

Vitis HLS Tool Flow

2022.1

Abstract

This lab introduces how to perform the most common processes related to the Vitis™ High-Level Synthesis (HLS) tool design flow. This lab targets the ZCU104 and VCK190 boards, which are NOT required to complete the lab.

This lab should take approximately 60 minutes.

CloudShare Users Only

You are provided three attempts to access a lab, and the time allotted to complete each lab is 2X the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Also, each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Create a new project in the Vitis HLS tool GUI
- Simulate a C design by using a self-checking test bench
- Synthesize the design
- Perform design analysis using the Analysis Perspective view
- Perform co-simulation on a generated RTL design by using a provided C test bench
- Implement the design

Introduction

This lab introduces major features of the Vitis High-Level Synthesis (HLS) tool GUI flow. You will use the Vitis HLS tool in GUI mode to create a project. You will also simulate, synthesize, and implement the provided design.

This design implements a discrete cosine transformation (DCT), and it is provided as C source. The function leverages a 2D DCT algorithm by first processing each row of the input array via a 1D DCT, then processing the columns of the resulting array through the same 1D DCT. It calls the *read_data*, *dct_2d*, and *write_data* functions.

The *read_data* function is defined at line 54 and consists of two loops: *RD_Loop_Row* and *RD_Loop_Col*. The *write_data* function is defined at line 66 and consists of two loops to perform

writing the result. The *dct_2d* function, defined at line 23, calls the *dct_1d* function and performs transpose.

Finally, the *dct_1d* function, defined in line 4, uses *dct_coeff_table* and performs the required function by implementing a basic iterative form of the 1D Type-II DCT algorithm.

The following figure shows the function hierarchy on the left-hand side, the loops in the order they are executed, and the flow of data on the right-hand side.

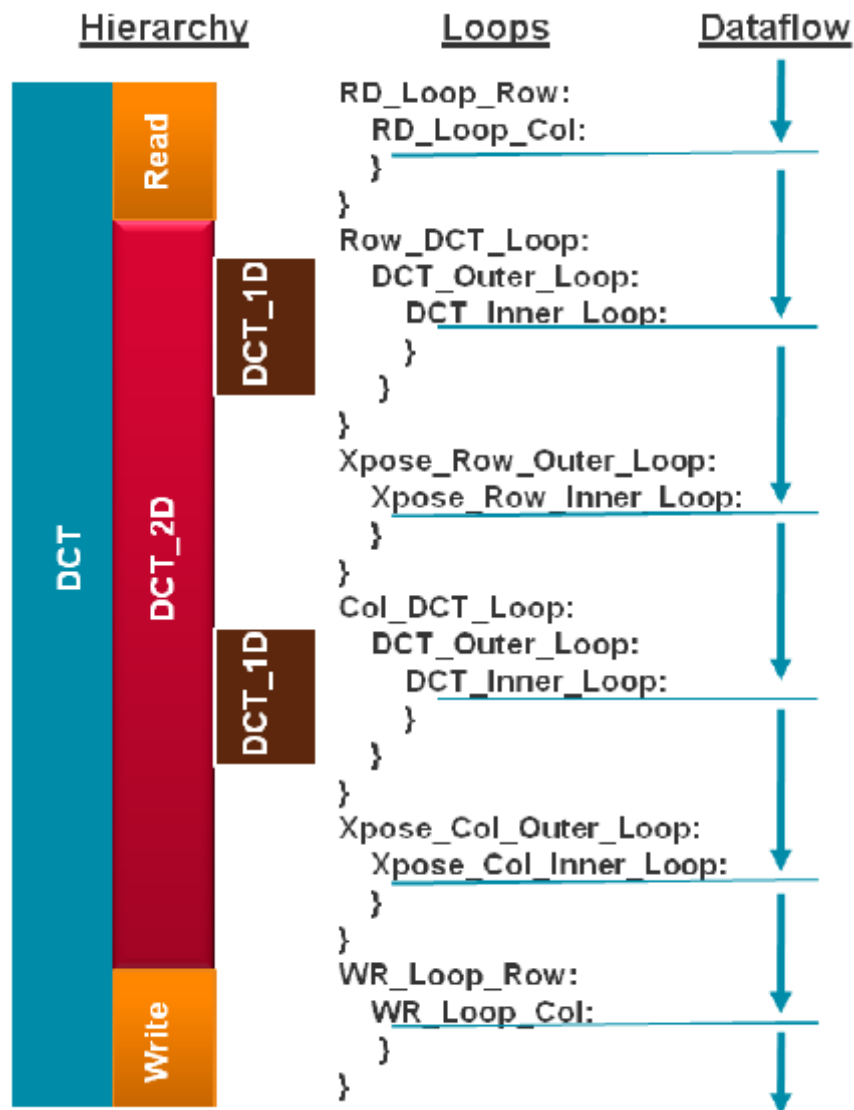


Figure 1-1: Design Hierarchy and Dataflow

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

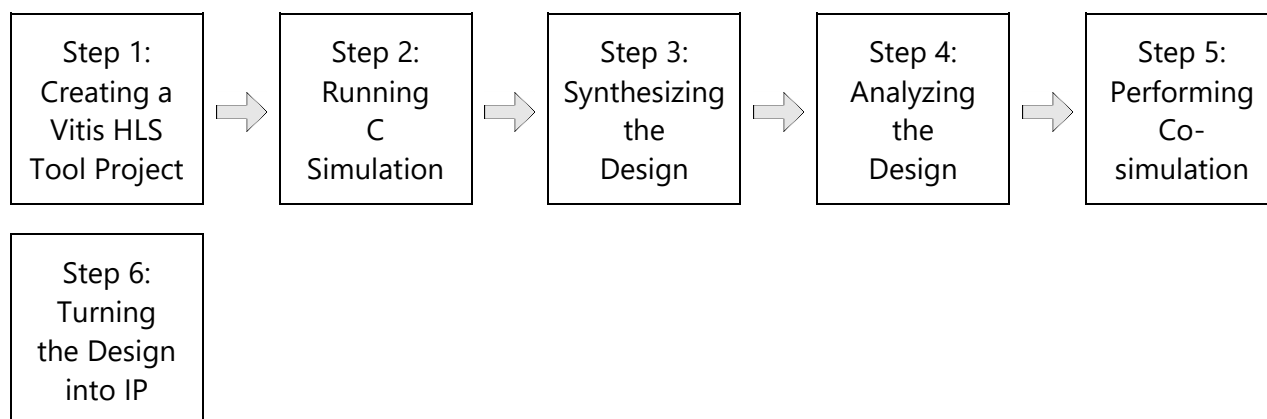
One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` will be expanded), and some tools require manual expansion (`/home/xilinx/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used by many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from a previous launch of the tools.

This means that if you sourced a Tcl script or manually set any Tcl variables and you closed the tool, then when you reopen the tool, you will need to source the Tcl script again and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

General Flow




Creating a Vitis HLS Tool Project

Step 1

In this step, you will launch the Vitis HLS tool GUI and create a new project for the provided C-based discrete cosine transformation (DCT) design.

There are several ways to launch the Vitis HLS tool. The most popular are shown here.

1-1. Launch the Vitis HLS tool.

1-1-1. Click the desktop or toolbar icon ().

If the desktop or toolbar icon is not available, then you can launch the tool by:

[Windows 10 users:] Selecting **Start** > **Xilinx Design Tools** > **Vitis HLS 2022.1**.

[VM Linux users:] Clicking the **Show Applications** icon in the toolbar (), typing **HLS** into the search window, and clicking the **Vitis HLS** icon.

The Vitis HLS tool opens to the Welcome window. From the Welcome window, you can create a new project, open examples, and access documentation and examples.

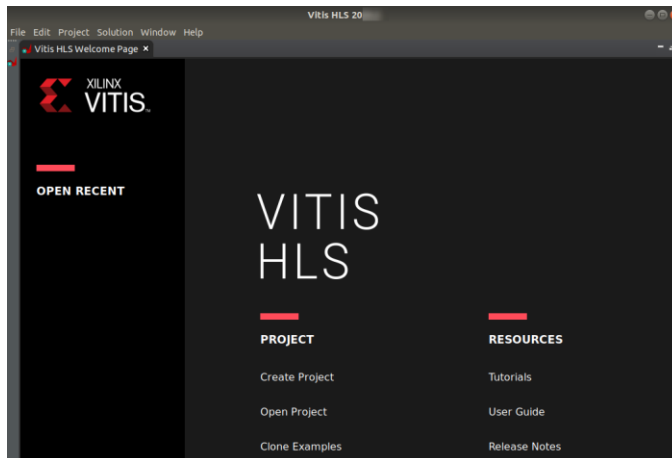


Figure 1-2: Vitis HLS Welcome Page

Here you will create a new Vitis HLS tool project from scratch.

1-2. Create a Vitis HLS tool project named *dct_prj*.

1-2-1. Click **Create Project** from the Welcome Page.

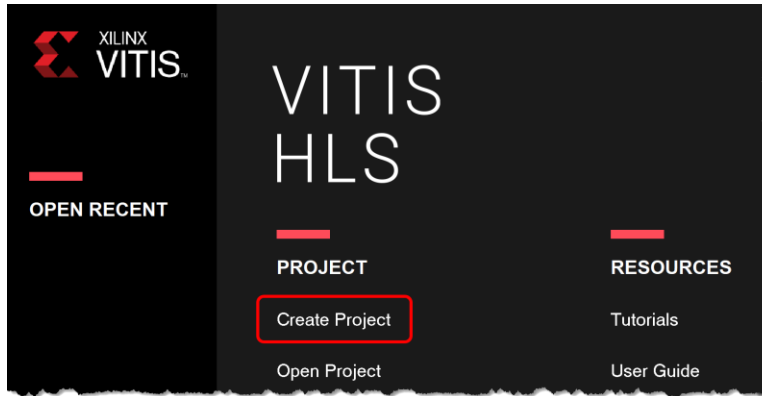


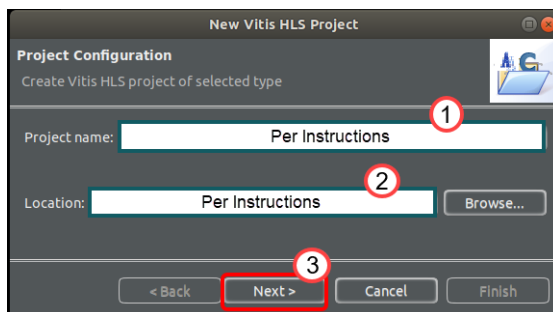
Figure 1-3: Creating a New Vitis HLS Tool Project

The Project Configuration dialog box asks for a project name and location.

1-2-2. Enter **dct_prj** in the Project name field (1).

1-2-3. Browse to or enter **\$TRAINING_PATH/hls_tool_flow/lab/[zcu104 | vck190]** in the Location field (2).

You can select either the ZCU104 or VCK190 target board to perform the lab.



1-2-4. Click **Next** (3).

The Add/Remove Design Files dialog box opens.

1-3. Add the provided files to the project.

1-3-1. Click **Add Files**.

1-3-2. Browse to **\$TRAINING_PATH/hls_tool_flow/support/[zcu104 | vck190]/dct**.

1-3-3. Double-click **dct.c** to select and add it to the project.

The Vitis HLS tool automatically adds the working directory (project directory) and any directory that contains C files to the search path. Hence, header files that reside in these

directories are automatically included in the project, so there is no need to explicitly specify them.

You must specify the path to other header files located in other paths (if any) by selecting the file and clicking **Edit CFLAGS**.

Note: You can add new files and list file-specific compiler directives for each entry here.

1-4. The tools need to be told which module is the top function. The tool will scan the project files and identify candidates to be the top of the program hierarchy. You will now choose one of these candidates.

1-4-1. Click **Browse** next to the Top Function field.

The Select Top function dialog box opens and lists the candidate functions available in the project as <function name> (<source file where the function is located>).

1-4-2. Select **dct (dct.c)** from the list to identify which function you want to serve as the "main".

1-4-3. Click **OK** to register the selection.

Note: You can also manually enter the name of the top function in the Top Function field.

C programs typically use *main()* as the top-level function. The Vitis HLS tool design flow supports using any sub-function below *main()* as the top-level function for synthesis. You cannot synthesize the top-level function *main()*.

The following are additional rules:

- Only one function is allowed as the top-level function for synthesis.
- The top-level function and all functions below it are synthesized as a unit.
- Additional functions that are not in the hierarchy under the top-level function must be merged into the top-level function for synthesis.

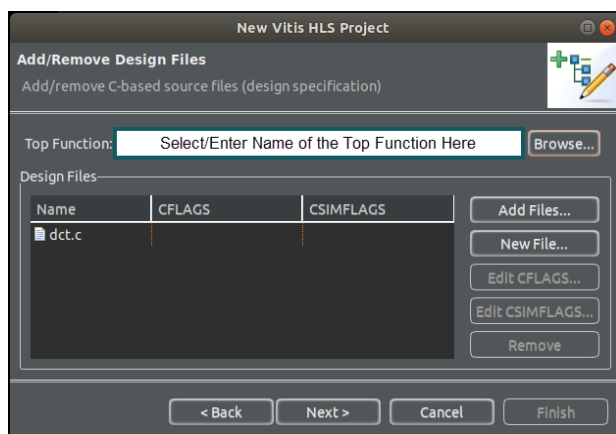


Figure 1-4: Adding Files to a New Vitis HLS Project

1-4-4. Click **Next** to add these files to the project and advance to the test bench files page.

1-5. Add the test bench files.

Test bench files are specified here.

- 1-5-1. Click **Add Files**.
- 1-5-2. Browse to **\$TRAINING_PATH/hls_tool_flow/support/[zcu104 | vck190]/dct**.
- 1-5-3. Select **dct_test.c**, **in.dat**, **out.golden.dat**.
- 1-5-4. Click **Open** to add these files.
- 1-5-5. Click **Next** to advance to the Solution Configuration page.

1-6. Specify some of the physical parameters of the design.

Physical parameters include clock speed, part selection, and other options.

By default, **solution1** is populated in the **Solution Name** field and can be overridden if desired.

- 1-6-1. Set the clock period to **10**.

Note: The default unit for the clock period is nanosecond.

The Uncertainty field enables you to specify a stringency (constraint) that will cause the tools to work harder to achieve a viable result.

