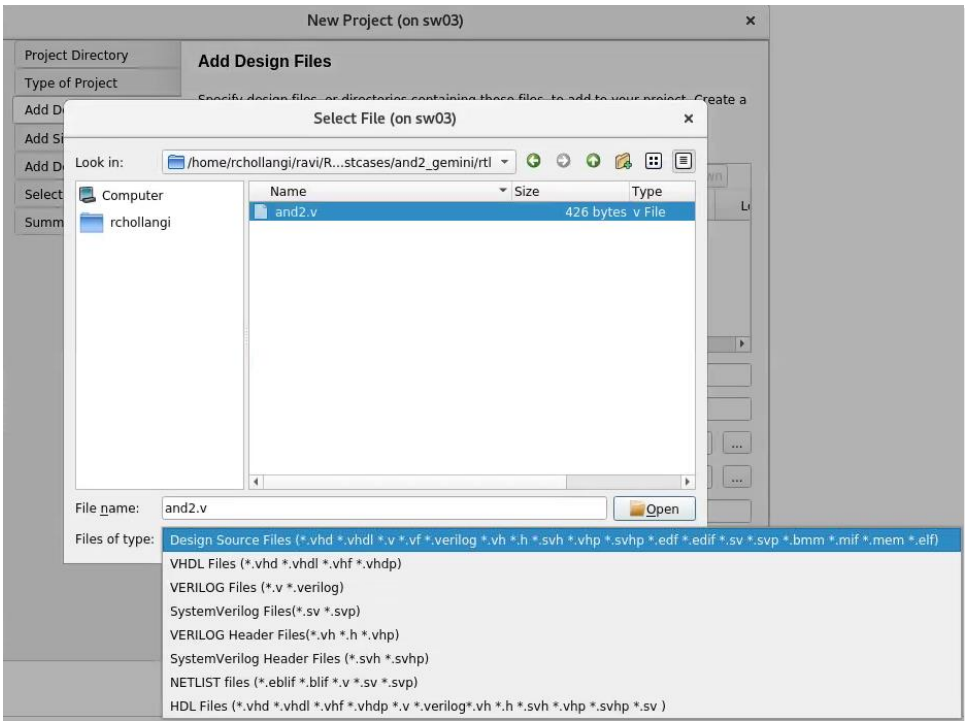


Figure 15. Add Individual Source Files

7. Once the design files have been added, the wizard shows the file(s) that will be added to the project. The user is also presented with various options as shown in Figure 16.

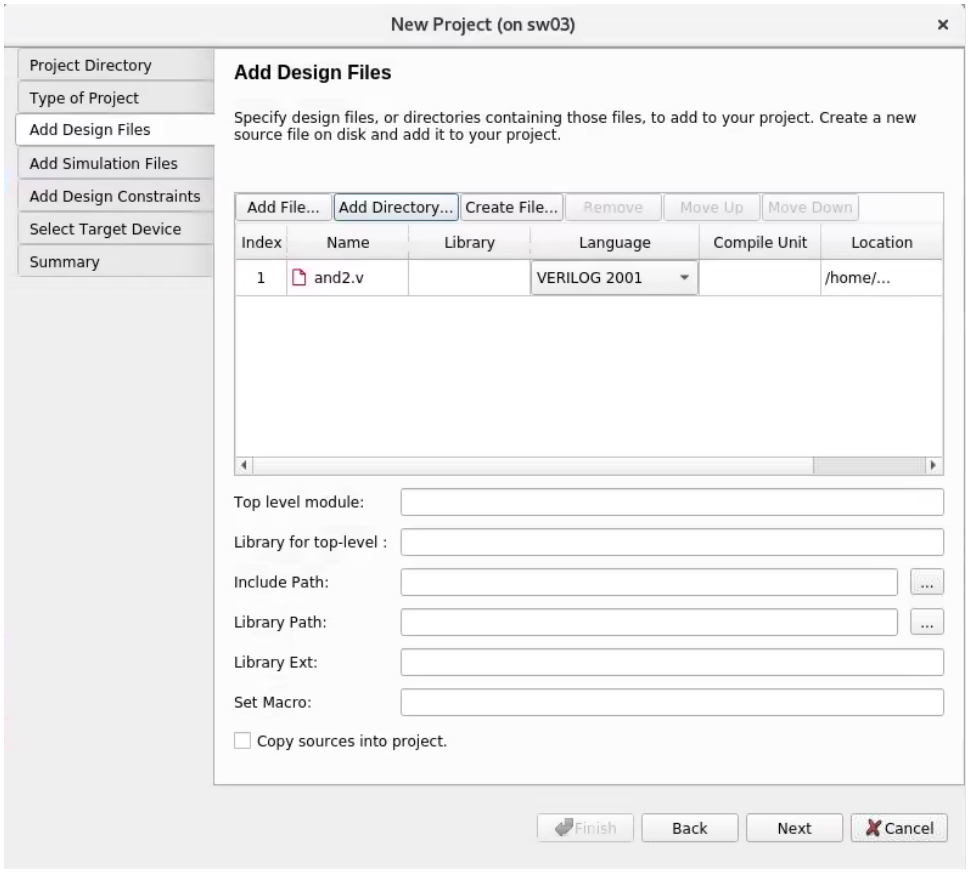


Figure 16. Add Files to the Project

8. Add Design Constraints: Similarly, the user can add or skip design constraint files as shown in Figure 17.

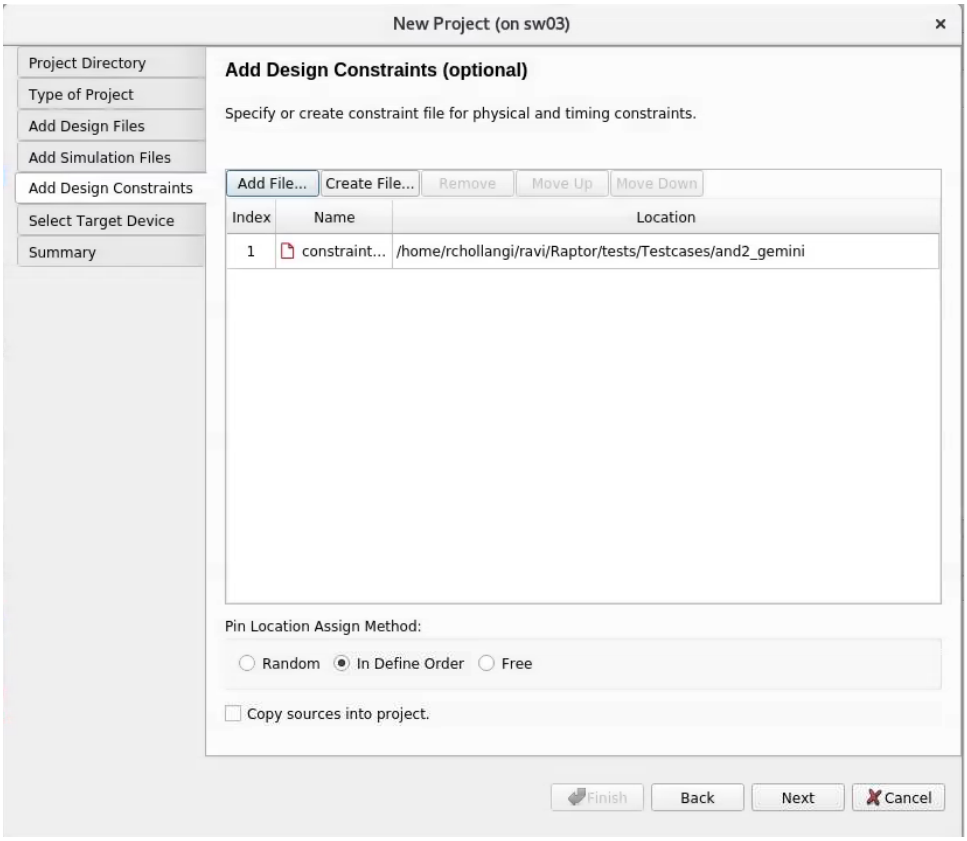


Figure 17. Add Design Constraints

9. Select Target Device: Select the supported target device in Raptor as shown in Figure 18.

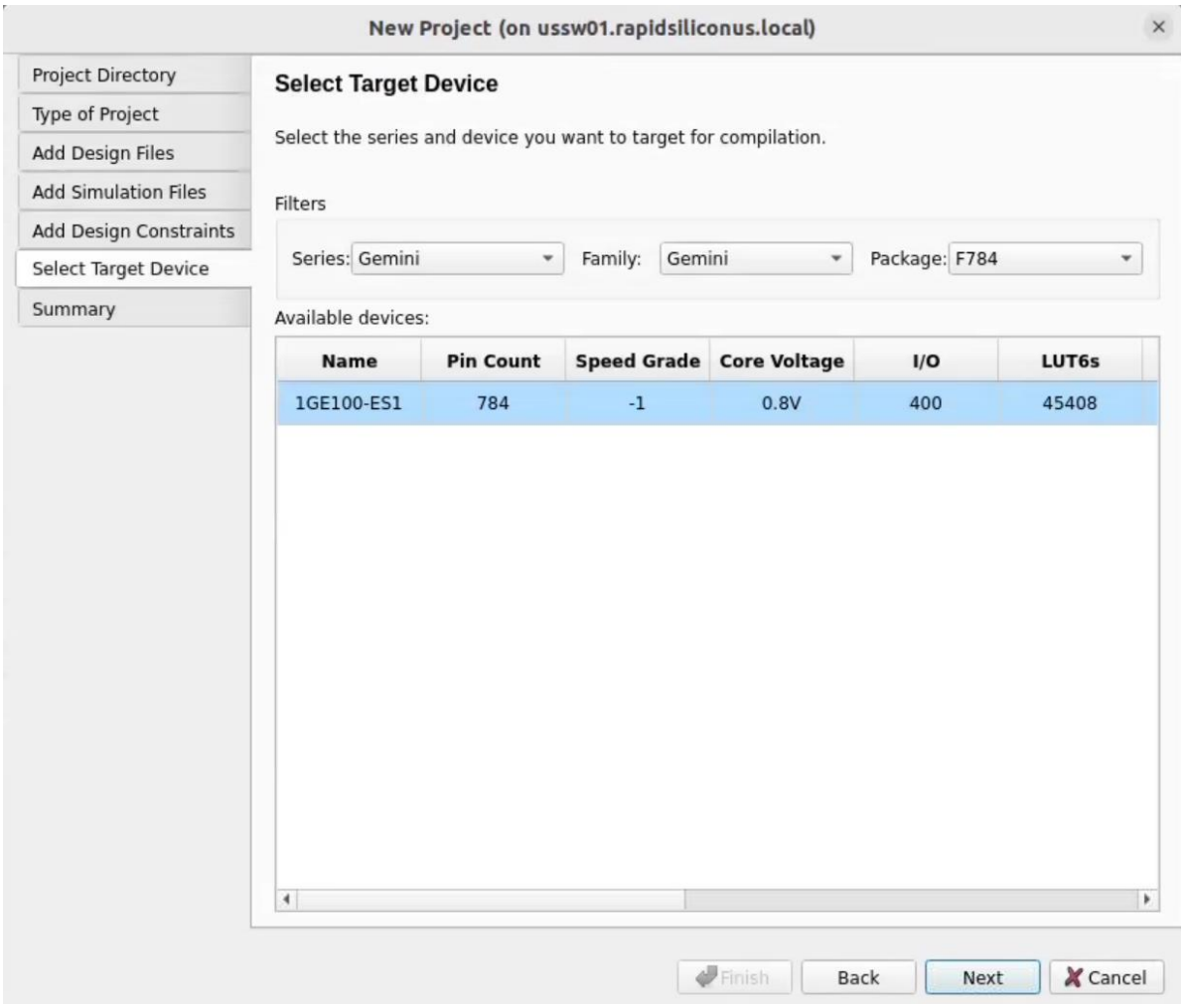


Figure 18. Select the Target Device

10. Summary: Displays the summary of project for user to finish the project creation process as shown in Figure 19.

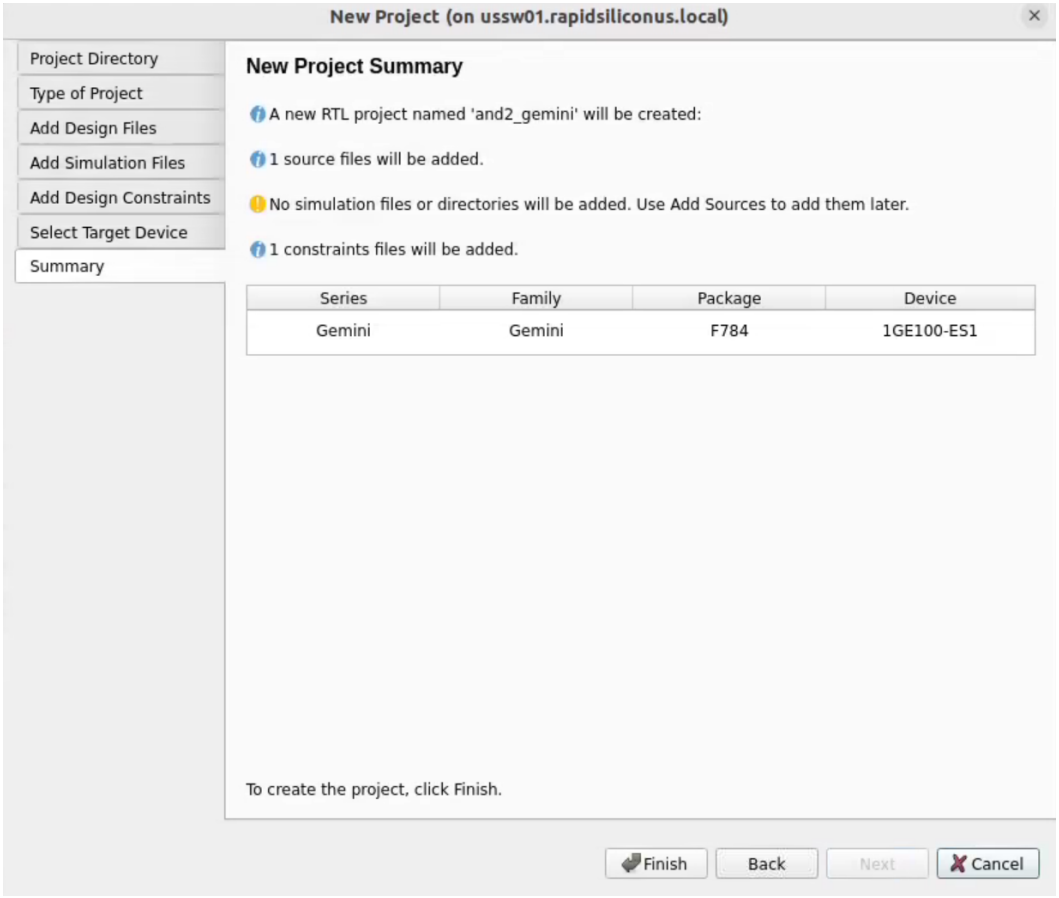


Figure 19. New Project Summary

11. The user can also edit the settings as shown below after a project has been created as shown in Figure 20.

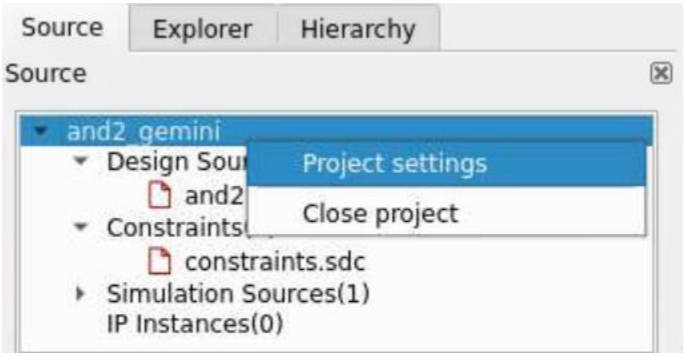


Figure 20. Edit Project Settings

12. Compile the design: Hit the play button from Menu tool bar to start the compilation as shown in Figure 21.

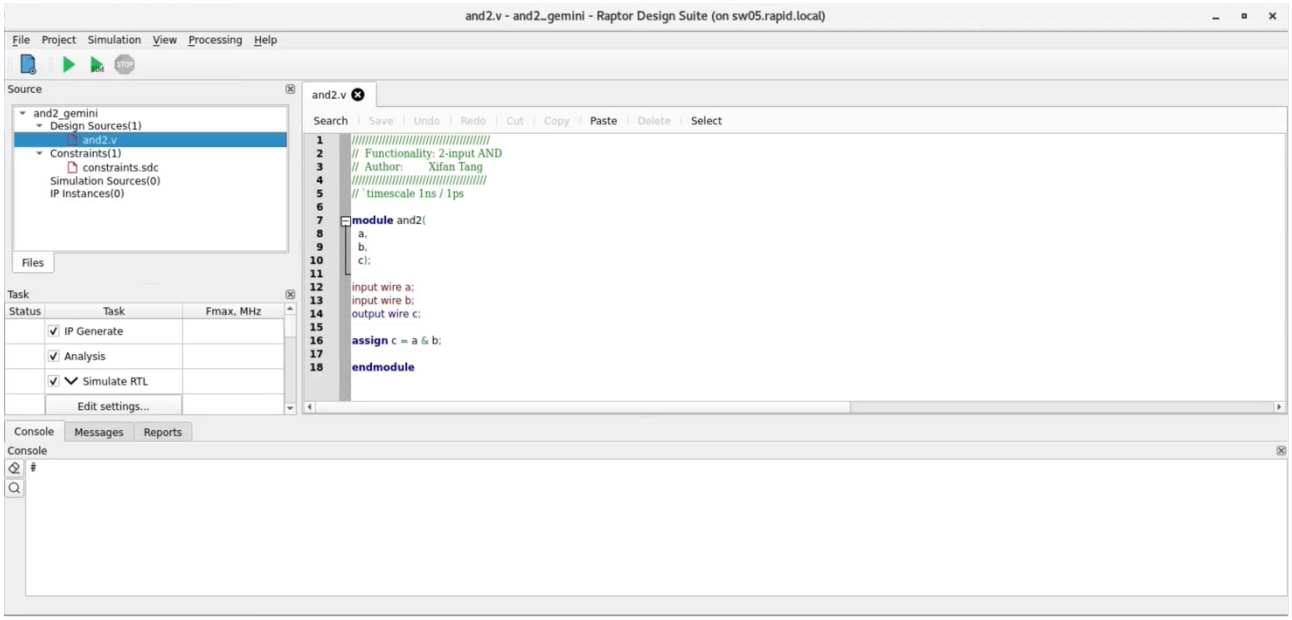


Figure 21. Compile the Design

13. Compilation in progress: During compilation, messages will be printed to the Tcl console. The task view's status column shows the current compilation step and a progress bar on top right is active until completion as shown in Figure 22.

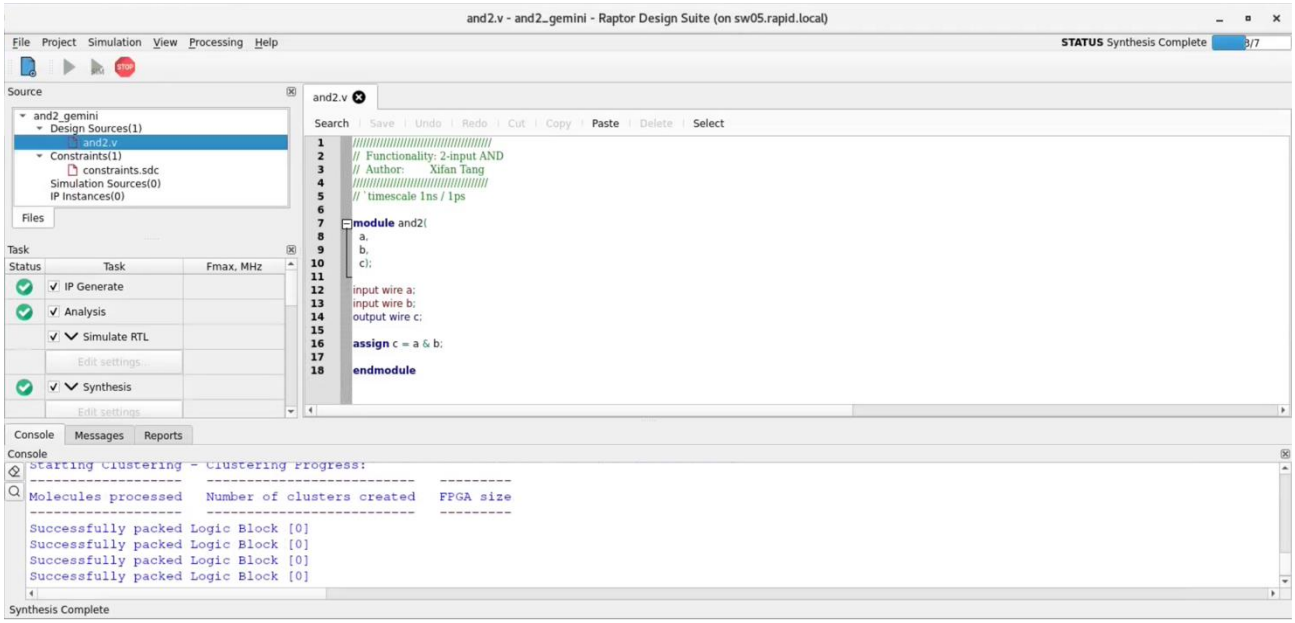


Figure 22. Compilation in Progress

14. Stop the compilation: The user can choose to stop the compilation by hitting the red Stop button in the tool bar. After stopping the user must clean the step at which the compilation was stopped before restarting the compilation from that stopped step as shown in Figure 23.

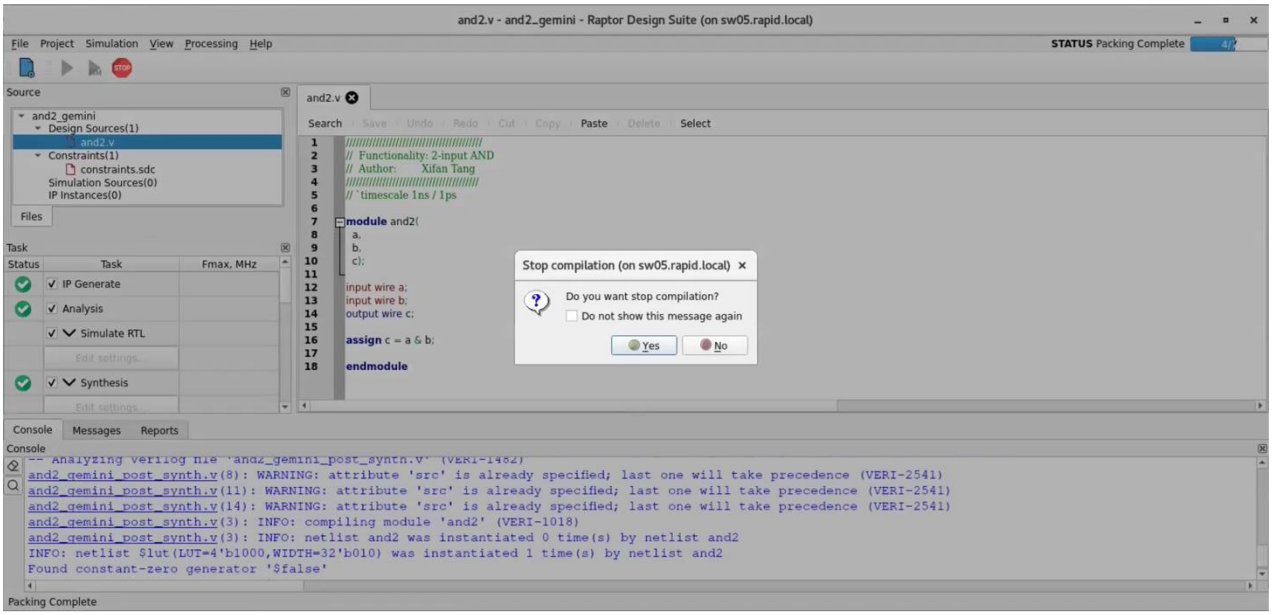


Figure 23. Stop the Compilation

15. Successful compilation: Once the compilation is finished, the user will see the bit generation completion message and all status in task view will be green as shown in Figure 24.

