NOKIA

FPGA HACKATHON

Project Guide

Hackathon – Project Guide

Introduction

In this instruction, you will find important information that is common to all tasks prepared for FPGA Hackathon 2025.

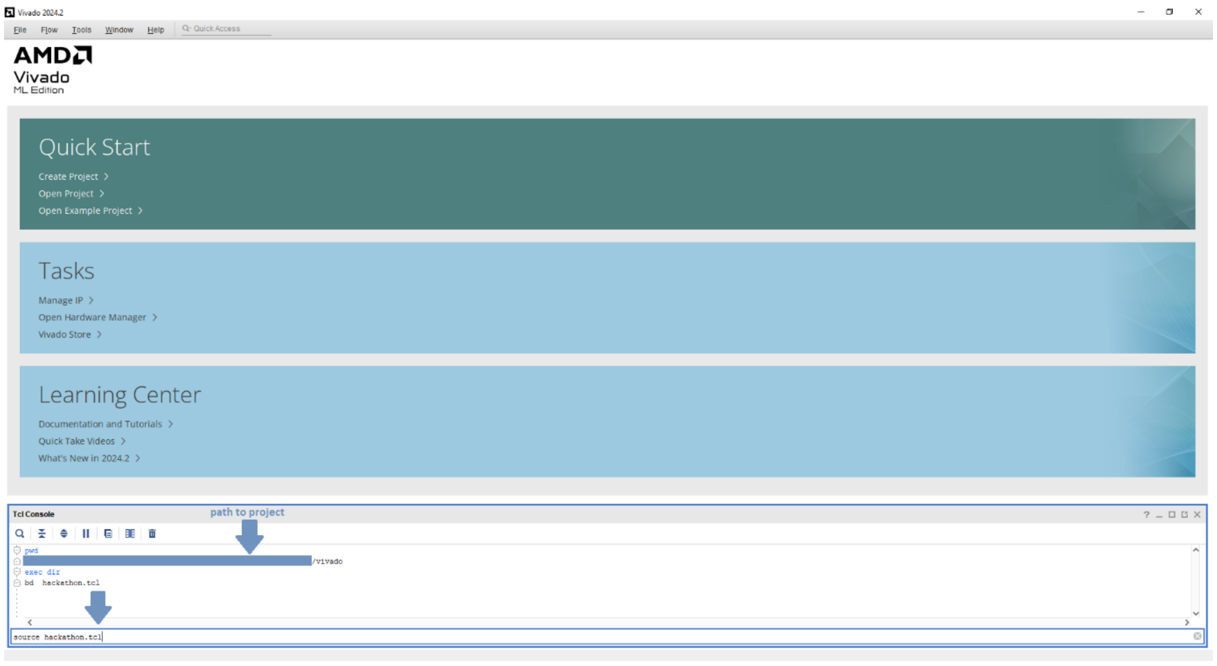
1. Project usage

1.1. How to launch the base project?

You can find instructions on how to download the base project in the **FPGA_Hackathon_2025_User_Manual.pdf**

To launch a project, open Vivado 2024.2 and use the Tcl Console to source the *hackathon.tcl* script from *vivado* subdirectory:

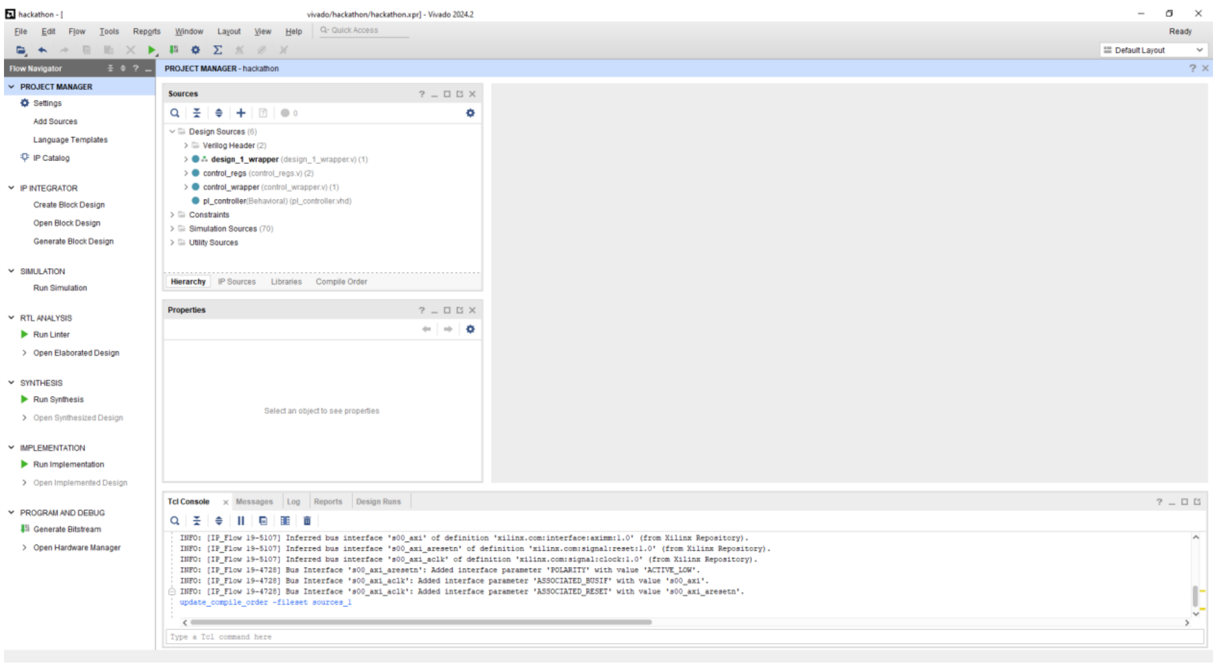
```
cd <path_to_project>/vivado/
source hackathon.tcl
```



```
pwd
[redacted]/vivado
exec dir
bd  hackathon.tcl
source hackathon.tcl
```

Alternatively, if you're already in the *vivado* directory, you can start Vivado and load the project directly.

After sourcing the Tcl script, the software will open the project (it may take a while - be patient). Please be aware that when working on the task solutions with your team members, you will most likely need to cooperate heavily with Git (source control management). Sometimes it may lead to difficulties when opening the Vivado project after pull/merge operations. If such a conflict occurs, try deleting the contents of the *vivado/hackathon* folder, relaunch Vivado, and source the project again using the Tcl file.

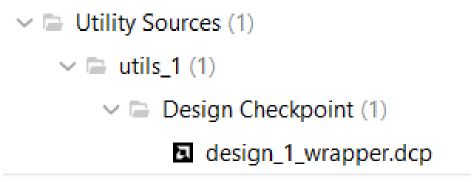


```
INFO: [IP_Flow 19-5107] Inferred bus interface 's00_axi' of definition 'xilinx.com:interface:aximm:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 's00_axi_aresetn' of definition 'xilinx.com:signal:reset:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-5107] Inferred bus interface 's00_axi_aclk' of definition 'xilinx.com:signal:clock:1.0' (from Xilinx Repository).
INFO: [IP_Flow 19-4728] Bus Interface 's00_axi_aresetn': Added interface parameter 'POLARITY' with value 'ACTIVE_LOW'.
INFO: [IP_Flow 19-4728] Bus Interface 's00_axi_aclk': Added interface parameter 'ASSOCIATED_BUSIF' with value 's00_axi'.
INFO: [IP_Flow 19-4728] Bus Interface 's00_axi_aclk': Added interface parameter 'ASSOCIATED_RESET' with value 's00_axi_aresetn'.
update_compile_order -fileset sources_1
```

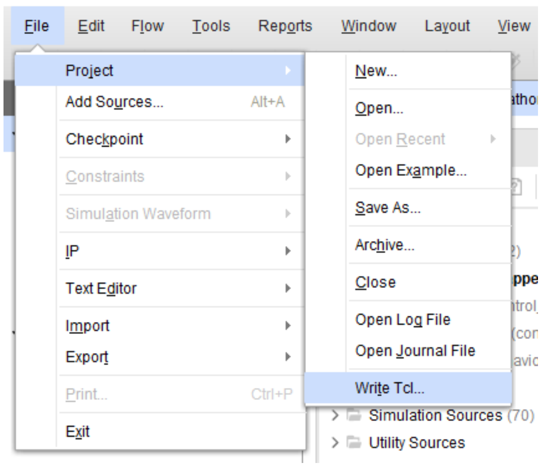
On the left-hand side of the main window, you will find the Flow Navigator, which helps you navigate various Vivado features grouped by FPGA development stages. The central panel shows your project sources (note that simulation sources like testbenches are categorized in a different section than design sources). At the bottom, the Tcl Console is opened by default, but you can switch tabs to view messages, logs, reports, or design runs, which may help when debugging errors.

1.2. How to upload the solutions?

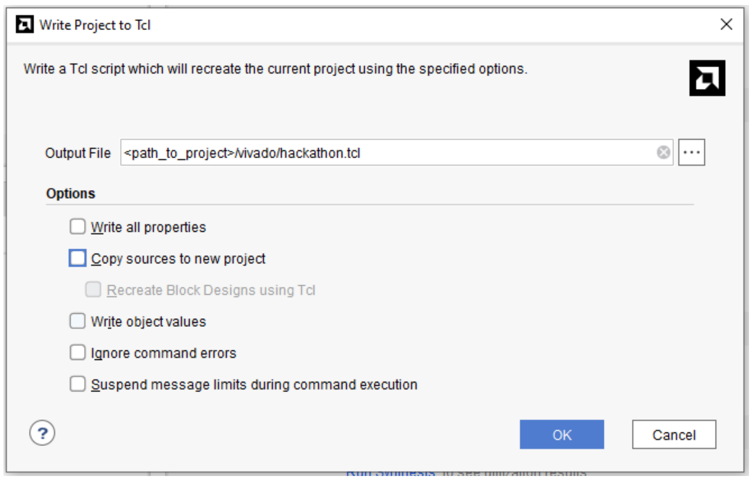
First, you need to create an updated Tcl file. Before doing this, **remove** all *.dcp files generated by Vivado:



To generate the Tcl file with your current project state, open Vivado and go to *File* -> *Project* -> *Write Tcl…*



In the new window, as the output file provide the same *hackathon.tcl* file that you used to generate the project initially (double-check the saving directory!). When asked, agree to overwrite the file (**do not change the file name**). Make sure to uncheck the options *Write all properties*, *Copy sources to new project*, and *Recreate Block Designs using Tcl* (as shown below). Select OK and confirm overwriting the file again.

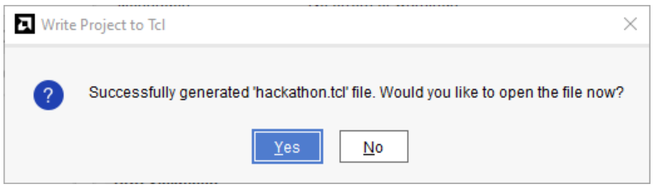

Write Project to Tcl

Write a Tcl script which will recreate the current project using the specified options.

Output File <path_to_project>/vivado/hackathon.tcl

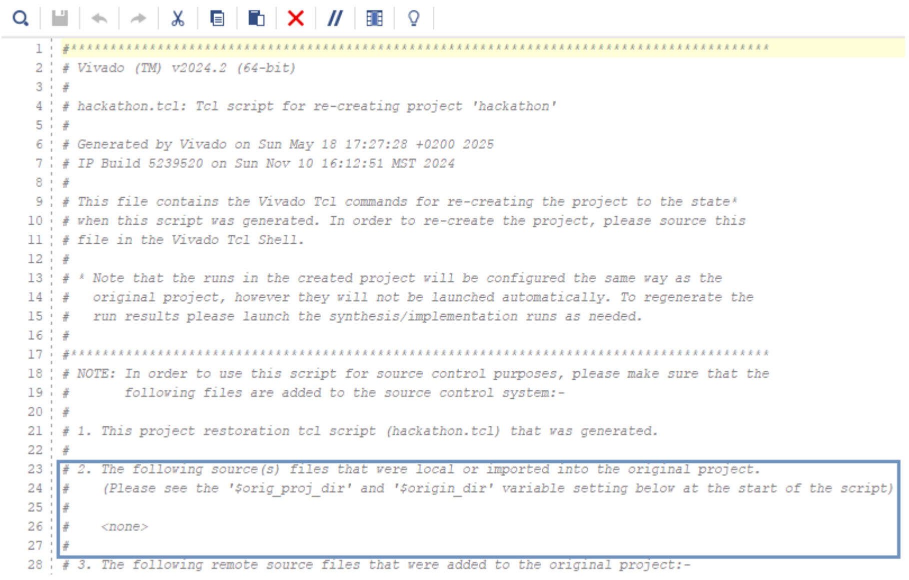
Options
- [ ] Write all properties
- [ ] Copy sources to new project
- [ ] Recreate Block Designs using Tcl
- [ ] Write object values
- [ ] Ignore command errors
- [ ] Suspend message limits during command execution

Once the generation is complete, open the generated file.


