Vitis HLS Tool Flow

2021.1

Abstract

This lab introduces how to perform basic actions using the Vitis™ High-Level Synthesis (HLS) tool design flow.

This lab should take approximately 45 minutes.

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

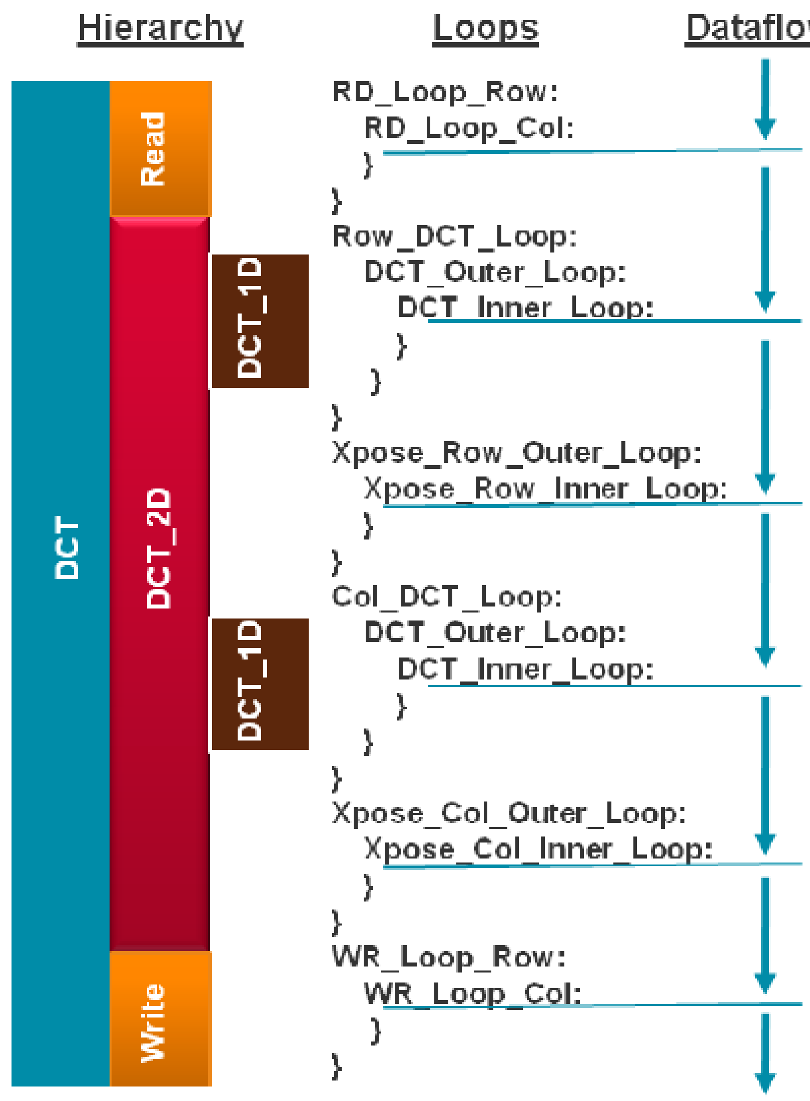
This lab provides an introduction to the major features of the Vitis™ High-Level Synthesis (HLS) tool GUI flow. You will use the Vitis HLS tool in GUI mode to create a project. You will also simulate, synthesize, and implement the provided design.

In this lab, you will be using a C design to implement a discrete cosine transformation (DCT). The function implements 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 at 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.



```
RD_Loop_Row:
   RD_Loop_Col:
   }
}
Row_DCT_Loop:
   DCT_Outer_Loop:
      DCT_Inner_Loop:
      }
   }
}
Xpose_Row_Outer_Loop:
   Xpose_Row_Inner_Loop:
   }
}
Col_DCT_Loop:
   DCT_Outer_Loop:
      DCT_Inner_Loop:
      }
   }
}
Xpose_Col_Outer_Loop:
   Xpose_Col_Inner_Loop:
   }
}
WR_Loop_Row:
   WR_Loop_Col:
   }
}
```

Figure 1-1: Design Hierarchy and Dataflow

Understanding the Lab Environment

The labs and demos provided can be successfully executed in the Windows environment or a native Linux environment as well.