As a quick reminder, clock uncertainty is the difference in time within a local clock network between the time the of rising edge's earliest arrival at a register and the rising edge's latest arrival at any register. This number is sometimes referred to as skew.

- 1-6-2. Leave the Uncertainty field blank.
- 1-6-3. Click the **Browse (. .)** icon from the Part Selection option to select a part or board.

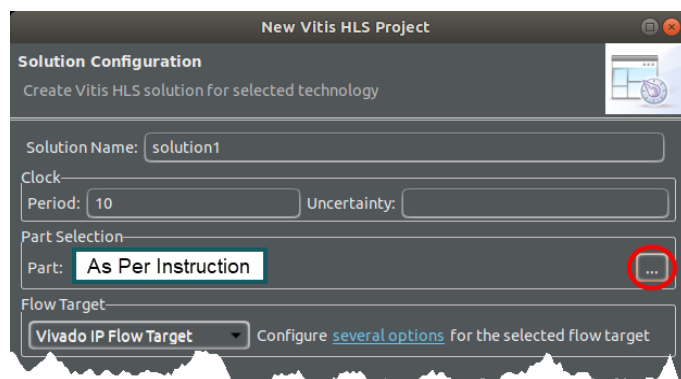


Figure 1-5: Locating the Board Browse Button

The Device Selection Dialog box opens.

1-6-4. Click **Boards** to make your selection based on the board name rather than specifying a part designation.

1-6-5. Enter **ZYNQ UltraScale+ or Versal** in the Search field.

Alternatively, you can type a part of the name, such as ZCU104 or VCK190, which is much quicker.

Note: For 2022.1, there is a known issue where the results may not appear. The quick fix to this is to click the Parts button and then click the Boards button.

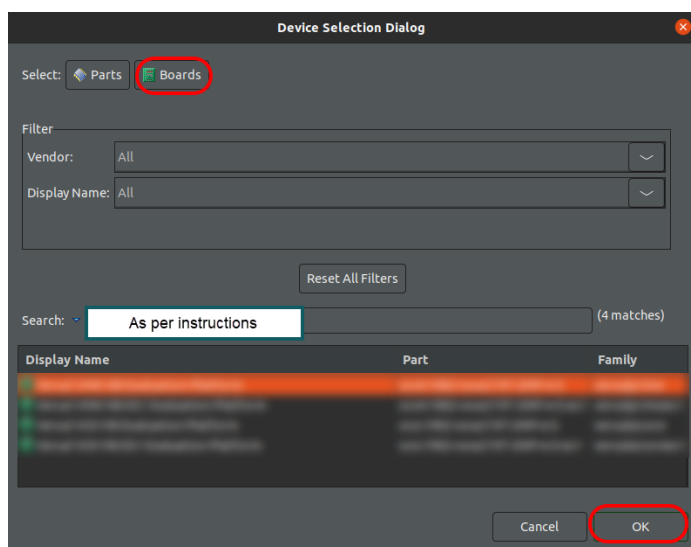


Figure 1-6: Filtering to Quickly Locate Target Platforms

1-6-6. Select **ZYNQ UltraScale+ ZCU104 Evaluation Board or Versal VCK190 Evaluation Platform** from the search list.

1-6-7. Click **OK** to select the board.

1-6-8. If not already set, select **Vivado IP Flow Target** from the drop-down list under the Flow Target field.

1-6-9. Click **Finish**.

You will see the created project in the Explorer tab.

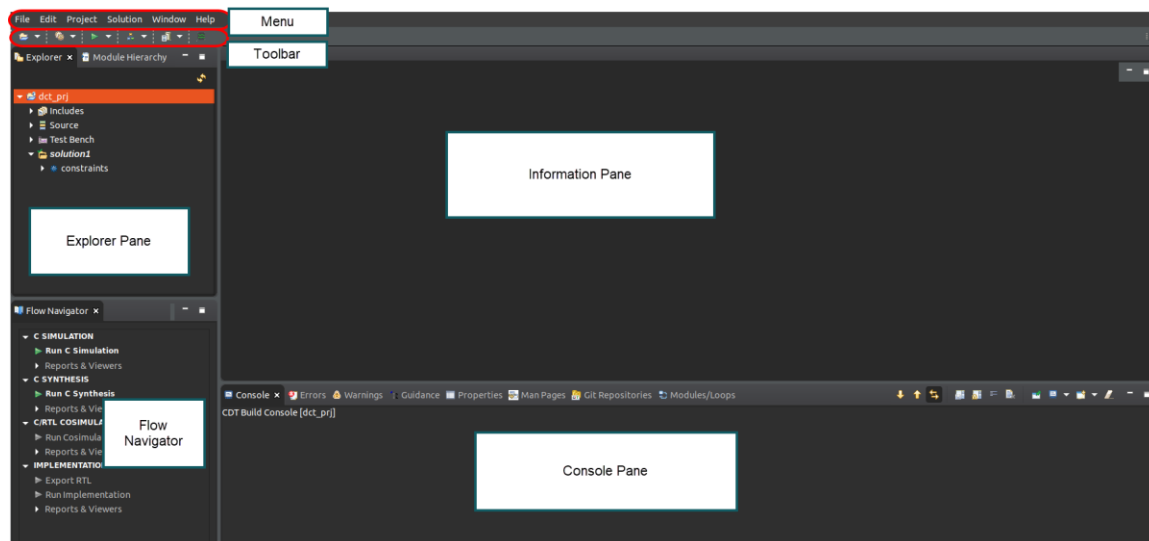


Figure 1-7: Vitis HLS with Newly Created Project

The Vitis HLS tool GUI consists of various panes for proceeding with development work:

- The Explorer view enables you to navigate through the project hierarchy. A similar hierarchy exists in the project directory on the disk. The expandable sub-folders organize various project components, such as source files and test benches.
- The Information area displays report summaries and shows the contents of open files. Files can be opened by double-clicking them in the Explorer view.
- The Console view displays the output when the Vitis HLS tool is running synthesis or simulation.
- The Flow Navigator view provides access to commands and processes to take your source code through simulation, synthesis, and exported output.
- Other views open dynamically, depending on the operations being performed, such as when source code is opened in the Information area, and the Outline and Directive views are displayed on the right side and show information related to the hierarchy of the code.

Running C Simulation

Step 2

With the project created, you will now validate the C code for both syntax and behavior. You will use the provided self-checking C test bench code to validate the behavior.

As a brief review: The discrete cosine transform algorithm expresses a finite sequence of data points in terms of a sum of different frequency cosine functions. DCT is commonly used to separate an image into spectral sub-bands. Like the FFT or DFT, there are many iterations performed in the algorithm.

2-1. Review the provided source and test bench files.

2-1-1. Expand the **Source** folder in the Explorer pane.

2-1-2. Double-click **dct.c** to open the file.

This will open the source file in the Information pane.

2-1-3. Review the code structures.

2-1-4. Double-click **dct.h** from the Outline view to open this header file in a new editor window.

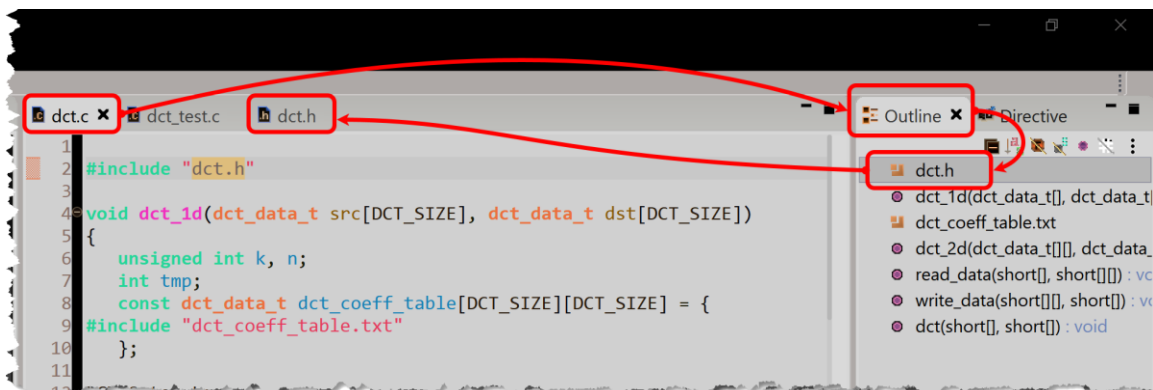


Figure 1-8: Opening a Header File from the Outline View

2-1-5. Review the contents of the header file.

2-1-6. Expand the **Test Bench** folder in the Explorer pane.

2-1-7. Double-click **dct_test.c** to open it in the Information pane.

This test bench is a self-checking test bench; i.e., the computed output is compared against a reference golden output and returns either pass or fail.

2-2. Simulate the Vitis HLS tool design.

2-2-1. Select **Project** > **Run C Simulation**.

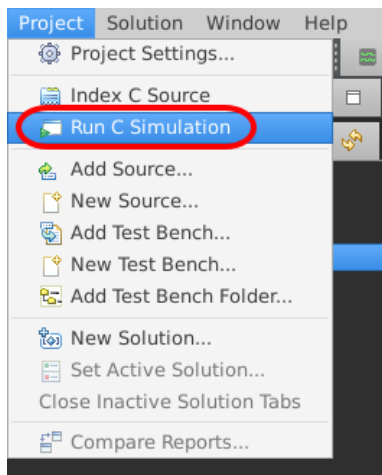


Figure 1-9: Launching the C Simulation

You can also run C simulation from the Flow Navigator by clicking **Run C Simulation** under C SIMULATION.

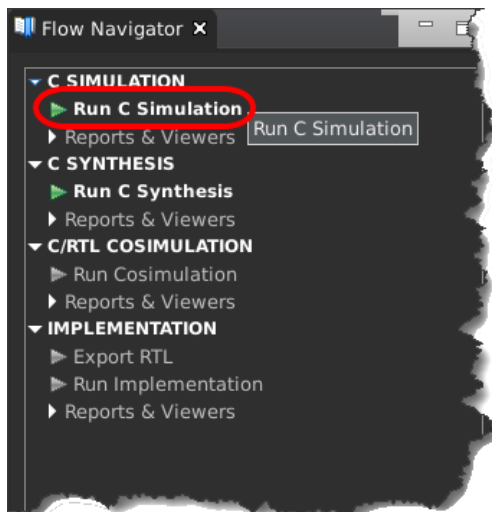


Figure 1-10: Launching the C Simulation Using the Flow Navigator

The Run C Simulation dialog box opens.

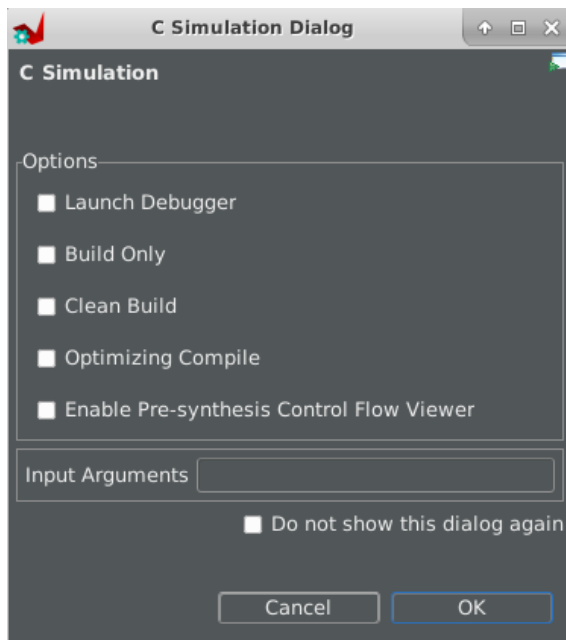


Figure 1-11: C Simulation Dialog Box

Each of the options controls how the simulation is run:

- Launch Debugger: After compilation the debug perspective automatically opens for you to step through the code.
- Build Only: Compiles the source code and the test bench, but the simulation does not run. This option can be used to test the compilation process and resolve any issues with the build before running the simulation.
- Clean Build: Removes any existing executable and object files before compiling the code.
- Optimizing Compile: By default, the design is compiled with debug information enabled, allowing the compilation to be analyzed and debugged. The Optimizing Compile option uses a higher level of optimization effort when compiling the design but does not add the information required by the debugger. This increases the compile time but should reduce the simulation runtime.
- Enable Pre-Synthesis Control Flow Viewer: Generates the Pre-synthesis Control Flow report.
- Input Arguments: Specify any inputs required by your test bench main() function.

2-2-2. Select Default Options (i.e. no additional selections required to run a C simulation).

2-2-3. Click OK.

The simulation log will be displayed in the editor pane.

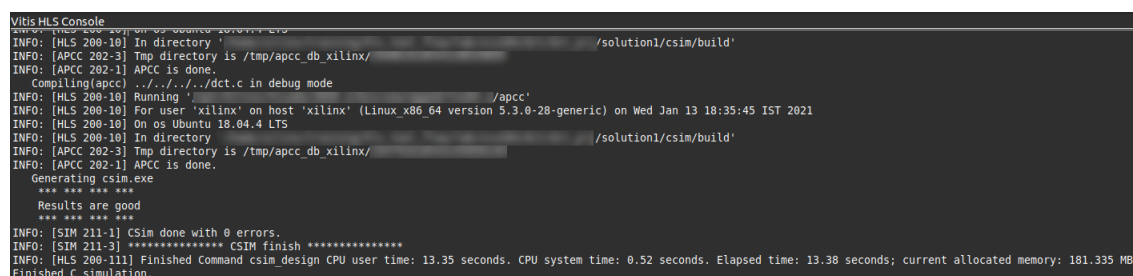
2-3. View the simulation report.

The information generated by the Vitis HLS tool can be found in two places, both described here.

The first is the **Console** window, which reports not only the output produced by the code, but the full set of simulation engine messages as well. The **simulation log** provides only key simulation engine messages and the simulated code output.

2-3-1. Select the **Console** tab in the lower portion of the tool's GUI.

You may need to scroll to view all the output produced by the simulation.



```

Vitis HLS Console
INFO: [HLS 200-10] In directory /solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/
INFO: [APCC 202-1] APCC is done.
Compiling(apcc) ../../../../dct.c in debug mode
INFO: [HLS 200-10] Running /apcc'
INFO: [HLS 200-10] For user 'xilinx' on host 'xilinx' (Linux_x86_64 version 5.3.0-28-generic) on Wed Jan 13 18:35:45 IST 2021
INFO: [HLS 200-10] On os Ubuntu 18.04.4 LTS
INFO: [HLS 200-10] In directory /solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/
INFO: [APCC 202-1] APCC is done.
Generating csim.exe
*** **
Results are good
*** **
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
INFO: [HLS 200-11] Finished Command csim_design CPU user time: 13.35 seconds. CPU system time: 0.52 seconds. Elapsed time: 13.38 seconds; current allocated memory: 181.335 MB.
Finished C simulation.
  