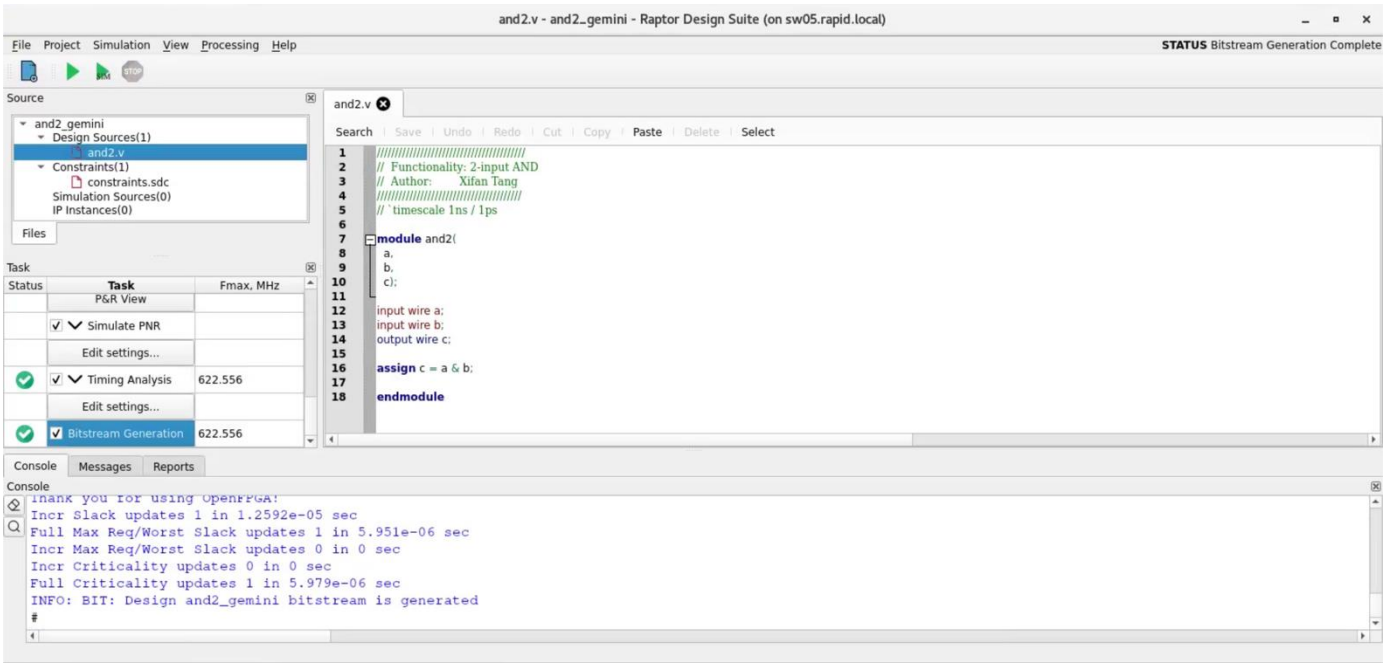
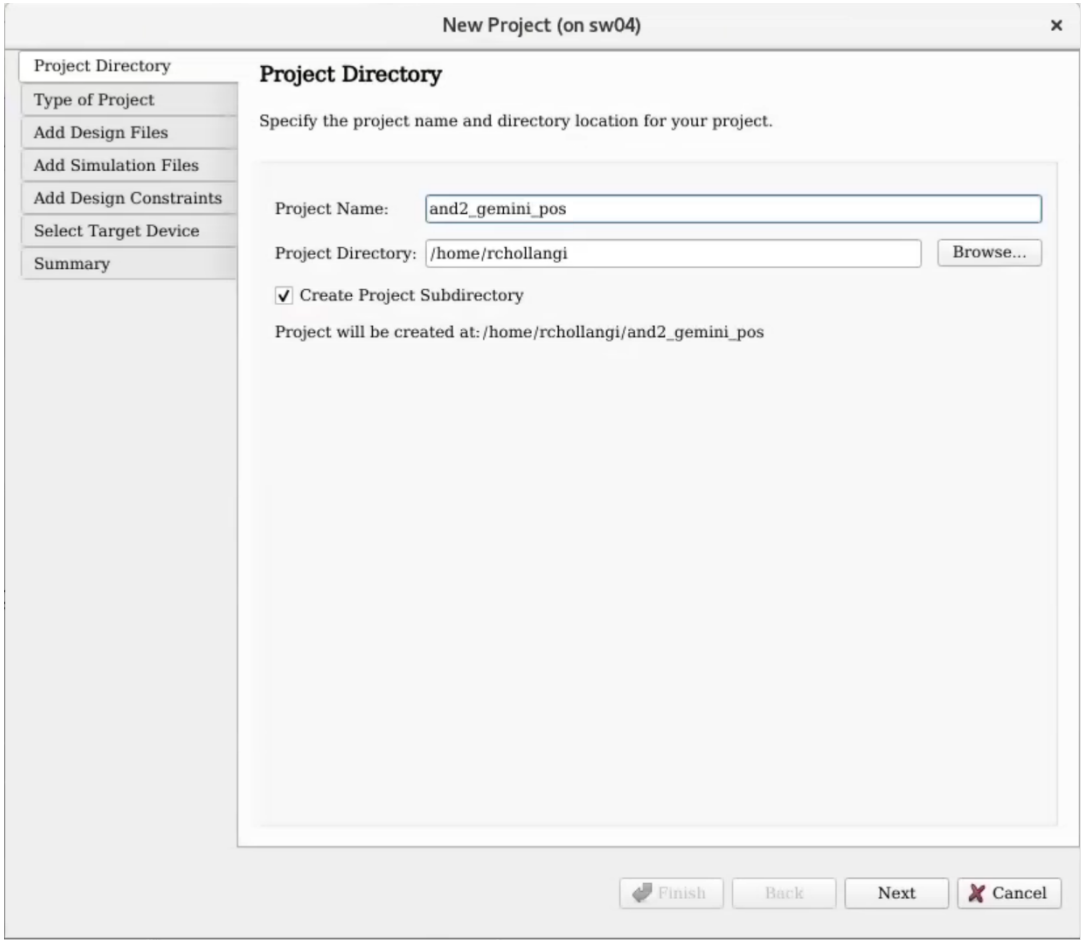


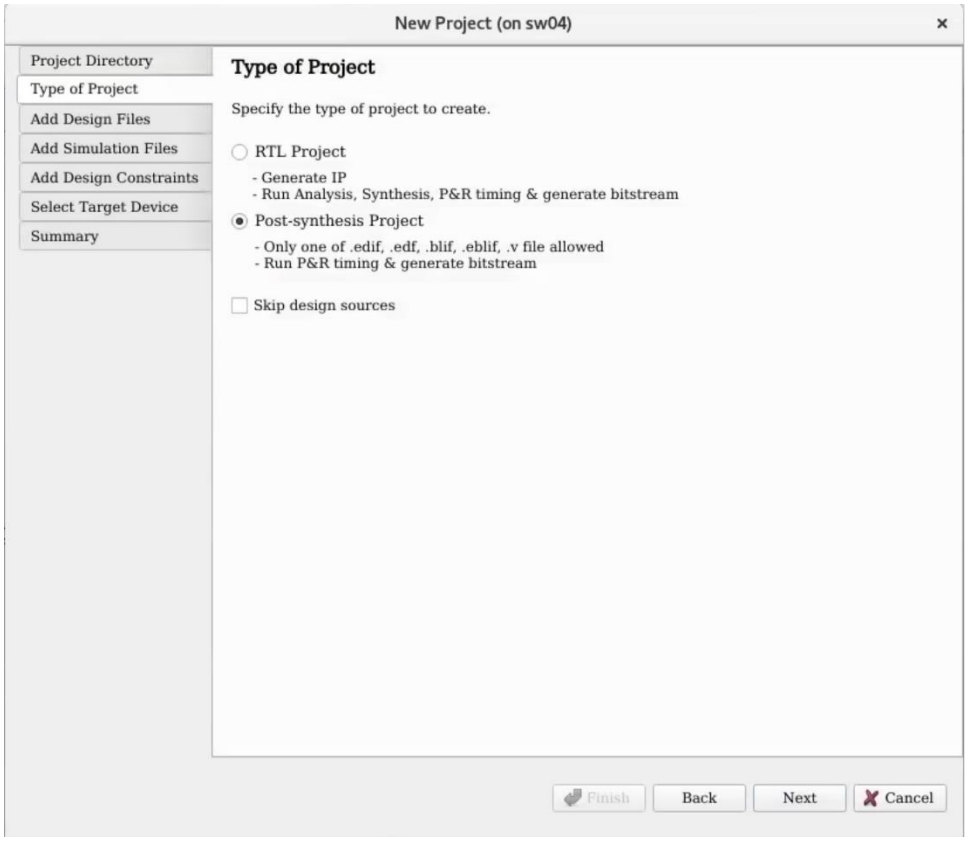
Figure 24. Successful Compilation

1.3.6.3 Creating a Post-Synthesis Project Using the Raptor Design Suite

1. Create a new project (Refer new project section)

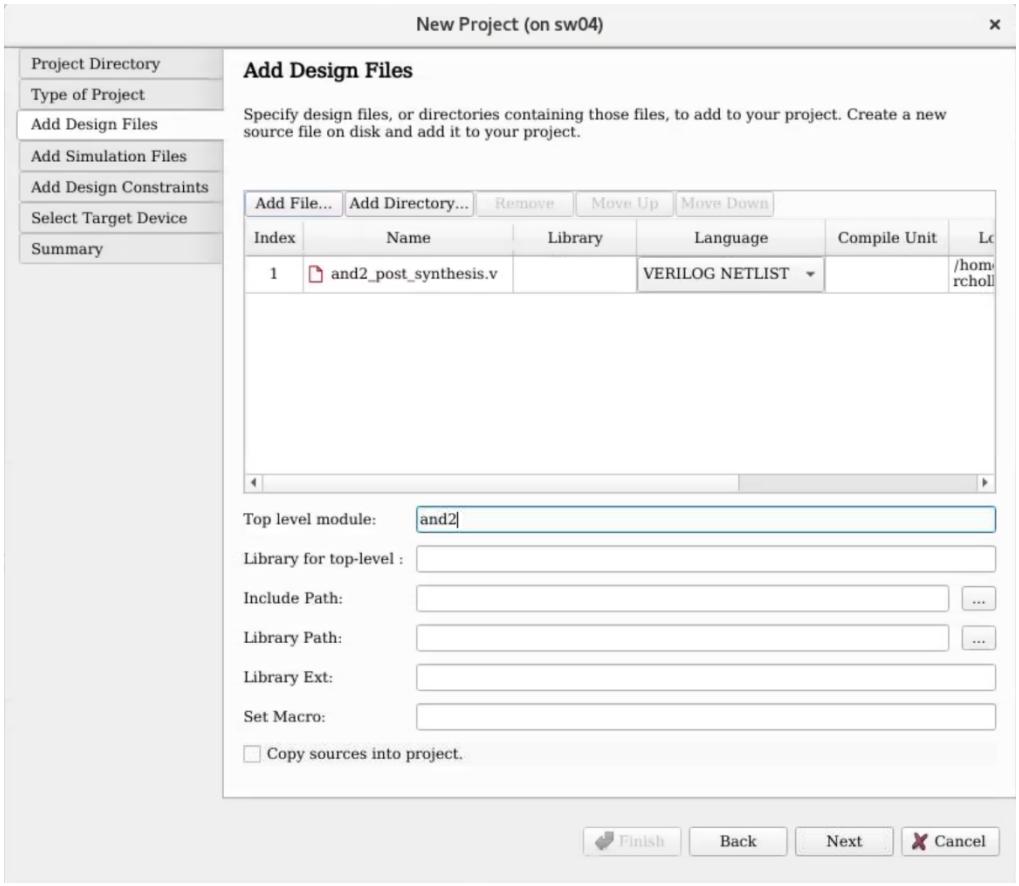


2. Select the project type as Post-synthesis project as shown below.

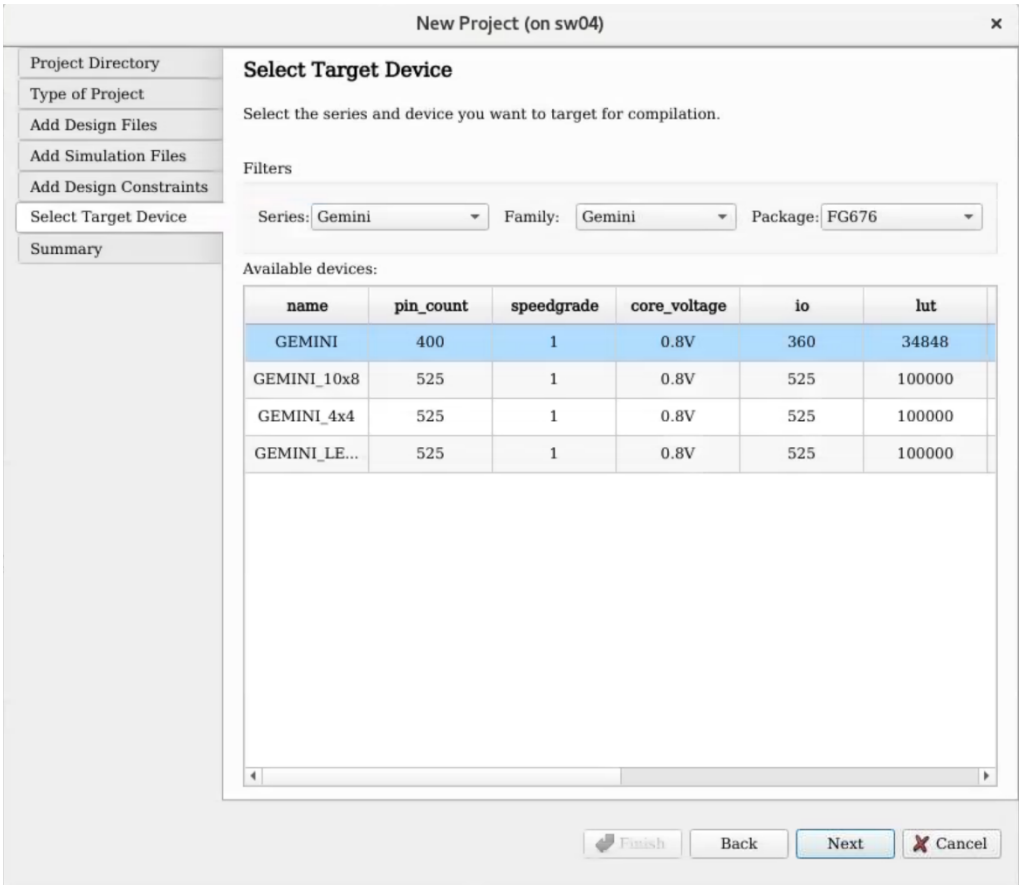


3. Add a post-synthesis Verilog netlist file (.v)

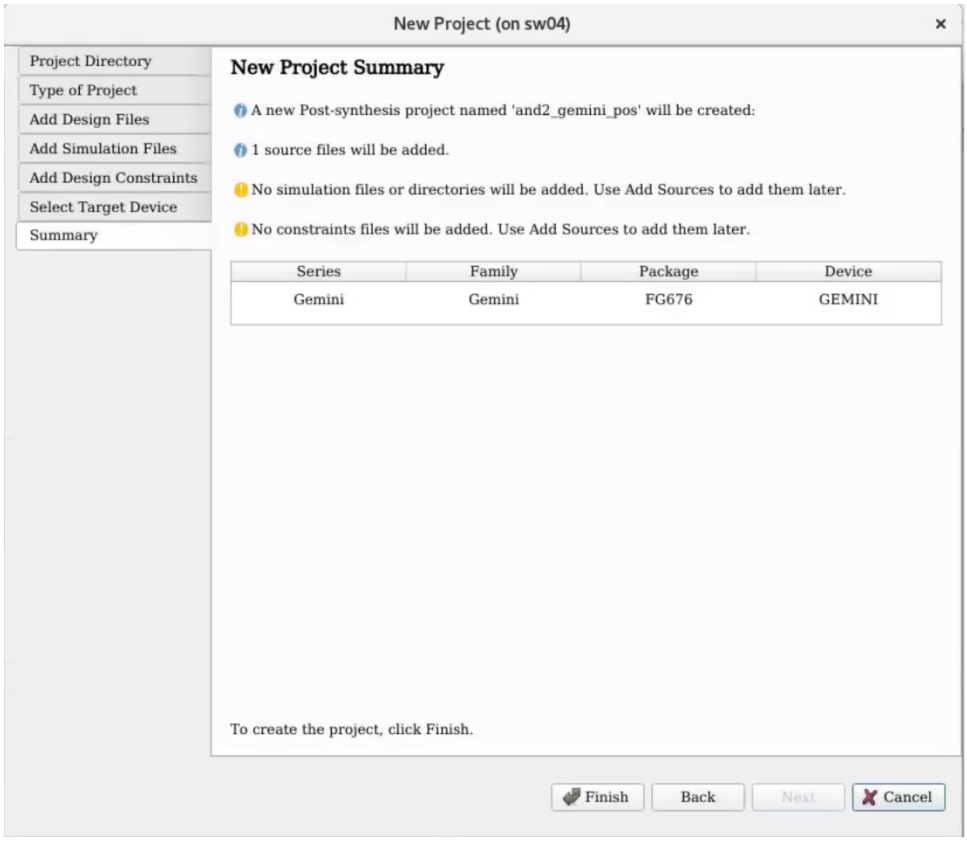
Note: We only support one type of .edif, .edf, eblif, .v for this project type.



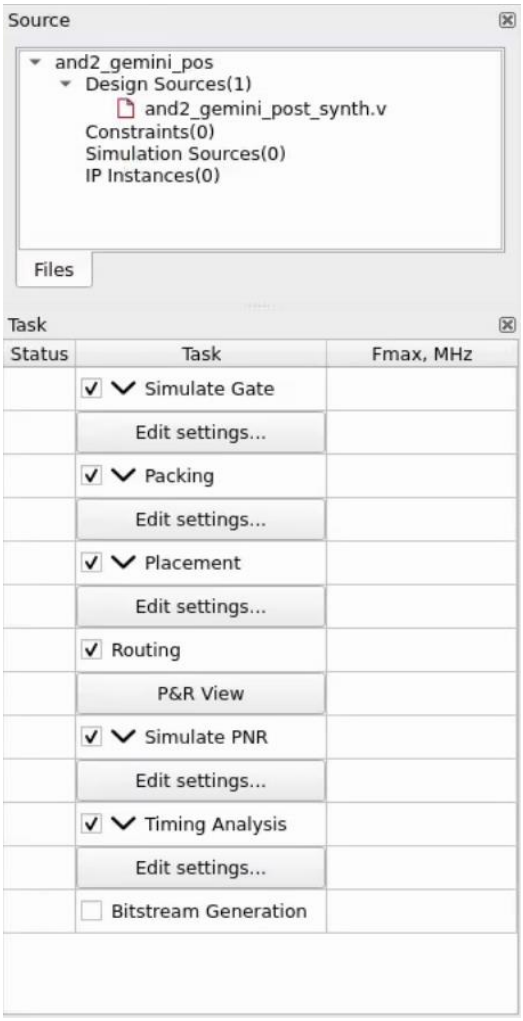
4. Add other files as needed. We will skip them and select the target device.



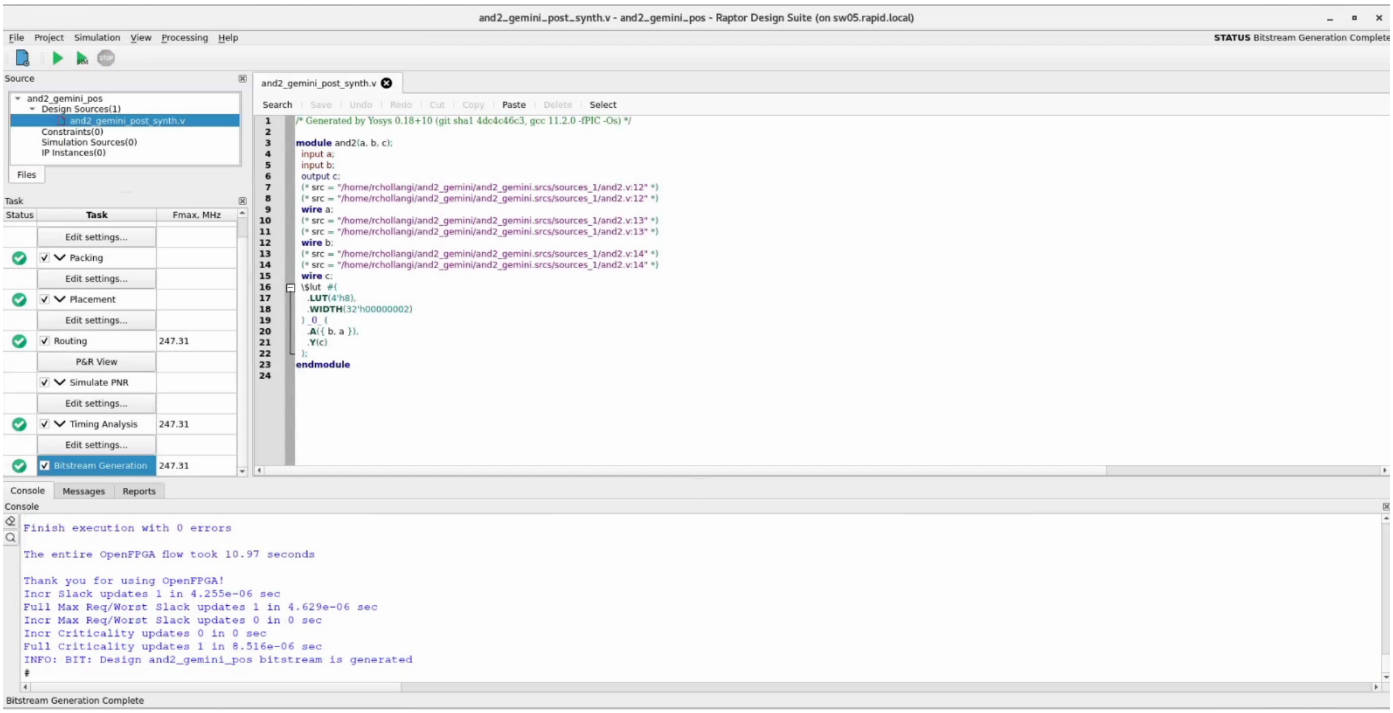
5. Review the project summary and hit "Finish"



6. In this project type, note the difference in task window below. The user can start compilation from "Simulation Gate" or "Packing".



7. Go through different compilation steps or use the run button to go through available tasks sequentially. At the end, the bitstream will be generated as shown below.

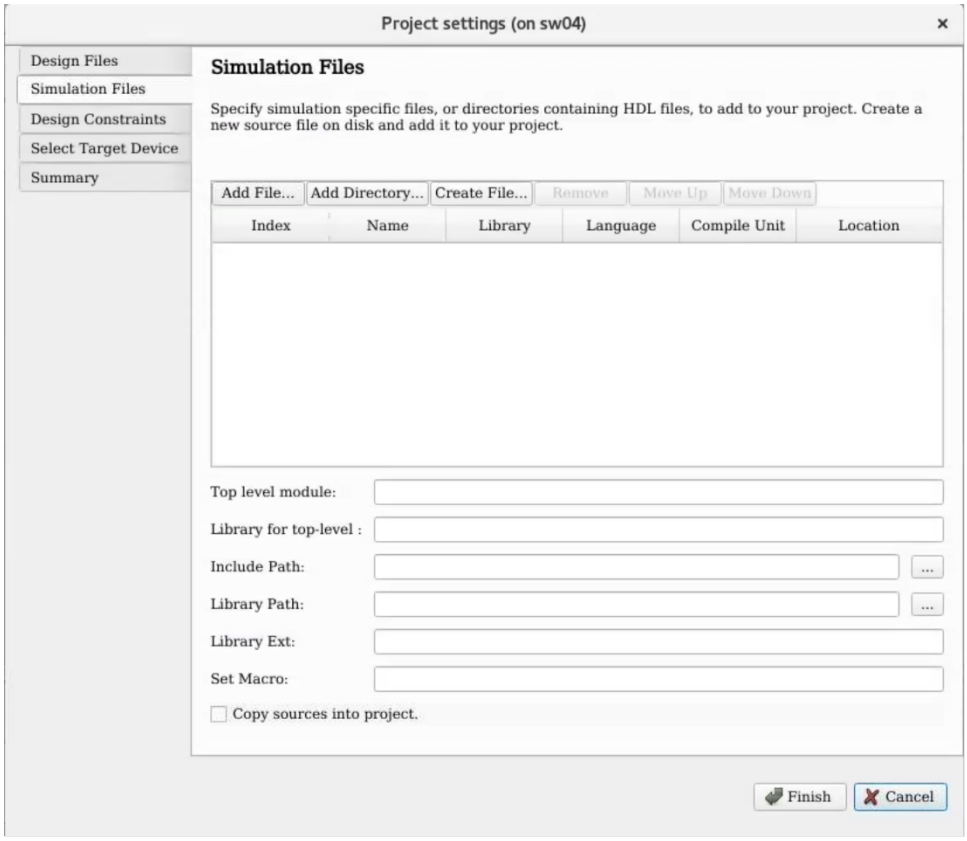


1.3.6.4 Creating Project with Simulation

Raptor provides simulation option in between compilation tasks as shown in Figure 4. Each simulation option can be configured separately using respective settings option in the task manager.

a. Simulate RTL
b. Simulate Gate
c. Simulate PNR

For the and2_gemini RTL example shown in Section 1.3.6.3 above, a user can add simulation files either during the project creation stage or using the project settings as shown below.



The following button can be used to run the design compilation tasks along with the simulation ones.

By default, all simulation tasks are enabled but a user can enable or disable them using the checkbox.

Once the simulation is complete, open the waveform file from the right click menu.

To generate waveforms, the testbench is expected to contain:

- $dumpvars statement for Icarus
- C waveform dump statement for Verilator (See Verilator documentation)
- command line options (Simulation runtime option) for GHDL.

Raptor Design Suite supports three native simulators:

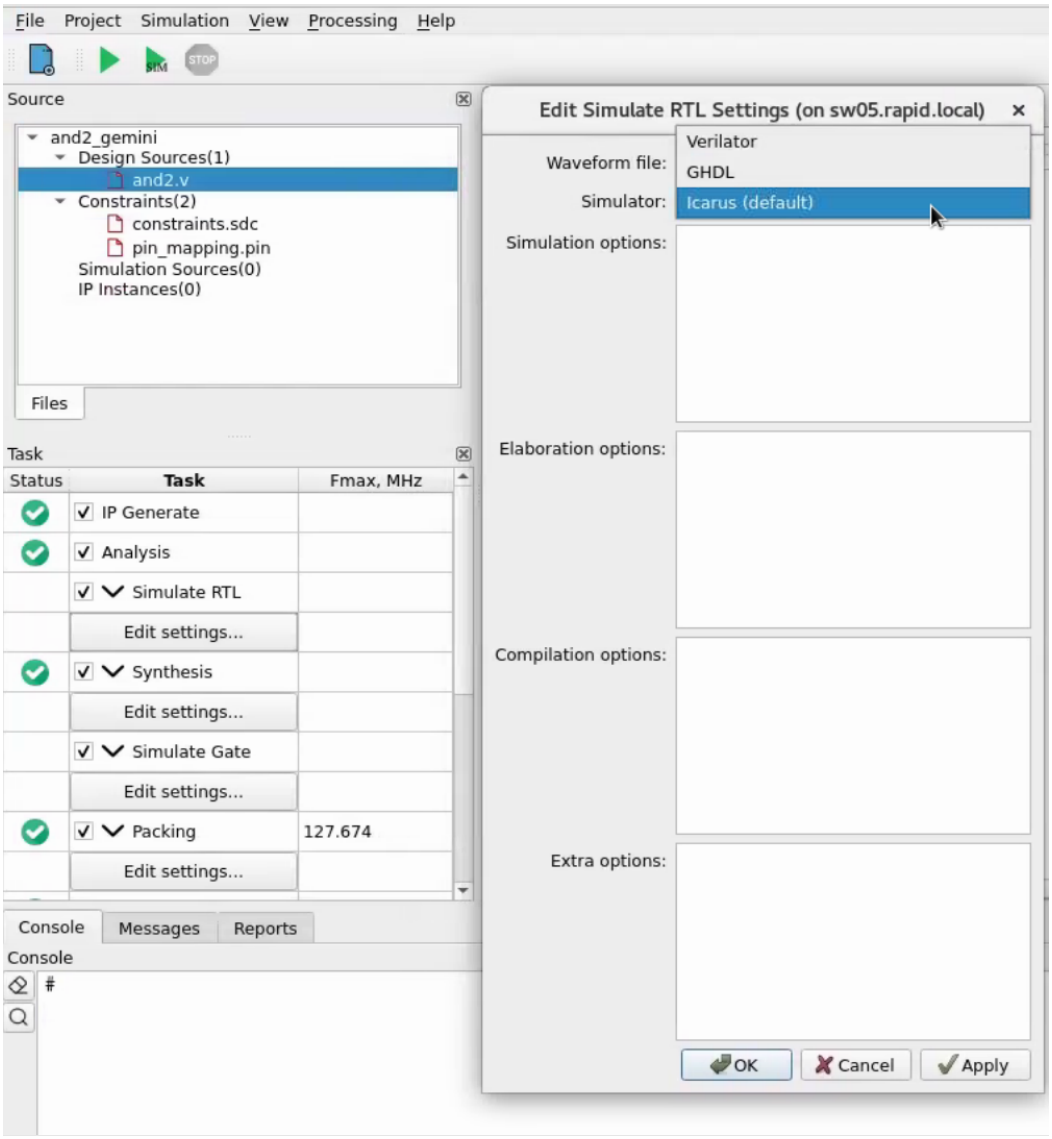
Simulator	Supported HDL	Reference Link
Icarus Verilog (**Iverilog**)	Verilog	Icarus Verilog
Verilator	Verilog & System Verilog	Overview — Verilator 5.007 documentation
GHDL	VHDL	Quick Start Guide — GHDL 0.36-dev documentation (gritbub-ghdl.readthedocs.io)

Note:

- Raptor UI by default is configured to use **Iverilog** to compile and simulate the design for functional simulations.
- Mix HDL language is not supported by Raptor's native simulators

How to select the simulator for your design simulations:

- Select the edit setting of Simulator RTL option, this will allow user to select between three different simulators



Note: simulator by default create simulation waveform in **.fst** file format in project repo, user should use GTKWAVE to load simulation waveform manually.

The file extension of the filename used in the "Waveform file" field (Or the "simulate" Tcl command) dictates the waveform file format. All formats are supported by GTKWave.