Write Project to Tcl

Successfully generated 'hackathon.tcl' file. Would you like to open the file now?

In the script, check section 2 and ensure that the value is *<none>*. If any file is listed there, it means that a local file is being referenced, and the judge will not be able to access it. In such a case, please update the project so that all required files are properly included. See section 3.1 of this instruction for details on adding files to the project.


```
#*******************************************************************************
# Vivado (TM) v2024.2 (64-bit)
#
# hackathon.tcl: Tcl script for re-creating project 'hackathon'
#
# Generated by Vivado on Sun May 18 17:27:28 +0200 2025
# IP Build 5239520 on Sun Nov 10 16:12:51 MST 2024
#
# This file contains the Vivado Tcl commands for re-creating the project to the state*
# when this script was generated. In order to re-create the project, please source this
# file in the Vivado Tcl Shell.
#
# * Note that the runs in the created project will be configured the same way as the
#   original project, however they will not be launched automatically. To regenerate the
#   run results please launch the synthesis/implementation runs as needed.
#
#*******************************************************************************
# NOTE: In order to use this script for source control purposes, please make sure that the
#       following files are added to the source control system:-
#
# 1. This project restoration tcl script (hackathon.tcl) that was generated.
#
# 2. The following source(s) files that were local or imported into the original project.
#    (Please see the '$orig_proj_dir' and '$origin_dir' variable setting below at the start of the script)
#
#    <none>
#
# 3. The following remote source files that were added to the original project:-
```

Add the modified *hackathon.tcl* file, along with all newly created and modified files to a new commit on the main branch. **Do not modify the content of *.gitignore* file.**

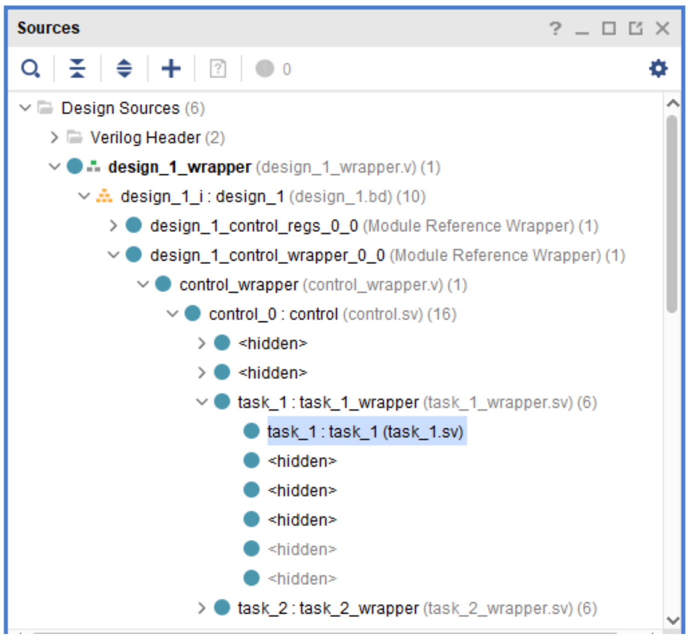
Push your commit to the main branch. If you need any help with source control, you can find a short Git guide in **FPGA_Hackathon_2025_User_Manual.pdf.**

1.3. Project structure

In the repository's main directory, there are three subfolders (src, tb, vivado):

- **src –** this folder contains the task manager, task configurations, and several control modules. You don't need to modify any of these files except for **task_enabler_pkg.sv** (see section 1.4 for more details).
- **src/task_wrappers –** this folder contains wrapper files responsible for feeding input data to each task and forwarding the results back to the task manager. **Do not modify any files** in this directory.
- **src/tasks/task_x –** each task has its own folder named task_x. In each of these folders, there is one file reserved for your task solution. This file contains the task module definition with its interface and is where you should implement your solution (check section 2.1). You are also allowed to add your own supporting files and instantiate them in the task file, if needed.
- **tb –** this directory contains the testbench. You are encouraged to use it to verify your solution locally before submitting it to the CI pipeline. You may also modify the testbench if needed while debugging. The testbench uses input and output test vector files (*.mem) as reference data, unless the task description specifies otherwise. Section 4.1 explains how to use the provided testbench.
- **vivado –** this folder contains the Tcl script used to generate the project, along with configuration files. Vivado will also generate additional project files in this directory automatically.

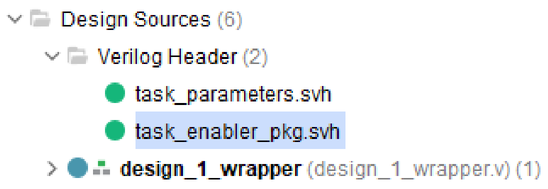
Since the project includes encrypted modules, some files may appear as <hidden> in the hierarchy. These correspond to encrypted content, are not meant to be accessed or modified, and can be safely ignored. Below you can find the file hierarchy to help you locate the task file more easily:



```
Sources
Design Sources (6)
  Verilog Header (2)
  design_1_wrapper (design_1_wrapper.v) (1)
    design_1_i : design_1 (design_1.bd) (10)
      design_1_control_regs_0_0 (Module Reference Wrapper) (1)
      design_1_control_wrapper_0_0 (Module Reference Wrapper) (1)
        control_wrapper (control_wrapper.v) (1)
          control_0 : control (control.sv) (16)
            <hidden>
            <hidden>
            task_1 : task_1_wrapper (task_1_wrapper.sv) (6)
              task_1 : task_1 (task_1.sv)
              <hidden>
              <hidden>
              <hidden>
              <hidden>
              <hidden>
            task_2 : task_2_wrapper (task_2_wrapper.sv) (6)
```

1.4.Project configuration

After finishing the implementation of a task, you need to explicitly enable it. Otherwise, the task manager will not report the task as ready for testing - no input data will be sent for that task, and no verification will be performed. To mark a task as completed, update the corresponding parameter in the *src/task_enabler_pkg.svh* file located in the src directory:



```
localparam task_enabler enable = '{
                                   task1  : 1'b1,
                                   task2  : 1'b0,
                                   task3  : 1'b0,
                                    . . .
                                    . . .
};
```

Set the value to 1’b1 for the task(s) you have completed and want to be tested. Leave the others as 1’b0. Multiple tasks can run simultaneously, and this will be required to maximize your score (see details below).

2. Tasks overview and evaluation

2.1.Tasks table

The order of tasks listed in the project does not reflect their difficulty and has no special meaning. You are free to solve the tasks in any order - start with whichever one you prefer:

Task number	Task name	Task file	Evaluation mode scoring*	Maximum amount of bonus points	Theoretical total points**
1	Maximum Finder	task_1.sv	3	3	6
2	Traverse to the far side of the Moon	task_2.sv	9	3	12
3	Sequence Counter	task_3.sv	9	3	12
4	Sorting Algorithm	task_4.sv	12	3+3	18
5	State Machine	task_5.sv	8	3	11
6	Maze	task_6.sv	21	3+3	27
7	Optimal Dot Product	task_7.sv	3	15	18
8	Life on the far side of the Moon	task_8.sv	15	3+3	21
9	Rover Navigation	task_9.sv	24	3+3	30
10	Quadratic Interpolator	task_10.sv	21	3	24
11	Hamming Correction	task_11.sv	15	3	18
12	MoonCompress	task_12.sv	21	3	24
13	Power Grid NDT	task_13.vhd	10	18	28
14	LIR	task_14.sv	18	3+3	24

*The maximum number of points for each task does not include any bonus points that may be awarded for resource utilization, timing performance, or latency. Detailed information on these criteria can be found in section 2.3.2 and in the individual task descriptions.

** This is a theoretical value due to latency, timing performance and resource utilization optimization trade-offs that are usually required.

You are allowed to edit only the main task files listed in the table (do not change their filenames or input/output interfaces). However, you may add new submodules and additional files as needed, making sure to place them in the appropriate folders within the *src/tasks/task_x* directory for each task. You can use any language you prefer - SystemVerilog or VHDL.

2.2. Judge overview

Judge is a software application running outside of the board, communicating with the board and verifying task solutions. Judge works in the following flow:

- Connection establishment – the Judge sends a ping message to the board and waits for the pong response, which includes the list of implemented tasks. This step is used to verify which tasks are ready for evaluation. The Judge attempts this step up to three times - if all attempts fail, the evaluation process is aborted. If you notice such scenario in the logs (no pong response), inform the Hackathon's technical team about it – perhaps some extraterrestrials took over your FPGA board!
- Task assessment - based on the list of ready tasks received in the pong message, the Judge proceeds to verify them one by one. The assessment process follows this sequence:
    - Judge sends input data to the task
    - It waits for the FPGA's response for up to 5s (extended to 60s for task 13)
    - It compares the FPGA output with the reference model
    - The score for the task is assigned and stored

  There are up to three attempts to obtain the solution for the task. If no result is obtained in that time, the task is scored with **0 points**.
- Report - after verifying all available tasks, the Judge sends the results to the database, updating the score table for the team.

2.3. Evaluation

2.3.1. Testing modes

Each time you trigger a job on Jenkins, your solution will first be tested in **Development mode** for debugging and verification purposes, and immediately afterwards in **Evaluation mode** for final scoring:

- **Development mode** is intended to help you verify and debug your solution. It performs a single set of data per task. The input data in this mode can be randomly generated. In this mode, you have access to logs. This allows the team to verify that their solution is functionally correct. **<u>Results from this mode do not affect your final score.</u>**

- **Evaluation mode** is used to obtain your final score. In this mode, the judge runs several test vectors per task (up to 3), with fixed input data the same for every team. You do not have access to logs in this mode. **Only the latest run will be considered for scoring**.

Each run will test all implemented tasks, and the score will depend on how many of them function correctly, so all tasks must be capable of running simultaneously. More information about the number of test vectors per task and their scoring can be found in the task-specific instructions.

2.3.2. Bonus Points

As part of the evaluation, selected tasks include additional scoring criteria: resource utilization, timing performance (slack), and latency (**applies only in Evaluation mode**). The table below shows which tasks include these extra evaluation aspects - marked with an X in the respective columns:

Task number	Task name	Resource utilization	Timing performance	Latency
1	Maximum Finder	X		
2	Traverse to the far side of the Moon	X		
3	Sequence Counter	X		
4	Sorting Algorithm	X		X
5	State Machine	X		
6	Maze	X		X
7	Optimal Dot Product*	X		
8	Life on the far side of the Moon	X	X	
9	Rover Navigation	X		X
10	Quadratic Interpolator			X
11	Hamming Correction	X		
12	MoonCompress	X		
13	Power Grid NDT*			X
14	LIR	X		X

Additional points are awarded only if the task is solved correctly (i.e. the maximum number of functional points is obtained):

- The best team in a given category receives 3 additional points
- The second-best team receives 2 additional points
- The third-best team receives 1 additional point

Note: Tasks marked with * are exceptions - please refer to the task-specific instructions for more information.

2.3.2.1. Additional points for resource utilization

Unless otherwise specified in the task description, resource utilization is evaluated using the following formula:

resource_utilization = CLB_LUTs + CLB_REGs + 300*DSP

The lower the resource_utilization value, the better.

2.3.2.2. Additional points for timing performance

The higher the slack value, the better the score.

2.3.2.3. Additional points for latency

Latency is measured as the number of clock cycles between the arrival of the first input sample and the appearance of the first output sample in the task module. The lower the latency value, the better.