```

Figure 1-12: Example Output After Simulation

The other location, described below, provides only a few simulation engine messages and the simulated code output. Typically this is opened after the simulation completes; however, if you need to access it after closing the log pane, here's how to access the simulation report.

2-3-2. Expand **dct_prj** > **solution1** > **csim** > **report** in the Explorer pane.

2-3-3. Double-click the log file name to open it in the editor pane.

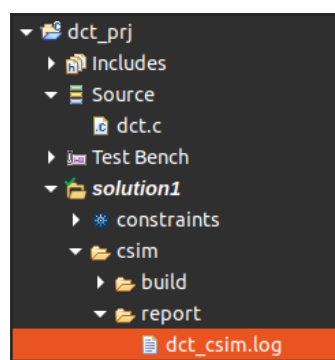


Figure 1-13: Locating the Simulation Log File

You should see a "Results are good" message in the simulation log file and the console area.


Synthesizing the Design

Step 3

Next, you will synthesize the design by using the default settings. You will then analyze the result to find the resources needed to implement the C function.

3-1. Synthesize the design.

3-1-1. Use one of the following methods to launch the C synthesis tool:

- Click the **Run Flow** icon () in the toolbar (1).
- Click the pull-down next to the Run Flow icon in the toolbar and select **C Synthesis** (2).
- From the Explorer tab, select the desired solution to synthesize and right-click and select **C Synthesis** > **Active Solution** (3).

Note: More complex projects allow multiple solutions to be batched which, is what the other options in this menu support.

- From Flow Navigator, select **C Synthesis** > **Run C Synthesis** (4).

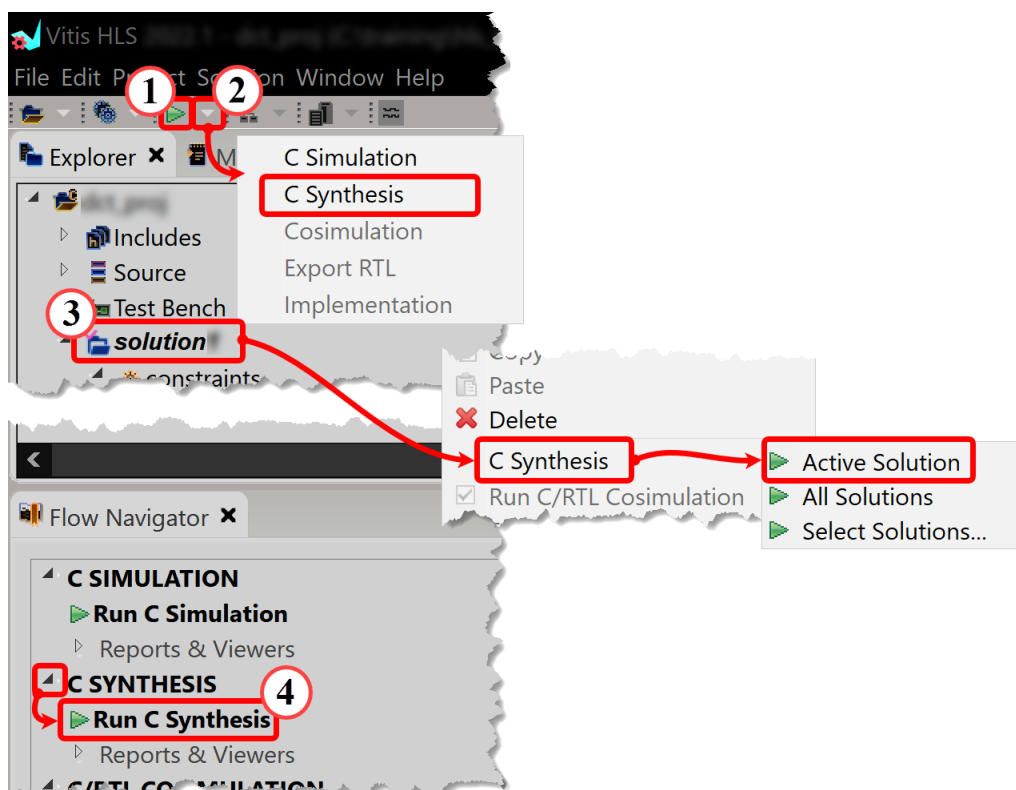


Figure 1-14: Multiple Methods for Launching C Synthesis

The C Synthesis - Active Solution dialog box opens.

The tool automatically fills in any information it already knows, such as the project's clock period, part selection, and flow type. You can modify these here if needed.

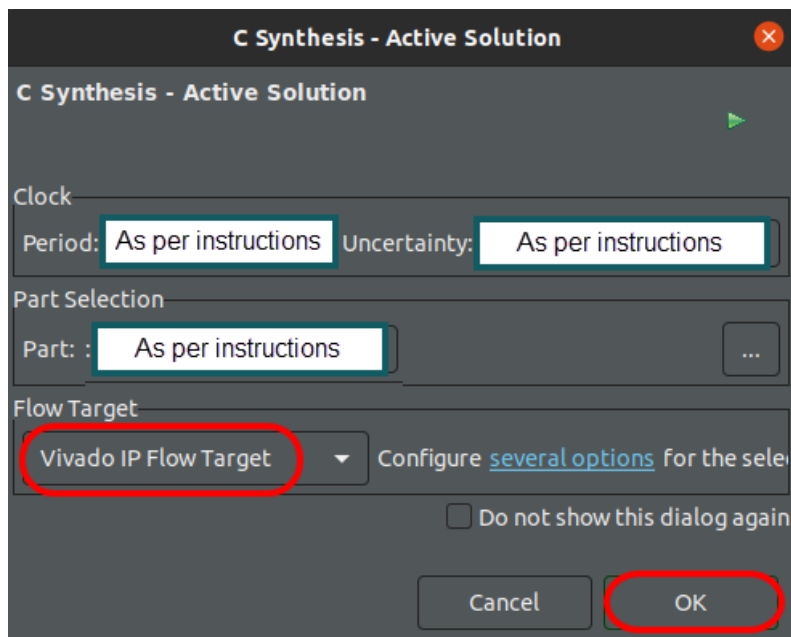


Figure 1-15: C Synthesis Dialog Box

3-1-2. Click **OK** to begin the synthesis process.

Note: Selecting the flow target as Vivado IP Flow Target causes the tools to generate RTL IP files for use in the Vivado Design Suite. In simpler terms, you are creating a piece of IP to be used in a larger design that you will create here and import into the Vivado Design Suite.

Another way to use this tool is to select Vitis Kernel Flow as the flow target, which generates a compiled kernel object (.xo) for the Vitis application acceleration flow. This is a narrower solution and is discussed in other labs.

When synthesis completes, the Synthesis Summary Report appears in the Information pane. Notice the Timing Estimate section, which shows the estimated clock period that this solution can be run. As long as the estimated time, which includes the uncertainty factor, is less than the target value, the design will meet timing.

The Performance & Resource Estimates section shows the total number of resources required to implement this design. The top-level shows these resources and the component elements in the code are listed below. The total and component latencies are provided to help determine system performance.

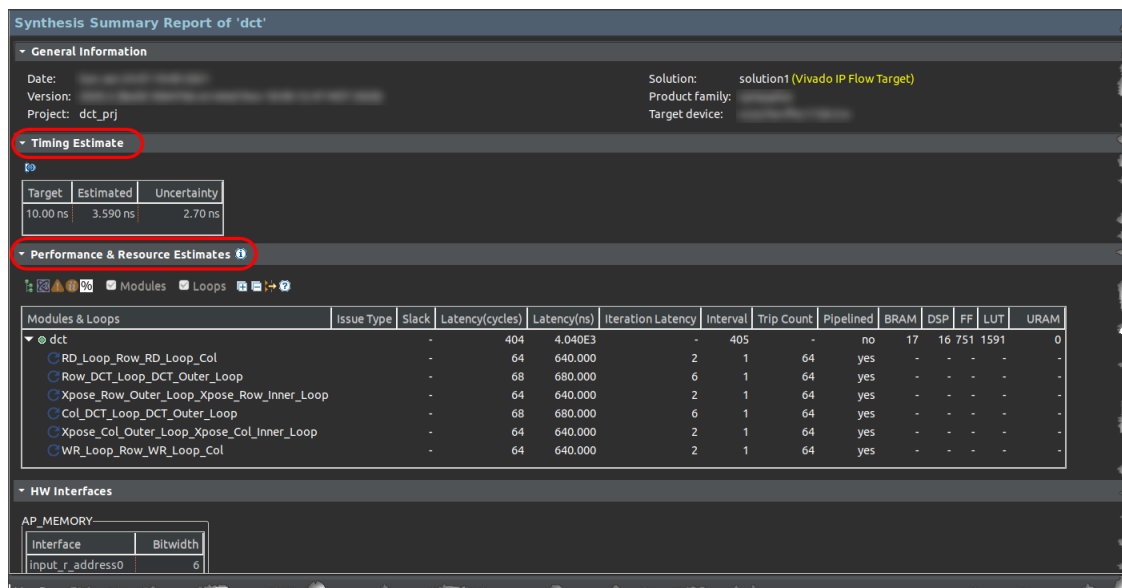


Figure 1-16: Synthesis Summary Report (Example)

3-1-3. Expand the **dct_prj > solution1 > syn > report** folder in the Explorer pane.

3-1-4. Double-click the **dct_csynth.rpt** file to view the Synthesis report.

3-1-5. Scroll to **Performance Estimates** and **Utilization Estimates** in the Synthesis report to answer the following question.

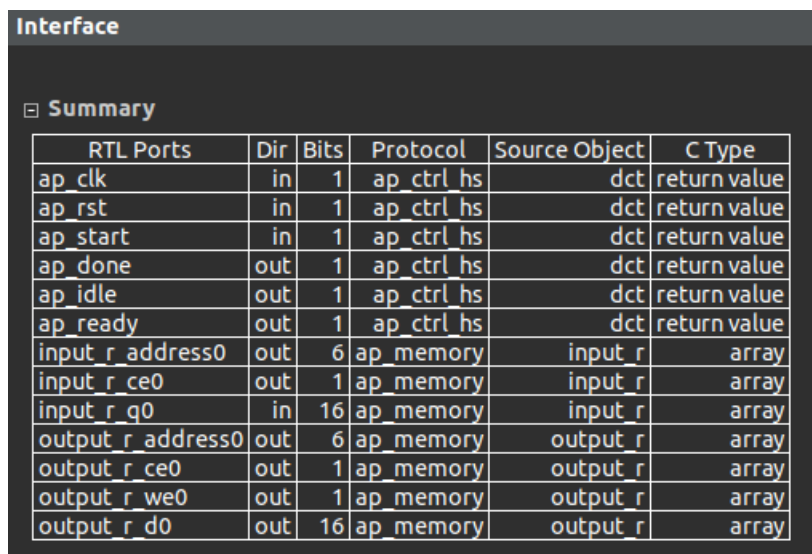
Question 1

Write down the following details from the Synthesis report:

- Estimated clock frequency:
- Worst case latency:
- Number of BRAM_18K:
- Number of DSP48E used:
- Number of FFs used:
- Number of LUTs used:

3-1-6. Select **Interface** > **Summary** in the Synthesis report.

This report shows the top-level interface signals generated by the tools.



The screenshot shows the 'Interface' tab selected in the Synthesis report. Under the 'Summary' section, a table lists the generated interface signals. The table has six columns: RTL Ports, Dir, Bits, Protocol, Source Object, and C Type.

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dct	return value
ap_rst	in	1	ap_ctrl_hs	dct	return value
ap_start	in	1	ap_ctrl_hs	dct	return value
ap_done	out	1	ap_ctrl_hs	dct	return value
ap_idle	out	1	ap_ctrl_hs	dct	return value
ap_ready	out	1	ap_ctrl_hs	dct	return value
input_r_address0	out	6	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	16	ap_memory	input_r	array
output_r_address0	out	6	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	16	ap_memory	output_r	array

Figure 1-17: Generated Interface Signals

The *ap_clk* and *ap_rst* signals are automatically added.

ap_start, *ap_done*, and *ap_idle* are top-level signals used as handshaking signals to indicate when the design can accept the next computation command (*ap_idle*), when the next computation is started (*ap_start*), and when the computation is completed (*ap_done*).

Other signals are generated based on the design itself.

3-1-7. Select the **Console** tab.

The Synthesis log is available in the Vitis HLS Console.

Note that when the *solution1* > *syn* folder is expanded in the Explorer view, it will show the *report*, *verilog*, and *vhdl* sub-folders under which the report files and generated source files (VHDL, Verilog, header, and cpp) are available. Double-clicking any of these entries will open the corresponding file in the Information pane.

The resulting design is structured hierarchically using functions, and reports corresponding to lower-level functions are created. By default, the report for the top-level function is displayed in the Information pane once synthesis is completed.

Analyzing the Design Using the Generated Reports

Step 4

With the synthesis phase now complete, you will analyze the synthesis reports in more detail to help you understand and analyze the performance of the implementation. These reports include the Synthesis Summary report, Schedule Viewer, Function Call Graph, and Dataflow Viewer.

These reports are accessed from the Flow Navigator:

- **Schedule Viewer:** Shows each operation and control step of the function and the clock cycle that it executes in.
- **Dataflow Viewer:** Shows the dataflow structure inferred by the tool, allowing you to inspect the channels (FIFO/PIPO) and examine the effect of channel depth on performance.
- **Function Call Graph Viewer:** Displays your full design after C synthesis or C/RTL co-simulation to show the throughput of the design in terms of latency and initiation interval (II).

By default, the Schedule Viewer is displayed. It provides both a tabular and graphical view of the design performance and resources.

4-1. Switch to the Analysis perspective and view the behavior through the Schedule Viewer.

4-1-1. Select **Solution** > **Open Schedule Viewer** to open the Schedule Viewer.

Alternatively, from the Flow Navigator, expand **C Synthesis** > **Reports & Viewers** and click **Schedule Viewer**.