.fst for FST

.vcd for VCD

.ghw for GHW (GHDL format

TCL Commands:

Tcl Command	Arguments	Description
set_top_testbench	<module>	Sets the top level testbench module/entity.
simulation_options <simulator>	<task>	Sets the simulator specific options for the specified simulator task Compilation, elaboration, simulation
simulate <level>	<level> rtl gate Pnr ?<simulator>? ?<waveform_file.ext>?	Simulates the design and testbench. RTL simulation Post-synthesis simulation Post place and route simulation verilator, ghdl, icarus (Default) .ext supported: .fst, .vcd, .ghw
wave_*		All wave commands will launch the GTKWave process if one has not been launched already.
wave_cmd …		Sends glven tcl commands to the GTKWave process. See the GTKWave documentation at: Gtkwave:: commands
wave_open	<filename>	Load a given file in the GTKWave process.
wave_refresh		Reloads the current active wave file.
wave_show	<signal>	Add the given signal to the GTKWave window and highlight it.
wave_time	<time>	Set the primary marker to <time>. Time units can be specified, without a space. For example: wave_time 100ps.

1.3.6.5 Compilation Logs

While compiling the project, each compilation step generates a log. It's a *.rpt file holding step-specific information. Logs can be opened manually from the project directory as shown in Figure 25.

Figure 25. Opening a Log Manually

Logs can also be opened via the RMB menu of the corresponding cell as shown in Figure 26.

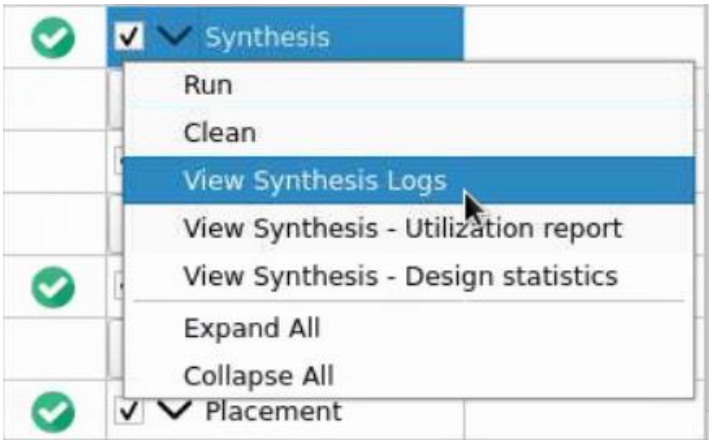


Figure 26. Opening a Log Via the RMB Menu

Log action is only available in the task column and corresponds to the selected task. Its name structure is as follows:

`View <TaskName> Logs`

Subtasks may also create log files. If they don't, Log action corresponds to the parent action, which creates the *.rpt log.

NOTE: In the current Raptor implementation, subtasks do not create log files.

In case a log file cannot be located, the action will be disabled. It may happen

if:

- The action hasn't been run yet
- The log file wasn't created (due to task failure or permission access)
- An action was executed
- The file was manually removed or cleaned

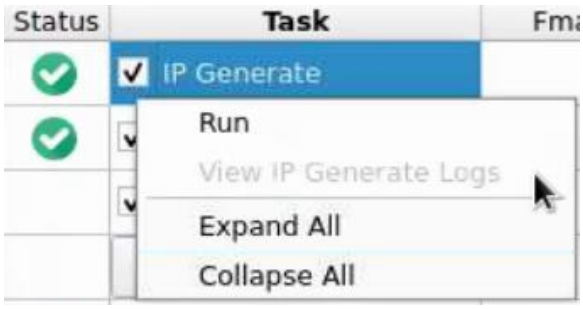


Figure 27. View Analysis Logs

Triggering a log action opens the log file in the application editor as shown in Figure 28 for synthesis log file

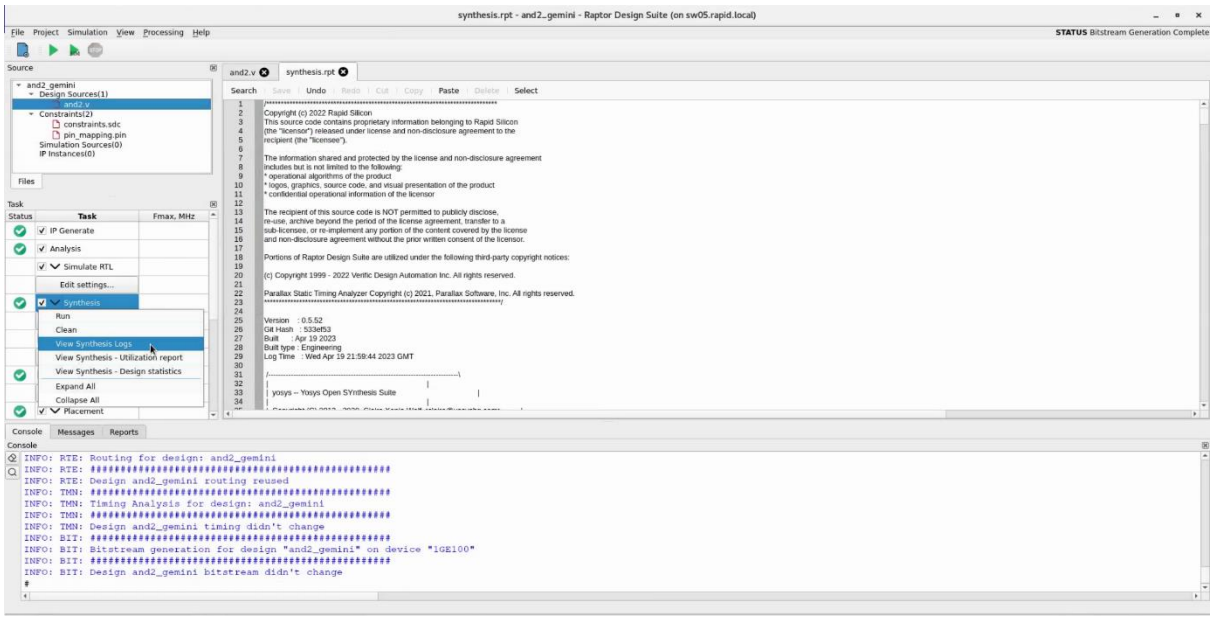
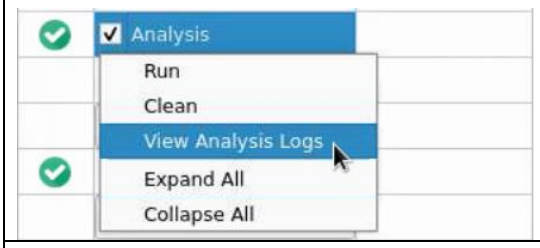
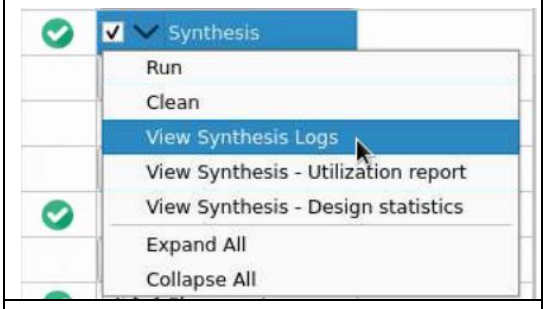
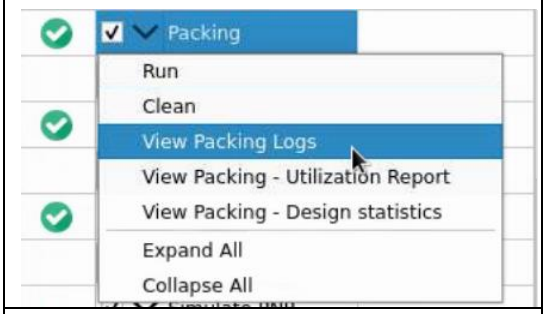
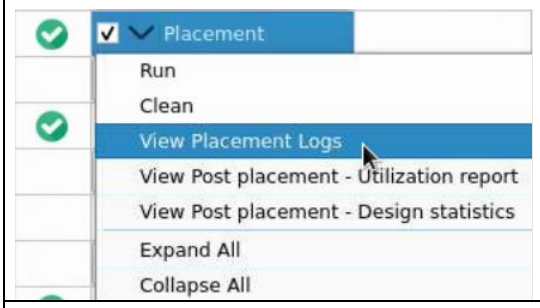
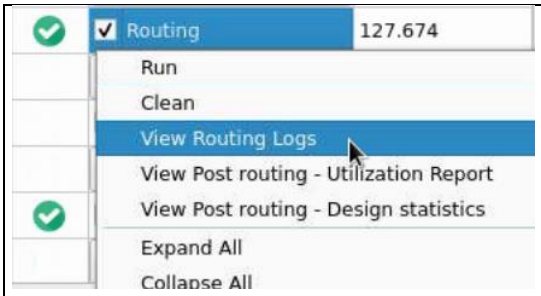
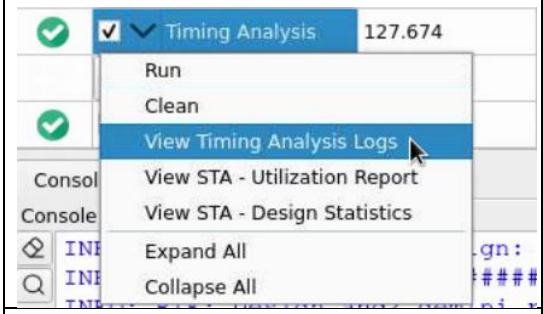


Figure 28. Open Log in Application Editor

Table 2 shows an overview of all tasks log files.

Table 2: Tasks Log Files Overview Screen

Task	File	Status
IP Generate 	ip_generate.rpt	Not Available
Analysis 	analysis.rpt	Available
Synthesis 	synthesis.rpt	Available
Packing 	packing.rpt	Available
Placement 	placement.rpt	Available
Routing	routing.rpt	Available

		
Timing Analysis 	timing_analysis.rpt	Available
BitStream Generation 	bitstream.rpt	Available

1.3.6.6 Compilation Messages

Important messages from each task log are available under the messages tab as shown in Figure 29.

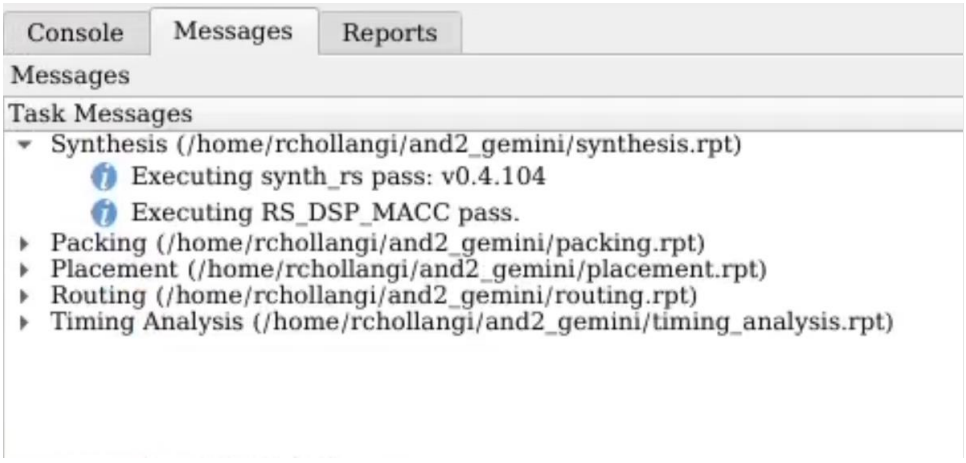


Figure 29. Compilation Messages

Messages can be expanded or collapsed as shown below in Figure 30. For example, the message highlighted under the Synthesis task, when double-clicked, opens the synthesis log file, and takes user to that line without the need for explicit search through the rpt file.

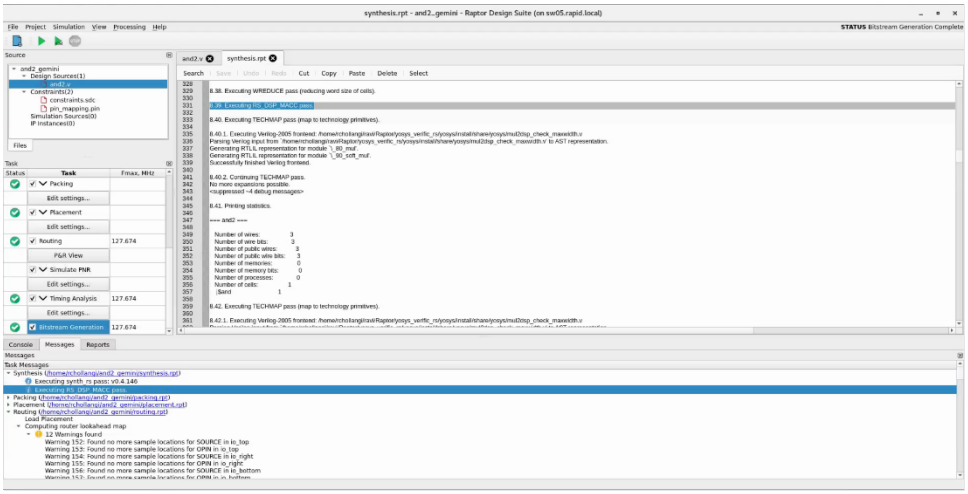


Figure 30. Expanding and Collapsing Messages

1.3.6.7 Compilation Reports

As shown in Table 2 (Task logs) above, the reports for some of the compilation tasks are also available by right-clicking on the task.

Figure 31 shows an example of a synthesis report.

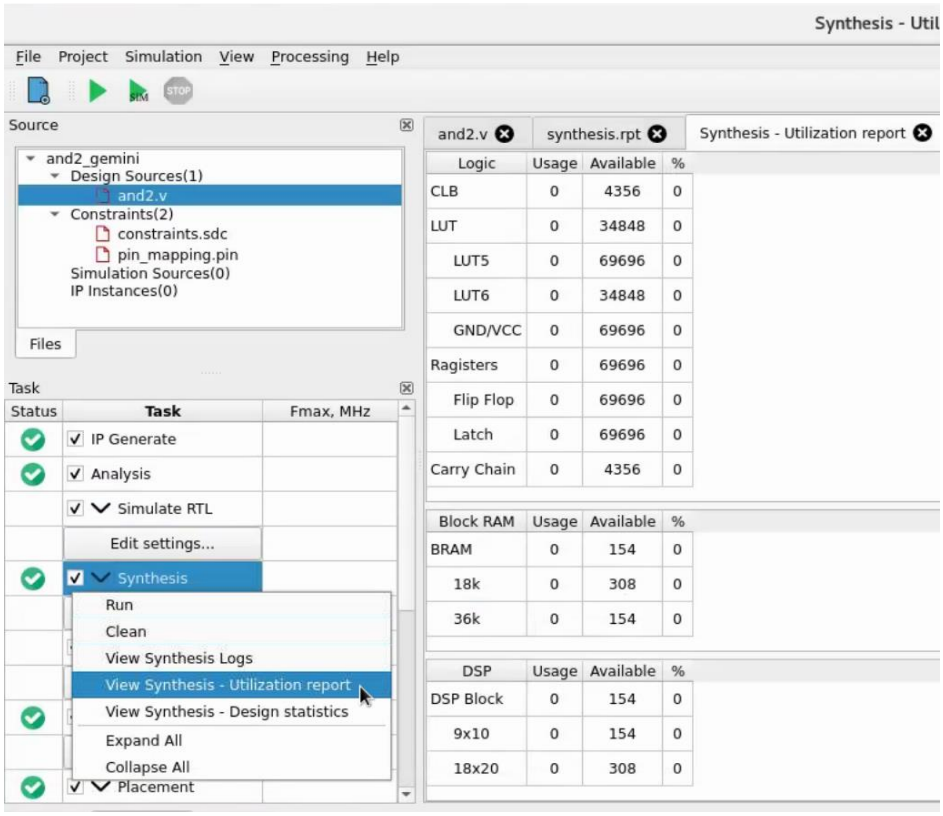


Figure 31. Example of a Synthesis Report

Some tasks might have multiple reports. Figure 32 is an example of two reports available under the synthesis task.

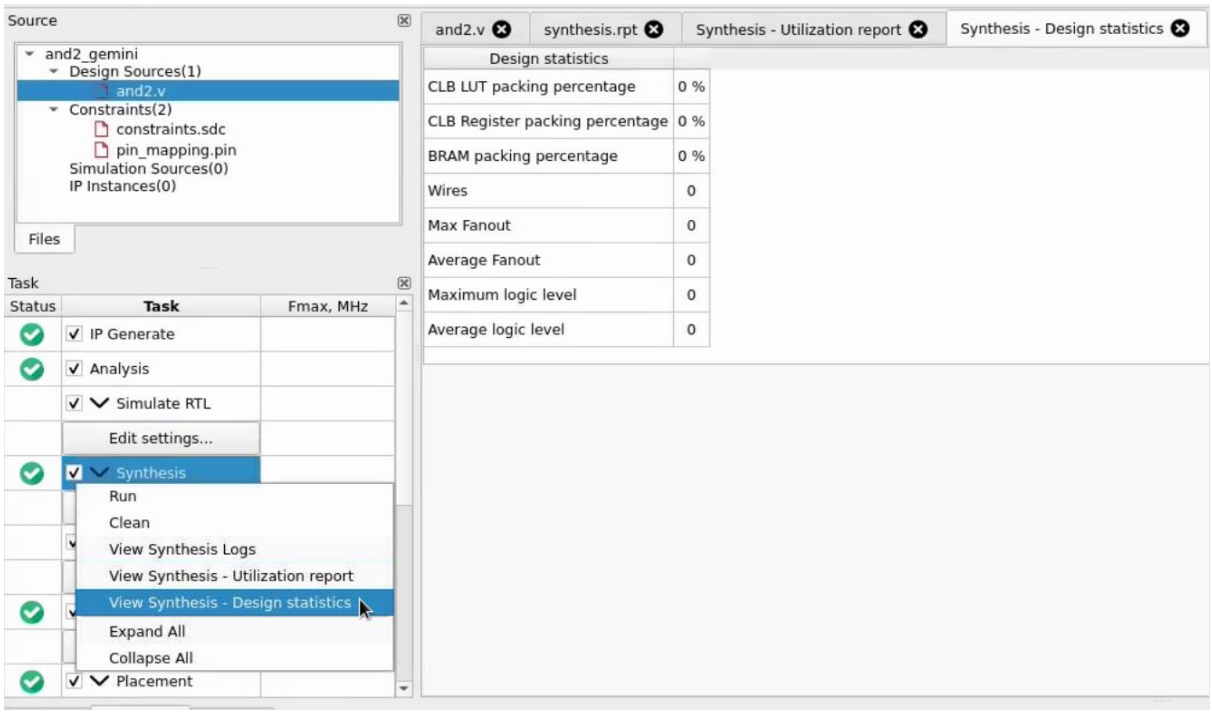


Figure 32. Viewing Multiple Reports

Reports for each task can also be accessed via the Report panel as shown in Figure 33 below.

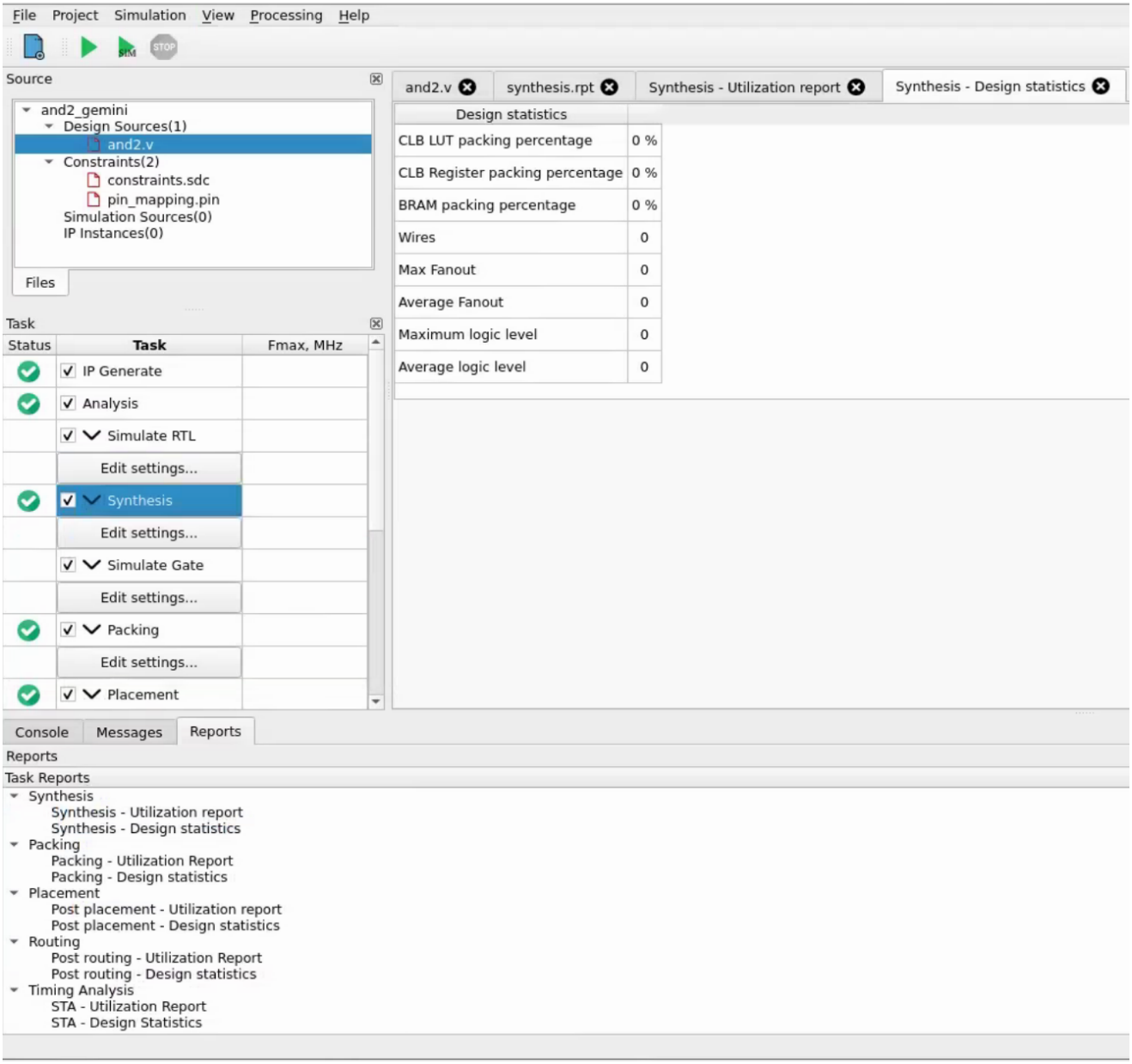


Figure 33. Reports panel

1.3.6.8 Compilation Compute Usage

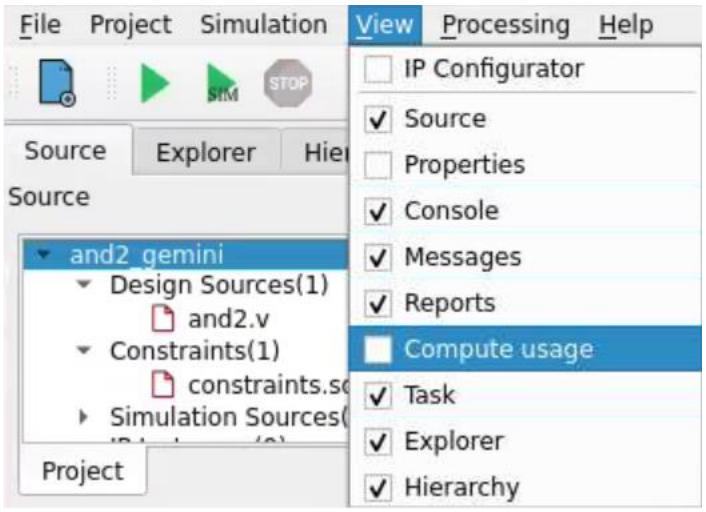


Figure 34. Enable Compute Usage View

Console | Messages | Reports | Compute usage

Compute usage

Task	Duration, s	Utilization, MB
Analysis	0.068	65.9141
Simulate RTL	N/A	N/A
Synthesis	8.178	102.727
Simulate Gate	N/A	N/A
Packing	23.425	1344.85
Placement	167.469	1344.82
Routing	81.86	1344.82
Simulate PNR	N/A	N/A
Timing Analysis	21.344	1344.82
Bitstream Generation	N/A	N/A

Figure 35. View Compute usage for current compilation process once completed

Understanding the data in the table:

1. Result is available only after task successfully done.
2. Result is N/A in case:
    a. Task was cleaned
    b. Task was skipped because of no design change.
    c. Project was opened.

3. For simulation task:
    a. Duration is a sum of three (elaboration, compile and simulate)
    b. Utilization is maximum of three (elaboration, compile and simulate)

1.3.7 Configuration Project Settings

The project settings can be configured using the Settings dialog box. These settings include:

- General settings
- Synthesis
- IP integration
- Simulation
- Elaboration
- Implementation
- Bit-stream

1.3.8 IP Configurator

The Raptor IP configurator is used to generate IPs. The IP configurator must be enabled to view the IP catalog. The IP configurator can be enabled as shown in Figure 36.

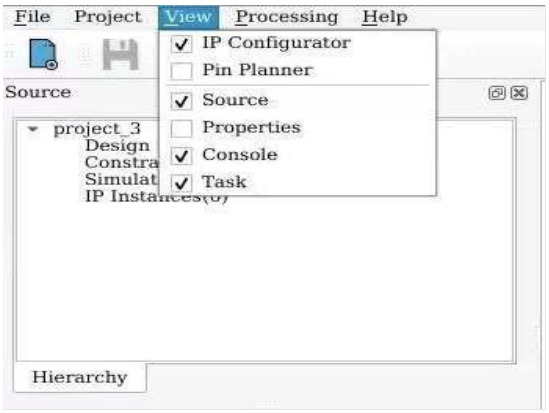


Figure 36. Enable IP Configurator

This action opens the list of IPs from the Raptor IP Catalog as shown in Figure 37. The right side of the figure shows a list of the available IP's.

Figure 37. List of Raptor Catalog IP's

Select the IP from the list of available IP's, then configure the selected IP and provide the output module name for the IP you want to generate. When finished, click *Generate IP* in the lower right-hand corner of Figure 38.

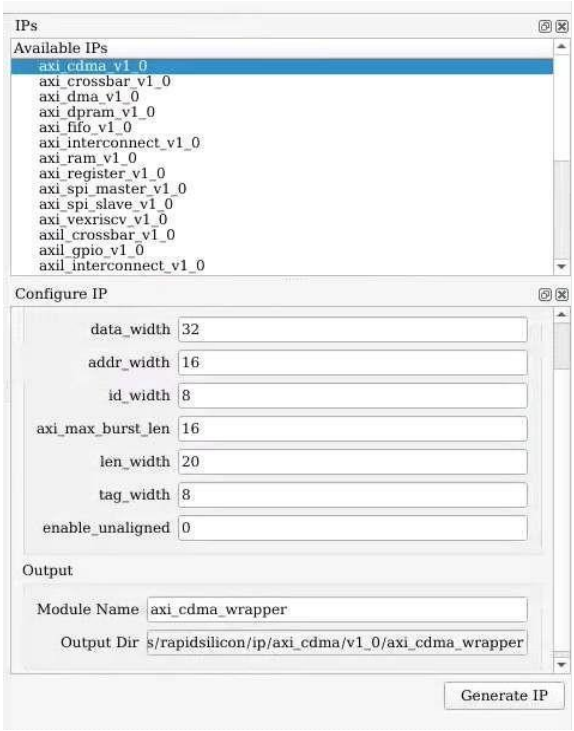


Figure 38. IP Configuration Window

Clicking on the Generate IP button above creates the IP in your Raptor project as shown in Figure 39.