In Evaluation Mode, where more than one test vector may be used for a single task, the maximum latency value across all test vectors is taken as the final latency result for that team. These final values (i.e. worst-case latencies) are then compared across all teams - the teams with the lowest maximum latencies receive additional points.

2.4. How are the results displayed?

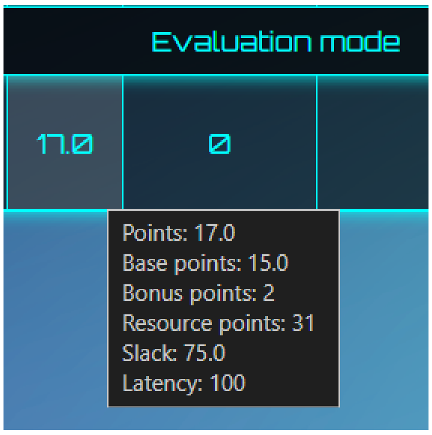
During the hackathon, you will be able to check your team results on the website. At the start of the event, you will obtain a unique link to your team's dashboard.

You will not have access to other teams' results, but general standings will be announced periodically on the Discord server.

On the main dashboard page, only your latest runs from Development mode and Evaluation mode will be displayed. There is also a link to a detailed table where all your previous runs (Development Mode only) and results are shown.

You can check detailed information about your results or task specifics by hovering over the respective items in the table. The details include:

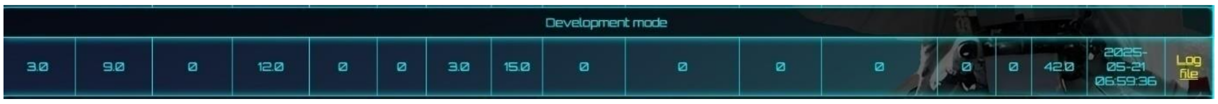
- **Points**: total points you earned for the task, divided into:
    - **Base points**: points awarded by the Judge for correct task functionality.
    - **Bonus points**: additional points (if applicable) awarded for resource utilization, timing performance, and latency (see section 2.3.2 for details):
        - **Resource points**: points calculated based on resource utilization
        - **Slack**: slack value achieved
        - **Latency**: latency value measured



Points: 17.0
Base points: 15.0
Bonus points: 2
Resource points: 31
Slack: 75.0
Latency: 100

2.5.Logs access

On the scoreboard website, you will also be able to check logs from your runs in Development mode. There will be a link in the table under the “Logs” column that opens the logs directly in your browser, with an option to download them:



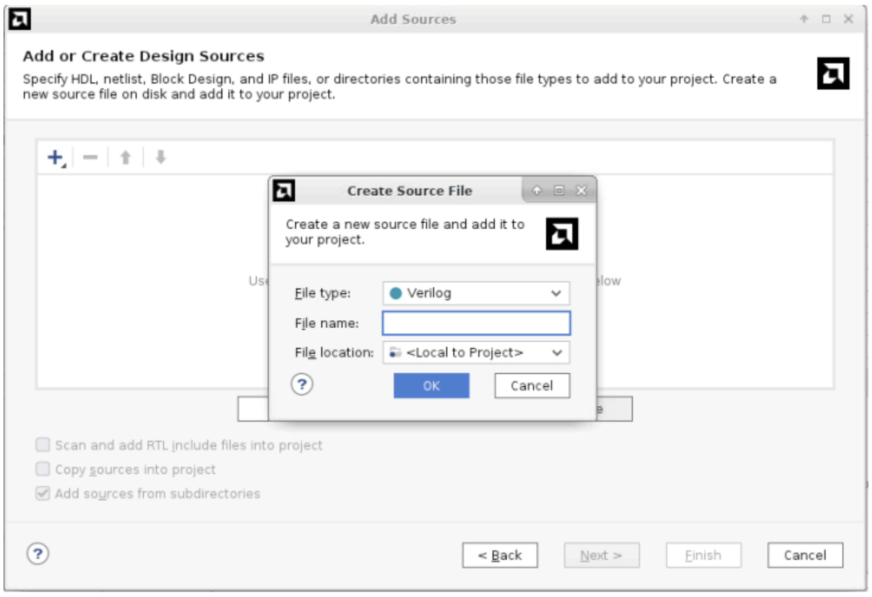
Development mode																
3.0	9.0	0	12.0	0	0	3.0	15.0	0	0	0	0	0	0	42.0	2025-05-21 06:59:36	Log file

If the link to the log file returns an error, please try again after one minute.

3. Vivado usage

3.1.How to create files and add them to the project?

If you decide to add some files to the project, please use the **ALT+A** shortcut or right- click on *Simulation source/Design* sources in *Sources* tab and choose *Add sources…* . When the window pops up, decide if you want to add a design (RTL) file or a simulation file (testbench). Then choose if you prefer to create a file from scratch (*Create file* option) or just add an existing one. Remember to keep those files in repository folders (design files in appropriate location in ***src/tasks/task_x*** subdirectory and tests in ***tb/tests*** subdirectory). **Do not check the box: *Copy sources into project* –** instead, make sure your files are stored in the locations mentioned above.



Add Sources

Add or Create Design Sources

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

Create Source File

Create a new source file and add it to your project.

File type: Verilog

File name:

File location: <Local to Project>

OK Cancel

☐ Scan and add RTL include files into project

☐ Copy sources into project

☑ Add sources from subdirectories

< Back Next > Finish Cancel

If you add an existing file and have trouble locating it, check the *File of type* field, which may filter files by their extensions. More information is available here: Creating and Adding Design Sources.

3.2.Synthesis and Implementation

We recommend launching a synthesis and implementation of the project before enabling our processing chain.

To launch a synthesis run, use one of the following methods:

- From the Flow Navigator, click Run Synthesis command.
- From the main menu, select Flow > Run Synthesis command.
- In the Design Runs window, right-click the run and select Launch Runs.

The first two options start the active synthesis run. The third option opens the Launch Selected Runs window.

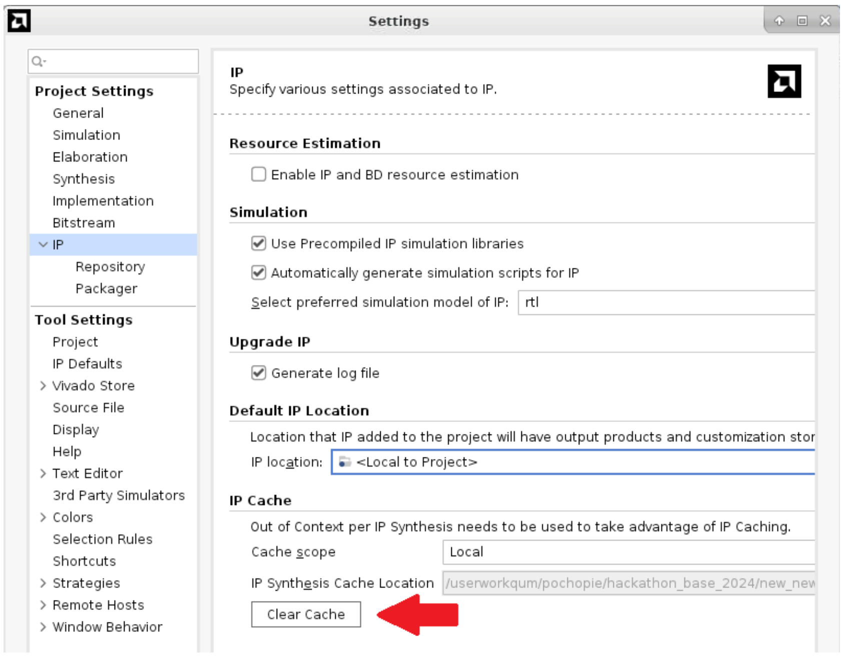
To launch an implementation run, do one of the following methods:

- Select Run Implementation in the Flow Navigator.
- Select Flow > Run Implementation from the main menu.
- Right-click a run in the Design Runs window and select Launch Runs.

To launch a run other than the active one, select the desired run in the Design Runs window before launching.

[TIP] If you use VM, we recommend using only 1 job for synthesis (more jobs may slow-down the machine).

[TIP] If changes you made to the design are not reflected in synthesis or implementation, clear the cache: Project Settings -> IP -> "Clear Cache" button at the bottom



IP
Specify various settings associated to IP.

Resource Estimation
☐ Enable IP and BD resource estimation

Simulation
☑ Use Precompiled IP simulation libraries
☑ Automatically generate simulation scripts for IP
Select preferred simulation model of IP: rtl

Upgrade IP
☑ Generate log file

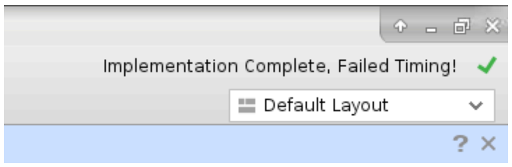
Default IP Location
Location that IP added to the project will have output products and customization stor
IP location: <Local to Project>

IP Cache
Out of Context per IP Synthesis needs to be used to take advantage of IP Caching.
Cache scope: Local
IP Synthesis Cache Location: /userworkqum/pochopie/hackathon_base_2024/new_new

Clear Cache

3.2.1. Failed timing [IMPORTANT]

There is a chance that synthesis and implementation in Vivado will be completed successfully but timing issues remain. We strongly encourage you to carefully review the synthesis and implementation logs, paying special attention to critical warnings. After synthesis and implementation finish, check the upper right corner:


Implementation Complete, Failed Timing!

If you see a warning similar to the one shown above or messages like this:

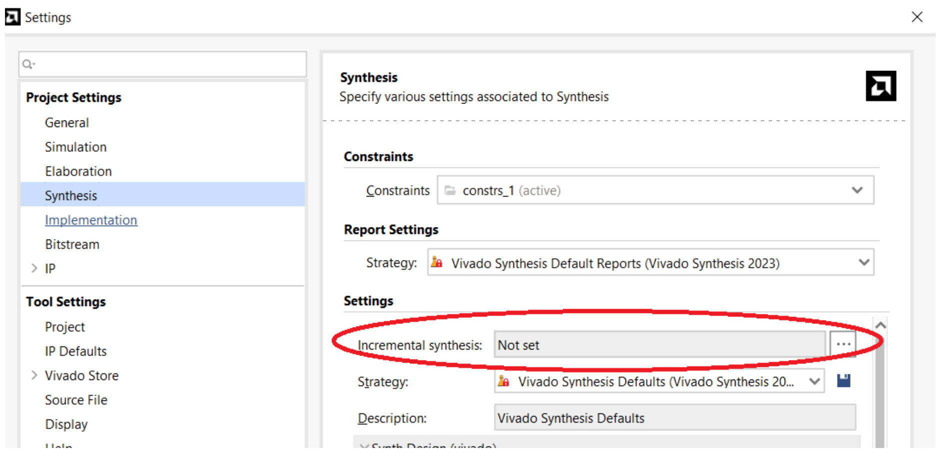
```
INFO: [Timing 38-91] UpdateTimingParams: Speed grade: -2LV, Temperature grade: C, Delay Type: min_max.
INFO: [Timing 38-191] Multithreading enabled for timing update using a maximum of 8 CPUs
CRITICAL WARNING: [Timing 38-282] The design failed to meet the timing requirements. Please see the timing summary report for details on the timing violations.
INFO: [runtcl-4] Executing : report_incremental_reuse -file design_1_wrapper_incremental_reuse_routed.rpt
INFO: [Vivado Tcl 4-1062] Incremental flow is disabled. No incremental reuse Info to report.
```

It means your design has timing violations and you need to improve your code to resolve them.

3.2.2. Project settings [IMPORTANT]

Vivado allows customizing project settings for each processing step: simulation, elaboration, synthesis, implementation, and bitstream generation. Due to the complexity and multiple processing stages in our Hackathon evaluation system, **please do not modify the “Incremental synthesis” and “Incremental implementation” settings**. Leave these settings as **Not set** (see screenshots below).

Changing these options causes additional files to be generated, which are **not supported** by our evaluation chain and will lead to failures when your solutions are tested.