Note: The working folder for the lab1_z0 is C:\Xilinx_trn\HLS2022\lab1_z0 in the Windows environment.

General Flow

$$\begin{tikzcd}
\text{Step 1: Creating a Vitis HLS Tool Project} \arrow[r] & \text{Step 2: Running C Simulation} \arrow[r] & \text{Step 3: Synthesizing the Design} \arrow[r] & \text{Step 4: Analyzing the Design} \arrow[r] & \text{Step 5: Performing Co-simulation}
\end{tikzcd}$$

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 a number of ways to launch the Vitis HLS tool. The two most popular mechanisms are shown here.

1-1. Launch the Vitis HLS tool.

1-1-1. **[Windows 10 users:]** Select **Start** > **Xilinx Design Tools** > **Vitis HLS 2021.1**.

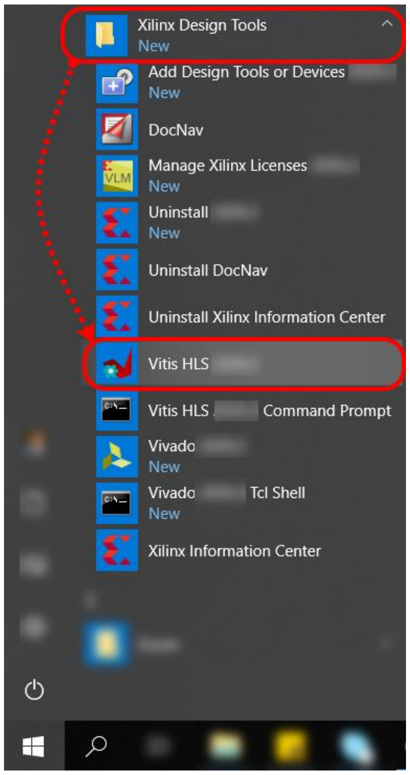


Figure 1-2: Launching the Vitis HLS Tool

You can also double-click the **Vitis HLS** shortcut icon ( ) from the Windows or Linux desktop or taskbar.

[Linux users]: Click the **Vitis HLS** ( ) icon from the taskbar to launch the tool.

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.

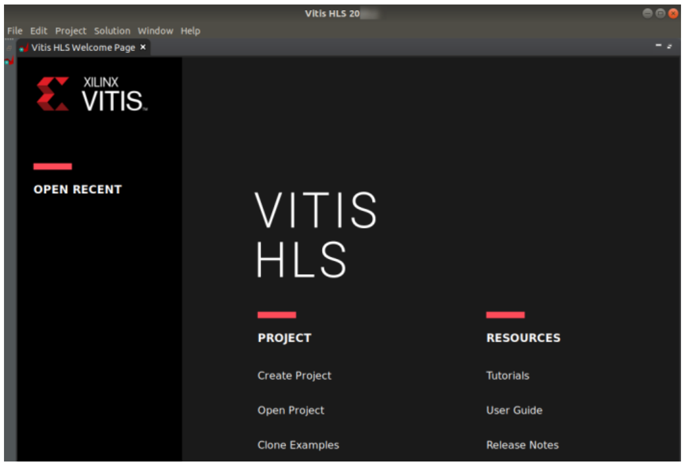


Figure 1-3: Vitis HLS Welcome Page

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.