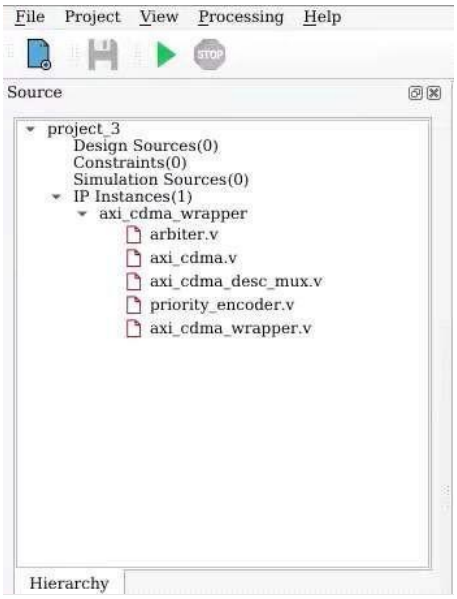


Figure 39. Create IP in the Raptor Project

After this you can add this generated IP part of your Raptor project using the Add Design Files option in the project wizard.

Note: All Soft IP collaterals are located in the Raptor Install area path: <Raptor Installation area>**/share/raptor/IP_Catalog**

File Type	Path of Directory
Verilog file	./IP_Catalog/rapidsilicon/ip/<IP_Name>/<IP_Version>/src
Simulation scripts	./IP_Catalog/rapidsilicon/ip/<IP_Name>/<IP_Version>/sim
Timing Constraints	./IP_Catalog/rapidsilicon/ip/<IP_Name>/<IP_Version>/sdc
IP User Guide	./IP_Catalog/rapidsilicon/ip/<IP_Name>/<IP_Version>/doc
LiteX Integration Python wrapper	./IP_Catalog/rapidsilicon/ip/<IP_Name>/<IP_Version>/litex_wrapper

1.3.9 Editing Properties

The synthesis and placement tasks have settings that can be edited via the "Edit settings…" button beneath the respective task in the Task Window as shown in Figure 40.

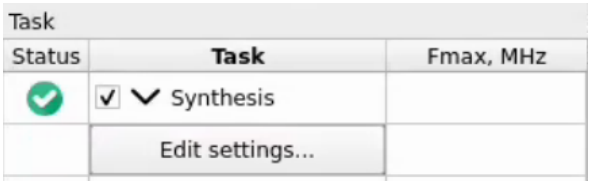


Figure 40. Edit Setting in Task Window

The synthesis settings can then be edited by selecting the options shown in Figure 41.

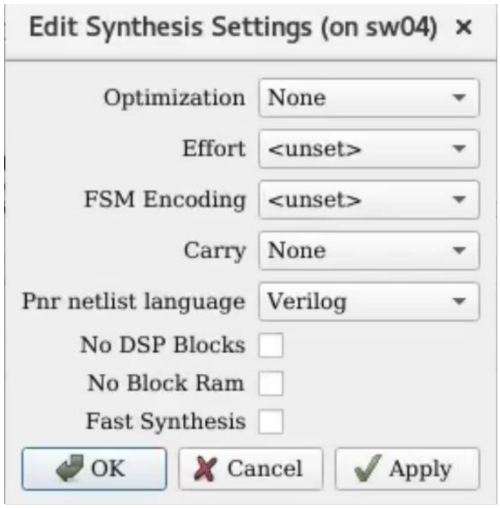


Figure 41. Edit the Synthesis Settings

The placement settings can be edited by selecting the options shown in Figure 42.

Figure 42. Edit Placement Settings

1.3.10 Using the Text Editor

The Raptor design suite text editor is a configurable, integrated text editor that supports:

- File comparison
- Code folding
- Code completion
- Syntax checking and highlighting
- Errors and warning indications

1.3.10.1 Text Editor Commands

The text editor provides the following actions via its tool bar.

Figure 43. Text Editor Command Tool Bar

Each selection on the tool bar is described in the following subsections.

Save

Save any changes made to the current file.

Undo

Undo the previous change to the text.

Redo

Restore the text that was changed by the previous Undo action.

Cut

Copy and remove the selected text. This text can be pasted elsewhere.

Delete

Delete the selected text without changing your currently copied text.

Select

Select all text in the current file.

1.3.10.2 Opening a Text File

Files can be opened from the menus via File → Open File as shown in Figure 44.

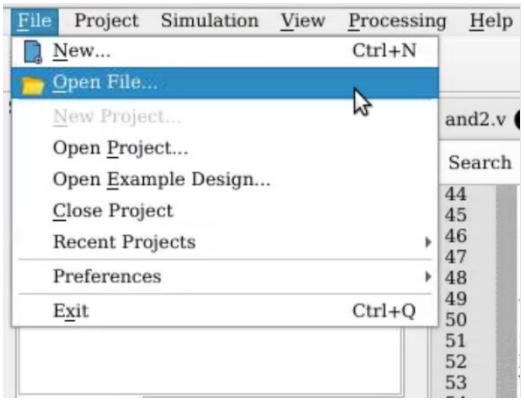


Figure 44. Opening an Existing Text File

1.3.10.3 Creating a New Text File

A new file can be created from the menus via File → New as shown in Figure 45.

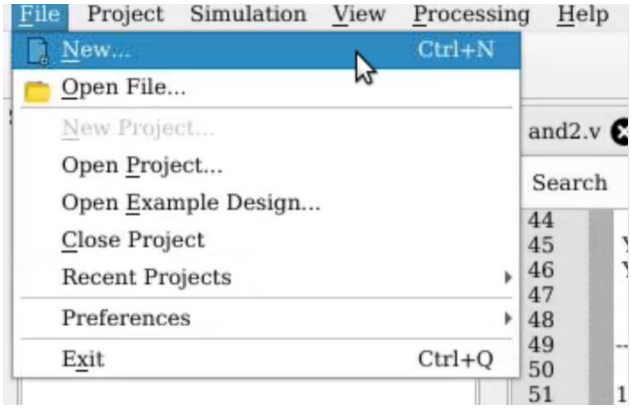


Figure 45. Creating a New Text File

A new file can also be created by clicking the "New" button in the tool bar as shown in Figure 46.

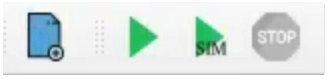


Figure 46. Click the Blue Folder Button to Create a New Text File

1.3.10.4 Using third-party editors with Raptor project files

Users can configure Raptor to edit the project files using third-party editors available on the user OS.

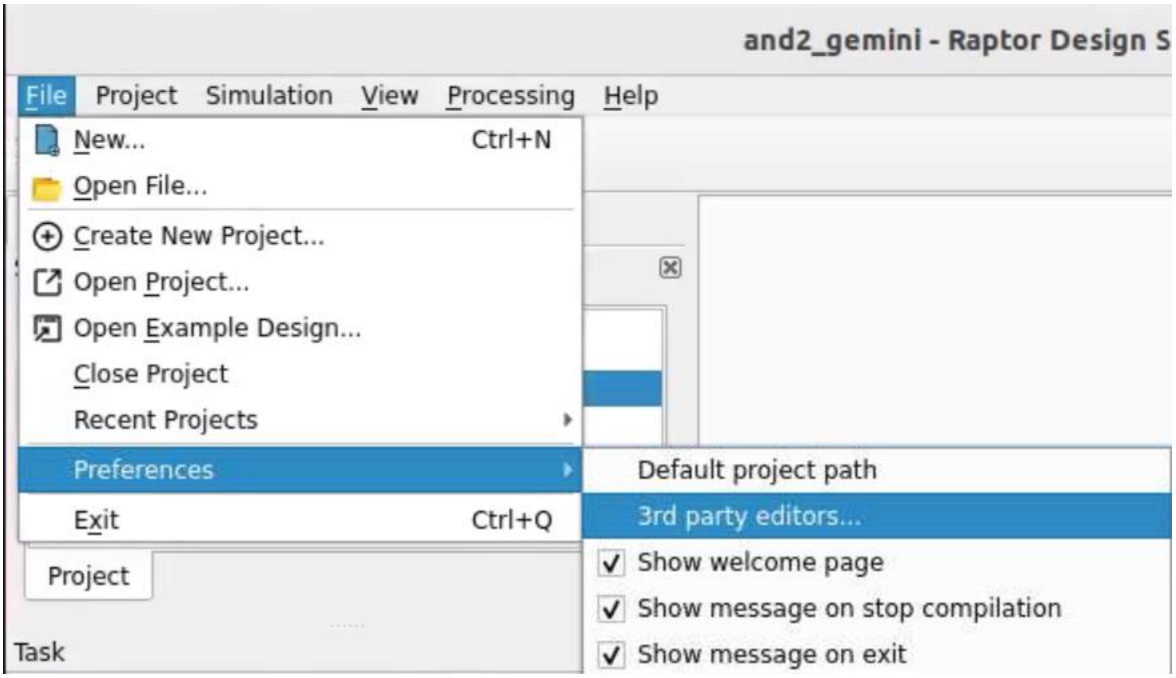


Figure 47. Set-up Third Party Editors using the Preferences Option

Set up the editors as shown below. Note that the editors must be available on the path for Raptor to find them.

Figure 48. Set-up gedit editor on Ubuntu

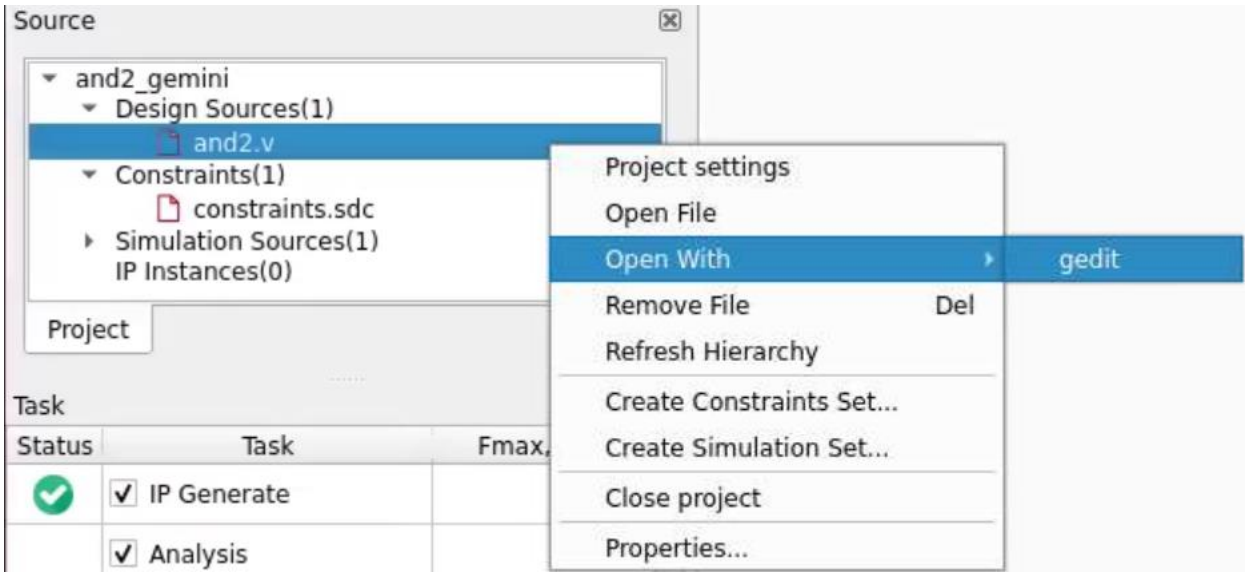


Figure 49. Open any project file using gedit

Figure 50. Interact with the project file opened with gedit

Note: In a future release, we plan to embed the third-party editors within Raptor windowing system itself.

2 Raptor Synthesis

2.1 Introduction

Synthesis is the process of converting an RTL design into a gate-level representation.

As mentioned in Section 1.2.1 "Project and Tcl Modes", there are two design flow modes available in the Raptor design suite:

- Project mode
- Tcl mode

In Project mode, a project is created using the Raptor EDA tool using the GUI. This approach automatically saves the design at regular intervals. It also automatically manages the source files, and also generates reports as necessary.

Tcl mode is executed using Tcl commands or scripts. In Tcl mode, the user still has full control of the design flow as with Project mode, but the Raptor EDA tool does not automatically manage source files or report the design state. Rather, in Tcl mode, the Raptor EDA tool can be opened at each stage of the design in order to manage the source files and perform design analysis and constraints assignments.

2.1.1 Using Synthesis

The Raptor design suite includes a synthesis tool used to synthesize the design. The tool manages the run data automatically and allows for repeated run attempts with various RTL source versions and target devices.

This section describes how to use the Raptor design suite to set up and run Raptor synthesis.

2.1.2 Running Synthesis

A *run* defines and configures the various aspects of the design that are used during synthesis. A *synthesis run* defines the following:

- Target device during synthesis
- Synthesis constraint to be applied
- Launching a synthesis run
- Controlling the synthesis results

2.1.3 Incremental Synthesis

Raptor synthesis can be run incrementally. In this mode, the tool detects when the design has changed and only then does it re-run synthesis on sections of the design that have changed.

Running in incremental synthesis mode means that for designs with small changes, the run time of the synthesis can be significantly reduced.

2.1.4 Analyzing Synthesis Results

After the synthesis run is complete, the results can be viewed and analyzed using the Reports window. This window contains the results of the synthesis.

2.1.5 Exploring the Logic

The Raptor design suite provides one physical view of the device, the P&R view. The P&R window provides a graphical view of the device, placed logic objects, and connectivity. It can be accessed from the Task menu when routing has finished.

2.1.6 Multi-Threading in RTL Synthesis

ABC-DE thread count can be adjusted.

2.2 Synthesis Attributes

In the Raptor Design Suite, the synthesis tool supports several types of attributes. These attributes have the same syntax and the same behavior.

- If Raptor synthesis supports the attribute, it uses that attribute and creates logic that reflects the used attribute.
- If the specified attribute is not recognized by the tool, the Raptor synthesis tool passes the attribute and its value to the generated netlist.

2.2.1 Supported Attributes

The following subsections show examples for the FULL_CASE and PARALLEL_CASE attributes.

2.2.1.1 FULL_CASE (Verilog Only)

```
module top (y, a, b, c, sel);
output y;
input [1:0] sel;
input a, b, c;
reg y;

always @(a or b or c or sel)
      case (sel) // synopsys full_case
         2'b00 : y = a;
         2'b01 : y = b;
         2'b10 : y = c;
      endcase
endmodule
```

2.2.1.2 PARALLEL_CASE (Verilog Only)

```
module top (res2, res1, res0, sel);
output res2, res1, res0;
input [2:0] sel;
reg res2, res1, res0;

      always @(sel) begin
         {res2, res1, res0} = 3'b0
         case2 (sel) // synopsys parallel_case
            3'b1??: res2 = 1'b1;
            3'b?1?: res1 = 1'b1;
            3'b??1: res0 = 1'b1;
         endcase
```

```
    end
endmodule
```

2.3 HDL Coding Techniques

The Raptor design suite contains Hardware Description Language (HDL) coding techniques which let the user:

- Take advantage of the various architectural features of Rapid Silicon devices.
- Easily describe common functionality found in digital logic circuits.
- Templates are available from the Raptor design suite.

2.3.1 Advantages of VHDL

- Enforces stricter rules, in particular strongly typed, less permissive and error-prone
- Initialization of RAM components in the HDL source code is easier (Verilog initial blocks are less convenient)
- Package support
- Custom types
- Enumerated types

2.3.2 Advantages of Verilog

- C-like syntax
- More compact code
- Block commenting
- No heavy component instantiation as in VHDL

2.3.3 Advantages of SystemVerilog

- More compact code compared to Verilog
- Structures and enumerated types for better scalability
- Interfaces for higher level of abstraction
- Supported in Raptor synthesis

2.3.4 Flip-Flops, Registers, and Latches

The Raptor synthesis tool recognizes flip-flops and registers with the following control elements:

- Rising or falling-edge clocks
- Asynchronous set/reset
- Synchronous set/reset
- Clock enable

2.3.5 Latches

The Raptor log file reports the type and size of all confirmed latches in the design. There are also inferred latches, which are often the result of HDL coding mistakes, such as incomplete if or case statements.

2.3.6 Shift Registers

A Shift Register is a series of Flip-Flops organized as a serial string that allows for the propagation of data across a fixed number of latency stages. In contrast, in Dynamic Shift Registers, the length of the propagation chain varies dynamically during circuit operation.

2.3.6.1 Static Shift Register Elements

A static Shift Register usually involves:

- A clock
- An optional clock enable
- A serial data input
- A serial data output

2.3.7 Dynamic Shift Registers

A dynamic shift register is a shift register the length of which can vary dynamically during circuit operation. A dynamic shift register can be viewed as a chain of flip-flops along with a multiplexer that selects, in a given clock cycle, the stage at which data is to be extracted from the chain.

The Raptor synthesis tool can infer dynamic shift registers of any length.

2.3.8 Multipliers

Raptor synthesis infers multiplier macros from multiplication operators in the source code. The resulting signal width equals the sum of the two operand sizes. For example, multiplying a 16-bit signal by an 8-bit signal produces a result of 24 bits.

2.3.9 Memories

Raptor synthesis infers Block RAM memories for the following kinds of behavioral memory examples:

Single port RAM with asynchronous read and with read address registered

Single port RAM with synchronous read

Single port RAM with synchronous read and read address registered

Simple dual port RAM with asynchronous read and with read address registered

Simple dual port RAM with synchronous read

Simple dual port RAM with synchronous read and with read address registered

True dual port RAM with asynchronous read and with read address registered

True dual port RAM with synchronous read

True dual port RAM with synchronous read and with read address registered

Each type of memory configuration is described in the following subsections.

2,3,9.1 Single Port RAM with Asynchronous Read and with Read Address Registered

The code used to support the single port RAM configuration with asynchronous read with read address registered is described below.

```
module rams_sp_async_read_reg_addr_1024x32 (clk, we, addr, di, dout);
input clk;
input we;
input [9:0] addr
input [31:0] di;
output [31:0] dout;
reg [9:0] addr_reg;
reg [31:0] RAM [1023:0];
always @(posedge clk)
begin
addr_reg <= addr;
if (we)
RAM[addr] <= di;
end
assign dout = RAM[addr_reg];
endmodule
```

2.3.9.2 Single Port RAM with Synchronous Read

The code used to support the single port RAM configuration with synchronous read is described below.

```
module rams_sp_sync_read_1024x32 (clk, we, addr, di, dout);
input clk;
input we;
input [9:0] addr;
input [31:0] di;
output reg [31:0] dout;
reg [31:0] RAM [1023:0];
always @(posedge clk)
begin
if (we)
RAM[addr] <= di;
else
dout = RAM[addr];
end
Endmodule
```

2.9.3.3 Single Port RAM with Synchronous Read and Read Address Registered

The code used to support the single port RAM configuration with synchronous read with read address registered is described below.

```
module rams_sp_sync_read_reg_addr_1024x32 (clk, we, addr, di, dout);
input clk, we;
input [9:0] addr;
input [31:0] di;
output reg [31:0] dout;
reg [9:0] addr_reg;
reg [31:0] RAM [1023:0];
always @(posedge clk)
begin
addr_reg <= addr;
if (we)
RAM[addr] <= di;
else
dout = RAM[addr_reg];
end
endmodule
```

2.9.3.4 Simple Dual Port RAM with Asynchronous Read and with Read Address Registered

The code used to support the simple dual port RAM configuration with asynchronous read with read address registered is described below.

```
module ram_simple_dp_async_reg_read_4096x32 (clk, we, read_addr, write_addr, din, dout);
input clk, we;
input [11:0] read_addr, write_addr;
input [31:0] din;
output [31:0] dout;
reg [31:0] ram [4095:0];
reg [11:0] read_addr_reg;
always @(posedge clk)
begin
read_addr_reg <= read_addr;
if (we)
ram[write_addr] <= din;
end
assign dout = ram[read_addr_reg];
endmodule
```

2.3.9.5 Simple Dual port RAM with Synchronous Read

The code used to support the simple dual port RAM configuration with synchronous read is described below.

```
module ram_simple_dp_sync_read_4096x32 (clk, we, read_addr, write_addr, din, dout);
input clk, we;
input [11:0] read_addr, write_addr;
input [31:0] din;
output reg [31:0] dout;
reg [31:0] ram [4095:0];
always @(posedge clk)
begin
if (we)
ram[write_addr] <= din;
else
dout = ram[read_addr];
end
endmodule
```

2.3.9.6 Simple Dual Port RAM with Synchronous Read and with Read Address Registered

The code used to support the simple dual port RAM configuration with synchronous read with read address registered is described below.

```
module ram_simple_dp_sync_reg_read_4096x32 (clk, we, read_addr, write_addr, din, dout);
input clk, we;
input [11:0] read_addr, write_addr;
input [31:0] din;
output reg [31:0] dout;
reg [31:0] ram [4095:0];
reg [11:0] read_addr_reg;
always @(posedge clk)
begin
read_addr_reg <= read_addr;
if (we)
ram[write_addr] <= din;
else
dout <= ram[read_addr_reg];
end
endmodule
```

2.3.9.7 True Dual Port RAM with Asynchronous Read and with Read Address Registered

The code used to support the true dual port RAM configuration with asynchronous read with read address registered is described below.

```
module ram_tdp_async_reg_read_4096x32 (clkA, clkB, weA, weB, addrA, addrB, dinA, dinB, doutA, doutB);
input clkA, clkB, weA, weB;
input [11:0] addrA, addrB;
input [31:0] dinA, dinB;
output [31:0] doutA, doutB;
reg [11:0] reg_addrA, reg_addrB;
reg [31:0] ram [4095:0];
always @(posedge clkA)
begin
reg_addrA <= addrA;
if (weA)
ram[addrA] <= dinA;
end
always @(posedge clkB)
begin
reg_addrB <= addrB;
if (weB)
ram[addrB] <= dinB;
end
assign doutA = ram[reg_addrA];
assign doutB = ram[reg_addrB];
endmodule
```

2.3.9.8 True Dual Port RAM with Synchronous Read

The code used to support the true dual port RAM configuration with synchronous read is described below.

```
module ram_tdp_sync_read_4096x32 (clkA, clkB, weA, weB, addrA, addrB, dinA, dinB, doutA, doutB);
input clkA, clkB, weA, weB;
input [11:0] addrA, addrB;
input [31:0] dinA, dinB;
output reg [31:0] doutA, doutB;
reg [31:0] ram [4095:0];
always @(posedge clkA)
begin
if (weA)
ram[addrA] <= dinA;
else
doutA = ram[addrA];
end
```

```
always @(posedge clkB)
begin
if (weB)
ram[addrB] <= dinB;
else
doutB <= ram[addrB];
end
endmodule
```

2.3.9.9 True Dual Port RAM with Synchronous Read and with Read Address Registered

The code used to support the true dual port RAM configuration with synchronous read with read address registered is described below.

```
module ram_tdp_sync_reg_read_4096x32 (clkA, clkB, weA, weB, addrA,addrB, dinA, dinB, doutA, doutB);
input clkA, clkB, weA, weB;
input [11:0] addrA, addrB;
input [31:0] dinA, dinB;
output reg [31:0] doutA, doutB;
reg [11:0] reg_addrA, reg_addrB;
reg [31:0] ram [4095:0];
always @(posedge clkA)
begin
reg_addrA <= addrA;
if (weA)
ram[addrA] <= dinA;
else
doutA <= ram[reg_addrA];
end
always @(posedge clkB)
begin
reg_addrB <= addrB;
if (weB)
ram[addrB] <= dinB;
else
doutB <= ram[reg_addrB];
end
endmodule
```

2.4 VHDL-2008 Hardware Language Support

Raptor synthesis supports a synthesizable subset of the VHDL-2008 standard.

2.4.1 Setting up Raptor to use VHDL-2008

There are several ways to run VHDL-2008 files with Raptor. The most common is to access the Properties window and set the type as "VHDL 2008" from the drop-down menu. This will configure the Raptor tool for VHDL-2008.

The Raptor design suite supports most of the VHDL-2008 features. It covers also previous releases such as VHDL87, VHDL93, and VHDL2k.

2.5 Verilog Language Support

Raptor supports most Verilog 2001 synthesis constructs

2.6 SystemVerilog Support

The Raptor design suite supports the synthesizable subset of SystemVerilog RTL versions 2005, 2009, 2012, 2017.

2.7 Mixed Language Support

Raptor synthesis supports VHDL and Verilog mixed language projects.

2.8 Litex Support

Litex is an open-source SoC generator. It provides a convenient and efficient infrastructure to create the Gemini FPGA core to explore various digital design architectures and create full FPGA based systems.

Litex uses the **Migen** package to translate the python to HDL. Migen is a Python-based tool that automates further the VLSI design process using FHDL libraries.

Litex framework contains a pool of IP Cores that can be used standalone and with the Litex framework to build complex systems. Moreover, Litex install contains a bunch of open-source CPUs that can be plugged in the system through the python packages installed in the Litex install area.

2.8.1 Litex SoC Integration

Litex offers multiple command line tools. **Litex_soc_gen** is the command line tool used to generate an SoC with custom CPU, Interconnect and peripheral options.

2.8.2.1 litex_soc Command

The Litex SoC generation command is as follows:

litex_soc_gen --cpu-type=vexriscv --bus-standard=axi-lite

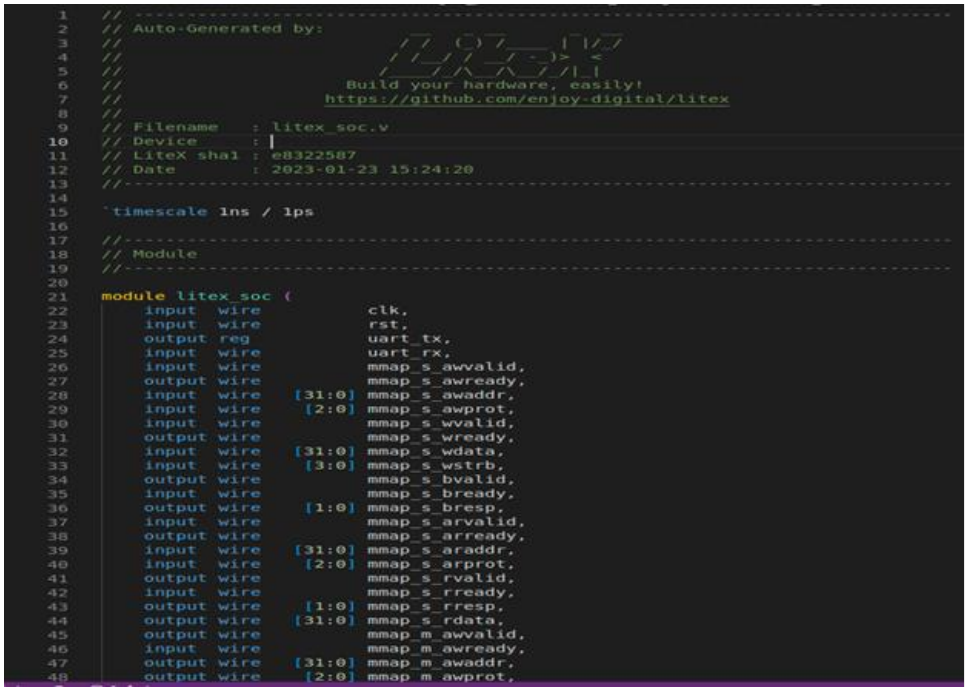


Figure 51. Litex Inputs and Outputs

2.8.2 Litex Simulation

The **Litex_sim** command line tool enables the user to create a custom SoC and the verification environment for it. Below command generates an SoC with a Vexriscv CPU and an axi-lite bus structure.