When the Schedule Viewer opens, the following windows appear:

- **Schedule Viewer:** Graphically represents how data moves through the design.
- **Module Hierarchy:** Shows the function hierarchy and the performance characteristics of the current hierarchy.
- **Modules/Loops:** Shows additional performance and resource metrics for each function/loop in the Modules/Loops table under the report.
- **Performance Profile:** Shows the loops from the top-level function without any performance information.
- **Resource Profile:** Shows the resource usage of different elements of the synthesized function.
- **Properties view:** Shows the properties of the currently selected control step or operation in the Schedule Viewer.

The Module Hierarchy pane provides an overview of the entire RTL design. It shows the resources and latency contribution for each block in the RTL hierarchy. This view directly indicates any II or timing violations. In case of timing violations, the hierarchy window will also show the total negative slack observed in a specific module.

Note that the module hierarchies are displayed unexpanded by default.

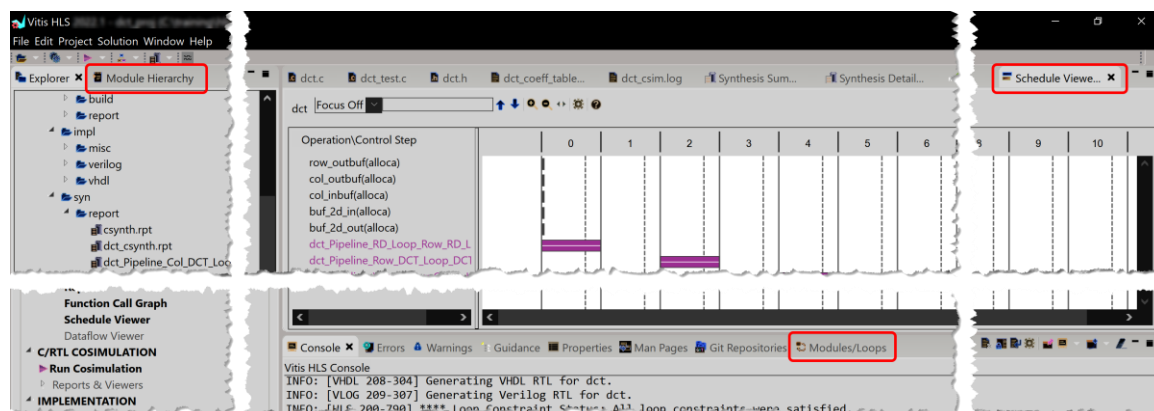


Figure 1-18: Schedule Viewer - Key Tabs

The Schedule Viewer illustrates the flow of data, showing each operation and control step of the function or loop and the clock cycle that it executes. The Schedule Viewer helps with identifying loop dependencies that prevent parallelism and cause timing violations and spotting data dependencies.

The left vertical axis shows the names of operations and loops in the RTL hierarchy in topological order, implying that an operation on that line can only be driven by operations from the previous line and can drive subsequent (later) lines.

As an example, signal 1 occurs prior to signal 2 and thus appears earlier—that is, higher in the signal list. Signal 3 follows signal 2 and appears after or below signal 2.



Figure 1-19: Understanding the Signal Chronology

The top horizontal axis shows the clock cycles in consecutive order.

The vertical dashed line in each clock cycle shows the reserved portion of the clock period due to clock uncertainty. This time is left by the tool for the place and route processes and any other backend tasks.

Each operation is shown as a box in the table. The box is horizontally sized according to the delay of the operation as a percentage of the total clock cycle. In the case of function calls, the provided cycle information is equivalent to the operation latency.

Multi-cycle operations are shown as boxes with a horizontal line through the center of the box.

General operator data dependencies are also displayed as solid blue lines. The green dotted line indicates an inter-iteration data dependency. Memory dependencies are displayed using golden lines.

Source code is associated with each operation in the Schedule Viewer report. Right-click the operation to access the Goto Source command, which opens the source code associated with the operation in a separate tab.

4-2. Analyze the performance of the dct module with the Schedule Viewer capability.

The dct module has three main resources:

- A loop called **RD_Loop_Row** that reads the data used in the computation from the **input.txt** file.
- A sub-block called **dct_2d**.
- A loop called **WR_Loop_Row** that writes the results to the **output.txt** file.

Notice that all the loops in the design are pipelined by default.

- 4-2-1. Right-click **dct_Pipeline_RD_Loop_Row_RD_Loop_Col** in the Module Hierarchy to open the context menu.
- 4-2-2. Select **Open Schedule Viewer** to open the Schedule Viewer focused on this module.
- 4-2-3. Expand **RD_Loop_Row_RD_Loop_Col** in the Schedule Viewer to see the **RD_Loop_Row_RD_Loop_Col** entry.

This causes the view in the Schedule Viewer to change and show the signals at this level of hierarchy.

4-2-4. Right-click **add_In61(+)** to both highlight the graph for this entry as well as open the context menu.

Note: this signal was chosen at random to illustrate some of the functionality that you are about to witness.

4-2-5. Select **Goto Source** to see the corresponding C source code.

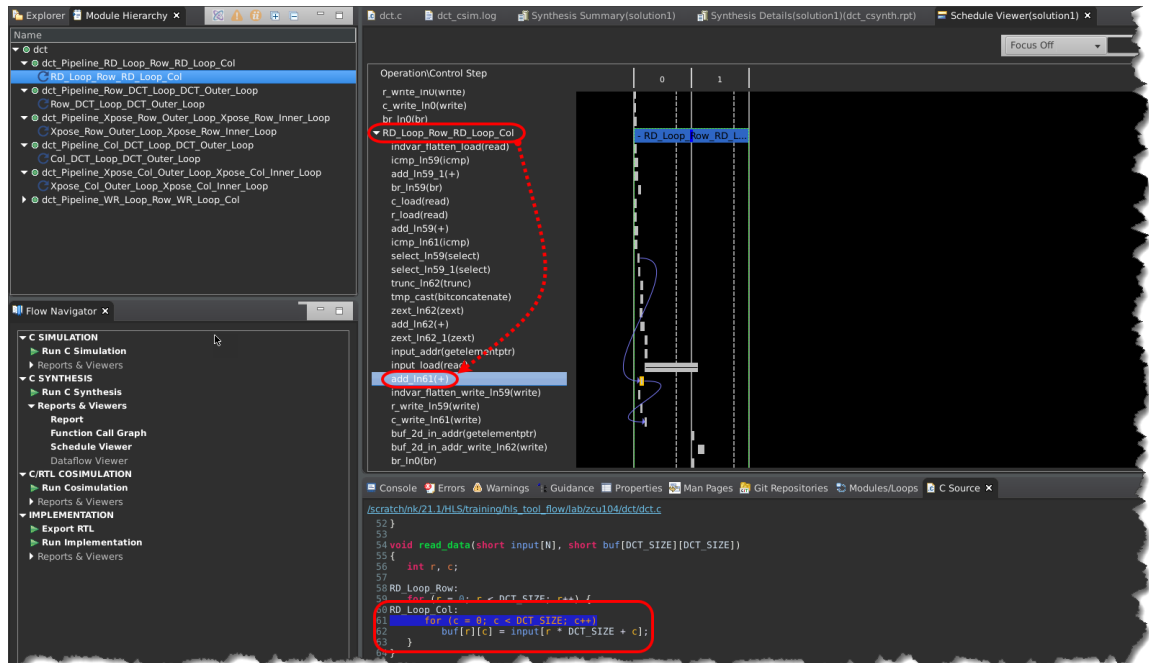


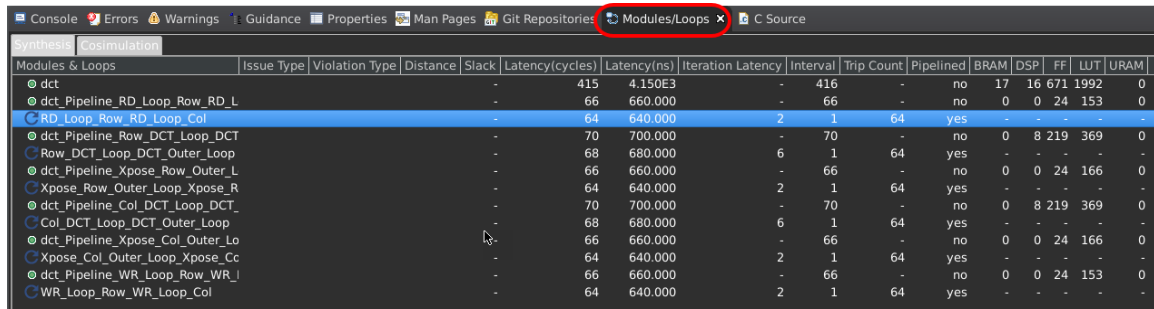
Figure 1-20: Performance of the dct Loop Operation

The information presented in the Schedule Viewer is explained below by reviewing the first set of resources to be executed:

- The design starts in the 0 state.
- It then starts to execute the logic in the loop `RD_Loop_Row`.
 - **Note:** In the first state of the loop, the exit condition is checked and there is an add operation.
- The loop executes over two states: 0, 1.

Question 2

What does the purple squiggly line represent?

4-2-6. Review the performance and resources used in this pane information.


Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dct				-	415	4.150E3	-	416	-	no	17	16	671	1992	0
dct_Pipeline_RD_Loop_Row_RD_L				-	66	660.000	-	66	-	no	0	0	24	153	0
RD_Loop_Row_RD_Loop_Col				-	64	640.000	2	1	64	yes	-	-	-	-	-
dct_Pipeline_Row_DCT_Loop_DCT				-	70	700.000	-	70	-	no	0	8	219	369	0
Row_DCT_Loop_DCT_Outer_Loop				-	68	680.000	6	1	64	yes	-	-	-	-	-
dct_Pipeline_Xpose_Row_Outer_L				-	66	660.000	-	66	-	no	0	0	24	166	0
Xpose_Row_Outer_Loop_Xpose_R				-	64	640.000	2	1	64	yes	-	-	-	-	-
dct_Pipeline_Col_DCT_Loop_DCT				-	70	700.000	-	70	-	no	0	8	219	369	0
Col_DCT_Loop_DCT_Outer_Loop				-	68	680.000	6	1	64	yes	-	-	-	-	-
dct_Pipeline_Xpose_Col_Outer_Lo				-	66	660.000	-	66	-	no	0	0	24	166	0
Xpose_Col_Outer_Loop_Xpose_Cc				-	64	640.000	2	1	64	yes	-	-	-	-	-
dct_Pipeline_WR_Loop_Row_WR_I				-	66	660.000	-	66	-	no	0	0	24	153	0
WR_Loop_Row_WR_Loop_Col				-	64	640.000	2	1	64	yes	-	-	-	-	-

Figure 1-21: Modules/Loops Pane (ZCU104)

This pane shows that this loop has a trip count of 64. Therefore, it iterates through these three states twice. It shows that the loop `RD_Loop_Row` takes 64 clock cycles to execute. Within the loop `RD_Loop_Col`, you can see that there are some adders, a two-cycle read operation, and a write operation.

This should make sense as there is both an adder needed for the loop along with a loop control mechanism, and some math must be performed to manage the indices of both the *buf* and *input* arrays.

This pane also shows the resources used at this level of hierarchy. In this design, you can see that most of the resources are due to the instances—blocks that are instantiated inside this block. As per the function selected in the Module Hierarchy tab, you can view the resources used by that function.

4-2-7. Select `dct_Pipeline_WR_Loop_Row_WR_Loop_Col` from the Module Hierarchy.

- 4-2-8. Expand the **WR_Loop_Row_WR_Loop_Col** loop in the Schedule Viewer (solution1).
- 4-2-9. Right-click **add_In73(+)** to both select the signal and open the context menu.
- 4-2-10. Select **Goto Source** to see the corresponding C source code.

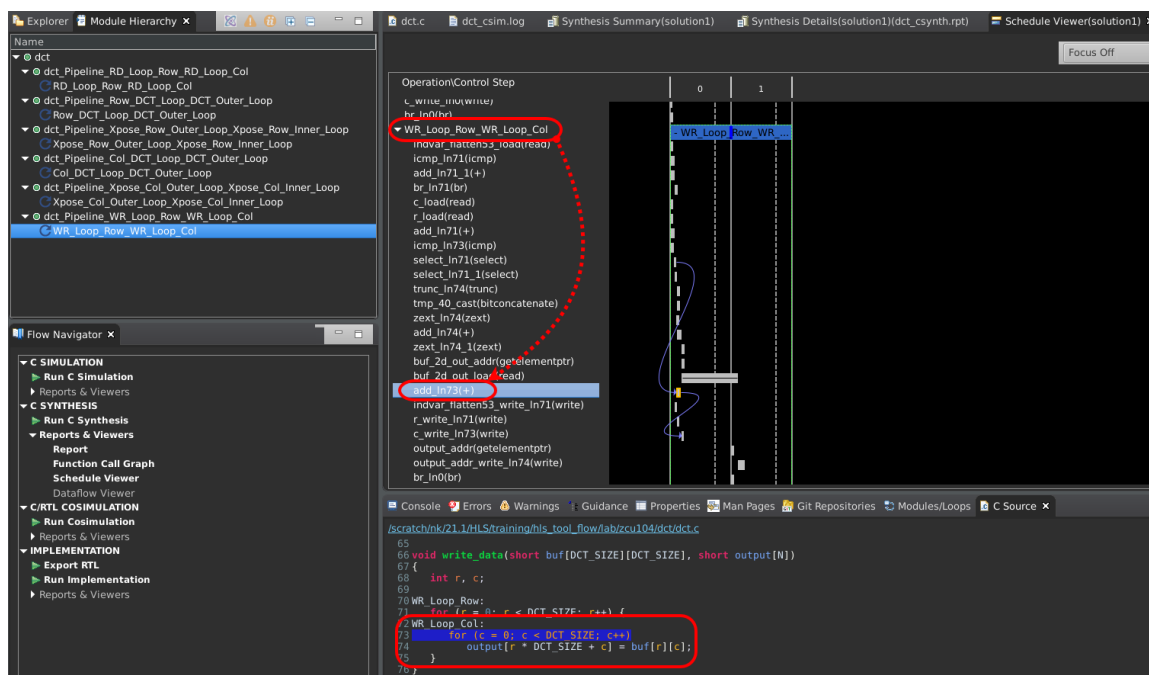


Figure 1-22: Performance of WR_Loop_Row_WR_Loop_Col

This write process is copying data from the buf array to the output array variable.