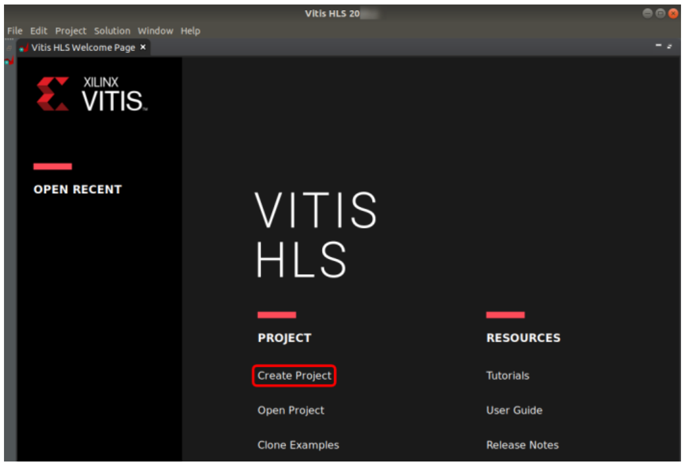


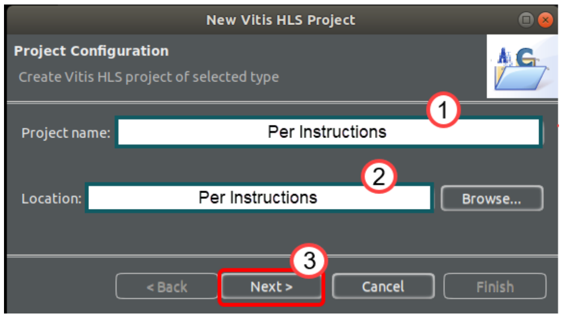
Figure 1-4: Creating a New Vitis HLS Tool Project

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. Enter **C:\Xilinx_trn\HLS2022\lab1_z0** in the Location field (2).

You can also browse to the desired path.



New Vitis HLS Project

Project Configuration

Create Vitis HLS project of selected type

Project name: Per Instructions

Location: Per Instructions

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

1-3. The Add/Remove Files dialog box opens. Here you will be invited to add existing files or create new sources.

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

1-3-2. Browse to **C:\Xilinx_trn\HLS2022\lab1_z0\source**.

1-3-3. Select **dct.c**.

The Vitis HLS tool automatically adds the working directory (project directory) and any directory that contains C files added to the project to the search path. Hence, header files that reside in these directories are automatically included in the project (no need to explicitly specify them). You must specify the path to all other header files (if any) by clicking **Edit CFLAGS**.

1-3-4. Click **OK** to add these files.

Note: If do not have existing files at this moment and you want to create new ones, click **New File**.

Note also that you can add compiler directives specific to each entry at this point.

1-3-5. Click **Browse** next to the Top Function field.

The Select Top function dialog box opens, which lists all the functions available from the specified source files.

1-3-6. Select **dct (dct.c)** from the list and click **OK**.

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

In any C program, the top-level function is called main(). In the Vitis HLS tool design flow, you can specify 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.
- Any sub-functions in the hierarchy under the top-level function for synthesis are also synthesized.
- If you want to synthesize functions that are not in the hierarchy under the top-level function for synthesis, you must merge the functions into a single top-level function for synthesis.

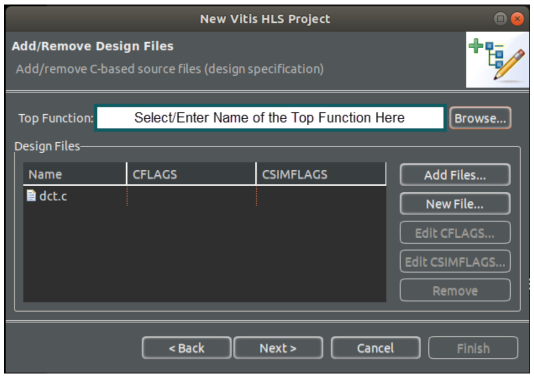


Figure 1-5: Adding Files to a New Vitis HLS Project

1-3-7. Click **Next**.

1-4. Add any existing test bench files.

If you have (or want) any test bench files, they can be entered here. Sometimes the test bench is built into the synthesizable file.

1-4-1. Click **Add Files**.

1-4-2. Navigate to **C:\Xilinx_trn\HLS2022\lab1_z0\source**.

1-4-3. Select **dct_test.c, in.dat, out.golden.dat**.