2.8.2.1 litex_sim Command

The Litex simulation command is as follows:

litex_sim_rs --cpu-type=vexriscv --bus-standard=axi-lite

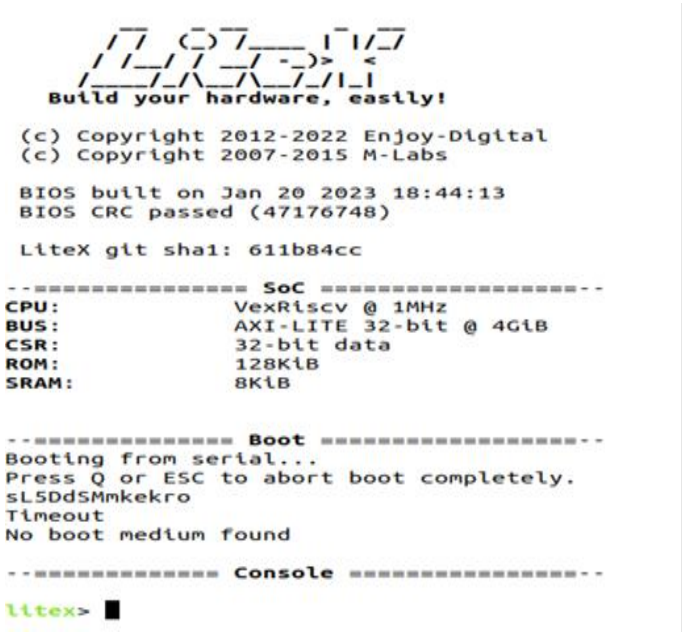


Figure 52. Litex Front Page

2.8.3 Adding Custom IP

In order to add a custom IP as a peripheral to the system, modify the litex_sim.py command as shown in the example below. For this exercise a migen wrapper must be around the verilog.

```
# AXI RAM ------------------------------------------------------------------------------------
if with_axi_ram:
    s_axi= axi.AXIInterface(data_width=32, address_width=16)
    self.submodules.axi_ram = AXIRAM(platform, s_axi)
    self.bus.add_slave("axi_ram", s_axi, region=SoCRegion(
    origin = 0x50000000,
    size = 0x1000,
    cached = True,
    ))
```

Figure 53. Adding Custom IP

The following example adds an axi_ram as a slave to the interconnect at the base address of 0x50000000 with a size of 0x1000. This can be simulated using the console as shown above and can also modify the firmware to test the system.

The RAM simulation is shown in Figure 54.

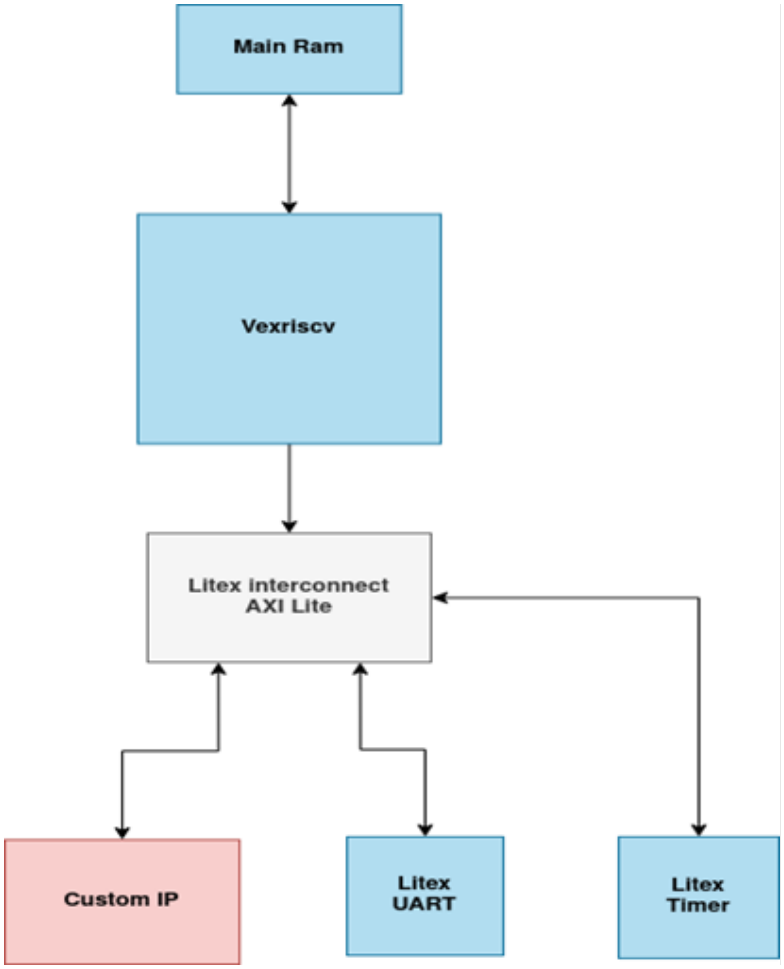


Figure 54. RAM Simulation Environment

2.8.4 Raptor IP Catalog

The Raptor IP catalog hosts a suite of IPs mostly based on AXI bus standard. All these IPs are parameterizable and can be generated using the IP Configurator window. These generated IPs are available to create system level designs and can also be used to integrate in LiteX SoC framework.

Raptor IP catalog contains a variety of IPs belonging to different categories.

- Processor IP (CPU, DSP)
- Memory IP (BRAM, ROM, FIFO)
- Interconnect IP (bus, switch, bridge)
- Peripheral IP (UART, SPI, I2C, Ethernet)

Generated IP hierarchy hosts, IP user guide document, constraint file, simulation source and the IP source files. IP Source files are visible in the IP Instance window under the name of the IP.

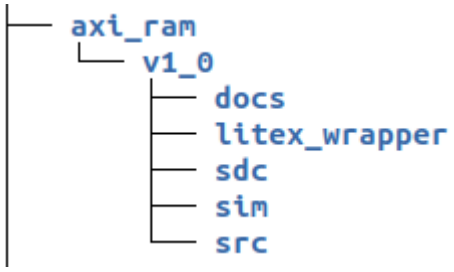


Figure 55. Example of axi_ram IP Structure

2.8.5. IP Catalog Simulations

Once an IP is generated a user can simulate the IP from Raptor. Generated IPs are available in the IP Instance window, Simulate IP button is available when you right click on any of the IP from IP Instance window.

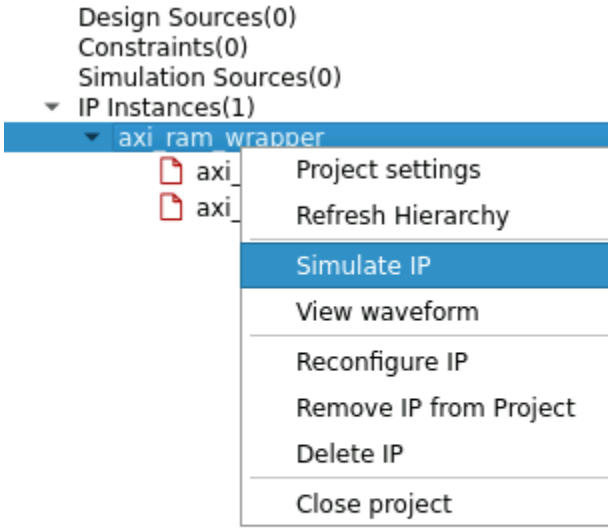


Figure 56. Simulating IP Catalog IP

Once the IP is simulated the user can view the generated waveform with the View waveform button as shown in the above figure. This will pop up the waveform of the IP you simulated.

3 Raptor Place and Route

3.1 Introduction

Raptor place and route (P&R) is the process of determining locations of post-synthesized gates and connecting them by following the netlist description. Raptor P&R is timing-driven.

There are two ways to setup and run P&R:

- Use Project Mode, selecting options from the Raptor Integrated Design Environment (IDE).
- Use Tcl Mode, applying Tool Command Language (Tcl) commands or scripts, and controlling your own design files.

3.2 P&R Methodology

The Raptor design suite invokes Versatile Place and Route (VPR) to do place and route. VPR is an open-source academic CAD tool designed for the exploration of new FPGA architectures and CAD algorithms, which performs:

- Packing
- Placement
- Routing

For details about VPR, please visit https://docs.verilogtorouting.org/en/latest/vpr/.

3.3 Use P&R

Each stage of P&R can be run within Raptor in 2 ways:

- Use IDE GUI task panel
- Use command lines
  - pack
  - place
  - route

Raptor supports all legal options of VPR. These options can be set with by executing the following command:

```
pnr_options <option_list>
```

All legal options and their description and usage can be found at: https://docs.verilogtorouting.org/en/latest/vpr/command_line_usage/

Below are some common and useful options:

3.3.1 General Options

This section lists the general place and route options.

--target_utilization <float>

Sets the target device utilization. This corresponds to the maximum target fraction of device grid-tiles to be used. A value of 1.0 means the smallest device (which fits the circuit) will be used.

Default: 1

--constant_net_method {global | route}

Specifies how constant nets (i.e., those driven to a constant value) are handled:

- global: Treat constant nets as globals (not routed)
- route: Treat constant nets

as normal nets (routed) Default:

global

--clock_modeling {ideal | route | dedicated_network}

Specifies how clock nets are handled:

- ideal: Treat clock pins as ideal (i.e., no routing delays on clocks)
- route: Treat clock nets as normal nets (i.e., routed using inter-block routing)
- dedicated_network: Use the architectures dedicated clock network

(experimental) Default: ideal (Only supported mode currently)

--terminate_if_timing_fails {on, off}

Controls whether VPR should terminate if timing is not met after routing.

Default: off

3.3.2 Filename Options

This section lists the filename place and route options.

--outfile_prefix <string>

Prefix for output files

3.3.3 Netlist Options

This section lists the netlist place and route options.

--absorb_buffer_luts {on | off}

Controls whether LUTs programmed as wires (i.e., implementing logical identity) should be absorbed into the down-stream logic.

Usually, buffer LUTS are introduced in BLIF circuits by upstream tools in order to rename signals (like assign statements in Verilog). Absorbing these buffers reduces the number of LUTs required to implement the circuit.

Ocassionally buffer LUTs are inserted for other purposes, and this option can be used to preserve them. Disabling buffer absorption can also improve the matching between the input and post-synthesis netlist/SDF.

Default: on

--const_gen_inference {none | comb | comb_seq}

Controls how constant generators are inferred/detected in the input circuit. Constant generators and the signals they drive are not considered during timing analysis.

- none: No constant generator inference will occur. Any signals which are actually constants will be treated as non-constants.

- comb: VPR will infer constant generators from combinational blocks with no non-constant inputs (always safe).
- comb_seq: VPR will infer constant generators from combinational and sequential blocks with only constant inputs (usually safe).

NOTE: In rare circumstances comb_seq could incorrectly identify certain blocks as constant generators. This would only occur if a sequential netlist primitive has an internal state which evolves completely independently of any data input (e.g., a hardened LFSR block, embedded thermal sensor).

Default: comb_seq

--sweep_dangling_primary_ios {on | off}

Controls whether the circuits dangling primary inputs and outputs (i.e., those who do not drive, or are not driven by anything) are swept and removed from the netlist.

Disabling sweeping of primary inputs/outputs can improve the matching between the input and post-synthesis netlists. This is often useful when performing formal verification.

See also:

--sweep_constant_primary_outputs

Default: on

--sweep_dangling_nets {on | off}

Controls whether dangling nets (i.e., those who do not drive, or are not driven by anything) are swept and removed from the netlist.

Default: on

--sweep_dangling_blocks {on | off}

Controls whether dangling blocks (i.e., those who do not drive anything) are swept and removed from the netlist.

Default: on

--sweep_constant_primary_outputs {on | off}

Controls whether primary outputs driven by constant values are swept and removed from the netlist.

See also:

--sweep_dangling_primary_ios

Default: off

--netlist_verbosity <int>

Controls the verbosity of netlist processing (constant generator detection, swept netlist components). High values produce more detailed output.

Default: 1

3.3.4 Packing Options

This section lists the packing place and route options.

--connection_driven_clustering {on | off}

Controls whether or not AAPack prioritizes the absorption of nets with fewer connections into a complex logic block over nets with more connections.

Default: on

--allow_unrelated_clustering {on | off | auto}

Controls whether primitives with no attraction to a cluster may be packed into it.

Unrelated clustering can increase packing density (decreasing the number of blocks required to implement the circuit) but can significantly impact routability.

When set to auto VPR automatically decides whether to enable unrelated clustring based on the targetted device and achieved packing density.

Default: auto

--alpha_clustering <float>

A parameter that weighs the optimization of timing vs area.

A value of 0 focuses solely on area, a value of 1 focuses entirely on timing.

Default: 0.75

--beta_clustering <float>

A tradeoff parameter that controls the optimization of smaller net absorption vs. the optimization of signal sharing.

A value of 0 focuses solely on signal sharing, while a value of 1 focuses solely on absorbing smaller nets into a cluster. This option is meaningful only when connection_driven_clustering is on.

Default: 0.9

--timing_driven_clustering {on|off}

Controls whether or not to do timing driven clustering

Default: on

--cluster_seed_type {blend | timing | max_inputs | max_pins | max_input_pins | blend2}

Controls how the packer chooses the first primitive to place in a new cluster.

- timing means that the unclustered primitive with the most timing-critical connection is used as the seed.
- max_inputs means the unclustered primitive that has the most connected inputs is used as the seed.
- blend uses a weighted sum of timing criticality, the number of tightly coupled blocks connected to the prim- itive, and the number of its external inputs.
- max_pins selects primitives with the most number of pins (which may be used, or unused).
- max_input_pins selects primitives with the most number of input pins (which may be used, or unused)

- blend2 An alternative blend formulation taking into account both used and unused pin counts, number of tightly coupled blocks and criticality.

Default: blend2 if timing_driven_clustering is on; max_inputs otherwise.

--clustering_pin_feasibility_filter {on | off}

Controls whether the pin counting feasibility filter is used during clustering. When enabled the clustering engine counts the number of available pins in groups/classes of mutually connected pins within a cluster. These counts are used to quickly filter out candidate primitives/atoms/molecules for which the cluster has insufficient pins to route (without performing a full routing). This reduces packing run-time.

Default: on

--balance_block_type_utilization {on, off, auto}

Controls how the packer selects the block type to which a primitive will be mapped if it can potentially map to multi-ple block types.

- on : Try to balance block type utilization by picking the block type with the (currently) lowest utilization.
- off : Do not try to balance block type utilization
- auto: Dynamically enabled/disabled (based on

density) Default: auto

--target_ext_pin_util { auto | <float> | <float>,<float> | <string>:<float> | <string>:<float>,<float> }

Sets the external pin utilization target (fraction between 0.0 and 1.0) during clustering. This determines how many pins the clustering engine will aim to use in a given cluster before closing it and opening a new cluster.

Setting this to 1.0 guides the packer to pack as densely as possible (i.e. it will keep adding molecules to the cluster until no more can fit). Setting this to a lower value will guide the packer to pack less densely, and instead create more clusters. In the limit setting this to 0.0 will cause the packer to create a new cluster for each molecule.

Typically packing less densely improves routability, at the cost of using more clusters. This option can take several different types of values:

auto VPR will automatically determine appropriate target utilizations.

- <float> specifies the target input pin utilization for all block types.

  For example: 0.7 specifies that all blocks should aim for 70% input pin utilization.
- <float>,<float> specifies the target input and output pin utilizations respectively for all block types.

  For example: 0.7, 0.9 specifies that all blocks should aim for 70% input pin utilization, and 90% output pin utilization.
- <string>:<float> and <string>:<float>,<float> specify the target pin utilizations for a specific block type (as above).

  For example: clb:0.7 specifies that only clb type blocks should aim for 70% input pin utilization. clb:0.7,0.9 specifies that only clb type blocks should aim for 70% input pin utilization, and 90% output pin utilization.

NOTE: If a pin utilization target is unspecified it defaults to 1.0 (i.e., 100% utilization)

For example:

1.7 leaves the output pin utilization unspecified, which is equivalent to 0.7,1.0.

clb:0.7,0.9 leaves the pin utilizations for all other block types unspecified, so they will assume a default utilization of 1.0,1.0.

This option can also take multiple space-separated values. For example:

--target_ext_pin_util clb:0.5 dsp:0.9,0.7 0.8

would specify that clb blocks use a target input pin utilization of 50%, dsp blocks use a target of 90% and 70% for inputs and outputs respectively, and all other blocks use an input pin utilization target of 80%.

NOTE: This option is only a guideline. If a molecule (e.g., a carry-chain with many inputs) would not otherwise fit into a cluster type at the specified target utilization the packer will fall back to using all pins (i.e. a target utilization of 1.0).

NOTE: This option requires --clustering_pin_feasibility_filter to be enabled.

Default: auto

--pack_prioritize_transitive_connectivity {on | off}

Controls whether transitive connectivity is prioritized over high-fanout connectivity during

packing. Default: on

--pack_high_fanout_threshold {auto | <int> | <string>:<int>}

Defines the threshold for high fanout nets within the packer.

This option can take several different types of values:

- auto VPR will automatically determine appropriate thresholds.
- <int> specifies the fanout threshold for all block

types. For example:

64 specifies that a threshold of 64 should be used for all blocks.

- <string>:<float> specifies the threshold for a specific block type.

For example:

clb:16 specifies that clb type blocks should use a threshold of 16.

This option can also take multiple space-separated values. For example:

--pack_high_fanout_threshold 128 clb:16

would specify that clb blocks use a threshold of 16, while all other blocks (e.g., DSPs/RAMs) would use a threshold of 128.

Default: auto

--pack_transitive_fanout_threshold <int>

Packer transitive fanout threshold.

Default: 4

--pack_feasible_block_array_size <int>

This value is used to determine the max size of the priority queue for candidates that pass the early filter legality test but not the more detailed routing filter.

Default: 30

--pack_verbosity <int>

Controls the verbosity of clustering output. Larger values produce more detailed output, which may be useful for debugging architecture packing problems.

Default: 2

--write_block_usage <file>

Writes out to the file under path <file> cluster-level block usage summary in machine readable (JSON or XML) or human readable (TXT) format. Format is selected based on the extension of <file>.

3.3.5 Placer Options

This section lists the placer place and route options.

--seed <int>

Sets the initial random seed used by the

placer. Default: 1

--enable_timing_computations {on | off}

Controls whether or not the placement algorithm prints estimate of the circuit speed of the placement it generates. This setting affects statistical output only, not optimization behavior.

Default: on if timing-driven placement is specified, off otherwise.

--inner_num <float>

The number of moves attempted at each temperature in placement can be calculated from inner_num scaled with circuit size or device-circuit size as specified in place_effort_scaling.

Changing inner_num is the best way to change the speed/quality tradeoff of the placer, as it leaves the highly-efficient automatic annealing schedule on and simply changes the number of moves per temperature.

Specifying -inner_num 10 will slow the placer by a factor of 10 while typically improving placement quality only by 10% or less (depends on the architecture). Hence users more concerned with quality than CPU time may find this a more appropriate value of inner_num.

Default: 0.5

--place_effort_scaling {circuit | device_circuit}

Controls how the number of placer moves level scales with circuit and device size:

- circuit: The number of moves attempted at each temperature is inner_num * num_blocks^(4/3) in the circuit.
- device_circuit: The number of moves attempted at each temperature is inner_num * grid_size^(2/3) * num_blocks^(4/3) in the circuit.

The number of blocks in a circuit is the number of pads plus the number of clbs.

Default: circuit

--init_t <float>

The starting temperature of the anneal for the manual annealing schedule.

Default: 100.0

--exit_t <float>

The manual anneal will terminate when the temperature drops below the exit

temperature. Default: 0.01

--alpha_t <float>

The temperature is updated by multiplying the old temperature by alpha_t when the manual annealing schedule is enabled.

Default: 0.8

--fix_pins {free | random}

Controls how the placer handles I/O pads during placement.

- free: The placer can move I/O locations to optimize the placement.
- random: Fixes I/O pads to arbitrary locations and does not allow the placer to move them during the anneal (models the effect of poor board-level I/O constraints).

NOTE: The fix_pins option is also used to accept a third argument - a place file that specifies where I/O pins should be placed. This argument is no longer accepted by fix_pins. Instead, the fix_clusters option can now be used to lock down I/O pins.

Default: free.

--fix_clusters {<file.place>}

Controls how the placer handles blocks (of any type) during placement.

<file.place>: A path to a file listing the desired location of blocks in the netlist.

This place location file is in the same format as a normal placement file but does not require the first two lines which are normally at the top of a placement file that specify the netlist file, netlist ID, and array size.

--place_algorithm {bounding_box | criticality_timing | slack_timing}

Controls the algorithm used by the placer.

- bounding_box Focuses purely on minimizing the bounding box wirelength of the circuit. Turns off timing analysis if specified.
- criticality_timing Focuses on minimizing both the wirelength and the connection timing costs (criticality * delay).
- slack_timing Focuses on improving the circuit slack values to reduce critical path

delay. Default: criticality_timing

--place_quench_algorithm {bounding_box | criticality_timing | slack_timing}

Controls the algorithm used by the placer during placement quench. The algorithm options have identical functionality as the ones used by the option --place_algorithm. If specified, it overrides the option --place_algorithm during placement quench.

Default: criticality_timing

--place_chan_width <int>

Tells VPR how many tracks a channel of relative width 1 is expected to complete routing of this circuit. VPR will then place the circuit only once, and repeatedly try routing the circuit as usual.

Default: 100

--place_rlim_escape <float>

The fraction of moves which are allowed to ignore the region limit. For example, a value of 0.1 means 10% of moves are allowed to ignore the region limit.

Default: 0.0

Setting any of the following options selects Dusty's annealing schedule.

--alpha_min <float>

The minimum (starting) update factor (alpha) used. Ranges between 0 and alpha_max.

Default: 0.2

--alpha_max <float>

The maximum (stopping) update factor (alpha) used after which simulated annealing will complete. Ranges between alpha_min and 1.

Default: 0.9

--alpha_decay <float>

The rate at which alpha will approach 1: alpha(n) = 1 - (1 - alpha(n-1)) * alpha_decay Ranges between 0 and 1.

Default: 0.7

--anneal_success_min <float>

The minimum success ratio after which the temperature will reset to maintain the target success ratio. Ranges between 0 and anneal_success_target.

Default: 0.1

--anneal_success_target <float>

The temperature after each reset is selected to keep this target success ratio. Ranges between anneal_success_target and 1.

Default: 0.25

3.3.6 Timing-Driven Placer Options

The following options are only valid when the placement engine is in timing-driven mode (timing-driven placement is used by default).

--timing_tradeoff <float>

Controls the trade-off between bounding box minimization and delay minimization in the placer.

A value of 0 makes the placer focus completely on bounding box (wirelength) minimization, while a value of 1 makes the placer focus completely on timing optimization.

Default: 0.5

--recompute_crit_iter <int>

Controls how many temperature updates occur before the placer performs a timing analysis to update its estimate of the criticality of each connection.

Default: 1

--inner_loop_recompute_divider <int>

Controls how many times the placer performs a timing analysis to update its criticality estimates while at a single temperature.

Default: 0

--td_place_exp_first <float>

Controls how critical a connection is considered as a function of its slack, at the start of the anneal.

If this value is 0, all connections are considered equally critical. If this value is large, connections with small slacks are considered much more critical than connections with small slacks. As the anneal progresses, the exponent used in the criticality computation gradually changes from its starting value of td_place_exp_first to its final value of --td_place_exp_last.

Default: 1.0

--td_place_exp_last <float>

Controls how critical a connection is considered as a function of its slack, at the end of the anneal.

See also

--td_place_exp_first

Default: 8.0

--place_delay_model {delta, delta_override}

Controls how the timing-driven placer estimates delays.

- delta The router is used to profile delay from various locations in the grid for various differences in position
- delta_override Like delta but also includes special overrides to ensure effects of direct connects between blocks are accounted for. This is potentially more accurate but is more complex and depending on the architecture (e.g. number of direct connects) may increase place run-time.

Default: delta

--place_delay_model_reducer {min, max, median, arithmean, geomean}

When calculating delta delays for the placement delay model how are multiple values combined?

Default: min

--place_delay_offset <float>

A constant offset (in seconds) applied to the placer's delay model.

Default: 0.0

--place_delay_ramp_delta_threshold <float>

The delta distance beyond which –place_delay_ramp is applied. Negative values disable the placer delay ramp.

Default: -1

--place_delay_ramp_slope <float>

The slope of the ramp (in seconds per grid tile) which is applied to the placer delay model for delta distance beyond --place_delay_ramp_delta_threshold.

Default: 0.0e-9

--place_tsu_rel_margin <float>

Specifies the scaling factor for cell setup times used by the placer. This effectively controls whether the placer should try to achieve extra margin on setup paths. For example, a value of 1.1 corresponds to requesting 10% setup margin.

Default: 1.0

--place_tsu_abs_margin <float>

Specifies an absolute offest added to cell setup times used by the placer. This effectively controls whether the placer should try to achieve extra margin on setup paths. For example, a value of 500e-12 corresponds to requesting an extra 500ps of setup margin.

Default: 0.0

--post_place_timing_report <file>

Name of the post-placement timing report file to generate (not generated if unspecfied).

3.3.7 Router Options

VPR uses a negotiated congestion algorithm (based on Pathfinder) to perform routing.

NOTE: By default the router performs a binary search to find the minimum routable channel width. To route at a fixed channel width use --route_chan_width.

See also Section 3.3.8 "Timing-Driven Router Options".

--max_router_iterations <int>

The number of iterations of a Pathfinder-based router that will be executed before a circuit is declared unrouteable (if it hasn't routed successfully yet) at a given channel width.

Speed-quality trade-off: reducing this number can speed up the binary search for minimum channel width, but at the cost of some increase in final track count. This is most effective if -initial_pres_fac is simultaneously increased. Increase this number to make the router try harder to route heavily congested designs.

Default: 50

--first_iter_pres_fac <float>

Similar to --initial_pres_fac. This sets the present overuse penalty factor for the very first routing iteration. --initial_pres_fac sets it for the second iteration.

NOTE: A value of 0.0 causes congestion to be ignored on the first routing iteration.

Default: 0.0

--initial_pres_fac <float>

Sets the starting value of the present overuse penalty factor.

Speed-quality trade-off: increasing this number speeds up the router, at the cost of some increase in final track count. Values of 1000 or so are perfectly reasonable.

Default: 0.5

--pres_fac_mult <float>

Sets the growth factor by which the present overuse penalty factor is multiplied after each router iteration.

Default: 1.3

--acc_fac <float>

Specifies the accumulated overuse factor (historical congestion cost factor).

Default: 1

--bb_factor <int>

Sets the distance (in channels) outside of the bounding box of its pins a route can go. Larger numbers slow the router somewhat but allow for a more exhaustive search of possible routes.

Default: 3

--base_cost_type {demand_only | delay_normalized | delay_normalized_length | delay_normalized_frequency | delay_normalized_length_frequency}

Sets the basic cost of using a routing node (resource).

- demand_only sets the basic cost of a node according to how much demand is expected for that type of node.
- delay_normalized is similar to demand_only but normalizes all these basic costs to be of the same magnitude as the typical delay through a routing resource.
- delay_normalized_length like delay_normalized but scaled by routing resource length.
- delay_normalized_frequency like delay_normalized, but scaled inversely by routing resource frequency.
- delay_normalized_length_frequency like delay_normalized, but scaled by routing resource length and scaled inversely by routing resource frequency.