4-3. Continue the design analysis using the Function Call Graph capability.

4-3-1. From the Flow Navigator select **Function Call Graph**.

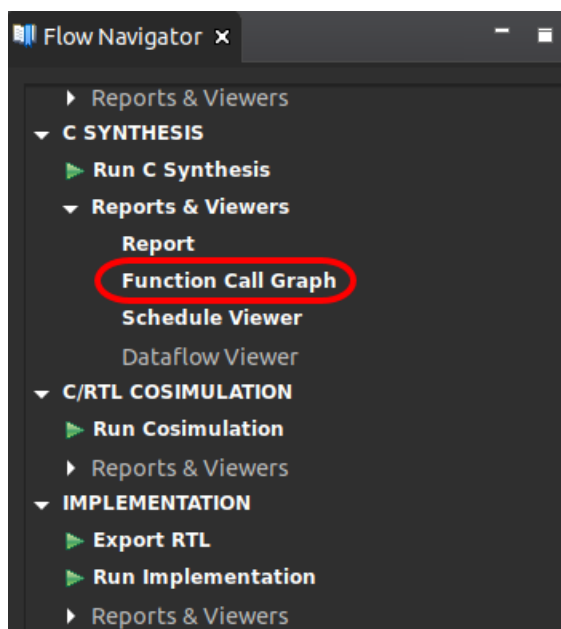


Figure 1-23: Opening the Function Call Graph Viewer

This opens the Function Call Graph Viewer and shows the throughput of the design in terms of latency and initiation interval (II) after C synthesis.

The default settings for the Function Call Graph Viewer has the Performance Metric set to **Synthesis** and Heat Map set to **NONE**.

4-3-2. Observe the latency and II values for each function and loop.

Hint: If the text in the blocks is not visible, zoom in on the block(s) of interest. When the zoom level is high enough to fit the text in the box, these details will appear.

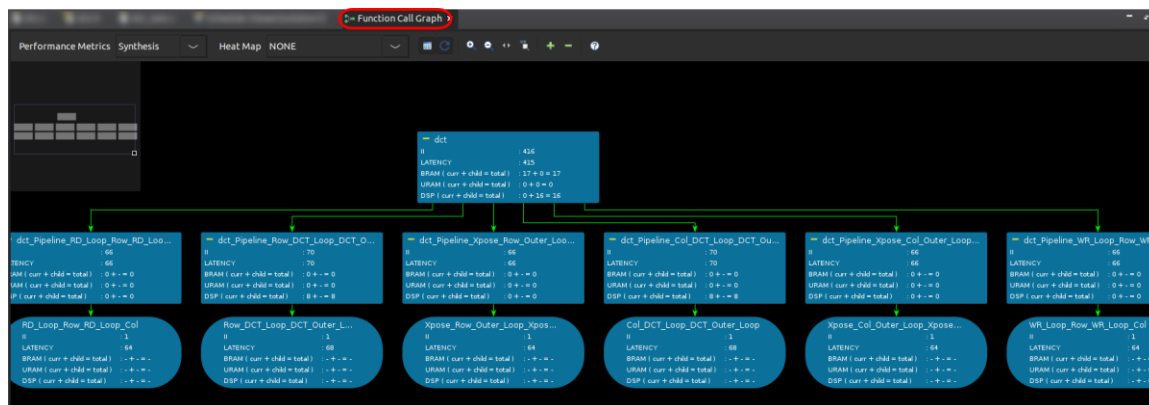


Figure 1-24: Output of Function Call Graph Viewer

The heat map feature highlights several metrics of interest:

- II (min, max, avg)
- Latency (min, max, avg)
- Stalling Time Percentage

The heat map uses color coding to highlight problematic modules with a color scale of red to green, where red indicates a high value (bad) of the metric (that is, the highest initiation interval or highest latency) while green (good) indicates a low value of the metric in question. The colors that are neither red nor green represent the range of values that are in between the highest and lowest values.

4-3-3. Click the **Heat Map** drop-down arrow and try changing the heat map options and observing the effects.

4-3-4. Close the window once done.

Performing C/RTL Co-simulation

Step 5

Now that you have performed high-level synthesis on the C design, you will perform RTL co-simulation on the generated RTL using the C test bench.

Co-simulation (or more accurately to what you are doing here, simulation verification) compares the behavior of the software (C/C++ code) with the behavior of the RTL code. The hope is that identical results are produced.

You will now run the C/RTL co-simulation using Verilog. This could also be done using VHDL; however, for the sake of time, only one flow is shown.

At the end of this process, you want to see that both simulations generate the same results.

5-1. Run co-simulation on the solution.

This will compare the simulation results of the C/C++ and RTL designs.

5-1-1. Select **Solution > Run C/RTL Cosimulation** to launch the co-simulation.

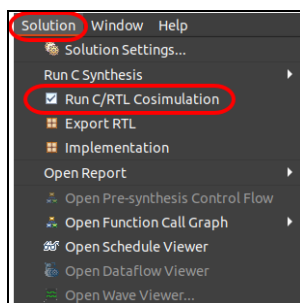


Figure 1-25: Launching from the Menu

Alternatively, from the Flow Navigator, expand **C/RTL Co-simulation** and click **Run Cosimulation**.

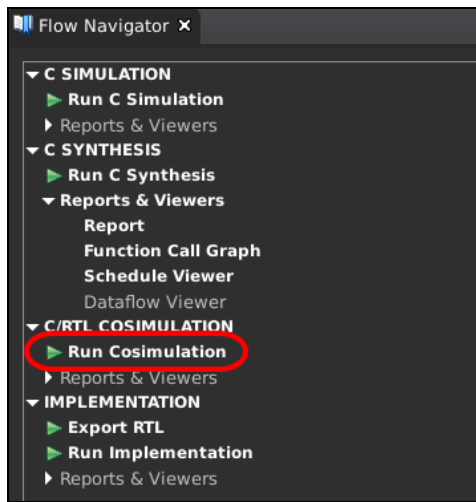


Figure 1-26: Launching Cosimulation from Flow Navigator

The Run C/RTL Co-simulation dialog box opens.

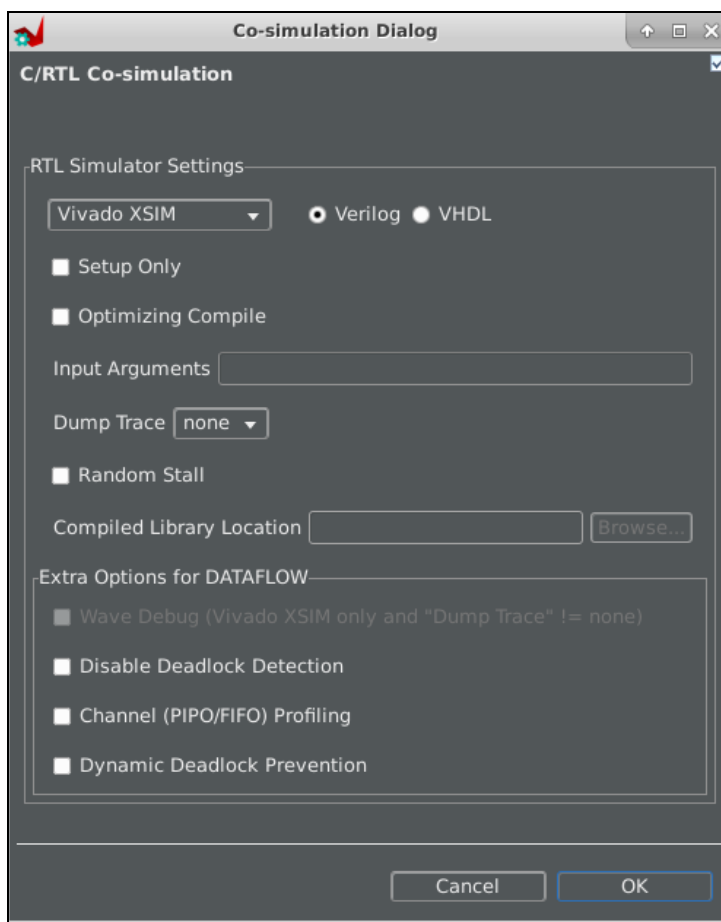


Figure 1-27: Co-simulation Dialog Box

The dialog box includes the following settings:

- Simulator: Choose from one of the supported HDL simulators in the Vivado Design Suite. The Vivado simulator (Vivado XSIM) is the default simulator. Licenses may be required for other simulators.
- Language: Specify Verilog or VHDL as the output language for simulation.
- Setup Only: Creates the required simulation files but does not run the simulation. The simulation executable can be run from a command shell at a later time.
- Optimizing Compile: Improve the runtime performance through enhanced optimization, if possible, which typically takes longer to compile.
- Input Arguments: Specify additional command-line arguments for the C test bench.
- Dump Trace: Specifies the level of trace file output written to the *sim/Verilog* or *sim/VHDL* directory of the current solution when the simulation executes. Options include:
 - all: Output all port and signal waveform data being saved to the trace file.
 - port: Output waveform trace data for the top-level ports only.
 - none: Do not output trace data.
- Random Stall: Applies a randomized stall for each data transmission.

5-1-2. Select **the default options (i.e. don't change anything)**.

5-1-3. Click **OK**.

The simulation log will be displayed in the Console pane.

Note: The Console pane shows the message "*Results are Good* ", which is generated by the test bench C code.

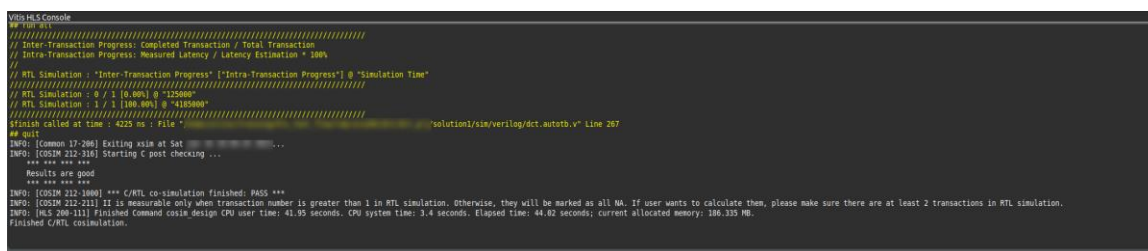
5-2. View the Cosimulation report.

The information generated by the Vitis HLS tool can be found in two places, both described here.

The first is the **Console** window, which reports not only the output produced by the code being simulated but all the simulation engine messages as well. The simulation log provides only a few simulation engine messages and the simulated code output.

- 5-2-1.** Double-click the **Console** tab to maximize it so that you can see more of the messages at one time.