1-4-4. Click **Open** to add these files.

1-4-5. Click **Next**.

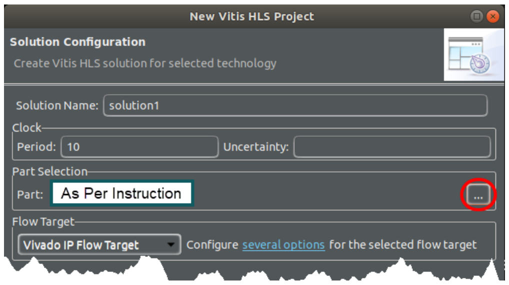
1-5. Finally, it is time to specify some of the physical parameters of the design.

By default, solution1 is populated in the Solution Name field. No changes are required.

1-5-1. Set the clock period to **10**.

You can leave the Uncertainty field blank.

1-5-2. Click the **Browse (. . .)** icon to select a part or board.



New Vitis HLS Project

Solution Configuration

Create Vitis HLS solution for selected technology

Solution Name: solution1

Clock

Period: 10 Uncertainty:

Part Selection

Part: As Per Instruction ...

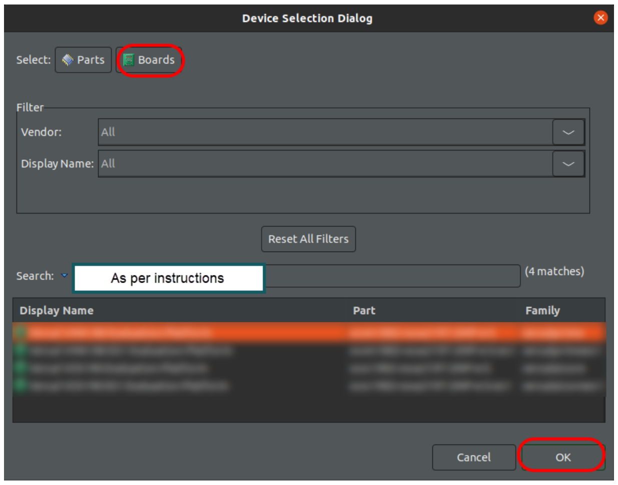
Flow Target

Vivado IP Flow Target Configure several options for the selected flow target

Figure 1-6: Locating the Board Browse Button

1-5-3. Click **Boards** as shown below.

1-5-4. Enter **ZYNQ UltraScale+** in the Search field.



Device Selection Dialog

Select: Parts Boards

Filter

Vendor: All

Display Name: All

Reset All Filters

Search: As per instructions (4 matches)

Display Name	Part	Family

Cancel OK

Figure 1-7: Filtering to Quickly Locate Target Platforms

1-5-5. Select **ZYNQ UltraScale+ ZCU104 Evaluation Board** from the list.

1-5-6. Click **OK** to select the board.

1-5-7. Make sure that **Vivado IP Flow Target** is selected from the drop-down list under **Flow Target** field.

1-5-8. Click **Finish**.

You will see the created project in the Explorer tab.