Default: delay_normalized_length for the timing-driven router and demand_only for the breadth-first router.

--bend_cost <float>

The cost of a bend. Larger numbers will lead to routes with fewer bends, at the cost of some increase in track count. If only global routing is being performed, routes with fewer bends will be easier for a detailed router to sub- sequently route onto a segmented routing architecture.

Default: 1 if global routing is being performed, 0 if combined global/detailed routing is being performed.

--route_type {global | detailed}

Specifies whether global routing or combined global and detailed routing should be

performed. Default: detailed (i.e., combined global and detailed routing)

--route_chan_width <int>

Tells VPR to route the circuit at the specified channel width.

Note

NOTE: If the channel width is >= 0, no binary search on channel capacity will be performed to find the minimum number of tracks required for routing. VPR simply reports whether or not the circuit will route at this channel width.

Default: -1 (perform binary search for minimum routable channel width)

--min_route_chan_width_hint <int>

Hint to the router what the minimum routable channel width is.

The value provided is used to initialize the binary search for minimum channel width. A good hint may speed up the binary search by avoiding time spent at congested channel widths which are not routable.

The algorithm is robust to incorrect hints (i.e., it continues to binary search), so the hint does not need to be

precise. This option may ocassionally produce a different minimum channel width due to the different initialization.

See also

--verify_binary_search

--verify_binary_search {on | off}

Force the router to check that the channel width determined by binary search is the minimum.

The binary search ocassionally may not find the minimum channel width (e.g., due to router sub-optimality, or rout- ing pattern issues at a particular channel width).

This option attempts to verify the minimum by routing at successively lower channel widths until two consecutive routing failures are observed.

--router_algorithm {breadth_first | timing_driven}

Selects which router algorithm to use.

Warning: The breadth_first router should NOT be used to compare the run-time/quality of alternate routing algorithms.

It is inferior to the timing_driven router from a circuit speed (2x - 10x slower) and run-time perspective (takes 10-100x longer on the large benchmarks). The breadth_first router is deprecated and may be removed in a future release.

The breadth_first router [BRM99] focuses solely on routing a design successfully, while the timing_driven router [BRM99, MZB20] focuses both on achieving a successful route and achieving good circuit speed.

The breadth-first router is capable of routing a design using slightly fewer tracks than the timing-driving router (typically 5% if the timing-driven router uses its default parameters. This can be reduced to about 2% if the router parameters are set so the timing-driven router pays more attention to routability and less to area). The designs produced by the timing-driven router are much faster, however, (2x - 10x) and it uses less CPU time to route.

Default: timing_driven

--min_incremental_reroute_fanout <int>

Incrementally re-route nets with fanout above the specified threshold.

This attempts to re-use the legal (i.e. non-congested) parts of the routing tree for high fanout nets, with the aim of reducing router execution time.

To disable, set value to a value higher than the largest fanout of any net.

Default: 16

--max_logged_overused_rr_nodes <int>

Prints the information on overused RR nodes to the VPR log file after each failed routing attempt.

If the number of overused nodes is above the given threshold N, then only the first N entries are printed to the log- file.

Default: 20

--generate_rr_node_overuse_report {on | off}

Generates a detailed report on the overused RR nodes' information: report_overused_nodes.rpt.

This report is generated only when the final routing attempt fails (i.e. the whole routing process has failed).

In addition to the information that can be seen via --max_logged_overused_rr_nodes, this report prints out all the net ids that are associated with each overused RR node. Also, this report does not place a threshold upon the number of RR nodes printed.

Default: off

--write_timing_summary <file>

Writes out to the file under path <file> final timing summary in machine readable (JSON or XML) or human readable (TXT) format. Format is selected based on the extension of <file>. The summary consists of parameters:

- cpd - Final critical path delay (least slack) [ns]
- fmax - Maximal frequency of the implemented circuit [MHz]
- swns - setup Worst Negative Slack (sWNS) [ns]
- stns - Setup Total Negative Slack (sTNS) [ns]

3.3.8 Timing-Driven Router Options

The following options are only valid when the router is in timing-driven mode (the default).

--astar_fac <float>

Sets how aggressive the directed search used by the timing-driven router is.

Values between 1 and 2 are reasonable, with higher values trading some quality for reduced CPU time.

Default: 1.2

--max_criticality <float>

Sets the maximum fraction of routing cost that can come from delay (vs. coming from routability) for any net.

A value of 0 means no attention is paid to delay; a value of 1 means nets on the critical path pay no attention to congestion.

Default: 0.99

--criticality_exp <float>

Controls the delay - routability tradeoff for nets as a function of their slack.

If this value is 0, all nets are treated the same, regardless of their slack. If it is very large, only nets on the critical path will be routed with attention paid to delay. Other values produce more moderate tradeoffs.

Default: 1.0

--router_init_wirelength_abort_threshold <float>

The first routing iteration wirelength abort threshold. If the first routing iteration uses more than this fraction of available wirelength routing is aborted.

Default: 0.85

--incremental_reroute_delay_ripup {on | off | auto}

Controls whether incremental net routing will rip-up (and re-route) a critical connection for delay, even if the routing is legal. auto enables delay-based rip-up unless routability becomes a concern.

Default: auto

--routing_failure_predictor {safe | aggressive | off}

Controls how aggressive the router is at predicting when it will not be able to route successfully and giving up early. Using this option can significantly reduce the runtime of a binary search for the minimum channel width.

- safe only declares failure when it is extremely unlikely a routing will succeed, given the amount of congestion existing in the design.
- aggressive can further reduce the CPU time for a binary search for the minimum channel width but can increase the minimum channel width by giving up on some routings that would succeed.
- off disables this feature, which can be useful if you suspect the predictor is declaring routing failure too quickly on your architecture.

See also:

--verify_binary_search

Default: safe

--routing_budgets_algorithm { disable | minimax | scale_delay }

Warning: Experimental

Controls how the routing budgets are created. Routing budgets are used to guide VPR's routing algorithm to consider both short path and long path timing constraints [FBC08].

- disable is used to disable the budget feature. This uses the default VPR and ignores hold time constraints.
- minimax sets the minimum and maximum budgets by distributing the long path and short path slacks depending on the current delay values. This uses the routing cost valleys and Minimax-PERT algorithm [FBC08, YLS92].
- scale_delay has the minimum budgets set to 0 and the maximum budgets is set to the delay of a net scaled by the pin criticality (net delay/pin criticality).

Default: disable

--save_routing_per_iteration {on | off}

Controls whether VPR saves the current routing to a file after each routing iteration. May be helpful for debugging. Default: off

--congested_routing_iteration_threshold CONGESTED_ROUTING_ITERATION_THRESHOLD

Controls when the router enters a high effort mode to resolve lingering routing congestion. Value is the fraction of max_router_iterations beyond which the routing is deemed congested.

Default: 1.0 (never)

--route_bb_update {static, dynamic}

Controls how the router's net bounding boxes are updated:

- static: bounding boxes are never updated
- dynamic: bounding boxes are updated dynamically as routing progresses (may improve routability of congested designs)

Default: dynamic

--router_high_fanout_threshold ROUTER_HIGH_FANOUT_THRESHOLD

Specifies the net fanout beyond which a net is considered high fanout. Values less than zero disable special behavior for high fanout nets.

Default: 64

--router_lookahead {classic, map}

Controls what lookahead the router uses to calculate the cost of completing a connection.

- classic: The classic VPR lookahead
- map: A more advanced lookahead which accounts for diverse wire types and their

connectivity Default: classic

--router_max_convergence_count <float>

Controls how many times the router is allowed to converge to a legal routing before halting. If multiple legal solutions are found the best quality implementation is used.

Default: 1

--router_reconvergence_cpd_threshold <float>

Specifies the minimum potential CPD improvement for which the router will continue to attempt re-convergent routing.

For example, a value of 0.99 means the router will not give up on reconvergent routing if it thinks a > 1% CPD reduction is possible.

Default: 0.99

--router_initial_timing {all_critical | lookahead}

Controls how criticality is determined at the start of the first routing iteration.

- all_critical: All connections are considered timing critical.
- lookahead: Connection criticalities are determined from timing analysis assuming (best-case) connection delays as estimated by the router's lookahead.

Default: all_critical for the classic --router_lookahead, otherwise lookahead

--router_update_lower_bound_delays {on | off}

Controls whether the router updates lower bound connection delays after the 1st routing iteration.

Default: on

--router_first_iter_timing_report <file>

Name of the timing report file to generate after the first routing iteration completes (not generated if unspecified).

--router_debug_net <int>

NOTE: This option is likely only of interest to developers debugging the routing algorithm

Controls which net the router produces detailed debug information for.

- For values >= 0, the value is the net ID for which detailed router debug information should be produced.
- For value == -1, detailed router debug information is produced for all nets.
- For values < -1, no router debug output is produced.

Warning: VPR must have been compiled with VTR_ENABLE_DEBUG_LOGGING on to get any debug output from this option.

Default: -2

--router_debug_sink_rr ROUTER_DEBUG_SINK_RR

NOTE: This option is likely only of interest to developers debugging the routing algorithm

Controls when router debugging is enabled for the specified sink RR.

- For values >= 0, the value is taken as the sink RR Node ID for which to enable router debug output.
- For values < 0, sink-based router debug output is disabled.

Warning: VPR must have been compiled with VTR_ENABLE_DEBUG_LOGGING on to get any debug output from this option.

Default: -2

3.3.9 Analysis Options

--full_stats

Print out some extra statistics about the circuit and its routing useful for wireability analysis.

Default: off

--gen_post_synthesis_netlist { on | off }

Generates the Verilog and SDF files for the post-synthesized circuit. The Verilog file can be used to perform functional simulation and the SDF file enables timing simulation of the post-synthesized circuit.

The Verilog file contains instantiated modules of the primitives in the circuit. Currently VPR can generate Verilog files for circuits that only contain LUTs, Flip Flops, IOs, Multipliers, and BRAMs. The Verilog description of these primitives are in the primitives.v file. To simulate the post-synthesized circuit, one must include the generated Verilog file and also the primitives.v Verilog file, in the simulation directory.

See also Post-Implementation Timing Simulation

If one wants to generate the post-synthesized Verilog file of a circuit that contains a primitive other than those mentioned above, he/she should contact the VTR team to have the source code updated. Furthermore, to perform simulation on that circuit the Verilog description of that new primitive must be appended to the primitives.v file as a separate module.

Default: off

--gen_post_implementation_merged_netlist { on | off }

This option is based on --gen_post_synthesis_netlist. The difference is that --gen_post_implementation_merged_netlist generates a single verilog file with merged top module multi-bit ports of the implemented circuit. The name of the file is <basename>_merged_post_implementation.v

Default: off

--post_synth_netlist_unconn_inputs { unconnected | nets | gnd | vcc }

Controls how unconnected input cell ports are handled in the post-synthesis netlist

- unconnected: leave unconnected
- nets: connect each unconnected input pin to its own separate undriven net named: __vpr unconn<ID>, where <ID> is index assigned to this occurrence of unconnected port in design
- gnd: tie all to ground (1'b0)
- vcc: tie all to VCC

(1'b1) Default: unconnected

--post_synth_netlist_unconn_outputs { unconnected | nets }

Controls how unconnected output cell ports are handled in the post-synthesis netlist

- unconnected: leave unconnected
- nets: connect each unconnected output pin to its own separate undriven net named: vpr__unconn<ID>, where <ID> is index assigned to this occurrence of unconnected port in design

Default: unconnected

--timing_report_npaths <int>

Controls how many timing paths are reported.

NOTE: The number of paths reported may be less than the specified value, if the circuit has fewer paths.

Default: 100

--timing_report_detail { netlist | aggregated | detailed | debug}

Controls the level of detail included in generated timing reports.

We obtained the following results using the k6_frac_N10_frac_chain_mem32K_40nm.xml architecture and multi-clock.blif circuit.

- netlist: Timing reports show only netlist primitive pins.
- aggregated: Timing reports show netlist pins, and an aggregated summary of intra-block and inter-block routing delays.
- detailed: Like aggregated, the timing reports show netlist pins, and an aggregated summary of intra-block. In addition, it includes a detailed breakdown of the inter-block routing delays.
- debug: Like detailed, but includes additional VPR internal debug information such as timing graph node IDs (tnode) and routing SOURCE/SINK nodes.

Default: netlist

--echo_dot_timing_graph_node { string | int }

Controls what subset of the timing graph is echoed to a GraphViz DOT file when vpr --echo_file is enabled.

Value can be a string (corresponding to a VPR atom netlist pin name), or an integer representing a timing graph node ID. Negative values mean the entire timing graph is dumped to the DOT file.

Default: -1

--timing_report_skew { on | off }

Controls whether clock skew timing reports are generated.

Default: off

3.4 Use Pin Location Constraints

Raptor allows users to bind their design I/O ports to specific device pins, by using pin location constraint commands in the SDC file.

The list of commands is as follows:

set_pin_loc <design_io_name> <device_io_name>

Constrains design pin to device pin.

Arguments:

<design_io_name>

The IO port name in user design.

<device_io_name>

The device package pin name.

set_property mode <io_mode_name> <device_io_name>

Constrains IO operation mode.

Arguments:

<io_mode_name>

Available IO operation mode.

<device_io_name>

Device IO package pin name.

Here is an example:

```
set_property mode Mode_BP_SDR_A_TX
Bank_VL_1_19
set_pin_loc a Bank_VL_1_19

set_property mode Mode_BP_SDR_A_TX
Bank_VL_1_21

set_pin_loc b Bank_VL_1_21

set_property mode  Mode_BP_SDR_A_TX
Bank_VL_1_23
set_pin_loc c Bank_VL_1_23
```

Where:

a, b and c are design ports

Bank_* are device package pins

Mode_* are IO operation modes

NOTE: To set the proper and available modes for pin constraint, please refer to the device pin table. In case there is no pin location constraint given, Raptor is able to assign pin location in the method specified by user through below command:

pin_loc_assign_method <method>

Method can be one of below:

- in_define_order: Pins are assigned to available device pin by the order the ports in design are defined
- fandom: Pins are assigned to available device pin by the order the ports randomly
- free: Pins are assigned by Raptor placer (VPR)

4. Raptor Timing Analysis

Raptor invokes the VPR/tatum timing engine to do static timing analysis (STA). Tatum is a block-based Static Timing Analysis (STA) engine suitable for integration with Computer-Aided Design (CAD) tools, which analyze, implement and optimize digital circuits. Tatum supports both setup (max-delay) and hold (min-delay) analysis, clock skew, multiple clocks and a variety of timing exceptions.

There are two ways to setup and run timing analysis:

- Use Project Mode, selecting options from the Raptor Integrated Design Environment (IDE).
- Use Tcl Mode, applying Tool Command Language (Tcl) commands or scripts, and controlling your own design files.

4.1 STA Methodology

Tatum timing engine is an open project which has been used in different tools and projects. VPR uses it as timing engine starting in version v8.0.0-rc1(2019-06-13), where tatum performs:

- Apply timing constraints (through SDC commands)
- Time design (Delay calculation)
- Timing violation check

For details about VPR/tatum, https://github.com/verilog-to-routing/tatum

4.2 Using STA

Each stage of STA can be run within Raptor in 2 ways:

- Use IDE GUI task panel
- Use command lines
  - add_constraint_file <SDC_FILE>
  - sta

NOTE: Static timing analysis should be performed after P&R.

4.2.1 SDC Commands

To do timing analysis, the user must provide SDC file for VPR/tatum to start with. Raptor passes these commands in SDC file to VPR/tatum to allow it to do the job.

Tatum supports a subset of SDC commands. For details and usage examples, please refer to: https://docs.verilog-torouting.org/en/latest/vpr/sdc_commands/.

The following subset of SDC syntax is supported by VPR.

create_clock

Creates a netlist or virtual clock.

Assigns a desired period (in nanoseconds) and waveform to one or more clocks in the netlist (if the –name option is omitted) or to a single virtual clock (used to constrain input and outputs to a clock external to the design). Netlist clocks can be referred to using regular expressions, while the virtual clock name is taken as-is.

Example Usage:

```
#Create a netlist clock
create_clock -period <float> <netlist clock list or regexes>

#Create a virtual clock
create_clock -period <float> -name <virtual clock name>

#Create a netlist clock with custom waveform/duty-cycle
create_clock -period <float> -waveform {rising_edge falling_edge} <netlist clock list or regexes>
```

Omitting the waveform creates a clock with a rising edge at 0 and a falling edge at the half period and is equivalent to using **-waveform {0 <period/2>}**. Non-50% duty cycles are supported but behave no differently than 50% duty cycles, since falling edges are not used in analysis. If a virtual clock is assigned using a create_clock command, it must be referenced elsewhere in a set_input_delay or set_output_delay constraint.

create_clock

-period <float>

Specifies the clock period.

Required: Yes

-waveform {<float> <float>}

Overrides the default clock waveform.

The first value indicates the time the clock rises, the second the time the clock falls.

Required: No

Default: 50% duty cycle (i.e. **-waveform {0 <period/2>}**).

-name <string>

Creates a virtual clock with the specified name.

Required: No

<netlist clock list or regexes>

Creates a netlist clock

Required: No

Note

One of -name or <netlist clock list or regexes> must be specified.

Warning:

If a netlist clock is not specified with a create_clock command, paths to and from that clock domain will not be analysed.

set_clock_groups

Specifies the relationship between groups of clocks. May be used with netlist or virtual clocks in any combination.

Since VPR supports only the **-exclusive** option, a **set_clock_groups** constraint is equivalent to a **set_false_path** constraint (see below) between each clock in one group and each clock in another.

For example, the following sets of commands are equivalent:

```
#Do not analyze any timing paths between clk1 and clk2
set_clock_groups -exclusive -group {clk1} -group {clk2}
and
set_false_path -from [get_clocks {clk1}] -to [get_clocks {clk2}]
set_false_path -from [get_clocks {clk2}] -to [get_clocks {clk1}]
```

set_clock_groups

-exclusive

Indicates that paths between clock groups should not be analyzed.

Required: Yes

Note:

VPR currently only supports exclusive clock groups

-group {<clock list or regexes>}

Specifies a group of clocks.

Note:

At least 2 groups must be specified.

Required: Yes

set_false_path

Cuts timing paths unidirectionally from each clock in **-from** to each clock in **–to.** Otherwise, equivalent to **set_clock_groups**.

Example Usage:

```
#Do not analyze paths launched from clk1 and captured by clk2
set_false_path -from [get_clocks {clk1}] -to [get_clocks {clk2}]

#Do not analyze paths launched from clk2 and captured by clk
set_false_path -from [get_clocks {clk2}] -to [get_clocks {clk1}]
```

Note:

False paths are supported between entire clock domains, but not between individual registers.

set_false_path

-from [get_clocks <clock list or regexes>]

Specifies the source clock domain(s).

Required: No

Default: All clocks

-to [get_clocks <clock list or regexes>]

Specifies the sink clock domain(s).

Required: No

Default: All clocks

set_max_delay/set_min_delay

Overrides the default setup (max) or hold (min) timing constraint calculated using the information from **create_clock** with a user-specified delay.

Example Usage:

```
#Specify a maximum delay of 17 from input_clk to output_clk
set_max_delay 17 -from [get_clocks {input_clk}] -to [get_clocks {output_clk}]

#Specify a minimum delay of 2 from input_clk to output_clk
set_min_delay 2 -from [get_clocks {input_clk}] -to [get_clocks {output_clk}]
```

Note

Max/Min delays are supported between entire clock domains, but not between individual netlist elements.

set_max_delay/set_min_delay

<delay>

The delay value to apply.

Required: Yes

-from [get_clocks <clock list or regexes>]

Specifies the source clock domain(s).

Required: No

Default: All clocks

-to [get_clocks <clock list or regexes>]

Specifies the sink clock domain(s).

Required: No

Default: All clocks

---

set_multicycle_path

Sets how many clock cycles elapse between the launch and capture edges for setup and hold checks.

The default the setup mutlicycle value is 1 (i.e., the capture setup check is performed against the edge one cycle

after the launch edge).

The default hold multicycle is one less than the setup multicycle path (e.g. the capture hold check occurs in the same cycle as the launch edge for the default setup multicycle).

Example Usage:

```
#Create a 4 cycle setup check, and 0 cycle hold check from clkA to clkB
set_multicycle_path -from [get_clocks {clkA}] -to [get_clocks {clkB}] 4

#Create a 3 cycle setup check from clk to clk2
# Note that this moves the default hold check to be 2 cycles
set_multicycle_path -setup -from [get_clocks {clk}] -to [get_clocks {clk2}] 3

#Create a 0 cycle hold check from clk to clk2
# Note that this moves the default hold check back to it's original
# position before the previous setup setup_multicycle_path was applied
set_multicycle_path -hold -from [get_clocks {clk}] -to [get_clocks {clk2}] 2

#Create a multicycle to a specific pin
set_multicycle_path -to [get_pins {my_inst.in\[0\]}] 2
```

Note:

Multicycles are supported between entire clock domains and ending at specific registers.

set_multicycle_path

-setup

Indicates that the multicycle-path applies to setup analysis.

Required: No

-hold

Indicates that the multicycle-path applies to hold analysis.

Required: No

-from [get_clocks <clock list or regexes>]

Specifies the source clock domain(s).

Required: No

Default: All clocks

-to [get_clocks <clock list or regexes>]

Specifies the sink clock domain(s).

Required: No

Default: All clocks

-to [get_pins <pin list or regexes>]

Specifies the sink/capture netlist pins to which the multicycle is applied.

See also

VPR's pin naming convention.

Required: No

<path_multiplier>

The number of cycles that apply to the specified path(s).

Required: Yes

Note:

If neither -setup nor -hold the setup multicycle is set to path_multiplier and the hold multicycle offset to 0.

Note:

Only a single -to option can be specified (either clocks or pins, but not both).

set_input_delay/set_output_delay

Use **set_input_delay** if you want timing paths from input I/Os analyzed, and **set_output_delay** if you want timing paths to output I/Os analyzed.

Note:

If these commands are not specified in your SDC, paths from and to I/Os will not be timing analyzed.

These commands constrain each I/O pad specified after **get_ports** to be timing-equivalent to a register clocked on the clock specified after **-clock**. This can be either a clock signal in your design or a virtual clock that does not exist in the design but which is used only to specify the timing of I/Os.

The specified delays are added to I/O timing paths and can be used to model board level delays.

For single-clock circuits, **-clock** can be wildcarded using ***** to refer to the single netlist clock, although this is not supported in standard SDC. This allows a single SDC command to constrain I/Os in all single-clock circuits.

Example Usage:

```
#Set a maximum input delay of 0.5 (relative to input_clk) on
#ports in1, in2 and in3
set_input_delay -clock input_clk -max 0.5 [get_ports {in1 in2 in3}]

#Set a minimum output delay of 1.0 (relative to output_clk) on
#all ports matching starting with 'out*'
set_output_delay -clock output_clk -min 1 [get_ports {out*}]

#Set both the maximum and minimum output delay to 0.3 for all I/Os
#in the design
set_output_delay -clock clk2 0.3 [get_ports {*}]
```

set_input_delay/set_output_delay

-clock <virtual or netlist clock>

Specifies the virtual or netlist clock the delay is relative to.

Required: Yes

-max

Specifies that the delay value should be treated as the maximum delay.

Required: No

-min

Specifies that the delay value should be treated as the minimum delay.

Required: No

<delay>

Specifies the delay value to be applied

Required: Yes

[get_ports {<I/O list or regexes>}]

Specifies the port names or port name regex.

Required: Yes

Note:

If neither **-min** nor **-max** are specified the delay value is applied to both.

---

set_clock_uncertainty

Sets the clock uncertainty between clock domains. This is typically used to model uncertainty in the clock arrival times due to clock jitter.

Example Usage:

```
#Sets the clock uncertainty between all clock domain pairs to 0.025
set_clock_uncertainty 0.025

#Sets the clock uncertainty from 'clk' to all other clock domains to 0.05
set_clock_uncertainty -from [get_clocks {clk}] 0.05

#Sets the clock uncertainty from 'clk' to 'clk2' to 0.75
set_clock_uncertainty -from [get_clocks {clk}]  -to [get_clocks {clk2}] 0.75
```

set_clock_uncertainty

-from [get_clocks <clock list or regexes>]

Specifies the source clock domain(s).

Required: No

Default: All clocks

-to [get_clocks <clock list or regexes>]

Specifies the sink clock domain(s).

Required: No

Default: All clocks

-setup

Specifies the clock uncertainty for setup analysis.

Required: No

-hold

Specifies the clock uncertainty for hold analysis.

Required: No

<uncertainty>

The clock uncertainty value between the from and to clocks.

Required: Yes

Note

If neither **-setup** nor **-hold** are specified the uncertainty value is applied to both.

---

set_clock_latency

Sets the latency of a clock. VPR automatically calculates on-chip clock network delay, and so only source latency is supported.

Source clock latency corresponds to the delay from the true clock source (e.g. off-chip clock generator) to the on-

chip clock definition point.

```
#Sets the source clock latency of 'clk' to 1.0
set_clock_latency -source 1.0 [get_clocks {clk}]
```

set_clock_latency

-source

Specifies that the latency is the source latency.

Required: Yes

-early

Specifies that the latency applies to early paths.

Required: No

-late

Specifies that the latency applies to late paths.

Required: No

<latency>

The clock's latency.

Required: Yes

[get_clocks <clock list or regexes>]

Specifies the clock domain(s).

Required: Yes

Note

If neither -**early** nor **-late** are specified the latency value is applied to both.

_____

set_disable_timing

Disables timing between a pair of connected pins in the netlist. This is typically used to manually break combinational loops.

```
#Disables the timing edge between the pins 'FFA.Q[0]' and 'to_FFD.in[0]' on
set_disable_timing -from [get_pins {FFA.Q\\[0\\]}] -to [get_pins {to_FFD.in\\[0\\]}]
```

set_disable_timing

-from [get_pins <pin list or regexes>]

Specifies the source netlist pins.

See also

VPR's pin naming convention.

Required: Yes

-to [get_pins <pin list or regexes>]

Specifies the sink netlist pins.

See also

VPR's pin naming convention.

Required: Yes

Note

Make sure to escape the characters in the regexes.

Special Characters

 # (comment), \\ (line continued), * (wildcard), {} (string escape)

starts a comment – everything remaining on this line will be ignored.

**** at the end of a line indicates that a command wraps to the next line.

***** is used in a **get_clocks/get_ports** command or at the end of **create_clock** to match all netlist clocks. Partial wildcarding (e.g. **clk*** to match **clk** and **clk2**) is also supported. As mentioned above, ***** can be used in set_input_delay and set_output_delay to refer to the netlist clock for single-clock circuits only, although this is not supported in standard SDC.

{} escapes strings, e.g. **{top^clk}** matches a clock called **top^clk**, while **top^clk** without braces gives an error because of the special **^** character.

SDC Examples

The following are sample SDC files for common non-default cases (assuming netlist clock domains clk and clk2).