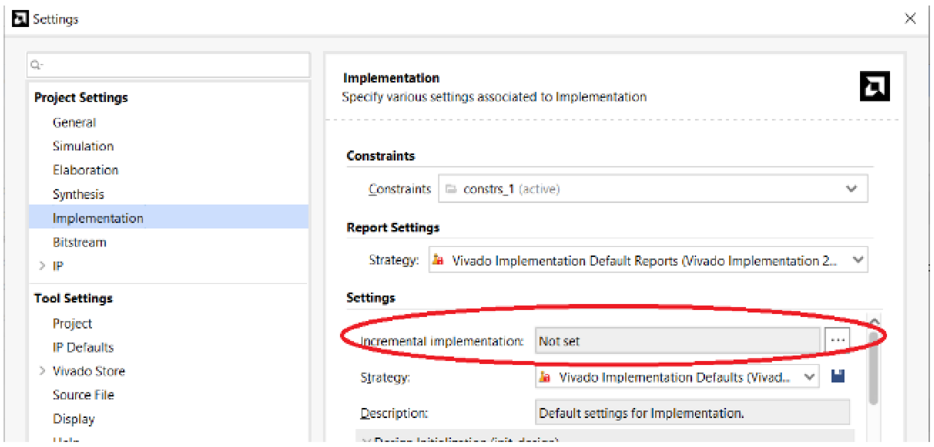

Settings

Synthesis
Specify various settings associated to Synthesis

Constraints
Constraints: constrs_1 (active)

Report Settings
Strategy: Vivado Synthesis Default Reports (Vivado Synthesis 2023)

Settings
Incremental synthesis: Not set
Strategy: Vivado Synthesis Defaults (Vivado Synthesis 20...
Description: Vivado Synthesis Defaults


Settings

Implementation
Specify various settings associated to Implementation

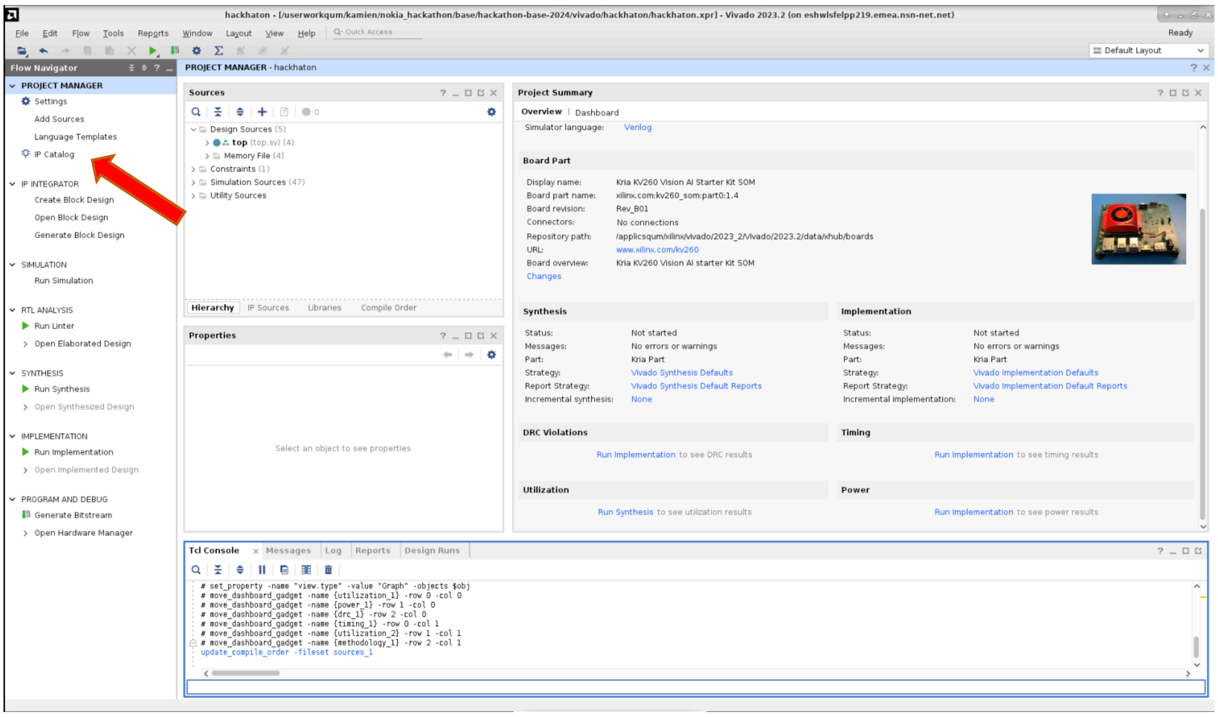
Constraints
Constraints: constrs_1 (active)

Report Settings
Strategy: Vivado Implementation Default Reports (Vivado Implementation 2...

Settings
Incremental implementation: Not set
Strategy: Vivado Implementation Defaults (Vivad...
Description: Default settings for Implementation.

3.3. Adding IP from the IP Catalog

If you want to add an IP from Vivado built-in IP Catalog, navigate to the *Project Manager* section in the Flow Navigator and click on *IP Catalog*. The IP Catalog tab will open.



Simulator language: Verilog

Board Part

Display name:	Kria KV260 Vision AI Starter Kit SOM
Board part name:	xilinx.com:kv260_som:part0:1.4
Board revision:	Rev_B01
Connectors:	No connections
Repository path:	/applicsqum/xilinx/vivado/2023_2/Vivado/2023.2/data/xhub/boards
URL:	www.xilinx.com/kv260
Board overview:	Kria KV260 Vision AI starter Kit SOM

Changes

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	Kria Part
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	Kria Part
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization

Run Synthesis to see utilization results

Power

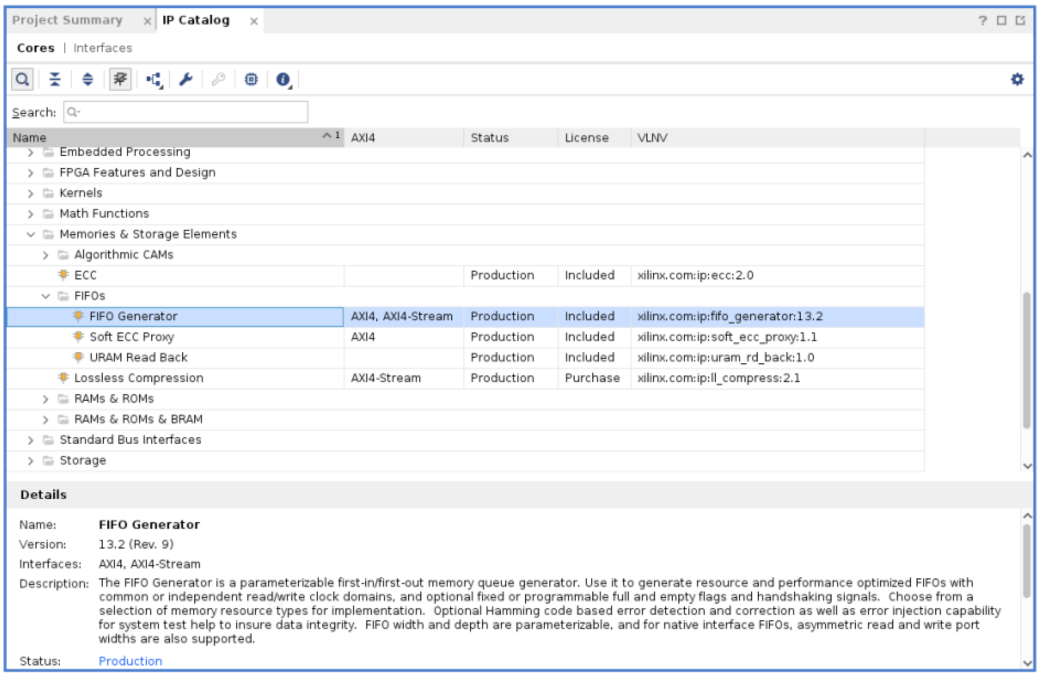
Run Implementation to see power results

Tcl Console

```
# set_property -name "view.type" -value "Graph" -objects $obj
# move_dashboard_gadget -name {utilization_1} -row 0 -col 0
# move_dashboard_gadget -name {power_1} -row 1 -col 0
# move_dashboard_gadget -name {drc_1} -row 2 -col 0
# move_dashboard_gadget -name {timing_1} -row 0 -col 1
# move_dashboard_gadget -name {utilization_2} -row 1 -col 1
# move_dashboard_gadget -name {methodology_1} -row 2 -col 1
update_compile_order -fileset sources_1
```

Choose the IP you want to instantiate (you can use the search field to find a specific IP), read the brief description shown at the bottom, then double-click the IP to customize its parameters according to your needs. After finishing the configuration, the IP will appear in your project sources and in the *IP Sources* tab alongside other IPs used in the project.

Save your IP in the *vivado/ip_repo* directory and remember to regenerate the Tcl file. Include all these files when committing your changes.

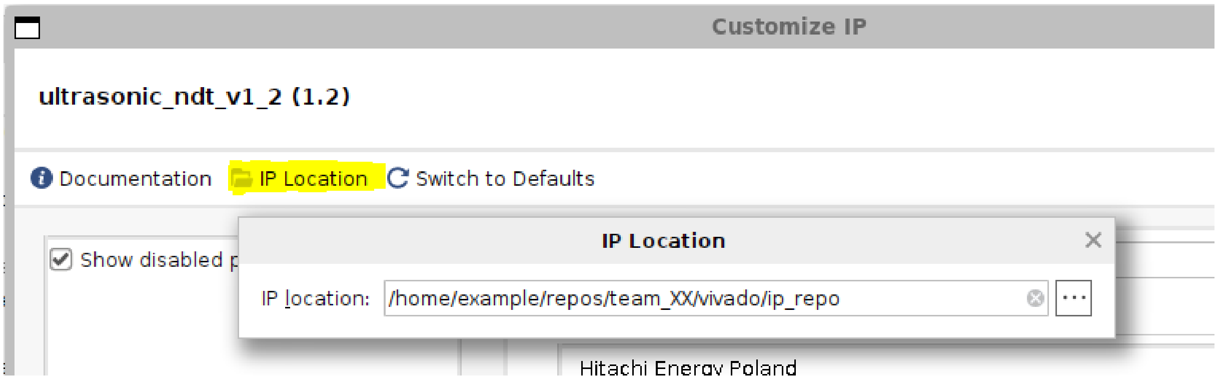


Name	AXI4	Status	License	VLNV
Embedded Processing				
FPGA Features and Design				
Kernels				
Math Functions				
Memories & Storage Elements				
Algorithmic CAMs				
ECC		Production	Included	xilinx.com:ip:ecc:2.0
FIFOs				
FIFO Generator	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:fifo_generator:13.2
Soft ECC Proxy	AXI4	Production	Included	xilinx.com:ip:soft_ecc_proxy:1.1
URAM Read Back		Production	Included	xilinx.com:ip:uram_rd_back:1.0
Lossless Compression	AXI4-Stream	Production	Purchase	xilinx.com:ip:ll_compress:2.1
RAMs & ROMs				
RAMs & ROMs & BRAM				
Standard Bus Interfaces				
Storage				

Details

Name:	**FIFO Generator**
Version:	13.2 (Rev. 9)
Interfaces:	AXI4, AXI4-Stream
Description:	The FIFO Generator is a parameterizable first-in/first-out memory queue generator. Use it to generate resource and performance optimized FIFOs with common or independent read/write clock domains, and optional fixed or programmable full and empty flags and handshaking signals. Choose from a selection of memory resource types for implementation. Optional Hamming code based error detection and correction as well as error injection capability for system test help to insure data integrity. FIFO width and depth are parameterizable, and for native interface FIFOs, asymmetric read and write port widths are also supported.
Status:	Production

Set the IP location to vivado/ip_repo:



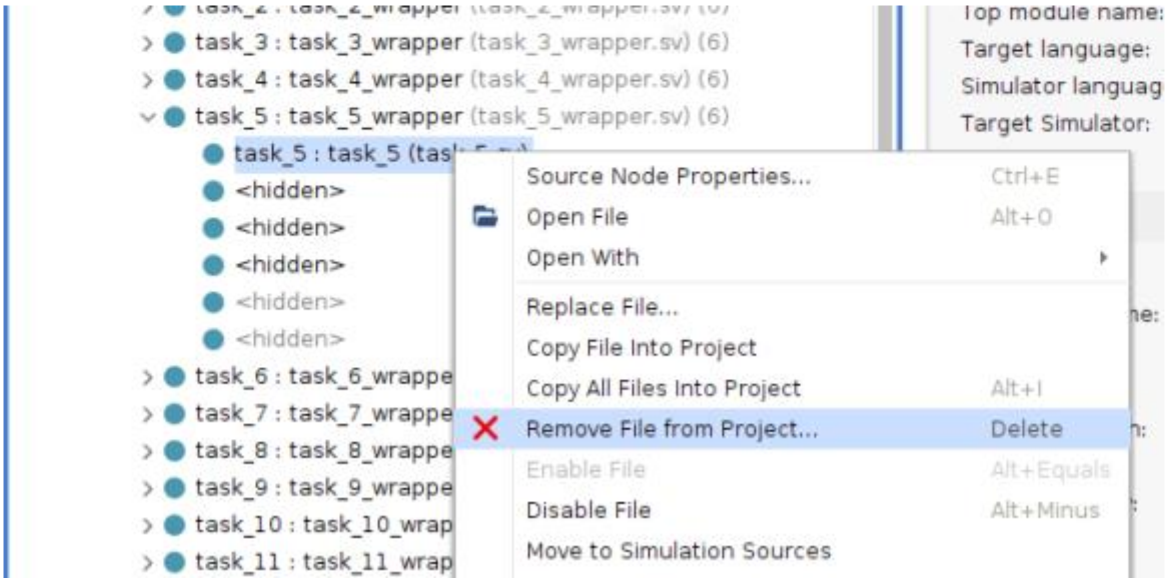
IP location: /home/example/repos/team_XX/vivado/ip_repo

For more details, please refer to AMD's official guide: [Adding IP from the IP Catalog.]()

3.4. Adding files generated by HDL Coder (MATLAB)

If you are using models that have been prepared for you (**do not modify any inputs or outputs**), follow these steps:

- Remove the existing **.sv* file from the project and delete it from the task folder - *src/task_**



- Copy the files generated by HDL Coder directly into the corresponding task folder - *src/task_**
- Add these new sources to the project (see chapter 3.1 for instructions)

For more details, refer to MathWorks documentation:

- [Basic HDL Code Generation Workflow]()
- [Generate HDL Code from Simulink Model]()
- [Basic HDL Code Generation and FPGA Synthesis from MATLAB - MATLAB & Simulink]()

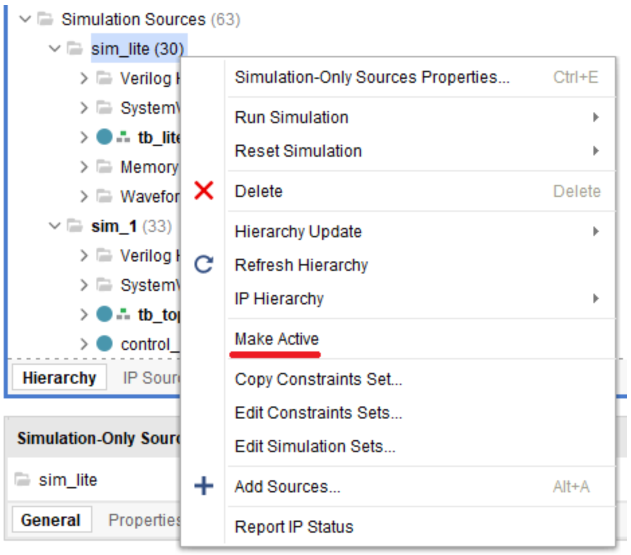
4. Debugging

4.1.Testbench

We have prepared a basic testbench file for you - *tb_top.sv*, located in the *tb* subdirectory. To test a specific task locally, you need to **uncomment the appropriate define** (TASK_N) inside the testbench (others tasks' defines should be commented-out – only one define can be active). When you launch a test, a predefined input dataset is sent to your task module, and the module's output will be automatically compared with the corresponding reference data.

```
// SET YOUR TASK HERE!
//`define TASK_1
`define TASK_2
//`define TASK_3
//`define TASK_4
//`define TASK_5
//`define TASK_6
//`define TASK_7
//`define TASK_8
```

Note: We have also prepared a **lite version** of the testbench – *tb_lite.sv*, which runs faster and may be more convenient. To use the lite version, you need to **explicitly activate** it:

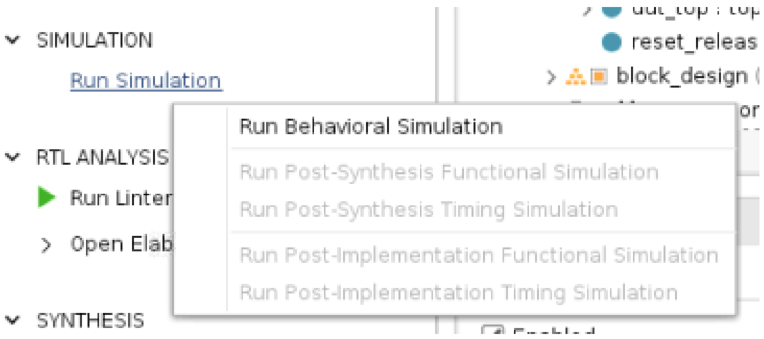


4.2. Simulators

There is a possibility to test the created RTL code using several simulators. It is entirely up to you which one you choose. The provided testbench (located in the *tb* subdirectory) is compatible with the most popular simulation tools. If you wish to get more detailed information about simulation workflows in Vivado, we recommend checking the following AMD resources: Preparing for Simulation and Logic Simulation. Basic simulation instructions and guidance on testbench usage for each supported simulator are provided in the following subchapters.

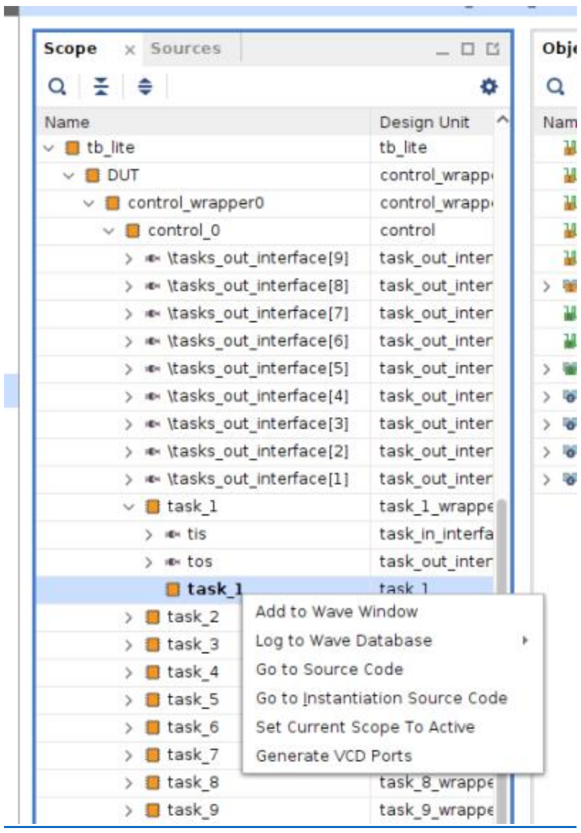
4.2.1. Vivado Simulator

To run a simulation, first verify that you followed the instructions from chapter 2.1 (enabling the correct test). Then, in the Flow Navigator, choose *Run Simulation* -> *Run Behavioral Simulation* (if you have already synthesized or implemented the design, you may also choose to run post-synthesis or post-implementation simulation).

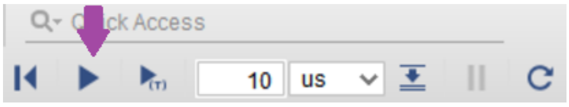

SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elab
SYNTHESIS
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation
reset_releas
block_design

Once the design is compiled, a waveform window should appear. We have prepared a pre-configured waveform file for you, with relevant task signals already set up. To load it (if not loaded automatically), go to: *File* -> *Simulation waveform* -> *Open configuration* and select the file *tb_behav.wcfg* (*tb_lite.wcfg* for lite version) located in the *tb* subfolder.

If you want to add more signals to the waveform, navigate to the *Scope* tab and locate the desired module. Then right-click on the module and choose *Add to Wave Window*. The selected signals will be added to the waveform view.

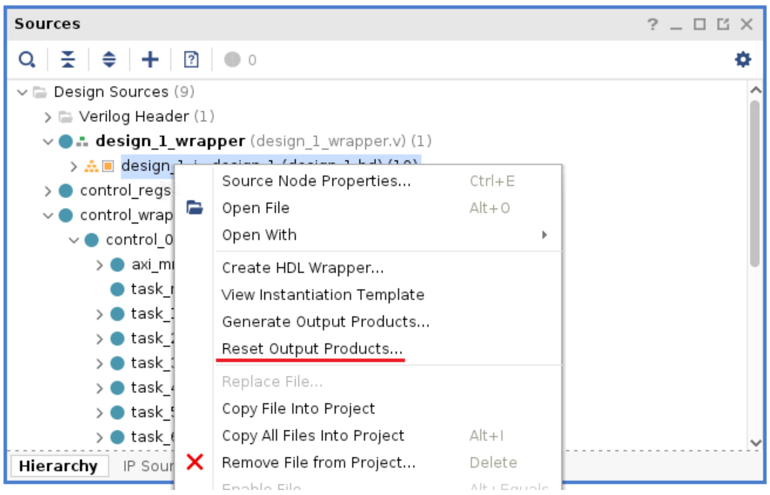


To run the simulation, click the *Run All* button:



[TIP] If the changes you made in the design are not reflected in the simulation (even after relaunching), try the following steps:

- Close the simulation
- Reset output products

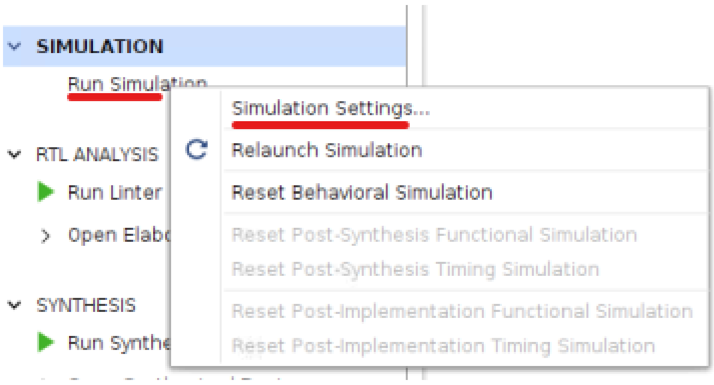


- Run the simulation again

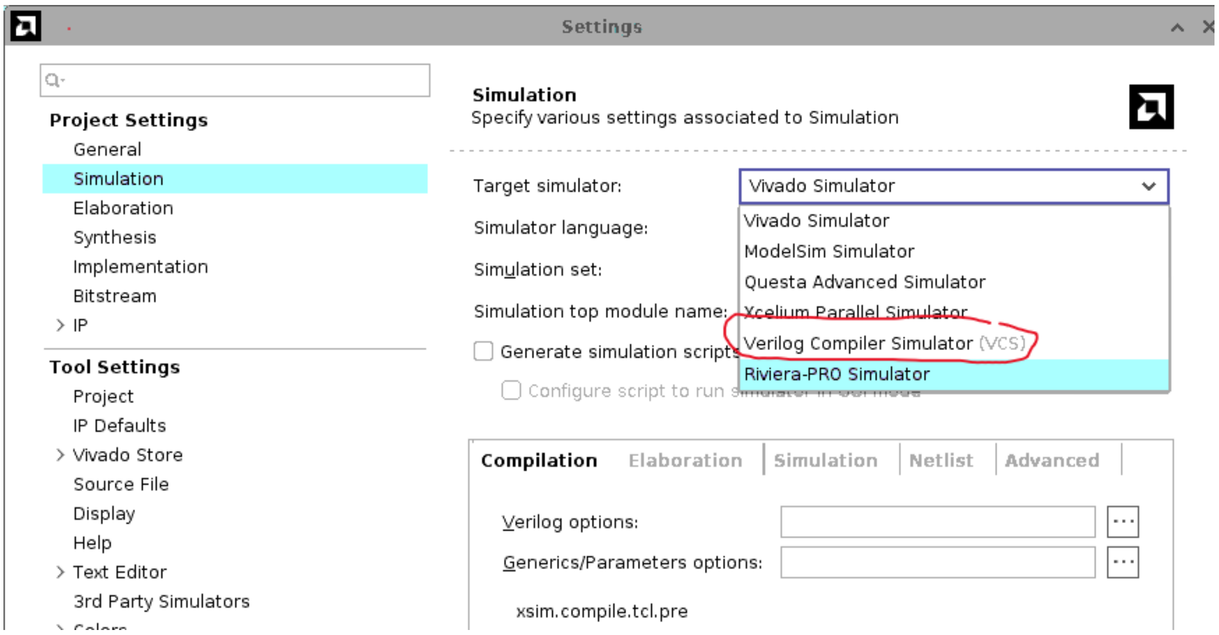
For more information, refer to: [Running the Vivado Simulator]() or [Analyzing Simulation Waveforms with Vivado Simulator.]()