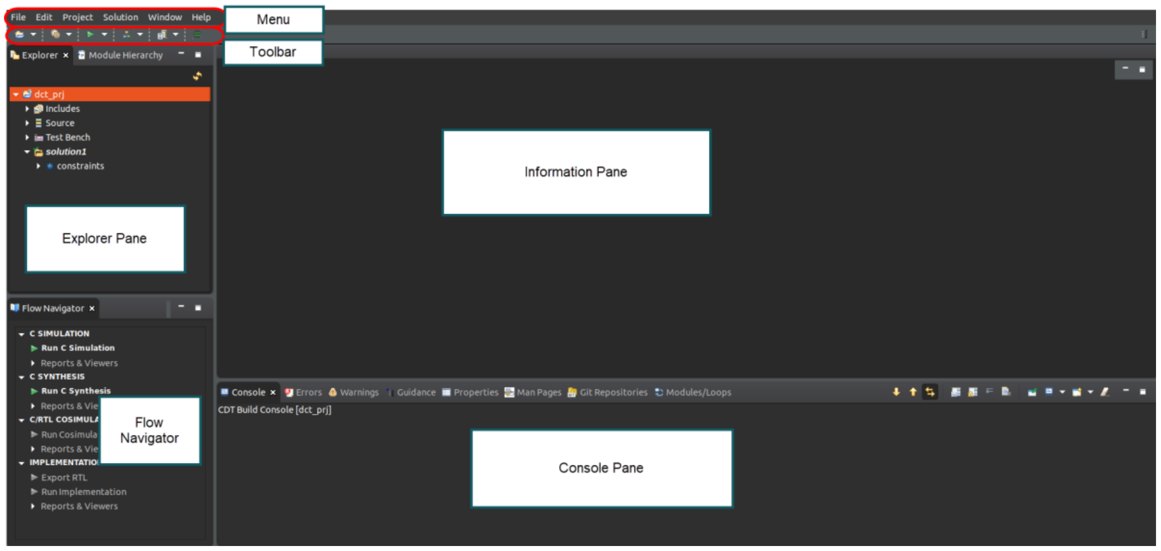


Figure 1-8: Vitis HLS with Newly Created Project

The Vitis HLS tool GUI consists of various panes to proceed with the development work.

- In the upper left-hand side, the Explorer view lets you navigate through the project hierarchy. A similar hierarchy exists in the project directory on the disk. You can expand various sub-folders to see the entries under each sub-folder. The Source folder consists of source files associated with the project, and the Test Bench folder consists of test bench files associated with the project.
- In the center, the Information area displays report summaries and open files. Files can be opened by double-clicking them in the Explorer view.
- At the bottom, the Console view displays the output when the Vitis HLS tool is running synthesis or simulation.
- In the lower left-hand side, the Flow Navigator view provides access to commands and processes to take your source code through simulation, synthesis, and exported output.
- Though not displayed by default, when source code is opened in the Information area, 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

After creating the project, the next step is to validate that the C function is correct before proceeding with synthesizing the design. In this step, you will validate the design by using the provided self-checking C test bench.

2-1. Become familiar with the provided source and test bench files.

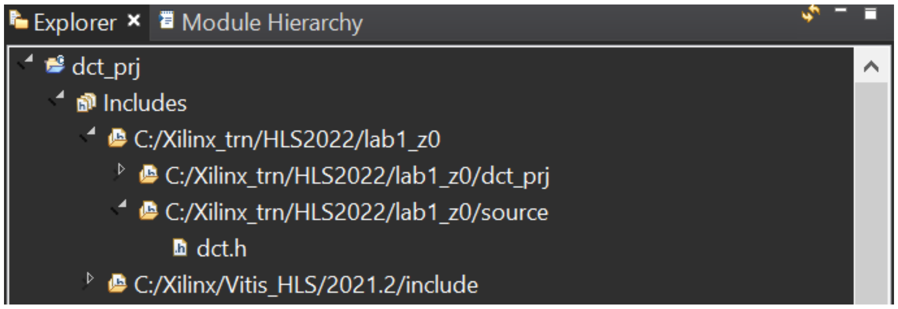
2-1-1. Expand the **Source** folder in the Explorer pane.

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 and data structures.

2-1-4. Expand the **Includes** > **C:\Xilinx_trn\HLS2022\lab1_z0\source** folder in the Explorer pane.


Explorer
Module Hierarchy
dct_prj
Includes
C:/Xilinx_trn/HLS2022/lab1_z0
C:/Xilinx_trn/HLS2022/lab1_z0/dct_prj
C:/Xilinx_trn/HLS2022/lab1_z0/source
dct.h
C:/Xilinx/Vitis_HLS/2021.2/include

2-1-5. Double-click **dct.h** to open the header file.

2-1-6. Review the contents of the header file.

2-1-7. Expand the **Test Bench** folder in the Explorer pane.

2-1-8. 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**.