You may need to scroll to view all the output produced by the cosimulation.



```

Vitis HLS Console
## Run Exit
=====
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
// RTL Simulation : 0 / 1 [0.00%] @ "0.00ns"
// RTL Simulation : 1 / 1 [100.00%] @ "418000ns"
// RTL Simulation : 2 / 2 [100.00%] @ "836000ns"
// RTL Simulation : 3 / 3 [100.00%] @ "1254000ns"
// RTL Simulation : 4 / 4 [100.00%] @ "1672000ns"
// RTL Simulation : 5 / 5 [100.00%] @ "2090000ns"
// RTL Simulation : 6 / 6 [100.00%] @ "2508000ns"
// RTL Simulation : 7 / 7 [100.00%] @ "2926000ns"
// RTL Simulation : 8 / 8 [100.00%] @ "3344000ns"
// RTL Simulation : 9 / 9 [100.00%] @ "3762000ns"
// RTL Simulation : 10 / 10 [100.00%] @ "4180000ns"
// RTL Simulation : 11 / 11 [100.00%] @ "4598000ns"
// RTL Simulation : 12 / 12 [100.00%] @ "5016000ns"
// RTL Simulation : 13 / 13 [100.00%] @ "5434000ns"
// RTL Simulation : 14 / 14 [100.00%] @ "5852000ns"
// RTL Simulation : 15 / 15 [100.00%] @ "6270000ns"
// RTL Simulation : 16 / 16 [100.00%] @ "6688000ns"
// RTL Simulation : 17 / 17 [100.00%] @ "7106000ns"
// RTL Simulation : 18 / 18 [100.00%] @ "7524000ns"
// RTL Simulation : 19 / 19 [100.00%] @ "7942000ns"
// RTL Simulation : 20 / 20 [100.00%] @ "8360000ns"
// RTL Simulation : 21 / 21 [100.00%] @ "8778000ns"
// RTL Simulation : 22 / 22 [100.00%] @ "9196000ns"
// RTL Simulation : 23 / 23 [100.00%] @ "9614000ns"
// RTL Simulation : 24 / 24 [100.00%] @ "10032000ns"
// RTL Simulation : 25 / 25 [100.00%] @ "10450000ns"
// RTL Simulation : 26 / 26 [100.00%] @ "10868000ns"
// RTL Simulation : 27 / 27 [100.00%] @ "11286000ns"
// RTL Simulation : 28 / 28 [100.00%] @ "11704000ns"
// RTL Simulation : 29 / 29 [100.00%] @ "12122000ns"
// RTL Simulation : 30 / 30 [100.00%] @ "12540000ns"
// RTL Simulation : 31 / 31 [100.00%] @ "12958000ns"
// RTL Simulation : 32 / 32 [100.00%] @ "13376000ns"
// RTL Simulation : 33 / 33 [100.00%] @ "13794000ns"
// RTL Simulation : 34 / 34 [100.00%] @ "14212000ns"
// RTL Simulation : 35 / 35 [100.00%] @ "14630000ns"
// RTL Simulation : 36 / 36 [100.00%] @ "15048000ns"
// RTL Simulation : 37 / 37 [100.00%] @ "15466000ns"
// RTL Simulation : 38 / 38 [100.00%] @ "15884000ns"
// RTL Simulation : 39 / 39 [100.00%] @ "16302000ns"
// RTL Simulation : 40 / 40 [100.00%] @ "16720000ns"
// RTL Simulation : 41 / 41 [100.00%] @ "17138000ns"
// RTL Simulation : 42 / 42 [100.00%] @ "17556000ns"
// RTL Simulation : 43 / 43 [100.00%] @ "17974000ns"
// RTL Simulation : 44 / 44 [100.00%] @ "18392000ns"
// RTL Simulation : 45 / 45 [100.00%] @ "18810000ns"
// RTL Simulation : 46 / 46 [100.00%] @ "19228000ns"
// RTL Simulation : 47 / 47 [100.00%] @ "19646000ns"
// RTL Simulation : 48 / 48 [100.00%] @ "20064000ns"
// RTL Simulation : 49 / 49 [100.00%] @ "20482000ns"
// RTL Simulation : 50 / 50 [100.00%] @ "20900000ns"
// RTL Simulation : 51 / 51 [100.00%] @ "21318000ns"
// RTL Simulation : 52 / 52 [100.00%] @ "21736000ns"
// RTL Simulation : 53 / 53 [100.00%] @ "22154000ns"
// RTL Simulation : 54 / 54 [100.00%] @ "22572000ns"
// RTL Simulation : 55 / 55 [100.00%] @ "22990000ns"
// RTL Simulation : 56 / 56 [100.00%] @ "23408000ns"
// RTL Simulation : 57 / 57 [100.00%] @ "23826000ns"
// RTL Simulation : 58 / 58 [100.00%] @ "24244000ns"
// RTL Simulation : 59 / 59 [100.00%] @ "24662000ns"
// RTL Simulation : 60 / 60 [100.00%] @ "25080000ns"
// RTL Simulation : 61 / 61 [100.00%] @ "25498000ns"
// RTL Simulation : 62 / 62 [100.00%] @ "25916000ns"
// RTL Simulation : 63 / 63 [100.00%] @ "26334000ns"
// RTL Simulation : 64 / 64 [100.00%] @ "26752000ns"
// RTL Simulation : 65 / 65 [100.00%] @ "27170000ns"
// RTL Simulation : 66 / 66 [100.00%] @ "27588000ns"
// RTL Simulation : 67 / 67 [100.00%] @ "28006000ns"
// RTL Simulation : 68 / 68 [100.00%] @ "28424000ns"
// RTL Simulation : 69 / 69 [100.00%] @ "28842000ns"
// RTL Simulation : 70 / 70 [100.00%] @ "29260000ns"
// RTL Simulation : 71 / 71 [100.00%] @ "29678000ns"
// RTL Simulation : 72 / 72 [100.00%] @ "30096000ns"
// RTL Simulation : 73 / 73 [100.00%] @ "30514000ns"
// RTL Simulation : 74 / 74 [100.00%] @ "30932000ns"
// RTL Simulation : 75 / 75 [100.00%] @ "31350000ns"
// RTL Simulation : 76 / 76 [100.00%] @ "31768000ns"
// RTL Simulation : 77 / 77 [100.00%] @ "32186000ns"
// RTL Simulation : 78 / 78 [100.00%] @ "32604000ns"
// RTL Simulation : 79 / 79 [100.00%] @ "33022000ns"
// RTL Simulation : 80 / 80 [100.00%] @ "33440000ns"
// RTL Simulation : 81 / 81 [100.00%] @ "33858000ns"
// RTL Simulation : 82 / 82 [100.00%] @ "34276000ns"
// RTL Simulation : 83 / 83 [100.00%] @ "34694000ns"
// RTL Simulation : 84 / 84 [100.00%] @ "35112000ns"
// RTL Simulation : 85 / 85 [100.00%] @ "35530000ns"
// RTL Simulation : 86 / 86 [100.00%] @ "35948000ns"
// RTL Simulation : 87 / 87 [100.00%] @ "36366000ns"
// RTL Simulation : 88 / 88 [100.00%] @ "36784000ns"
// RTL Simulation : 89 / 89 [100.00%] @ "37202000ns"
// RTL Simulation : 90 / 90 [100.00%] @ "37620000ns"
// RTL Simulation : 91 / 91 [100.00%] @ "38038000ns"
// RTL Simulation : 92 / 92 [100.00%] @ "38456000ns"
// RTL Simulation : 93 / 93 [100.00%] @ "38874000ns"
// RTL Simulation : 94 / 94 [100.00%] @ "39292000ns"
// RTL Simulation : 95 / 95 [100.00%] @ "39710000ns"
// RTL Simulation : 96 / 96 [100.00%] @ "40128000ns"
// RTL Simulation : 97 / 97 [100.00%] @ "40546000ns"
// RTL Simulation : 98 / 98 [100.00%] @ "40964000ns"
// RTL Simulation : 99 / 99 [100.00%] @ "41382000ns"
// RTL Simulation : 100 / 100 [100.00%] @ "41800000ns"
// RTL Simulation : 101 / 101 [100.00%] @ "42218000ns"
// RTL Simulation : 102 / 102 [100.00%] @ "42636000ns"
// RTL Simulation : 103 / 103 [100.00%] @ "43054000ns"
// RTL Simulation : 104 / 104 [100.00%] @ "43472000ns"
// RTL Simulation : 105 / 105 [100.00%] @ "43890000ns"
// RTL Simulation : 106 / 106 [100.00%] @ "44308000ns"
// RTL Simulation : 107 / 107 [100.00%] @ "44726000ns"
// RTL Simulation : 108 / 108 [100.00%] @ "45144000ns"
// RTL Simulation : 109 / 109 [100.00%] @ "45562000ns"
// RTL Simulation : 110 / 110 [100.00%] @ "45980000ns"
// RTL Simulation : 111 / 111 [100.00%] @ "46398000ns"
// RTL Simulation : 112 / 112 [100.00%] @ "46816000ns"
// RTL Simulation : 113 / 113 [100.00%] @ "47234000ns"
// RTL Simulation : 114 / 114 [100.00%] @ "47652000ns"
// RTL Simulation : 115 / 115 [100.00%] @ "48070000ns"
// RTL Simulation : 116 / 116 [100.00%] @ "48488000ns"
// RTL Simulation : 117 / 117 [100.00%] @ "48906000ns"
// RTL Simulation : 118 / 118 [100.00%] @ "49324000ns"
// RTL Simulation : 119 / 119 [100.00%] @ "49742000ns"
// RTL Simulation : 120 / 120 [100.00%] @ "50160000ns"
// RTL Simulation : 121 / 121 [100.00%] @ "50578000ns"
// RTL Simulation : 122 / 122 [100.00%] @ "50996000ns"
// RTL Simulation : 123 / 123 [100.00%] @ "51414000ns"
// RTL Simulation : 124 / 124 [100.00%] @ "51832000ns"
// RTL Simulation : 125 / 125 [100.00%] @ "52250000ns"
// RTL Simulation : 126 / 126 [100.00%] @ "52668000ns"
// RTL Simulation : 127 / 127 [100.00%] @ "53086000ns"
// RTL Simulation : 128 / 128 [100.00%] @ "53504000ns"
// RTL Simulation : 129 / 129 [100.00%] @ "53922000ns"
// RTL Simulation : 130 / 130 [100.00%] @ "54340000ns"
// RTL Simulation : 131 / 131 [100.00%] @ "54758000ns"
// RTL Simulation : 132 / 132 [100.00%] @ "55176000ns"
// RTL Simulation : 133 / 133 [100.00%] @ "55594000ns"
// RTL Simulation : 134 / 134 [100.00%] @ "56012000ns"
// RTL Simulation : 135 / 135 [100.00%] @ "56430000ns"
// RTL Simulation : 136 / 136 [100.00%] @ "56848000ns"
// RTL Simulation : 137 / 137 [100.00%] @ "57266000ns"
// RTL Simulation : 138 / 138 [100.00%] @ "57684000ns"
// RTL Simulation : 139 / 139 [100.00%] @ "58102000ns"
// RTL Simulation : 140 / 140 [100.00%] @ "58520000ns"
// RTL Simulation : 141 / 141 [100.00%] @ "58938000ns"
// RTL Simulation : 142 / 142 [100.00%] @ "59356000ns"
// RTL Simulation : 143 / 143 [100.00%] @ "59774000ns"
// RTL Simulation : 144 / 144 [100.00%] @ "60192000ns"
// RTL Simulation : 145 / 145 [100.00%] @ "60610000ns"
// RTL Simulation : 146 / 146 [100.00%] @ "61028000ns"
// RTL Simulation : 147 / 147 [100.00%] @ "61446000ns"
// RTL Simulation : 148 / 148 [100.00%] @ "61864000ns"
// RTL Simulation : 149 / 149 [100.00%] @ "62282000ns"
// RTL Simulation : 150 / 150 [100.00%] @ "62700000ns"
// RTL Simulation : 151 / 151 [100.00%] @ "63118000ns"
// RTL Simulation : 152 / 152 [100.00%] @ "63536000ns"
// RTL Simulation : 153 / 153 [100.00%] @ "63954000ns"
// RTL Simulation : 154 / 154 [100.00%] @ "64372000ns"
// RTL Simulation : 155 / 155 [100.00%] @ "64790000ns"
// RTL Simulation : 156 / 156 [100.00%] @ "65208000ns"
// RTL Simulation : 157 / 157 [100.00%] @ "65626000ns"
// RTL Simulation : 158 / 158 [100.00%] @ "66044000ns"
// RTL Simulation : 159 / 159 [100.00%] @ "66462000ns"
// RTL Simulation : 160 / 160 [100.00%] @ "66880000ns"
// RTL Simulation : 161 / 161 [100.00%] @ "67298000ns"
// RTL Simulation : 162 / 162 [100.00%] @ "67716000ns"
// RTL Simulation : 163 / 163 [100.00%] @ "68134000ns"
// RTL Simulation : 164 / 164 [100.00%] @ "68552000ns"
// RTL Simulation : 165 / 165 [100.00%] @ "68970000ns"
// RTL Simulation : 166 / 166 [100.00%] @ "69388000ns"
// RTL Simulation : 167 / 167 [100.00%] @ "69806000ns"
// RTL Simulation : 168 / 168 [100.00%] @ "70224000ns"
// RTL Simulation : 169 / 169 [100.00%] @ "70642000ns"
// RTL Simulation : 170 / 170 [100.00%] @ "71060000ns"
// RTL Simulation : 171 / 171 [100.00%] @ "71478000ns"
// RTL Simulation : 172 / 172 [100.00%] @ "71896000ns"
// RTL Simulation : 173 / 173 [100.00%] @ "72314000ns"
// RTL Simulation : 174 / 174 [100.00%] @ "72732000ns"
// RTL Simulation : 175 / 175 [100.00%] @ "73150000ns"
// RTL Simulation : 176 / 176 [100.00%] @ "73568000ns"
// RTL Simulation : 177 / 177 [100.00%] @ "73986000ns"
// RTL Simulation : 178 / 178 [100.00%] @ "74404000ns"
// RTL Simulation : 179 / 179 [100.00%] @ "74822000ns"
// RTL Simulation : 180 / 180 [100.00%] @ "75240000ns"
// RTL Simulation : 181 / 181 [100.00%] @ "75658000ns"
// RTL Simulation : 182 / 182 [100.00%] @ "76076000ns"
// RTL Simulation : 183 / 183 [100.00%] @ "76494000ns"
// RTL Simulation : 184 / 184 [100.00%] @ "76912000ns"
// RTL Simulation : 185 / 185 [100.00%] @ "77330000ns"
// RTL Simulation : 186 / 186 [100.00%] @ "77748000ns"
// RTL Simulation : 187 / 187 [100.00%] @ "78166000ns"
// RTL Simulation : 188 / 188 [100.00%] @ "78584000ns"
// RTL Simulation : 189 / 189 [100.00%] @ "79002000ns"
// RTL Simulation : 190 / 190 [100.00%] @ "79420000ns"
// RTL Simulation : 191 / 191 [100.00%] @ "79838000ns"
// RTL Simulation : 192 / 192 [100.00%] @ "80256000ns"
// RTL Simulation : 193 / 193 [100.00%] @ "80674000ns"
// RTL Simulation : 194 / 194 [100.00%] @ "81092000ns"
// RTL Simulation : 195 / 195 [100.00%] @ "81510000ns"
// RTL Simulation : 196 / 196 [100.00%] @ "81928000ns"
// RTL Simulation : 197 / 197 [100.00%] @ "82346000ns"
// RTL Simulation : 198 / 198 [100.00%] @ "82764000ns"
// RTL Simulation : 199 / 199 [100.00%] @ "83182000ns"
// RTL Simulation : 200 / 200 [100.00%] @ "83600000ns"
// RTL Simulation : 201 / 201 [100.00%] @ "84018000ns"
// RTL Simulation : 202 / 202 [100.00%] @ "84436000ns"
// RTL Simulation : 203 / 203 [100.00%] @ "84854000ns"
// RTL Simulation : 204 / 204 [100.00%] @ "85272000ns"
// RTL Simulation : 205 / 205 [100.00%] @ "85690000ns"
// RTL Simulation : 206 / 206 [100.00%] @ "86108000ns"
// RTL Simulation : 207 / 207 [100.00%] @ "86526000ns"
// RTL Simulation : 208 / 208 [100.00%] @ "86944000ns"
// RTL Simulation : 209 / 209 [100.00%] @ "87362000ns"
// RTL Simulation : 210 / 210 [100.00%] @ "87780000ns"
// RTL Simulation : 211 / 211 [100.00%] @ "88198000ns"
// RTL Simulation : 212 / 212 [100.00%] @ "88616000ns"
// RTL Simulation : 213 / 213 [100.00%] @ "89034000ns"
// RTL Simulation : 214 / 214 [100.00%] @ "89452000ns"
// RTL Simulation : 215 / 215 [100.00%] @ "89870000ns"
// RTL Simulation : 216 / 216 [100.00%] @ "90288000ns"
// RTL Simulation : 217 / 217 [100.00%] @ "90706000ns"
// RTL Simulation : 218 / 218 [100.00%] @ "91124000ns"
// RTL Simulation : 219 / 219 [100.00%] @ "91542000ns"
// RTL Simulation : 220 / 220 [100.00%] @ "91960000ns"
// RTL Simulation : 221 / 221 [100.00%] @ "92378000ns"
// RTL Simulation : 222 / 222 [100.00%] @ "92796000ns"
// RTL Simulation : 223 / 223 [100.00%] @ "93214000ns"
// RTL Simulation : 224 / 224 [100.00%] @ "93632000ns"
// RTL Simulation : 225 / 225 [100.00%] @ "94050000ns"
// RTL Simulation : 226 / 226 [100.00%] @ "94468000ns"
// RTL Simulation : 227 / 227 [100.00%] @ "94886000ns"
// RTL Simulation : 228 / 228 [100.00%] @ "95304000ns"
// RTL Simulation : 229 / 229 [100.00%] @ "95722000ns"
// RTL Simulation : 230 / 230 [100.00%] @ "96140000ns"
// RTL Simulation : 231 / 231 [100.00%] @ "96558000ns"
// RTL Simulation : 232 / 232 [100.00%] @ "96976000ns"
// RTL Simulation : 233 / 233 [100.00%] @ "97394000ns"
// RTL Simulation : 234 / 234 [100.00%] @ "97812000ns"
// RTL Simulation : 235 / 235 [100.00%] @ "98230000ns"
// RTL Simulation : 236 / 236 [100.00%] @ "98648000ns"
// RTL Simulation : 237 / 237 [100.00%] @ "99066000ns"
// RTL Simulation : 238 / 238 [100.00%] @ "99484000ns"
// RTL Simulation : 239 / 239 [100.00%] @ "99902000ns"
// RTL Simulation : 240 / 240 [100.00%] @ "100320000ns"
// RTL Simulation : 241 / 241 [100.00%] @ "100738000ns"
// RTL Simulation : 242 / 242 [100.00%] @ "101156000ns"
// RTL Simulation : 243 / 243 [100.00%] @ "101574000ns"
// RTL Simulation : 244 / 244 [100.00%] @ "101992000ns"
// RTL Simulation : 245 / 245 [100.00%] @ "102410000ns"
// RTL Simulation : 246 / 246 [100.00%] @ "102828000ns"
// RTL Simulation : 247 / 247 [100.00%] @ "103246000ns"
// RTL Simulation : 248 / 248 [100.00%] @ "103664000ns"
// RTL Simulation : 249 / 249 [100.00%] @ "104082000ns"
// RTL Simulation : 250 / 250 [100.00%] @ "104500000ns"
// RTL Simulation : 251 / 251 [100.00%] @ "104918000ns"
// RTL Simulation : 252 / 252 [100.00%] @ "105336000ns"
// RTL Simulation : 253 / 253 [100.00%] @ "105754000ns"
// RTL Simulation : 254 / 254 [100.00%] @ "106172000ns"
// RTL Simulation : 255 / 255 [100.00%] @ "106590000ns"
// RTL Simulation : 256 / 256 [100.00%] @ "107008000ns"
// RTL Simulation : 257 / 257 [100.00%] @ "107426000ns"
// RTL Simulation : 258 / 258 [100.00%] @ "107844000ns"
// RTL Simulation : 259 / 259 [100.00%] @ "108262000ns"
// RTL Simulation : 260 / 260 [100.00%] @ "108680000ns"
// RTL Simulation : 261 / 261 [100.00%] @ "109098000ns"
// RTL Simulation : 262 / 262 [100.00%] @ "109516000ns"
// RTL Simulation : 263 / 263 [100.00%] @ "109934000ns"
// RTL Simulation : 264 / 264 [100.00%] @ "110352000ns"
// RTL Simulation : 265 / 265 [100.00%] @ "110770000ns"
// RTL Simulation : 266 / 266 [100.00%] @ "111188000ns"
// RTL Simulation : 267 / 267 [100.00%] @ "111606000ns"
// RTL Simulation : 268 / 268 [100.00%] @ "112024000ns"
// RTL Simulation : 269 / 269 [100.00%] @ "112442000ns"
// RTL Simulation : 270 / 270 [100.00%] @ "112860000ns"
// RTL Simulation : 271 / 271 [100.00%] @ "113278000ns"
// RTL Simulation : 272 / 272 [100.00%] @ "113696000ns"
// RTL Simulation : 273 / 273 [100.00%] @ "114114000ns"
// RTL Simulation : 274 / 274 [100.00%] @ "114532000ns"
// RTL Simulation : 275 / 275 [100.00%] @ "114950000ns"
// RTL Simulation : 276 / 276 [100.00%] @ "115368000ns"
// RTL Simulation : 277 / 277 [100.00%] @ "115786000ns"
// RTL Simulation : 278 / 278 [100.00%] @ "116204000ns"
// RTL Simulation : 279 / 279 [100.00%] @ "116622000ns"
// RTL Simulation : 280 / 280 [100.00%] @ "117040000ns"
// RTL Simulation : 281 / 281 [100.00%] @ "117458000ns"
// RTL Simulation : 282 / 282 [100.00%] @ "117876000ns"
// RTL Simulation : 283 / 283 [100.00%] @ "118294000ns"
// RTL Simulation : 284 / 284 [100.00%] @ "118712000ns"
// RTL Simulation : 285 / 285 [100.00%] @ "119130000ns"
// RTL Simulation : 286 / 286 [100.00%] @ "119548000ns"
// RTL Simulation : 287 / 287 [100.00%] @ "119966000ns"
// RTL Simulation : 288 / 288 [100.00%] @ "120384000ns"
// RTL Simulation : 289 / 289 [100.00%] @ "120802000ns"
// RTL Simulation : 290 / 290 [100.00%] @ "121220000ns"
// RTL Simulation : 291 / 291 [100.00%] @ "121638000ns"
// RTL Simulation : 292 / 292 [100.00%] @ "122056000ns"
// RTL Simulation : 293 / 293 [100.00%] @ "122474000ns"
// RTL Simulation : 294 / 294 [100.00%] @ "122892000ns"
// RTL Simulation : 295 / 295 [100.00%] @ "123310000ns"
// RTL Simulation : 296 / 296 [100.00%] @ "123728000ns"
// RTL Simulation : 297 / 297 [100.00%] @ "124146000ns"
// RTL Simulation : 298 / 298 [100.00%] @ "124564000ns"
// RTL Simulation : 299 / 299 [100.00%] @ "124982000ns"
// RTL Simulation : 300 / 300 [100.00%] @ "125400000ns"
// RTL Simulation : 301 / 301 [100.00%] @ "125818000ns"
// RTL Simulation : 302 / 302 [100.00%] @ "126236000ns"
// RTL Simulation : 303 / 303 [100.00%] @ "126654000ns"
// RTL Simulation : 304 / 304 [100.00%] @ "127072000ns"
// RTL Simulation : 305 / 305 [100.00%] @ "127490000ns"
// RTL Simulation : 306 / 306 [100.00%] @ "127908000ns"
// RTL Simulation : 307 / 307 [100.00%] @ "128326000ns"
// RTL Simulation : 308 / 308 [100.00%] @ "128744000ns"
// RTL Simulation : 309 / 309 [100.00%] @ "129162000ns"
// RTL Simulation : 310 / 310 [100.00%] @ "129580000ns"
// RTL Simulation : 311 / 311 [100.00%] @ "129998000ns"
// RTL Simulation : 312 / 312 [100.00%] @ "130416000ns"
// RTL Simulation : 313 / 313 [100.00%] @ "130834000ns"
// RTL Simulation : 314 / 314 [100.00%] @ "131252000ns"
// RTL Simulation : 315 / 315 [100.00%] @ "131670000ns"
// RTL Simulation : 316 / 316 [100.00%] @ "132088000ns"
// RTL Simulation : 317 / 317 [100.00%] @ "132506000ns"
// RTL Simulation : 318 / 318 [100.00%] @ "132924000ns"
// RTL Simulation : 319 / 319 [100.00%] @ "133342000ns"
// RTL Simulation : 320 / 320 [100.00%] @ "133760000ns"
// RTL Simulation : 321 / 321 [100.00%] @ "134178000ns"
// RTL Simulation : 322 / 322 [100.00%] @ "134596000ns"
// RTL Simulation : 323 / 323 [100.00%] @ "135014000ns"
// RTL Simulation : 324 / 324 [100.00%] @ "135432000ns"
// RTL Simulation : 325 / 325 [100.00%] @ "135850000ns"
// RTL Simulation : 326 / 326 [100.00%] @ "136268000ns"
// RTL Simulation : 327 / 327 [100.00%] @ "136686000ns"
// RTL Simulation : 328 / 328 [100.00%] @ "137104000ns"
// RTL Simulation : 329 / 329 [100.00%] @ "137522000ns"
// RTL Simulation : 330 / 330 [100.00%] @ "137940000ns"
// RTL Simulation : 331 / 331 [100.00%] @ "138358000ns"
// RTL Simulation : 332 / 332 [100.00%] @ "138776000ns"
// RTL Simulation : 333 / 333 [100.00%] @ "139194000ns"
// RTL Simulation : 334 / 334 [100.00%] @ "139612000ns"
// RTL Simulation : 335 / 335 [100.00%] @ "140030000ns"
// RTL Simulation : 336 / 336 [100.00%] @ "140448000ns"
// RTL Simulation : 337 / 337 [100.00%] @ "140866000ns"
// RTL Simulation : 338 / 338 [100.00%] @ "141284000ns"
// RTL Simulation : 339 / 339 [100.00%] @ "141702000ns"
// RTL Simulation : 340 / 340 [100.00%] @ "142120000ns"
// RTL Simulation : 341 / 341 [100.00%] @ "142538000ns"
// RTL Simulation : 
```