4.2.2. VCS/Verdi

To run VCS simulator from Vivado gui first right click the SIMULATION/*Run Simulation* tab and go to *Simulation Settings…*


SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elab
SYNTHESIS
Run Synthe
Simulation Settings...
Relaunch Simulation
Reset Behavioral Simulation
Reset Post-Synthesis Functional Simulation
Reset Post-Synthesis Timing Simulation
Reset Post-Implementation Functional Simulation
Reset Post-Implementation Timing Simulation

In the Simulation Settings menu select *Target simulator:* to **Verilog Compiler Simulator (VCS)**, and then click *Yes*



Settings

Project Settings: General, Simulation, Elaboration, Synthesis, Implementation, Bitstream, IP

Tool Settings: Project, IP Defaults, Vivado Store, Source File, Display, Help, Text Editor, 3rd Party Simulators

Simulation
Specify various settings associated to Simulation

Target simulator: Vivado Simulator

Vivado Simulator
ModelSim Simulator
Questa Advanced Simulator
Xcelium Parallel Simulator
Verilog Compiler Simulator (VCS)
Riviera-PRO Simulator

Simulator language:
Simulation set:
Simulation top module name:
Generate simulation scripts
Configure script to run simulator in GUI mode

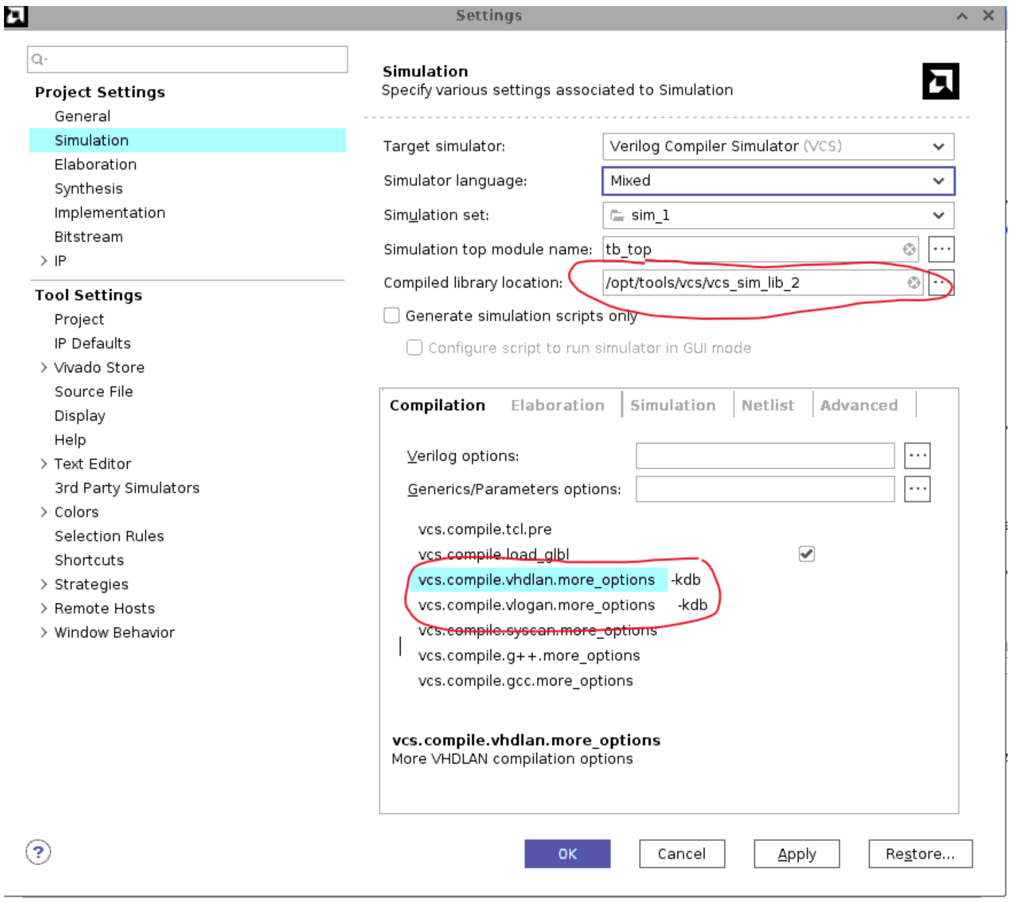
Compilation | Elaboration | Simulation | Netlist | Advanced

Verilog options:
Generics/Parameters options:

xsim.compile.tcl.pre

Next, remember to change Compiled library location to precompiled library:

/opt/tools/vcs/vcs_sim_lib_2, and add **-kdb** option to compilation tools (vhdlan and vlogan).



Settings

Project Settings: General, Simulation, Elaboration, Synthesis, Implementation, Bitstream, IP

Tool Settings: Project, IP Defaults, Vivado Store, Source File, Display, Help, Text Editor, 3rd Party Simulators, Colors, Selection Rules, Shortcuts, Strategies, Remote Hosts, Window Behavior

Simulation
Specify various settings associated to Simulation

Target simulator:	Verilog Compiler Simulator (VCS)
Simulator language:	Mixed
Simulation set:	sim_1
Simulation top module name:	tb_top
Compiled library location:	/opt/tools/vcs/vcs_sim_lib_2

Generate simulation scripts only
Configure script to run simulator in GUI mode

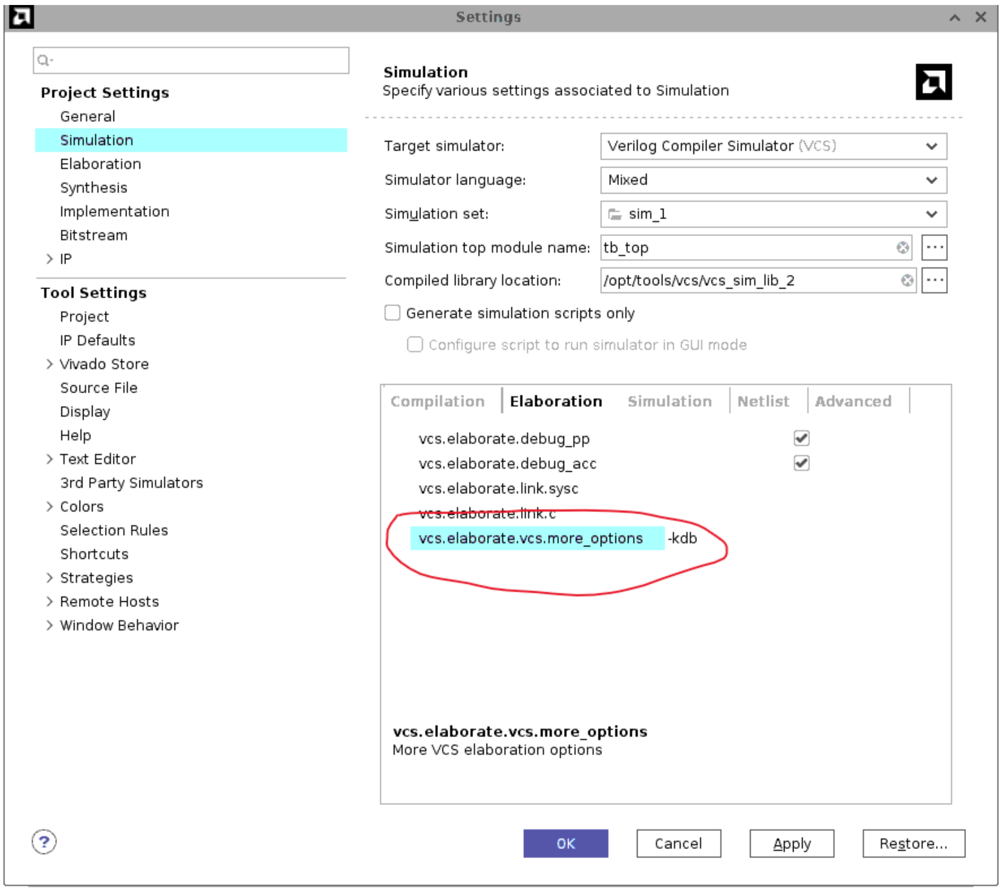
Compilation | Elaboration | Simulation | Netlist | Advanced

Verilog options:
Generics/Parameters options:

vcs.compile.tcl.pre	
vcs.compile.load_glbl	✓
vcs.compile.vhdlan.more_options	-kdb
vcs.compile.vlogan.more_options	-kdb
vcs.compile.syscan.more_options	
vcs.compile.g++.more_options	
vcs.compile.gcc.more_options	

vcs.compile.vhdlan.more_options
More VHDLAN compilation options

Next, switch to elaborate tab and add **-kdb** option to elaboration and click OK button.



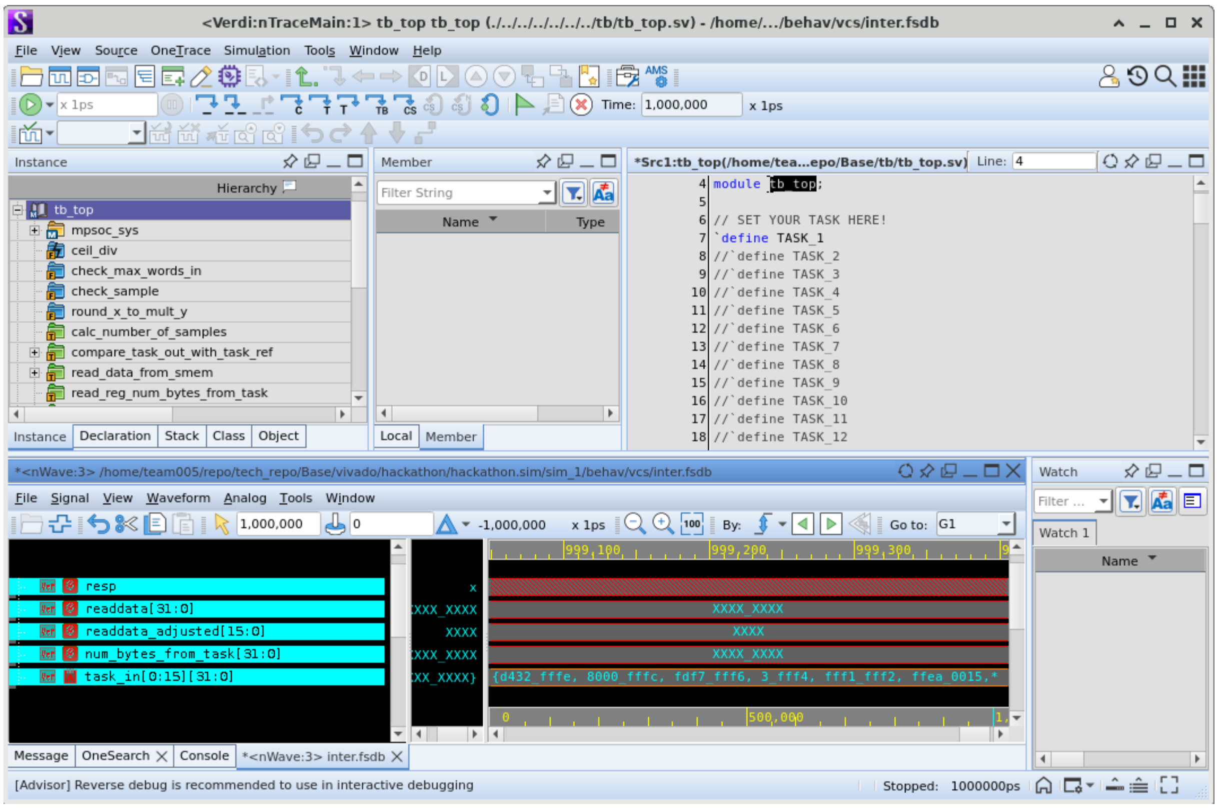
Simulation
Specify various settings associated to Simulation