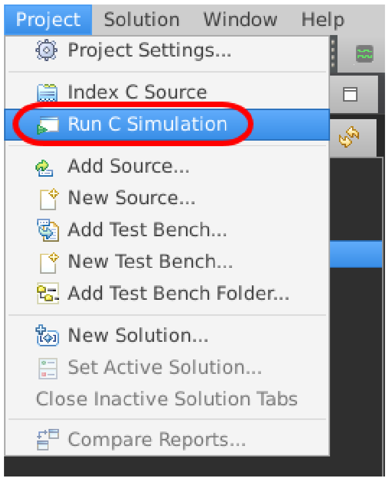


Figure 1-9: Launching the C Simulation

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

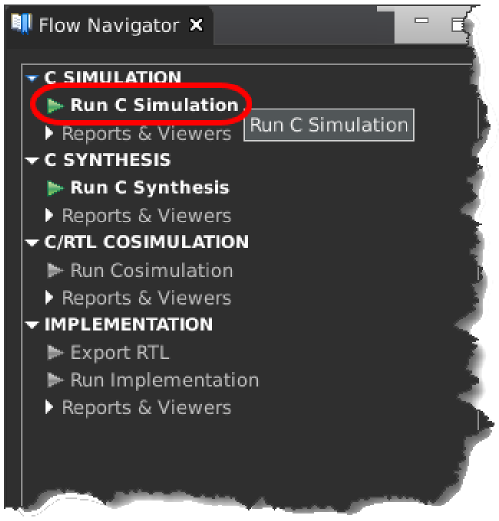
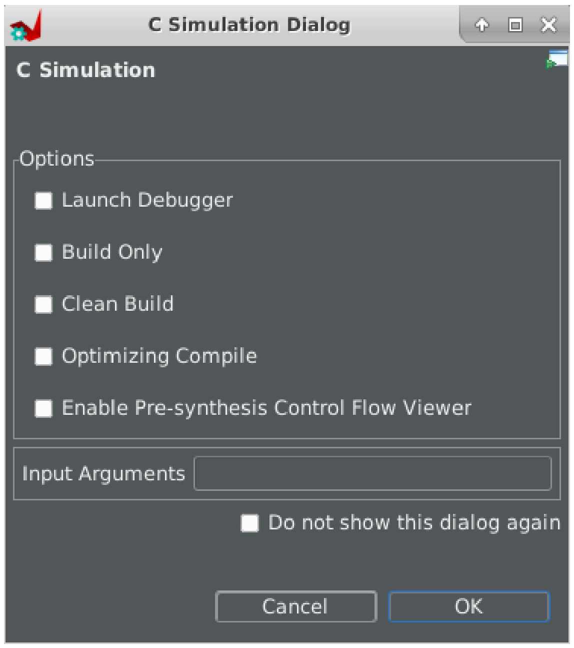


Figure 1-10: Launching the C Simulation Using the Flow Navigator

The Run C Simulation dialog box opens.



C Simulation Dialog

C Simulation

Options

- ☐ Launch Debugger
- ☐ Build Only
- ☐ Clean Build
- ☐ Optimizing Compile
- ☐ Enable Pre-synthesis Control Flow Viewer

Input Arguments

☐ Do not show this dialog again

Figure 1-11: C Simulation Dialog Box

Each of the options controls how 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 prior to running 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 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. select nothing 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.

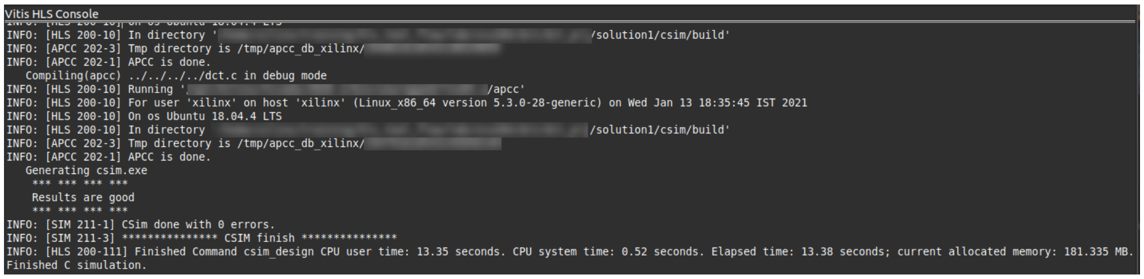
The information generated by the Vitis HLS tool can be found in two places, both described here.