5-2-2. Using the Explorer pane, expand **dct_prj** > **solution1** > **sim** > **report**.

5-2-3. Double-click the log file to open it in the editor pane.

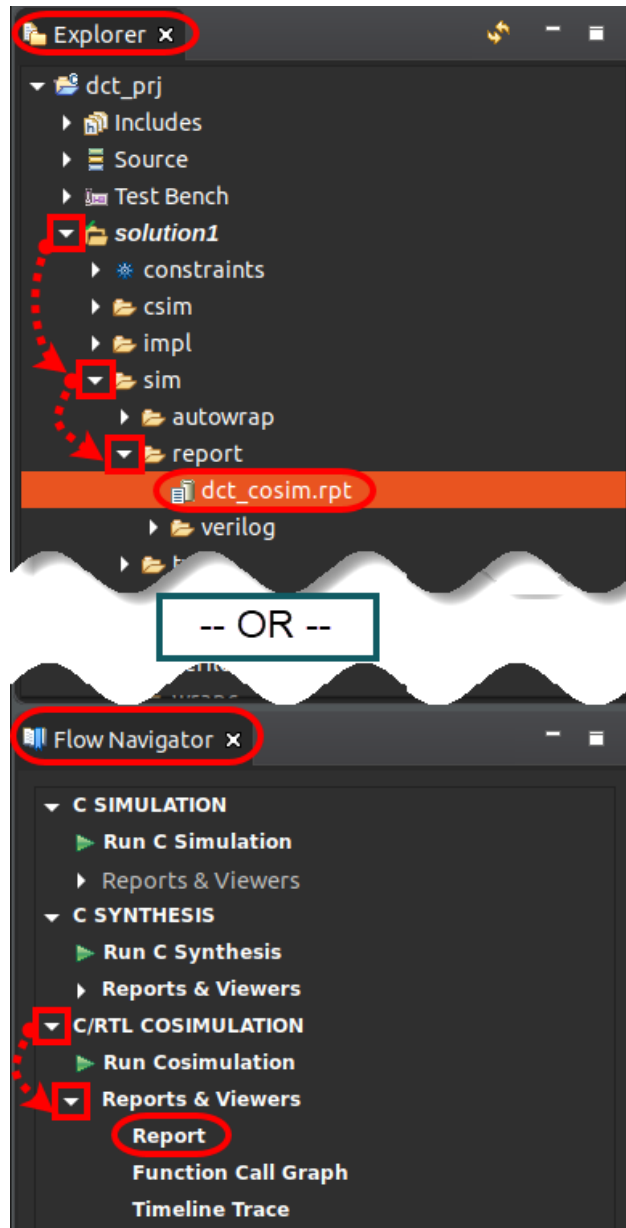


Figure 1-29: Locating the Co-simulation Report File

Alternatively, the report can be found using the Flow Navigator tab under **C/RTL COSIMULATION** > **Reports & Viewers** > **Report**

The Cosimulation Report in HTML format will be displayed in the main viewing area. You can also open the cosimulation report from Flow Navigator by expanding the **Reports & Viewers** section under the C/RTL COSIMULATION section.

Cosimulation Report for 'dct'

General Information

Date:

Version:

Project: dct.prj

Status: Pass

Solution:

Product family:

Target device:

solution1 (Vivado IP Flow Target)

Cosim Options

Tool: Vivado XSIM

RTL: Verilog

Performance Estimates

Modules	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
▼ dct				403	403	403
▶ dct_Pipeline_RD_Loop_Row_RD_Loop_Col				64	64	64
▶ dct_Pipeline_Row_DCT_Loop_DCT_Outer_Loop				68	68	68
▶ dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop				64	64	64
▶ dct_Pipeline_Col_DCT_Loop_DCT_Outer_Loop				68	68	68
▶ dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop				64	64	64
▶ dct_Pipeline_WR_Loop_Row_WR_Loop_Col				64	64	64

Figure 1-30: Cosimulation Report – HTML

You can quickly verify the cosimulation status here.

This process will take a few minutes to complete. After C/RTL cosimulation has been completed, the Cosimulation report will be accessible in the Information pane, including the latency information.

Turning the Design into an IP Core

Step 6

Now that you have successfully implemented and validated the design, you will now convert it to re-usable IP.

6-1. Export the RTL design as IP.

6-1-1. Select **Solution** > **Export RTL** to open the Export Wizard.

Alternatively, from the **Flow Navigator** > **IMPLEMENTATION**, click **Export RTL**.

6-1-2. Select **Vivado IP (.zip)** from the Export Format drop-down list as this defines a compatible format for the IP to be usable from within the Vivado tool.

Alternatively, selecting **Vitis Kernel (.xo)** produces an XO file that can be used for linking by the Vitis compiler in the application acceleration development flow.