A

Cut I/Os and analyse only register-to-register paths, including paths between clock domains; optimize to run as fast as possible.

```
create_clock -period 0 *
```

B

Same as A, but with paths between clock domains cut. Separate target frequencies are specified.

```
create_clock -period 2 clk
create_clock -period 3 clk2
set_clock_groups -exclusive -group {clk} -group {clk2}
```

C

Same as B, but with paths to and from I/Os now analyzed. This is the same as the multi-clock default, but with custom period constraints.

```
create_clock -period 2 clk
create_clock -period 3 clk2
create_clock -period 3.5 -name virtual_io_clock
set_clock_groups -exclusive -group {clk} -group {clk2}
set_input_delay -clock virtual_io_clock -max 0 [get_ports {*}]
set_output_delay -clock virtual_io_clock -max 0 [get_ports {*}]
```

D

Changing the phase between clocks, and accounting for delay through I/Os with set_input/output delay constraints.

```
#Custom waveform rising edge at 1.25, falling at 2.75
create_clock -period 3 -waveform {1.25 2.75} clk
create_clock -period 2 clk2
create_clock -period 2.5 -name virtual_io_clock
set_input_delay -clock virtual_io_clock -max 1 [get_ports {*}]
set_output_delay -clock virtual_io_clock -max 0.5 [get_ports {*}]
```

E

Sample using many supported SDC commands. Inputs and outputs are constrained on separate virtual clocks.

```
create_clock -period 3 -waveform {1.25 2.75} clk
create_clock -period 2 clk2
create_clock -period 1 -name input_clk
create_clock -period 0 -name output_clk
set_clock_groups -exclusive -group input_clk -group clk2
set_false_path -from [get_clocks {clk}] -to [get_clocks {output_clk}]
set_max_delay 17 -from [get_clocks {input_clk}] -to [get_clocks {output_clk}]
set_multicycle_path -setup -from [get_clocks {clk}] -to [get_clocks {clk2}] 3
set_input_delay -clock input_clk -max 0.5 [get_ports {in1 in2 in3}]
set_output_delay -clock output_clk -max 1 [get_ports {out*}]
```

F

Sample using all remaining SDC commands.

```
create_clock -period 3 -waveform {1.25 2.75} clk
create_clock -period 2 clk2
create_clock -period 1 -name input_clk
create_clock -period 0 -name output_clk
set_clock_latency -source 1.0 [get_clocks{clk}]

#if neither early nor late is specified then the latency applies to early paths
set_clock_groups -exclusive -group input_clk -group clk2set_false_path -from [get_clocks{clk}] -to
[get_clocks{output_clk}]
set_input_delay -clock input_clk -max 0.5 [get_ports{in1 in2 in3}]
set_output_delay -clock output_clk -min 1 [get_ports{out*}]
set_max_delay 17 -from [get_clocks{input_clk}] -to [get_clocks{output_clk}]
set_min_delay 2 -from [get_clocks{input_clk}] -to [get_clocks{output_clk}]
set_multicycle_path -setup -from [get_clocks{clk}] -to [get_clocks{clk2}] 3
#For multicycle_path, if setup is specified then hold is also implicitly specified
set_clock_uncertainty -from [get_clocks{clk}] -to [get_clocks{clk2}] 0.75
#For set_clock_uncertainty, if neither setup nor hold is unspecified then uncertainty is applied to both
set_disable_timing -from [get_pins {FFA.Q\\[0\\]}] -to [get_pins {to_FFD.in\\[0\\]}]
```

5. Device Configuration

The Raptor Device Programmer allows you to program and configure Rapid Silicon FPGA devices. After compiling your design, the next step is generating the device image and download into the target FPGA device or flash. Use the Raptor device programming Tcl commands to program your target device. The supported programmer's modes are JTAG and Master SPI interface programming via third-party QSPI serial configuration devices.

5.1 Device programming via JTAG mode

Device Configuration Steps

1. Generate the bitstream using Raptor compilation tool.
2. Connect the board with a JTAG USB cable.
3. Connect the power cable attached to the board to a power source.
4. Turn on the power to the board.
5. Launch Raptor software.
6. In the TCL console, type the following command:

programmer fpga config <bitstream file path>

<bitstream file path>

Specifies the bitstream file to be programmed into the FPGA target device. The specified bitstream file (.cfgbit) must be a valid full path of the file. The bitstream file can be located anywhere in the machine.

Required: Yes

7. Example of programming a Gemini device with helloworld.cfgbit bitstream file.

programmer fpga_config /home/asic01/development/helloworld.cfgbit

A successful programming figure is as below

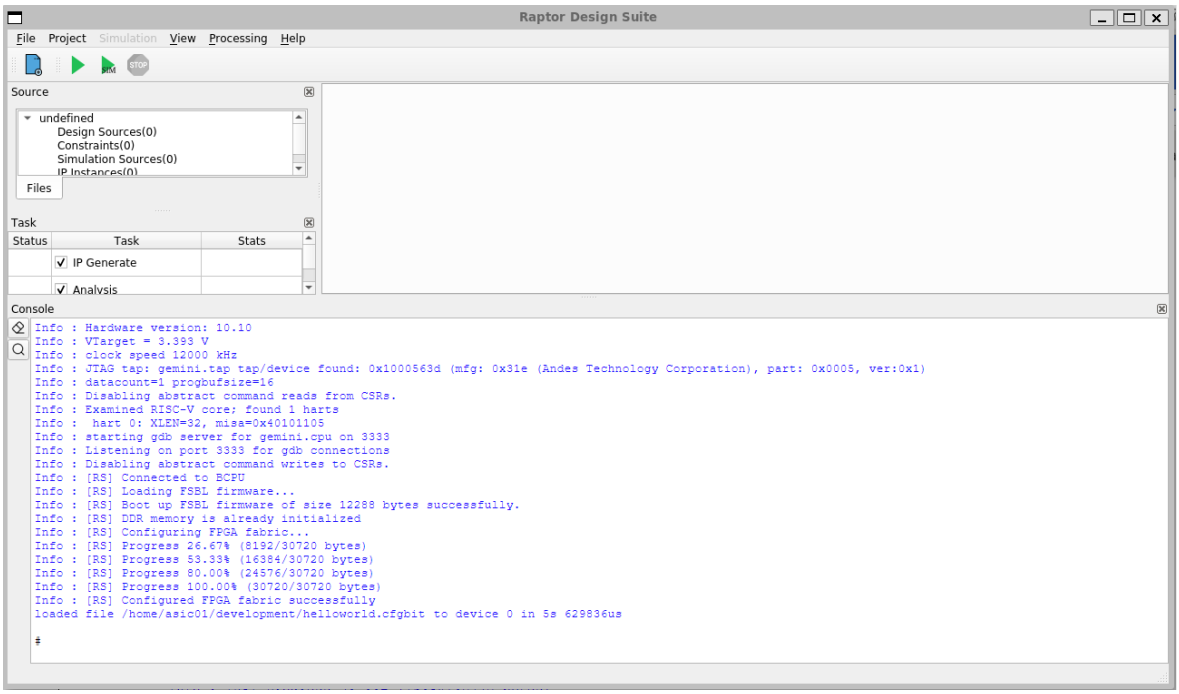


Figure 57. Example of a Successful Programming

The following figure demonstrates an error is returned when there is invalid bitstream input

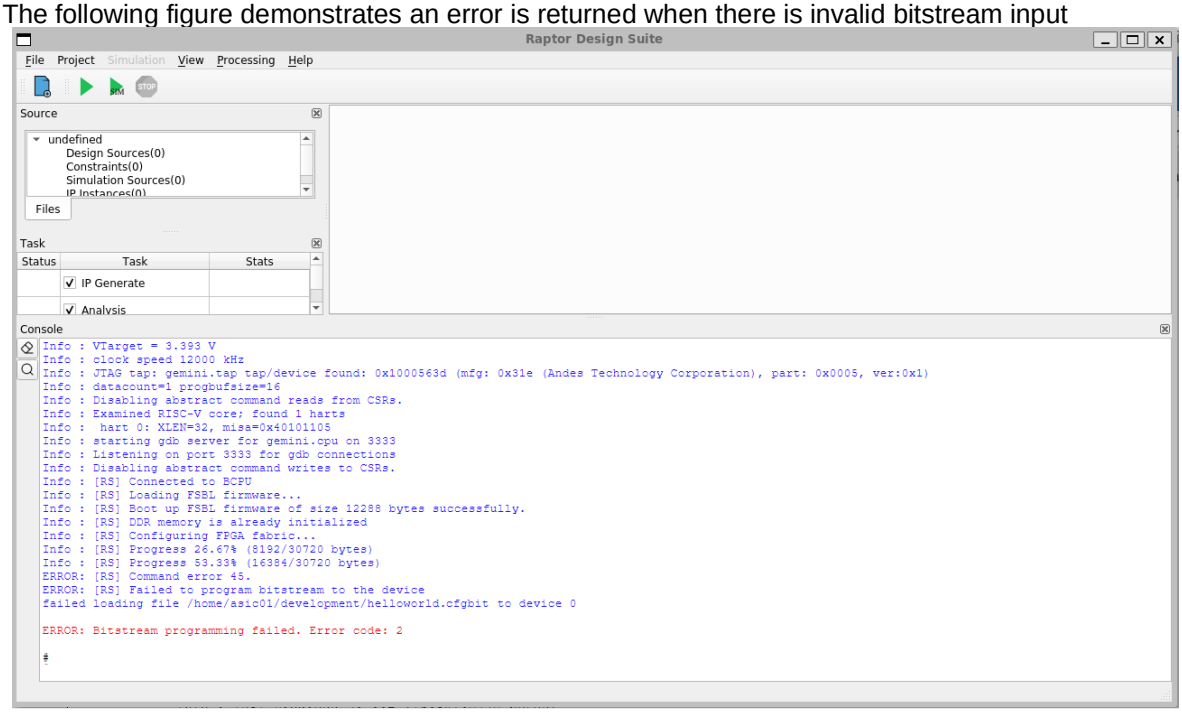


Figure 58. Invalid Bitstream Error

The following figure demonstrates an error is returned when JTAG cable is not attached.

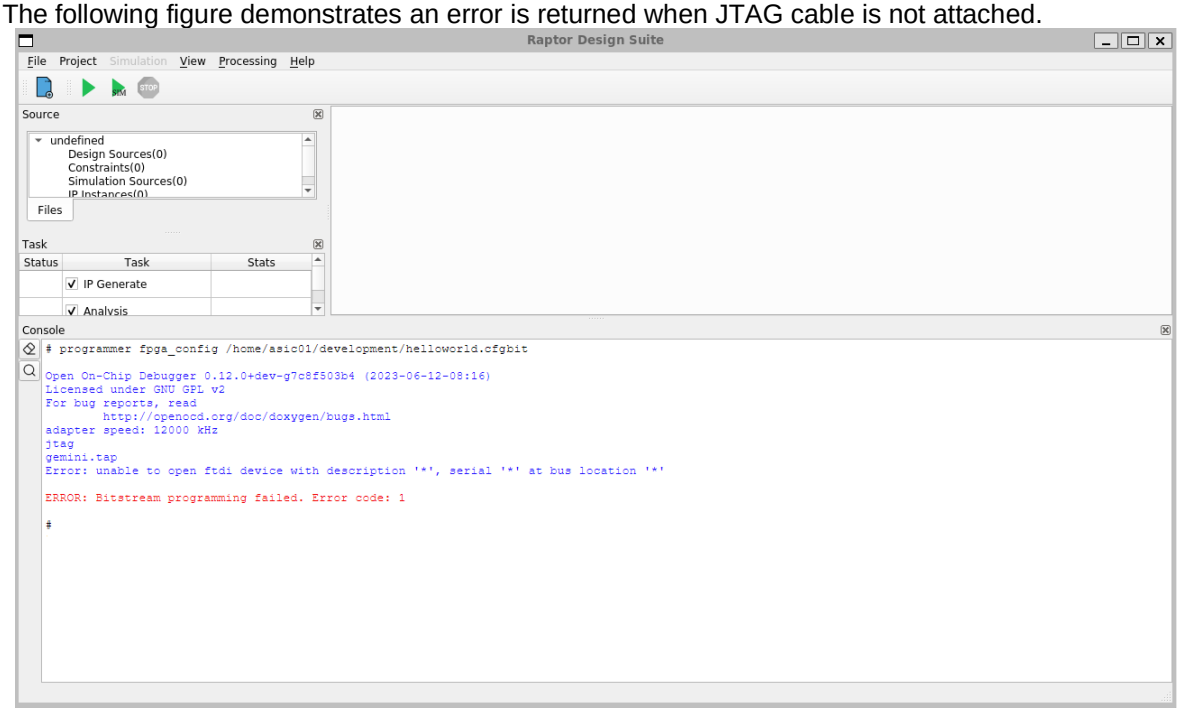


Figure 59. JTAG Cable not Attached Error

The following figure demonstrates an error is returned when there is no device attached.

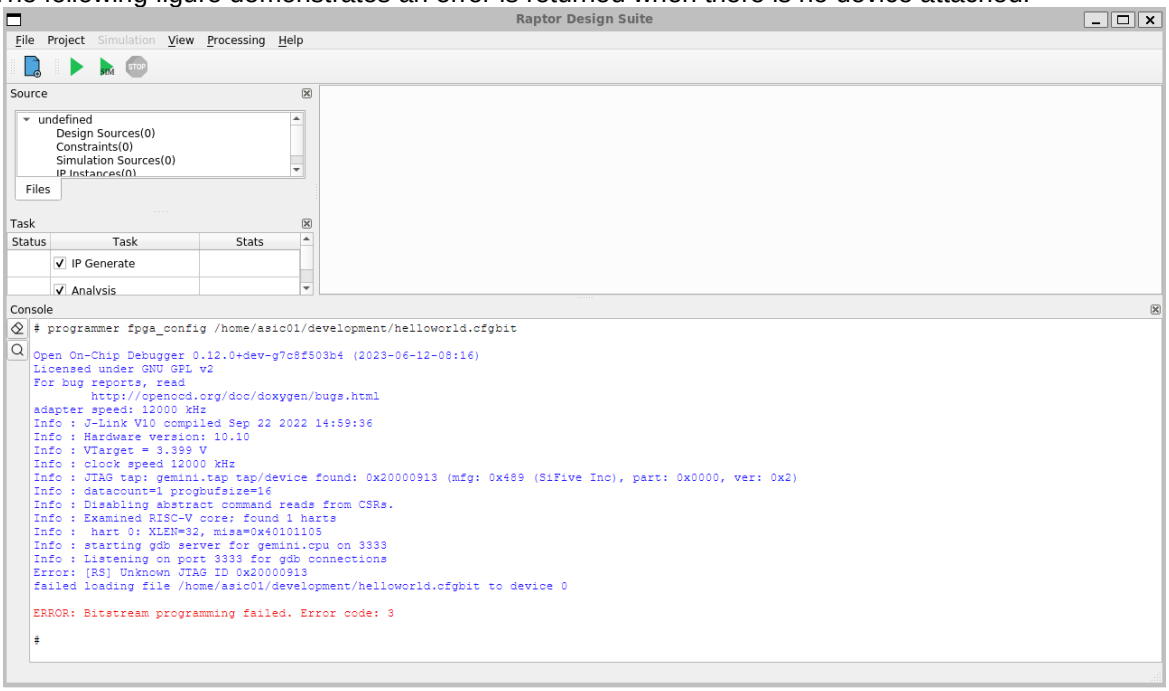


Figure 60. No Device Attached Error

5.2 Flash programming via JTAG mode

The Raptor Device Programmer also supports flash programming. You can use the Programmer to load an FPGA configuration bitstream file into a Quad SPI flash memory device. The Quad SPI flash memory device subsequently loads the configuration data into the target.

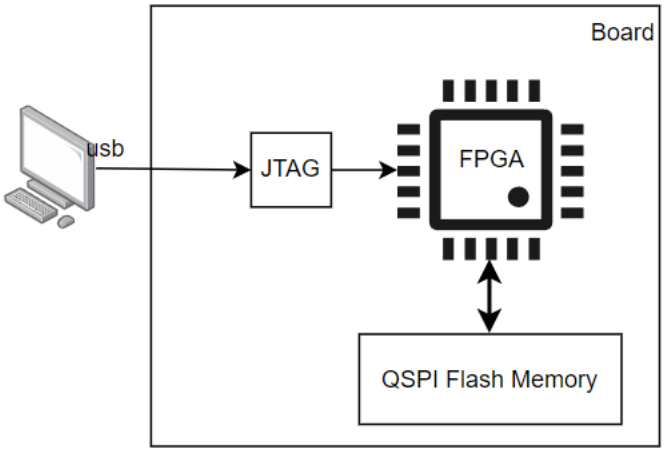


Figure 61. Programming Flash using JTAG

Flash Programming Steps

1. Generate the bitstream using Raptor compilation tool.
2. Connect the board with a JTAG USB cable.
3. Connect the power cable attached to the board to a power source.
4. Turn on the power to the board.
5. Launch Raptor software.
6. In the TCL console, type the following command:

   programmer flash <bitstream file path>

7. Example of programming a Flash device with helloworld.cfgbit bitstream file

   programmer flash /home/asic01/development/helloworld.cfgbit

5.3 FPGA Status

Configuration signals "cfg_done" and "cfg_error" from FPGA can be monitored. To query these signals, use the following command:

programmer fpga_status

Example of the output from *programmer fpga_status* command.

```
            | Device              |  cfg_done | cfg_error
----------- --------------------- ----------- ----------
Found        1 Gemini               1           0
```

Figure 62. Programmer Fpga_status Output

For "cfg_done" column, it indicates the status of the FPGA configuration process. The value "1" means that the configuration was successful. Additionally, if there were no errors during FPGA configuration, it is indicated by "cfg_error" being value "0".

6 Using Tcl Scripting

The Tool Command Language, or Tcl, is one of the languages used to interface to the Rapid Silicon Raptor design suite. Rapid Silicon has adopted Tcl as a standard Application Programming Interface (API). The Tcl interpreter built-in to the Raptor Design Suite provides Tcl the ability to control the application, access design objects, and create reports.

Tcl provides built-in commands to read and write files within the local Raptor file system. These commands allow users to perform tasks such as creating directories, adding files, starting design projects, and running synthesis. The results of the project run are output to a file that can be reviewed by the designer to check for any issues.

6.1 Tcl Overview

A Tcl script is a series of Tcl commands. Each command is separated by either a new-line or a semicolon. The built-in Tcl interpreter described above breaks individual commands into segments and performs substitutions as needed. The Tcl interpreter reads a line from left to right. Similarly, command and variable substitutions are also performed from left to right.

A Tcl word is a string that can be represented as follows:

- A single word
- Multiple words within braces, { }
- Multiple words within quotation marks, " "

Semicolons, brackets, tabs, spaces, and new lines within quotation marks or braces are treated as ordinary characters.

The first word identifies the command, and all subsequent words are passed to the command as arguments.

NOTE: In Tcl, a backslash \ is treated as a special character even within braces and quotation marks.

6.2 Loading and Running Tcl Scripts

The Raptor Design Suite offers three ways to load and run a Tcl script during a design session

- Load script files automatically when the Raptor design suite is launched
- Run a script from the Tcl command line
- Add the Tcl script to the menu selections in the Raptor design suite.

6.2.1 Automatically Load Tcl Scripts

The Raptor Design Suite can automatically load Tcl scripts that are defined in the raptor_init.tcl file. The benefit of this method is that Tcl procedures which define new commands are made available to all Raptor design sessions.

6.2.2 Invoking a Tcl Script at Power-up

A Tcl script to be executed at power-up from the command line using the -script option:

```
raptor --script <path-to-script>
```

6.2.3 Tcl Script Source Command

The source command allows the Tcl script files to be manually loaded into the Raptor design suite.

6.2.4 Adding Constraints to Tcl Scripts

Tcl scripts can be added to project constraint sets like any regular Raptor file. However, since the Raptor files are managed by the tool and not Tcl scripts, any constraints defined by a Tcl script and edited by the tool cannot be saved back to the Tcl script automatically. Rather, the edited file must be saved to memory, then it can be used to manually update the Tcl script.

6.3 Writing a Tcl Script

When writing a Tcl script, the script should provide the same type of functionality as the embedded Raptor commands. An example would be the Help command. When the user types Help on the command line, the output should provide the same relative amount of information as would be displayed when clicking on the Help button in the GUI.

6.3.1. Adding Arguments to a Tcl Script

When writing a Tcl script, it is advantageous for the programmer to add arguments to a procedure. These arguments help to reduce the amount of redundant code, while at the same time allowing the procedure to handle multiple contexts.

6.3.2. Types of Variables

A Tcl script typically contains both local and global variables. A local variable is created inside a Tcl procedure and is only accessible within that procedure. As such, it is not subject to name collision with the same variable name outside the procedure.

6.3 Loop Control and Iteration

The Tcl language uses the following commands to implement loops and perform iterations through a loop:

- for
- foreach
- while

6.4. Error Handling

The Raptor design suite stops at the first Tcl error it encounters. If the error is coming from a script, it points to the file and line number. It also shows a Tcl stack trace.

6.5.1 Corner Cases and Incorrect Script Usage

When writing a Tcl scripts, Rapid Silicon recommends that the user check for corner cases or other conditions that could cause the code to fail. These checks can sometimes indicate the script was being used incorrectly. Checking for these conditions is important to avoid having the script stop abruptly. Without these checks, the user will not know what went wrong or what corrective action should be taken.

6.5.2. Tcl Script Errors

As mentioned in the previous section, an error in the Tcl script can abruptly halt the execution of the script. If this occurs, the Raptor design suite will generate a Tcl error message.

6.6 Interfacing to External Programs

The Tcl interpreter inside the Raptor design suite allows for calling external programs from within Tcl and capturing the result of that program. The user should be explicit when calling external programs or commands to ensure data integrity.

6.7 Tcl Commands

Raptor executable options:

--help : This help

--version : Version

--batch : Tcl only, no GUI

--script <script>: Execute a Tcl script

--project <project file>: Open a project

--mute : mutes stdout in batch mode

The following is a list of Tcl commands supported by the Raptor design suite. Tcl commands are available in GUI mode, Batch console, or Batch script. For bellow table, arguments inside two question marks is optional: ?-optional argument?

General Tcl Commands:

Tcl Command & Arguments	Description
help	Help
chatgpt <command> "<message>" ?-c <path>? send reset -c <path>	Send message to ChatGPT Command to send message Command to reset history Specify ini file path with API key. The key needs to be set only once for a session [OpenAI] API_KEY: <api key>

Project Tcl Commands:

Tcl Command & Arguments	Description
create_design <name> ?clean? ?-type <project type>? <project type> clean	Creates a design with <name> name rtl (Default), gate-level If project folder already exists, remove recursively folder content
open_project <file>	Opens a project
run_project <file>	Opens and immediately runs the project
target_device <name>	Targets a device with <name> (default is 1GE75)
add_design_file <file list> ?type? ?-work <libName>? ?-L <libName>? <type> -work <libName> -L <libName>	 -VHDL_1987, -VHDL_1993, -VHDL_2000, -VHDL_2008, -VHDL_2019, - V_1995, -V_2001, -SV_2005, -SV_2009, -SV_2012, -SV_2017, default auto-detect Compiles the compilation unit into library <libName>, default is "work" Import the library <libName> needed to compile the compilation unit, default is "work"
set_top_module <top> ?-work <libName>?	Sets the top-level design module/entity for synthesis
add_include_path <paths>	Specify paths for Verilog includes (Not applicable to VHDL)
add_library_path <paths>	Specify paths for libraries (Not applicable to VHDL)
add_library_ext <ext>	Spcify library extensions (Not applicable to VHDL)
set_macro <name>=<value>	As in -D<macro>=<value>
read_netlist <file>	Read a netlist (.blif/.eblif) instead of an RTL design (Skip Synthesis)
add_constraint_file <file>	Sets constraints file (SDC) filename and location
add_simulation_file <file list> ?type? ?-work <libName>? ?-L <libName>? <type> -work <libName> -L <libName>	 -VHDL_1987, -VHDL_1993, -VHDL_2000, -VHDL_2008, -VHDL_2019, - V_1995, -V_2001, -SV_2005, -SV_2009, -SV_2012, -SV_2017, -C, -CPP Compiles the compilation unit into library <libName>, default is "work" Import the library <libName> needed to compile the compilation unit, default is "work"
set_top_testbench <module>	Sets the top-level testbench module/entity for simulation
clear_simulation_files	Remove all simulation files
script_path	Returns the path of the Tcl script passed with --script
max_threads <-1/[2:64]>	Maximum number of threads to be used (-1 is for automatic selection)
set_device_size XxY	Device fabric size selection

custom_openfpga_script <file>	Uses a custom OpenFPGA templatized script
architecture <vpr_file.xml> ?<openfpga_file.xml>?	Uses the architecture file and optional openfpga arch file (For bitstream generation)

Constraints Tcl Commands:

Tcl Command & Arguments	Description
keep <signal list> OR all_signals	Keeps the list of signals or all signals through Synthesis unchanged (unoptimized in certain cases)
message_severity <message_id> <ERROR/WARNING/INFO/IGNORE>	Upgrade/downgrade message severity
set_pin_loc <design_io_name> <device_io_name> ?<internal_pinname> ?	Constraints pin location (Use in constraint.pin file)
set_property mode <io_mode_name> <device_io_name>	Constraints pin mode (Use in constraint.pin file)
set_clock_pin -device_clock <device_clock_name> -design_clock <design_clock_name>	like gemini has 16 clocks clk[0],clk[1]....,clk[15] and e.g. user clocks are clk_a, clk_b and want to map clk_a with clk[15] constraints: set clocks for repacking constraints (Use in constraint.pin file)

IP Tcl Commands:

Tcl Command & Arguments	Description
add_litex_ip_catalog <directory>	Browses directory for LiteX IP generators, adds the IP(s) to the IP Catalog
ip_catalog ?<ip_name>?	Lists all available IPs, and their parameters if <ip_name> is given
ip_configure <IP_NAME> -mod_name <name> -out_file <filename> -version <ver_name> -P<param>="<value>"...	Configures an IP <IP_NAME> and generates the corresponding file with module name
ipgenerate ?clean? clean	Generates all IP instances set by ip_configure Deletes files generated from this task
simulate_ip <module name>	Simulate IP with module name <module name>

Simulation Tcl Commands:

Tcl Command & Arguments	Description
simulation_options <simulator> <phase> ?<level>? <options> <phase>	Sets the simulator specific options for the specified phase compilation, elaboration, simulation, extra_options
simulate<level> ?<simulator>? ?<waveform_file>? <level> <simulator>	Simulates the design and testbench rtl, gate, pnr, rtl : RTL simulation, gate: post-synthesis simulation, pnr: post-pnr simulation verilator, ghdl, icarus
wave_*	All wave commands will launch a GTKWave process if one hasn't been launched already. Subsequent commands will be sent to the launched process
wave_cmd ...	Sends given tcl commands to GTKWave process. See GTKWave docs for gtkwave :: commands

wave_open <filename>	Load given file in current GTKWave process
wave_refresh	Reloads the current active wave file
wave_show <signal>	Add the given signal to the GTKWave window and highlight it
wave_time <time>	Set the primary marker to <time>. Time units can be specified, without a space. Ex: wave_time 100ps

Synthesis Tcl Commands:

Tcl Command & Arguments	Description
synth_options <option list>	
-effort <level>	Optimization effort level (high, medium, low)
high	Most compute, generally impacting runtime (default)
medium	Balanced compute
low	least compute, least runtime
-fsm_encoding <encoding>	FSM encoding (binary, onehot)
binary	Compact encoding - using minimum of registers to cover the N states
onehot	One hot encoding - using N registers for N states (default)
-carry <mode>	Carry logic inference mode (all, auto, none)
all	Infer as much as possible
auto	Infer carries based on internal heuristics (default)
none	Do not infer carries
-clke_strategy <strategy>	Clock enable extraction strategy for FFs (early, late)
early	Perform early extraction (default)
late	Perform late extraction
-fast	Perform the fastest synthesis. QoR can be impacted
-no_flatten	Do not flatten design
-no_simplify	Do not run special simplification algorithms in synthesis
-clke_strategy <strategy>	Clock enable extraction strategy for FFs
early	Perform early extraction
late	Perform late extraction
-cec	Dump verilog after key phases and use internal equivalence checking (ABC based)
set_limits <type> <int>	Sets a user limit on object of type (dsp, bram), specify 0 to disable block inferrence
dsp	Maximum number of usable DSPs
bram	Maximum number of usable BRAMs
carry_length	Maximum carry length
analyze ?clean?	Analyzes the RTL design, generates top-level, pin and hierarchy information
clean	Deletes files generated from this task
synthesize <optimization> ?clean?	RTL Synthesis, optional opt. (area, delay, mixed)
<optimization>	area, delay, mixed
area	Optimize for reduce resource area
delay	Optimize for performance
mixed	Optimize for area and performance (default)

clean	Deletes files generated from this task
synthesis_type Yosys/QL/RS	Selects Synthesis type
custom_synth_script <file>	Uses a custom Yosys templatized script

Packing Tcl Commands

Tcl Command & Arguments	Description
pnr_netlist_lang <format>	Chooses post-synthesis netlist format, (blif, edif, verilog, vhdl)
packing_options <option list>	Packing options
-clb_packing <directive>	Performance optimization flags (auto, dense, timing_driven)
auto	CLB packing automatically determined (default)
dense	Pack logic more densely into CLBs resulting in fewer utilized CLBs however may negatively impact timing
timing_driven	Pack logic to optimize timing
packing ?clean?	Packing
clean	Deletes files generated from this task

Place Tcl Commands:

Tcl Command & Arguments	Description
pin_loc_assign_method <method>	Algortihm for automatic pin assignment (in_define_order, random, free)
in_define_order	Port order pin assignment (default)
random	Random pin assignment
free	No automatic pin assignment
place ?clean?	Placer
clean	Deletes files generated from this task
pnr_options <option list>	PnR Options

Route Tcl Commands:

Tcl Command & Arguments	Description
route ?clean?	Router
clean	Deletes files generated from this task
set_channel_width <int>	VPR Routing channel setting

Static Timing Analysis Tcl Commands:

Tcl Command & Arguments	Description
sta ?clean? clean	Statistical Timing Analysis Deletes files generated from this task

Bitstream Tcl Commands:

Tcl Command & Arguments	Description
bitstream ?force? ?clean? ?enable_simulation? ?write_xml? ?write_fabric_independent? ?pb_pin_fixup? clean	Bitstream generation Deletes files generated from this task
bitstream_config_files -bitstream <bitstream_setting.xml> -sim <sim_setting.xml> -repack <repack_setting.xml> -key <fabric_key.xml>	Uses alternate bitstream generation configuration files

Program & Debug Tcl Commands:

Tcl Command & Arguments	Description
programmer <command> command fpga_config <bitstream_file> flash <bitstream_file> fpga_status list_devices	fpga_config, flash, fpga_status, list_devices Program the device via JTAG Program the flash via JTAG Get the status of the device List all available devices
diagnostic <type>	Debug mode. Types: packer

6.8 Using ChatGPT in Raptor

1. Get an API key from ChatGPT website and save it to a *.ini file as shown below

```
[OpenAI]
API_KEY: <your_api_key>
```

2. For first communication, the API key needs to be set using the "-c" flag for "chatgpt" tcl commands. Once set, setting the key again is optional unless it changes or expires. The user can change API key with any ChatGPT tcl commands. Example is shown below.

```
chatgpt send "Hello world" -c <path-to-config>.ini
chatgpt send "Hello world again"
```

3. The responses from ChatGPT will be available in the Raptor's editor. One can right-click and copy the responses to another file/s.