The first is the Console window, which reports not only the output produced by the code being simulated, but all of the simulation engine messages as well.

The second is the simulation log, which provides only a few simulation engine messages and the simulated code output.

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

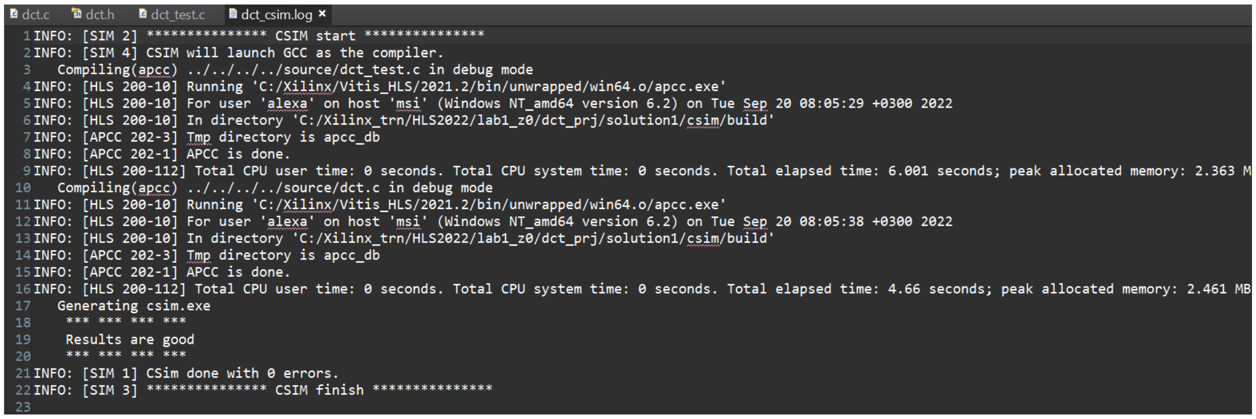
You may need to scroll to view all the output produced by the simulation.



```
INFO: [HLS 200-10] In directory '[redacted]/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/[redacted]
INFO: [APCC 202-1] APCC is done.
   Compiling(apcc) ../../../../dct.c in debug mode
INFO: [HLS 200-10] Running '[redacted]/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 [redacted]/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is /tmp/apcc_db_xilinx/[redacted]
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 200-111] 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 (the simulation log), described below, provides only a few simulation engine messages and the simulated code output.



```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling(apcc) ../../../../source/dct_test.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vitis_HLS/2021.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'alexa' on host 'msi' (Windows NT_amd64 version 6.2) on Tue Sep 20 08:05:29 +0300 2022
INFO: [HLS 200-10] In directory 'C:/Xilinx_trn/HLS2022/lab1_z0/dct_prj/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
INFO: [HLS 200-112] Total CPU user time: 0 seconds. Total CPU system time: 0 seconds. Total elapsed time: 6.001 seconds; peak allocated memory: 2.363 M
   Compiling(apcc) ../../../../source/dct.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vitis_HLS/2021.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'alexa' on host 'msi' (Windows NT_amd64 version 6.2) on Tue Sep 20 08:05:38 +0300 2022
INFO: [HLS 200-10] In directory 'C:/Xilinx_trn/HLS2022/lab1_z0/dct_prj/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
INFO: [HLS 200-112] Total CPU user time: 0 seconds. Total CPU system time: 0 seconds. Total elapsed time: 4.66 seconds; peak allocated memory: 2.461 MB
   Generating csim.exe
    *** *** *** ***
    Results are good
    *** *** *** ***
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

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.