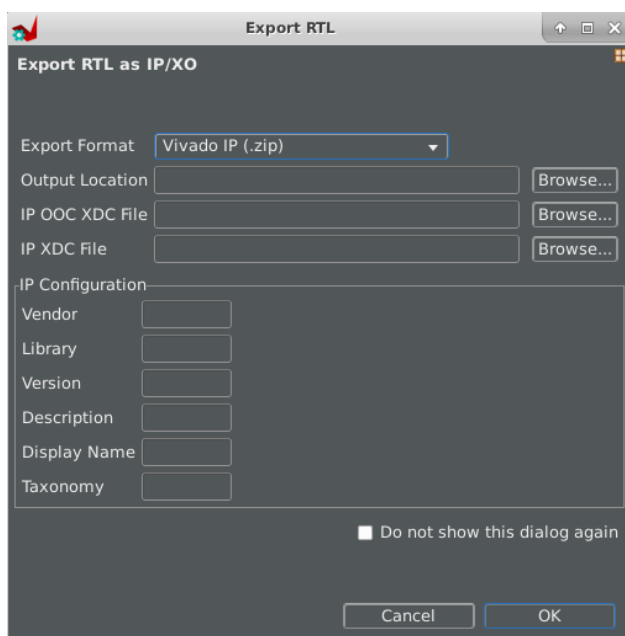


Figure 1-31: Export RTL Dialog Box

Note the following fields:

- **Output Location:** Lets you specify the path and file name for the exported RTL design. By default, the zipped design is placed under the project's solution directory under impl.
- **IP OOC XDC File:** Specifies an XDC file to be used for the RTL IP for out-of-context (OOC) synthesis.
- **IP XDC File:** Lets you specify an XDC file for use during Vivado IDE place and route.
- **IP Configuration:** Provides information about the IP, such as the vendor, library, version, and description of the IP.

6-1-3. Leave the settings at their defaults.

The Vitis HLS tool is limited in terms of the estimations it can provide about the RTL design that it generates. It can project resource utilization and timing of the final result, but these are just projections.

More accurate values are generated by running Vivado synthesis and place and route on the generated RTL design and then reviewing the timing and resource utilization in the Vivado Design Suite.

6-1-4. Click **OK** in the Export RTL dialog box.

Once the RTL is exported successfully, you can move to the implementation step.

6-2. Implement the design.

6-2-1. Select **Solution > Implementation** to open the Run Implementation dialog box.

Alternatively, from the Flow Navigator, click **Run Implementation**.

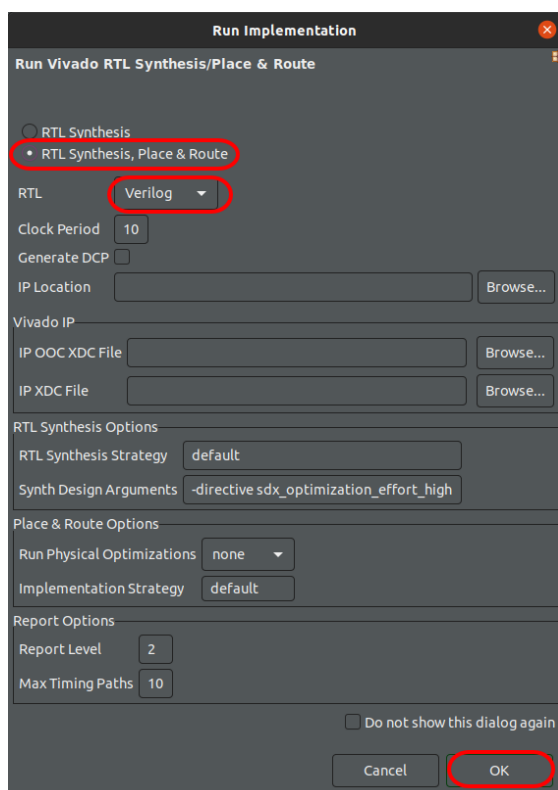


Figure 1-32: Run Implementation Dialog Box

This dialog box presents the choice of running RTL synthesis or RTL synthesis, place and route. The dialog box is largely unchanged with either selection except that the place-and-route options appear at the bottom.

- **RTL:** Generates the RTL in Verilog or VHDL form.
- **Clock Period:** Specifies the clock period, which is defined by the active solution by default.

- **Generate DCP:** Generates a DCP file for the synthesized or implemented design.
- **DCP Location:** If the previous option is selected, then this option specifies the location to write the generated DCP file.
- **IP Location:** Specifies the location of the IP generated.
- **IP OOC XDC File:** Specifies an XDC file to be used for the RTL IP for out-of-context (OOC) synthesis.
- **IP XDC File:** Let you specify an XDC file for use during Vivado place and route.
- **RTL Synthesis Strategy:** Specifies the strategy to employ in the synthesis run.
- **Synth Design Arguments:** Specifies options for the `synth_design` command.
- **Report Level:** Defines the report level generated during synthesis or implementation.
- **Max Timing Paths:** Specifies the number of timing paths to extract from the Timing Summary report. The worst-case paths are returned as defined by the specified value.
- **Run Physical Optimizations:** Specifies the physical optimization to run. Choices include: none, place, route, and all.
- **Implementation Strategy:** Specifies the strategy to employ in the implementation run.

6-2-2. Select the **RTL Synthesis, Place & Route** option.

6-2-3. Make sure that **Verilog** is selected as the RTL to be evaluated.

These options will enable Vivado RTL synthesis and implementation to be performed on the generated IP. Implementation is run to evaluate and provide confidence that the RTL will meet its estimated timing and area goals and that these results are not included as part of the exported package.

6-2-4. Click **OK** in the Run Implementation dialog box.

You can observe the progress in the Console tab. This may take 4-5 minutes to complete. Once implementation completes, the Implementation report will open in the Information pane.

Observe the "**Timing met**" message in the report in the Final Timing section. This shows that the final timing of the implemented design has been achieved.

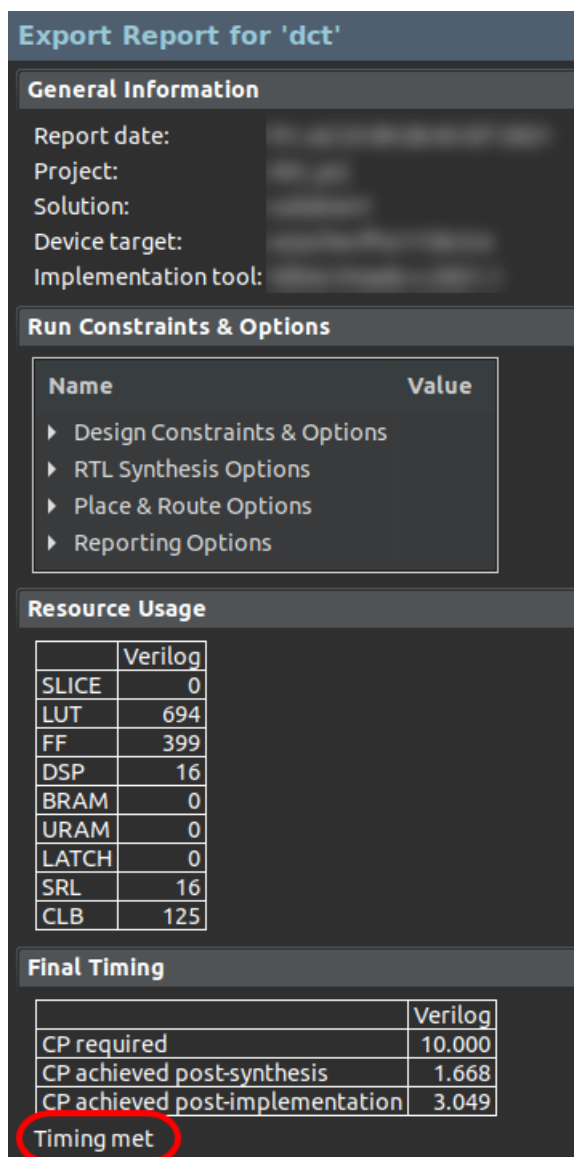


Figure 1-33: Export Report - Timing Met

6-2-5. Go to the **Fail Fast** section of the Implementation report.

The Fail Fast reports that the Vivado Design Suite provides can guide your investigation into specific issues encountered by the tool. In the Fail Fast report, you should look into anything with the status of REVIEW to improve the implementation and timing closure.

6-2-6. Go to the **Timing Paths** section of the Implementation report.

The Timing Paths reports show the timing-critical paths that result in the worst slack for the design. By default, the tool will show the top 10 worst negative slack paths.

Each path in the table has detailed information that shows the combination path between one flip-flop to another. Breaking these long combinational paths will be required to address the timing issues.

Fail Fast

Created on [redacted]
Design Summary
impl_1

Criteria	Guideline	Actual	Status
LUT	70%	0.30%	OK
FD	50%	0.09%	OK
LUTRAM+SRL	25%	0.36%	OK
CARRY8	25%	0.03%	OK
MUXF7	15%	0.00%	OK
DSP	80%	0.93%	OK
RAMB/FIFO	80%	0.00%	OK
URAM	80%	0.00%	OK
DSP+RAMB+URAM (Avg)	70%	0.93%	OK
BUFGCE* + BUFGCTRL	24	0	OK
DONT_TOUCH (cells/nets)	0	0	OK
MARK_DEBUG (nets)	0	0	OK
Control Sets	4320	30	OK
Average Fanout for modules > 100k cells	4	0	OK
Non-FD high fanout nets > 10k loads	0	0	OK
TIMING-6 (No common primary clock between related clocks)	0	0	OK
TIMING-7 (No common node between related clocks)	0	0	OK
TIMING-8 (No common period between related clocks)	0	0	OK
TIMING-14 (LUT on the clock tree)	0	0	OK
TIMING-35 (No common node in paths with the same clock)	0	0	OK
Number of paths above max LUT budgeting (0.300ns)	0	0	OK
Number of paths above max Net budgeting (0.208ns)	0	0	OK

Timing Paths

Worst Negative Slack: 6.575ns(met)
Total Negative Slack: 0.000ns(met)
Max levels: 3
Max fanout: 77
Full Timing Report: [verilog/report/dct_timing_routed.rpt](#)

Name	Value
▶ Path 1	slack=6.575ns(met) levels=3 fanout=63
▶ Path 2	slack=6.801ns(met) levels=3 fanout=63
▶ Path 3	slack=6.944ns(met) levels=2 fanout=77
▶ Path 4	slack=7.015ns(met) levels=2 fanout=77
▶ Path 5	slack=7.060ns(met) levels=3 fanout=63

Figure 1-34: Fail Fast & Timing Paths Sections of the Implementation Report

Note: Your values may slightly vary depending upon your system.

6-2-7. Select **File > Exit** to close the Vitis HLS tool.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/hls_tool_flow` directory.

6-3. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

6-3-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/hls_tool_flow`.

6-3-2. Select `hls_tool_flow`.

6-3-3. Press <**Delete**>.

-- OR --

Using the command line:

6-3-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

6-3-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/hls_tool_flow`

[Linux users]: `rm -rf $TRAINING_PATH/hls_tool_flow`

Summary

You just created a piece of hardware IP from a C program following the basic development flow, including simulating, synthesizing, co-simulating, and exporting the design as an IP.

You examined some of the generated reports and determined how the design was implemented. Other labs will expand on what you have learned here.

Answers

1. Write down the following details from the Synthesis report:

ZCU104:

Estimated clock frequency:	3.59
Worst case latency:	415
Number of BRAM_18K:	17
Number of DSP used:	16
Number of FFs used:	671
Number of LUTs used:	1992

VCK190:

Estimated clock frequency:	3.155
Worst case latency:	415
Number of BRAM_18K:	0
Number of DSP used:	16
Number of FFs used:	632
Number of LUTs used:	1209

2. What does the purple squiggly line represent?

This line represents the chronological flow through the various blocks in the hardware.

Each heavy vertical line indicates a clock stage enumerated at the top of the column. Everything happening between the solid vertical lines and the next solid line to the right illustrates all the events that take place during one clock cycle. The dotted vertical line represents the clock uncertainty.

The figure below focuses on the very first clock cycle indicated by the 0 and 1 columns.

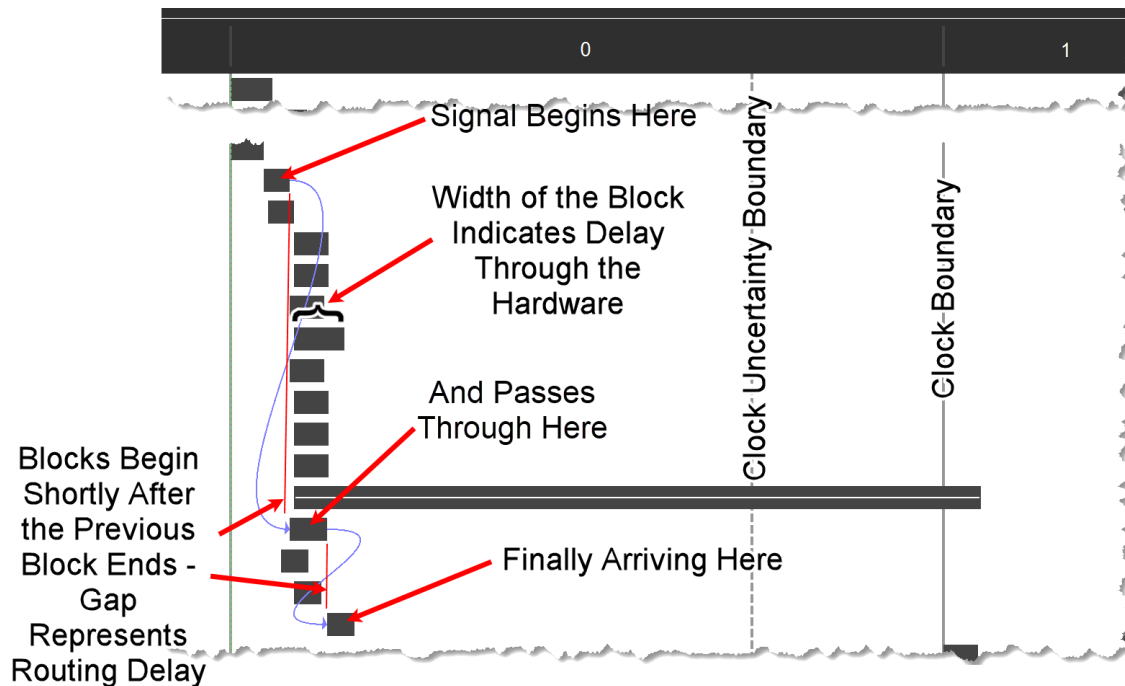


Figure 1-35: Detailed View of the Signal Flow Using the Schedule Viewer

If you were to zoom in on this clock cycle, you would notice that the little blocks are not perfectly aligned. The width of the block represents the delay through this hardware element, and the skew from the end of one block to the beginning of the next represents the delay due to routing (which is approximated at this stage of development).

Note that the color of the squiggly line is the same color as the highlighted line in the source code. This indicates that this "add" operation somehow relates to the RTL implementation of the for loop—most likely the column increment (`c++`).