Target simulator: Verilog Compiler Simulator (VCS)
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: tb_top
Compiled library location: /opt/tools/vcs/vcs_sim_lib_2
☐ Generate simulation scripts only
☐ Configure script to run simulator in GUI mode

vcs.elaborate.debug_pp	✓
vcs.elaborate.debug_acc	✓
vcs.elaborate.link.sysc	
vcs.elaborate.link.c	
vcs.elaborate.vcs.more_options	-kdb

vcs.elaborate.vcs.more_options
More VCS elaboration options

After left-clicking the *Run Simulation* button again, your design will compile and elaborate for VCS and will run the Verdi gui:



<Verdi:nTraceMain:1> tb_top tb_top (./../../../../../../tb/tb_top.sv) - /home/.../behav/vcs/inter.fsdb

Time: 1,000,000 x 1ps

tb_top
mpsoc_sys
ceil_div
check_max_words_in
check_sample
round_x_to_mult_y
calc_number_of_samples
compare_task_out_with_task_ref
read_data_from_smem
read_reg_num_bytes_from_task

*Src1:tb_top(/home/tea...epo/Base/tb/tb_top.sv) Line: 4

```
module tb_top;

// SET YOUR TASK HERE!
`define TASK_1
//`define TASK_2
//`define TASK_3
//`define TASK_4
//`define TASK_5
//`define TASK_6
//`define TASK_7
//`define TASK_8
//`define TASK_9
//`define TASK_10
//`define TASK_11
//`define TASK_12
```

*<nWave:3> /home/team005/repo/tech_repo/Base/vivado/hackathon/hackathon.sim/sim_1/behav/vcs/inter.fsdb

```
resp                          x
readdata[31:0]                XXXX_XXXX   XXXX_XXXX
readdata_adjusted[15:0]       XXXX        XXXX
num_bytes_from_task[31:0]     XXXX_XXXX   XXXX_XXXX
task_in[0:15][31:0]           XX_XXXX}    {d432_fffe, 8000_fffc, fdf7_fff6, 3_fff4, fff1_fff2, ffea_0015,*
```

1,000,000 0 -1,000,000 x 1ps By: Go to: G1
999,100 999,200 999,300
0 500,000

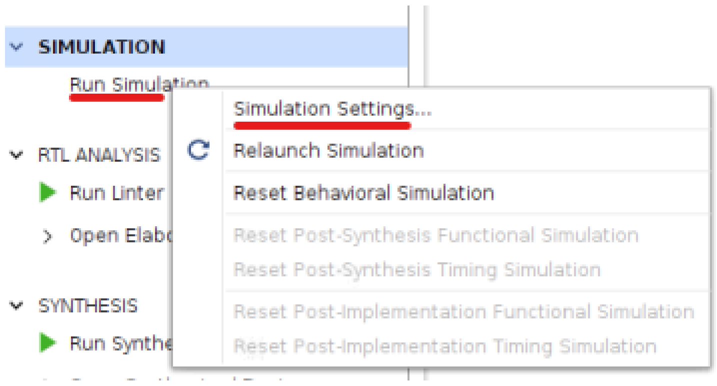
*<nWave:3> inter.fsdb

[Advisor] Reverse debug is recommended to use in interactive debugging   Stopped: 1000000ps

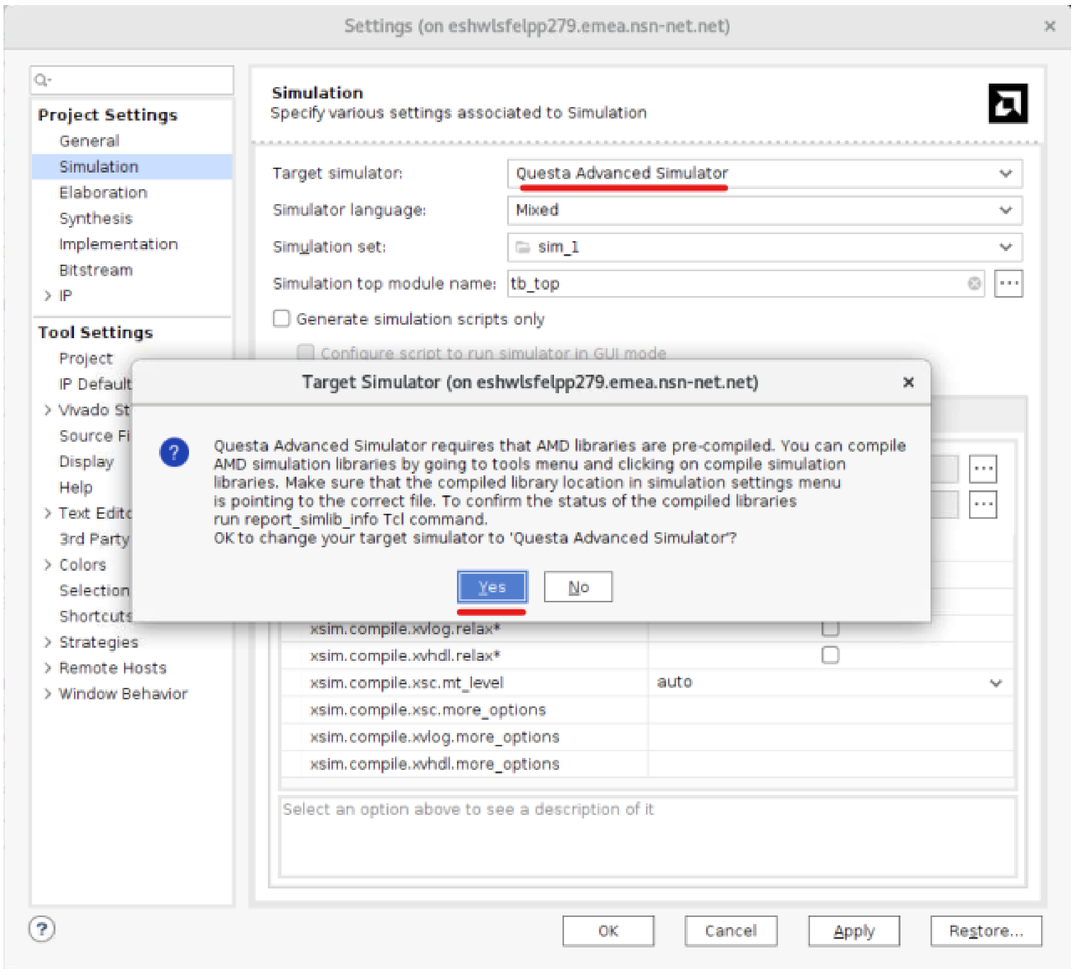
4.2.3. QuestaSim or Visualizer

There are 2 ways of running QuestaSim simulations, one is from Vivado Gui and the other is by using Makefile present in the repository.

To run QuestaSim simulator from Vivado gui first right click the SIMULATION/*Run Simulation* tab and go to *Simulation Settings…*


SIMULATION
Run Simulation
Simulation Settings...
Relaunch Simulation
Reset Behavioral Simulation
Reset Post-Synthesis Functional Simulation
Reset Post-Synthesis Timing Simulation
Reset Post-Implementation Functional Simulation
Reset Post-Implementation Timing Simulation
RTL ANALYSIS
Run Linter
Open Elab
SYNTHESIS
Run Synthe

In the Simulation Settings menu select *Target simulator:* to Questa Advanced Simulator, and then click *Yes*



Simulation
Specify various settings associated to Simulation

Target simulator: Questa Advanced Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: tb_top
Generate simulation scripts only
Configure script to run simulator in GUI mode

xsim.compile.xvlog.relax*
xsim.compile.xvhdl.relax*
xsim.compile.xsc.mt_level auto
xsim.compile.xsc.more_options
xsim.compile.xvlog.more_options
xsim.compile.xvhdl.more_options

Select an option above to see a description of it

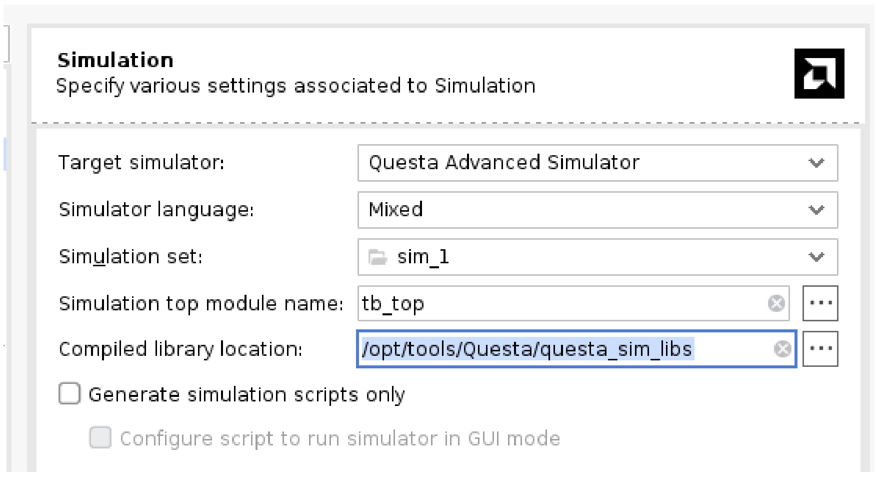
Target Simulator (on eshwlsfelpp279.emea.nsn-net.net)

Questa Advanced Simulator requires that AMD libraries are pre-compiled. You can compile AMD simulation libraries by going to tools menu and clicking on compile simulation libraries. Make sure that the compiled library location in simulation settings menu is pointing to the correct file. To confirm the status of the compiled libraries run report_simlib_info Tcl command.
OK to change your target simulator to 'Questa Advanced Simulator'?

Yes No

Lastly, remember to change Compiled library location to precompiled library:

/opt/tools/Questa/questa_sim_libs



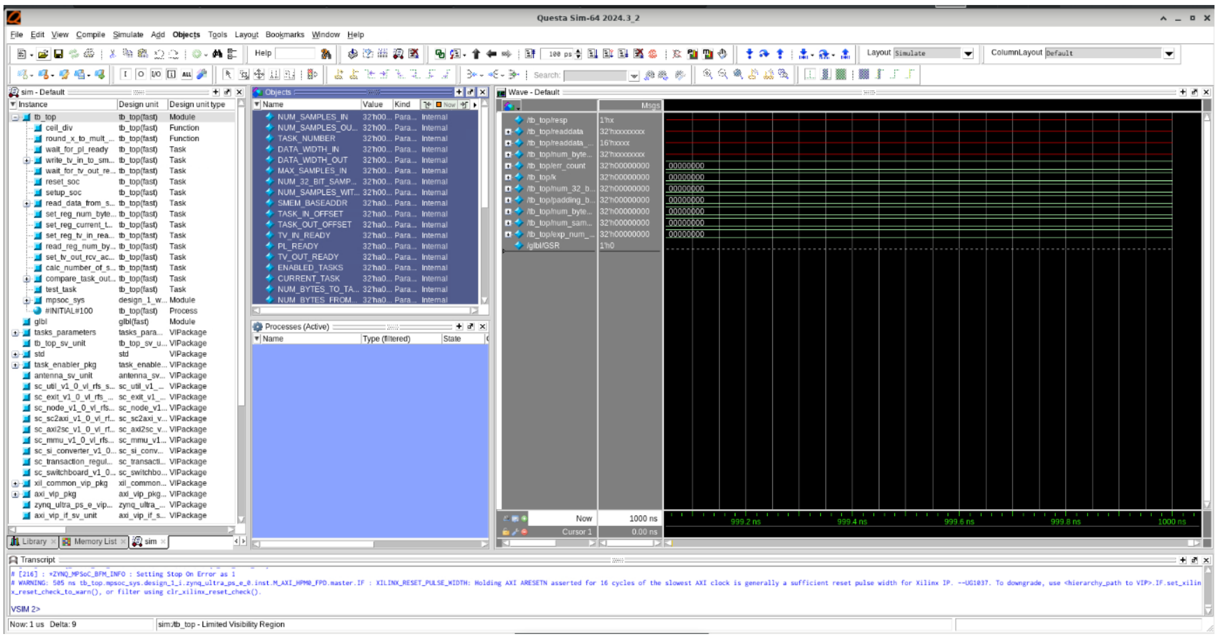
Simulation
Specify various settings associated to Simulation

Target simulator: Questa Advanced Simulator
Simulator language: Mixed
Simulation set: sim_1
Simulation top module name: tb_top
Compiled library location: /opt/tools/Questa/questa_sim_libs
☐ Generate simulation scripts only
☐ Configure script to run simulator in GUI mode

You can also target different *Simulation set:* sim_1 and sim_lite

Click *Apply* and then *OK* to close the window.