In figure below, we asked ChatGPT to create a 10 bit adder followed by a request to create a test bench for it.

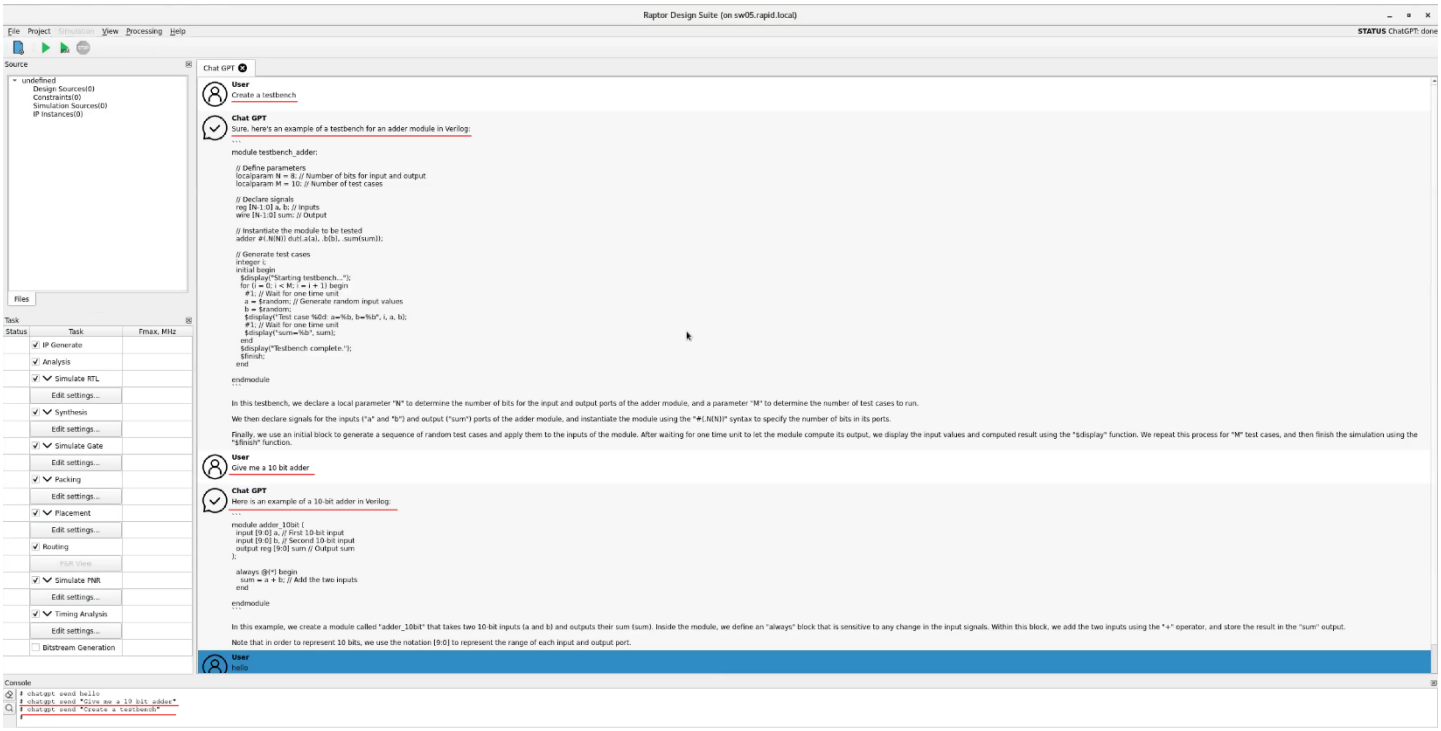


Figure 63: ChatGPT Window in Raptor

4. The chat history can be reset as shown below

```
chatgpt reset
```

7. Examples

7.1 Tcl Examples

The installation directory contains several Tcl script example designs under;

```
<install_path>/share/raptor/tcl_examples
```

For instance, refer to:

```
raptor   --script   <install_path>/share/raptor/tcl_examples/aes_decrypt_fpga/aes_decrypt.tcl
```

(GUI mode)

Or

```
raptor --script   <install_path>/share/raptor/tcl_examples/aes_decrypt_fpga/aes_decrypt.tcl --batch
```

(Batch mode)

7.2 Project Examples

The installation directory contains several GUI project examples. From the GUI you can select them using the menu:

```
File → Select example design
```

7.3 Example Designs

Raptor installation contains example designs, there are two type of example design.

- LiteX SoC based example design
  - Based on VEXRISCV Soft CPU, LiteX, Migen generated IPs and Native Verilog IP
  - Design Name:
    - vexrisc_axi_gpio
    - vexrisc_axi_ram
    - vexrisc_axi_hello_world
- Native RTL example SoC Design:
  - This has VEXRISCV Soft CPU, AXI interconnect and AXI BRAM, It also has Bare Metal C Code to perform test on the design. User can simulate this design using Verilator simulator and compile on Raptor
  - Design name: vexriscv_axi_ram
- How to Run LiteX SoC simulation:
  - Command: <Raptor Installation area> /share/envs/litex/bin/python3 litex_sim_rs.py
  - Each of example design has README file, which describes the steps to simulate with Verilator and compile the design on Raptor
- Path of example designs: <Raptor Installation area> /share/litex_reference_design

Notes:

- LiteX is installed on user machine as a part of Raptor Installation setup. For more information refer to *Raptor Installation Guide*.
  - LiteX installation path: `<Raptor Installation area> /share/envs/litex/bin`
- LiteX dependency and how to install these:
  - Python dependency for LiteX:
    - Command to install: `pip3 install meson ninja`
  - Verilator simulator & other OS dependency:
    - Command to install: `sudo apt install libevent-dev libjson-c-dev verilator`
  - RISC-V Tool Chain:
    - CentOS: https://static.dev.sifive.com/dev-tools/freedom-tools/v2020.08/riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux-centos6.tar.gz
    - Ubuntu: https://static.dev.sifive.com/dev-tools/freedom-tools/v2020.08/riscv64-unknown-elf-gcc-10.1.0-2020.08.2-x86_64-linux-ubuntu14.tar.gz

8 Formal Verification

The Raptor post-synthesis netlist can be formally verified through Formal Verification tools like OneSpin or Formality. This formal verification consists to compare the original RTL design description versus the post-synthesis netlist. Note that this verification requires to add to the netlist description the associated Rapid Silicon primitives that are defined in file "cells_sim.v" or "cells_sim.vhd" depending if the netlist output format is Verilog or VHDL.

8.1 Formal Verification Using VHDL

The following figure is a typical tcl script to provide to OneSpin in order to compare for instance the RTL description of design "rtea" versus the post-synthesis netlist in VHDL:

```
load_settings ec_fpga_rtl

read_vhdl -golden -pragma_ignore {} -version 2008 {
        rtea.vhdl
}
set_elaborate_option -golden -top {vhdl!work.rtea(behave)}

read_vhdl -revised -pragma_ignore {} -version 2008 {
        cells_sim.vhd \
        post_synthesis_netlist.vhd
}
set_elaborate_option -revised -top {vhdl!work.rtea(arch)}

elaborate -both

compile -both
set_mode ec
map
compare

exit -force
```

Figure 64. OneSpin Tcl Script – VHDL

8.2 Formal Verification Using Verilog

The following figure is a typical tcl script to provide to OneSpin in order to compare for instance the RTL description of design "rtea" versus the post-synthesis netlist in Verilog:

```
load_settings ec_fpga_rtl

read_verilog -golden -pragma_ignore {} -version sv2012 {
        rtea.v
}
set_elaborate_option -golden -top {verilog!work.rtea}

read_verilog -revised -pragma_ignore {} -version sv2012 {
        cells_sim.v \
        post_synthesis_netlist.v
}

set_elaborate_option -revised -top {verilog!work.rtea}

elaborate -both

compile -both
set_mode ec
map
compare

exit -force
~
```

Figure 65. OneSpin Tcl Script - Verilog

8.3 Formal Verification Using Formality

Formality can also be used. A classic Formality script with VHDL RTL language can look like that shown in the following figure:

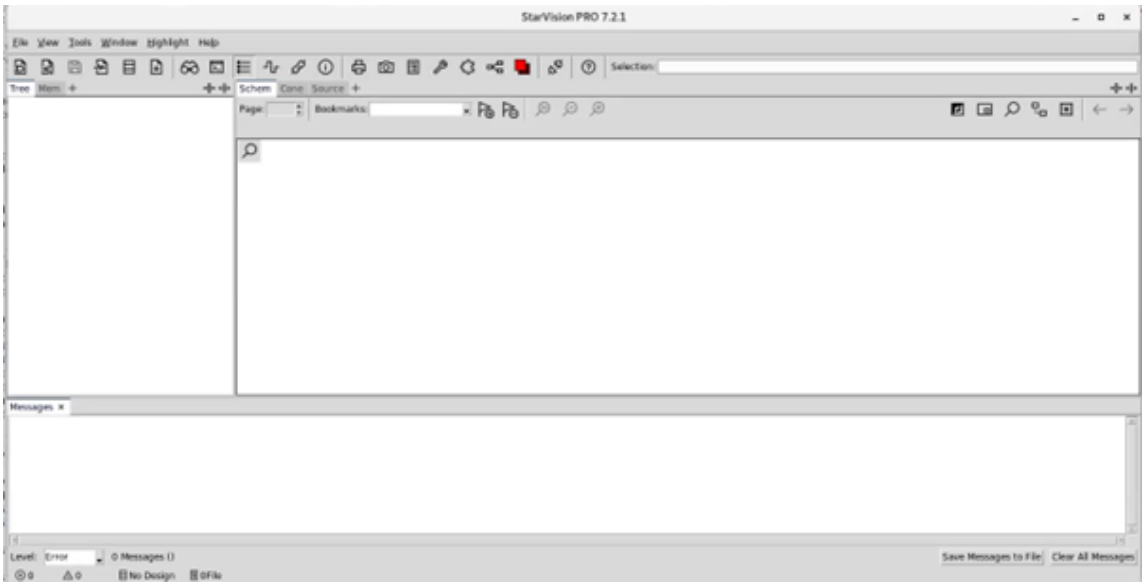
```
set synopsys_auto_setup true
set hdlin_vhdl_auto_file_order False

# = = = = = = = = = = = = = = = = = = = = = = = =
#  Ignore Design Elaboration Warnings
# = = = = = = = = = = = = = = = = = = = = = = = =
set_mismatch_message_filter -warn FMR_ELAB-116
set_mismatch_message_filter -warn FMR_ELAB-147
set_mismatch_message_filter -warn FMR_ELAB-149

set ReadRef [read_vhdl -container r -libname WORK  {\
 key_schedule.vhd, round_f.vhd, rtea.vhd
}]

set ref_top [set_top r:/WORK/rtea]

set ReadRev [read_vhdl -container i -libname WORK {
 cells_sim.vhd, post_synthesis_netlist.vhd
}]

set rev_top [set_top i:/WORK/rtea]

# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
#  Setup the Tool's Setting For Compare Point Matching
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

set message_x_source_reporting True
set name_match_based_on_nets False
set name_match port

# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
#  Setup the Tool's Setting For Verification Stage
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

set verification_timeout_limit 04:00:00
set verification_run_analyze_points False
set verification_merge_duplicated_registers False
set verification_failing_point_limit 1000
set verification_assume_reg_init Auto
set debug_mode False

if {$ReadRef && $ref_top && $ReadRev && $rev_top} {
        verify
} else {
        puts "  ERROR Reading Design Files"
}
 report_status > output_rtea.log
```

Figure 66. Using Formality Script in VHDL

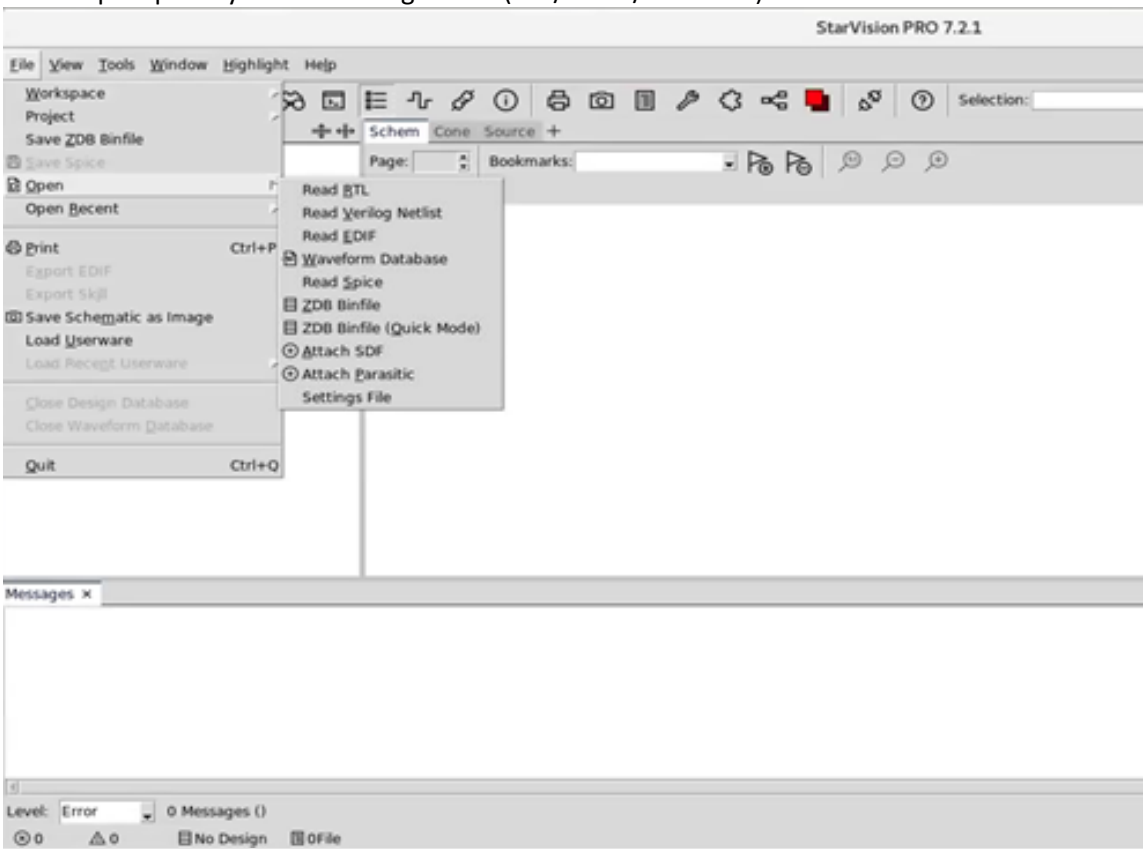
8.4 View Raptor Post-Synthesis Verilog Netlist in StarVision

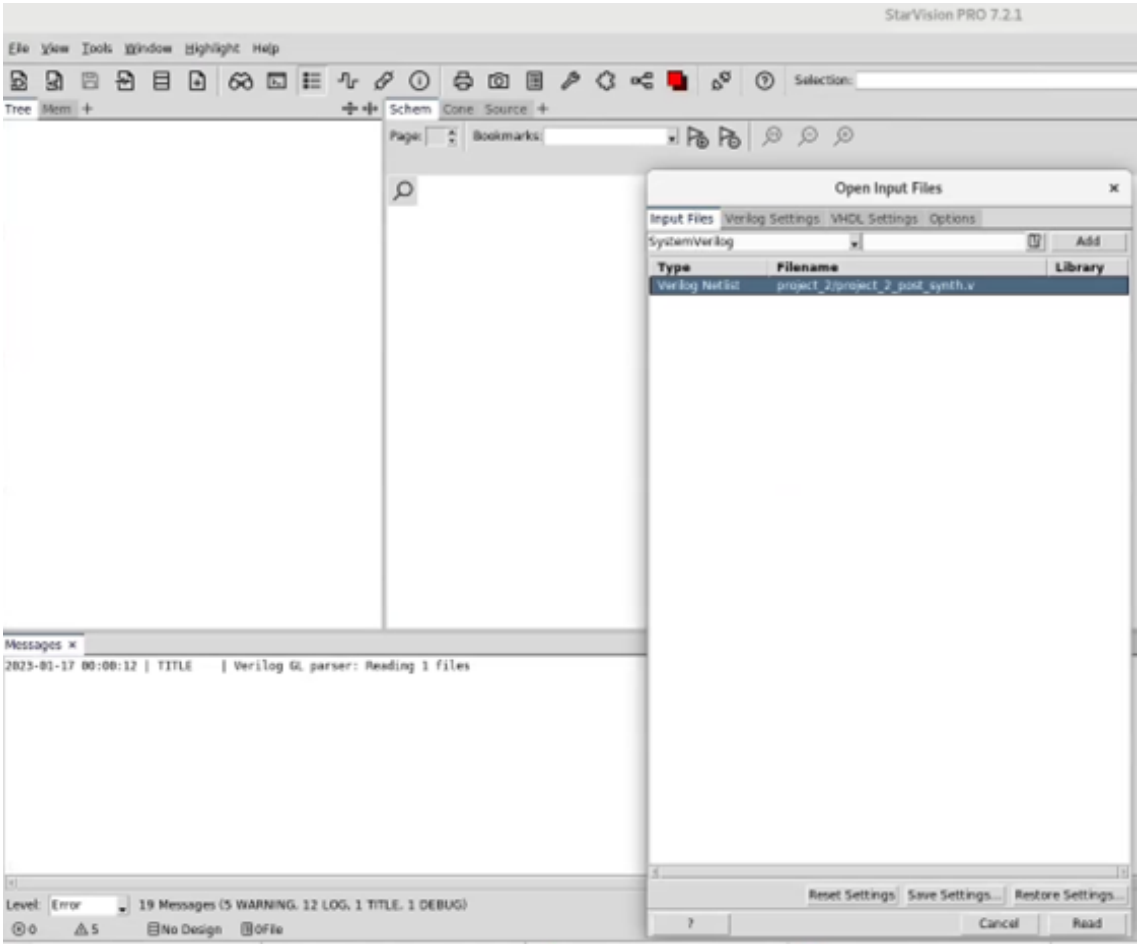
The following numbered procedure describes how to view the Raptor post-synthesis Verilog Netlist in the StarVision Pro software.

1. Compile the design using Rapid Silicon's Raptor Software
2. Generate post synthesis Verilog netlist
3. Invoke the StarVision Pro software as shown:

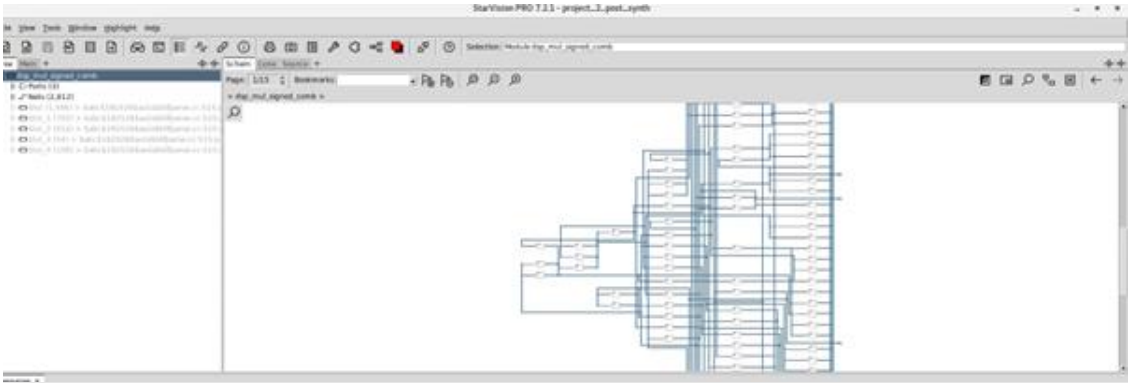


4. Load Raptor post Synthesis Verilog netlist (*.v /*.edf /*.edif file) in StarVision Pro UI





5. Analyse schematic view of Raptor Verilog netlist.



8.5 MVP Support

The Gemini device supports MVP equivalence checking using OneSpin. Customers should contact OneSpin for support.

Appendix A Raptor Design Suite Installation Guide

The access to Rapid silicon tools is given through the industry-standard FTP communication protocol. A username and a password provided by Rapid Silicon to be able to access our tool (in the form of a tar file). The downloaded
`*.tar.gz` file from FTP site contains following files:

1. README.md (this file)
2. Raptor installer exe which name is in the form "Raptor_x.x.run"

A.1 Installing Raptor

GUI mode:

Once downloaded, unzip the downloaded tar file either by double clicking on it and then extracting or using the below command:

tar –xvzf <Downloaded tar file>

Make sure to replace Downloaded tar file with correct name.

Once unzipping is done. Invoke the GUI mode of installer using the below command:

./<Raptor .run file>

Provide the Raptor .run file name.

This will pop up the GUI installer. Follow the on-screen instructions to do installation.

Note: When choosing installation folder, make sure you have correct write access.

Batch mode or non-gui mode:

You can also install the Raptor witout the GUI or in batch. In batch mode you must provide the following command line options:

- -b or --batch-mode -> running in interactive mode. Mandatory if you want to use -r or -i option
- -r or --raptor-home -> you are giving the absolute path of directory where Raptor will be installed
- --accept -> [Mandatory] Grant acceptance of license on command line.

Go to the directory where you have extracted the Raptor executable after downloading and execute it with any or with all above mentioned options like below

```
./Raptor_x.x.run -- -b  -r <absolute path where Raptor will be installed>
```

Replace `x.x` in the name above with the version of Raptor executable file you have received.

A.2 Raptor License Setup

Rapid Silicon uses FlexLM based license technology to license its product. The FlexLM depends upon lsb-core. Kindly install them as per your OS.

Depending upon your request, you may have received node located or floating type license.

Node Locked License

To setup node locked license using Raptor GUI, click on Help then on Manage license, a dialog box will pop up as shown below:

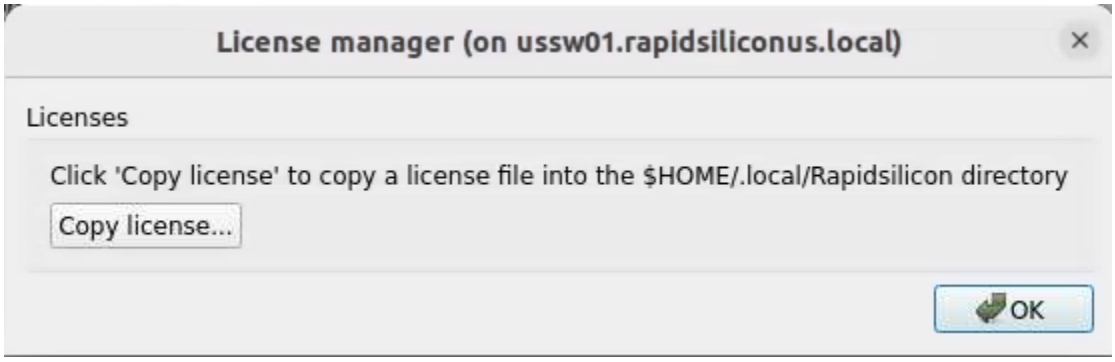


Figure 67. License Manager

Click on copy license and navigate to license file. Select the license file and click OK. This will copy the license file to default location.

To start a node locked license by setting an environment variable

```
export LM_LICENSE_FILE=<path to license file.lic>
```

Floating License

To start the floating type license file:

1. Place the license file in the directory of your choice.
2. In license file, write the name of your machine before host id and port after the host id
3. In the license file, correct the path of `rapidsil` daemon
4. Now start the FlexLM daemon per the command below

```
```

```
${Raptor install directory}/bin/Flex_LM/lmgrd -c <path of license file> -l <path to save the log file>
```

Now go to the machine where you have installed the Raptor and set the below variable:

```
export LM_LICENSE_FILE=<port>@<ip address of license server>
```

For further details, refer to the *FlexLM License Server Administration Guide*.

A.3 Verify the Installation

The user can verify the installation using the following commands.

```
source <install_path>/raptorenv_lin64.sh
raptor --version
raptor  (To verify the GUI compone
```