After left-clicking the *Run Simulation* button again, your design will compile and elaborate for QuestaSim and will run the QuestaSim gui:



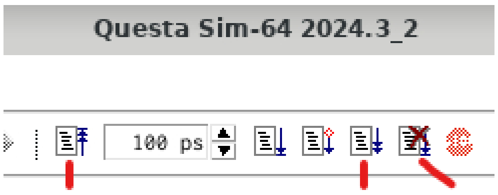
```
# [216] : *ZYNQ_MPSoC_BFM_INFO : Setting Stop On Error as 1
# WARNING: 505 ns tb_top.mpsoc_sys.design_1_i.zynq_ultra_ps_e_0.inst.M_AXI_HPM0_FPD.master.IF : XILINX_RESET_PULSE_WIDTH: Holding AXI ARESETN asserted for 16 cycles of the slowest AXI clock is generally a sufficient reset pulse width for Xilinx IP. --UG1037. To downgrade, use <hierarchy_path to VIP>.IF.set_xilinx_reset_check_to_warn(), or filter using clr_xilinx_reset_check().

VSIM 2>
```

Now: 1 us Delta: 9   sim:/tb_top - Limited Visibility Region

Helpful Questasim tips:

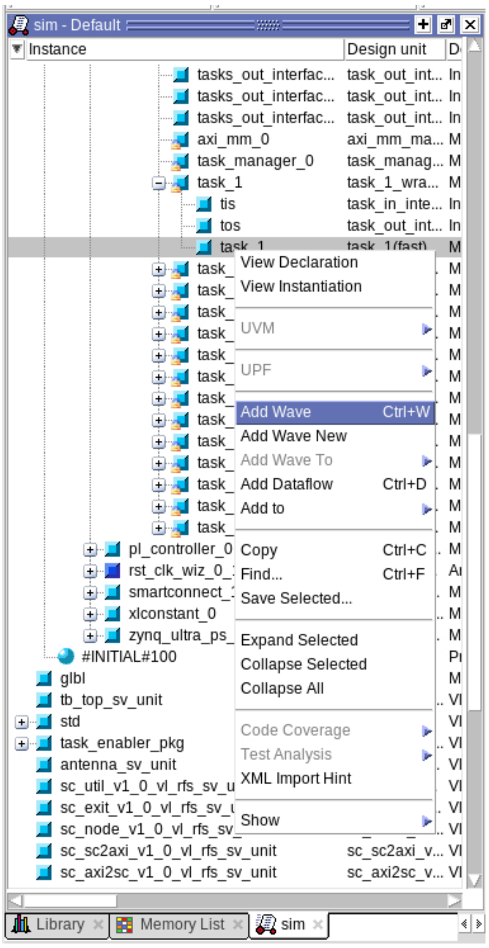
1: Running the simulation:



Reset Run All Break

2: Adding waves to waveform

Go to sim window and navigate to desired hierarchy and then right-click to then select *Add Wave*:



3: To maximize windows hit F5, then you can navigate through all the tabs the same size.

4: Waveform key bindings:

'F' - show full simulation waveform

'C' - Zoom in on the cursor

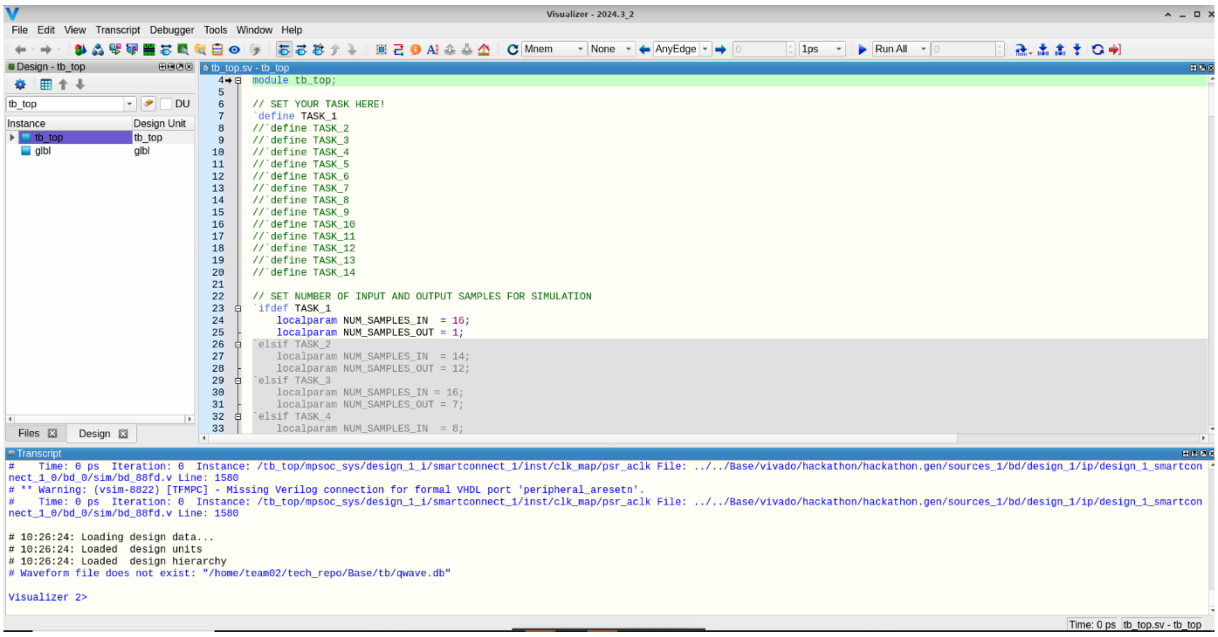
'I' and *'O'* - Zoom in and Zoom out in the center of the screen

To run QuestaSim and Visualizer from Makefile, go to /Base/tb/ of your repository in terminal and simply type *make help*, this will show all the available commands for running the batch simulation, Questasim gui and Visualizer gui. There are specific targets to different simulation environments. If you chose sim_1 you will be running regular flow (make batch, make gui_visualizer etc). If you chose sim_lite, you will be running lite flow (make batch_lite, make gui_visualizer_lite etc).

You will only be able to run Makefile based compilation after you run simulation from vivado gui first! It applies to both sim_1 and sim_lite!

Make sure to modify Makefile (/Base/tb/Makefile) whenever you're adding new files to your project.

Visualizer gui:



Instance	Design Unit
tb_top	tb_top
glbl	glbl

```
module tb_top;

// SET YOUR TASK HERE!
`define TASK_1
//`define TASK_2
//`define TASK_3
//`define TASK_4
//`define TASK_5
//`define TASK_6
//`define TASK_7
//`define TASK_8
//`define TASK_9
//`define TASK_10
//`define TASK_11
//`define TASK_12
//`define TASK_13
//`define TASK_14

// SET NUMBER OF INPUT AND OUTPUT SAMPLES FOR SIMULATION
`ifdef TASK_1
    localparam NUM_SAMPLES_IN  = 16;
    localparam NUM_SAMPLES_OUT = 1;
`elsif TASK_2
    localparam NUM_SAMPLES_IN  = 14;
    localparam NUM_SAMPLES_OUT = 12;
`elsif TASK_3
    localparam NUM_SAMPLES_IN = 16;
    localparam NUM_SAMPLES_OUT = 7;
`elsif TASK_4
    localparam NUM_SAMPLES_IN  = 8;
```

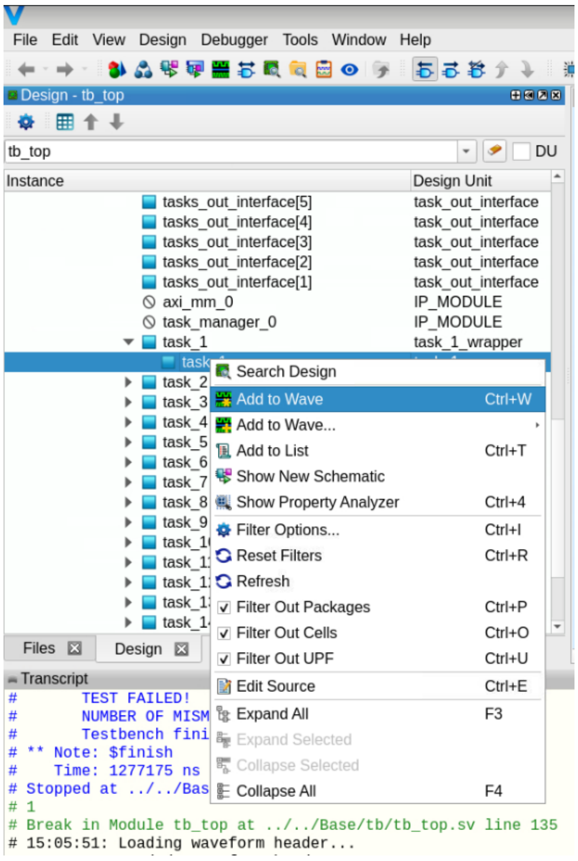
```
#    Time: 0 ps  Iteration: 0  Instance: /tb_top/mpsoc_sys/design_1_i/smartconnect_1/inst/clk_map/psr_aclk File: ../../Base/vivado/hackathon/hackathon.gen/sources_1/bd/design_1/ip/design_1_smartcon
nect_1_0/bd_0/sim/bd_88fd.v Line: 1580
# ** Warning: (vsim-8822) [TFMPC] - Missing Verilog connection for formal VHDL port 'peripheral_aresetn'.
#    Time: 0 ps  Iteration: 0  Instance: /tb_top/mpsoc_sys/design_1_i/smartconnect_1/inst/clk_map/psr_aclk File: ../../Base/vivado/hackathon/hackathon.gen/sources_1/bd/design_1/ip/design_1_smartcon
nect_1_0/bd_0/sim/bd_88fd.v Line: 1580

# 10:26:24: Loading design data...
# 10:26:24: Loaded  design units
# 10:26:24: Loaded  design hierarchy
# Waveform file does not exist: "/home/team02/tech_repo/Base/tb/qwave.db"

Visualizer 2>
```

To Run simulation click the button.

To add a waveform, navigate the *Design* tab and right click desired unit and *Add to Wave* :



Instance	Design Unit
tasks_out_interface[5]	task_out_interface
tasks_out_interface[4]	task_out_interface
tasks_out_interface[3]	task_out_interface
tasks_out_interface[2]	task_out_interface
tasks_out_interface[1]	task_out_interface
axi_mm_0	IP_MODULE
task_manager_0	IP_MODULE
task_1	task_1_wrapper
task	
task_2	
task_3	
task_4	
task_5	
task_6	
task_7	
task_8	
task_9	
task_1	
task_1	
task_1	
task_1	
task_1	

```
#       TEST FAILED!
#       NUMBER OF MISM
#       Testbench fini
# ** Note: $finish
#    Time: 1277175 ns
# Stopped at ../../Bas
# 1
# Break in Module tb_top at ../../Base/tb/tb_top.sv line 135
# 15:05:51: Loading waveform header...
```