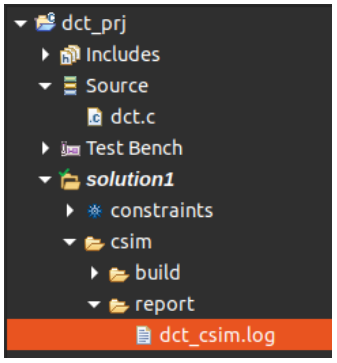


Figure 1-13: Locating the Simulation Log File

You should see a "Results are good" *message in the simulation log file and in the console area. If you do not see this message, ask for help from your instructor.*

Synthesizing the Design — Step 3

In this step, you will synthesize the design by using Vitis HLS tool defaults and analyze how many resources are utilized to implement the C design.

3-1. Synthesize the design.

3-1-1. Select **Solution** > **Run C Synthesis** > **Active Solution** or click the **C Synthesis** icon from the toolbar.

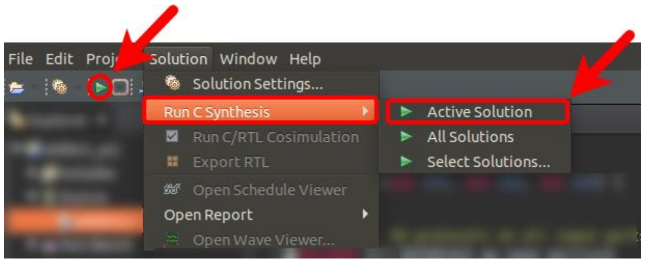


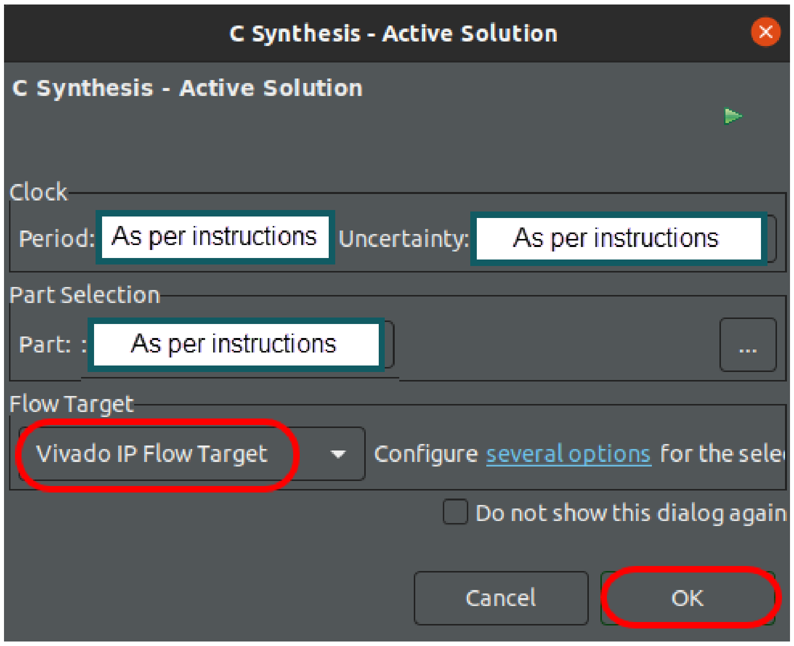
Figure 1-14: Launching Synthesis

You can also click **Run C Synthesis** under C SYNTHESIS from the Flow Navigator at the bottom.

3-1-2. Make sure that the clock period is the same as your project settings.

3-1-3. Make sure that the **Part** here matches with the part number of the project.

3-1-4. Select **Vivado IP Flow Target** from the Flow Target drop-down list.


C Synthesis - Active Solution

C Synthesis - Active Solution

Clock

Period: As per instructions Uncertainty: As per instructions

Part Selection

Part: : As per instructions

Flow Target

Vivado IP Flow Target Configure several options for the sele

Do not show this dialog again

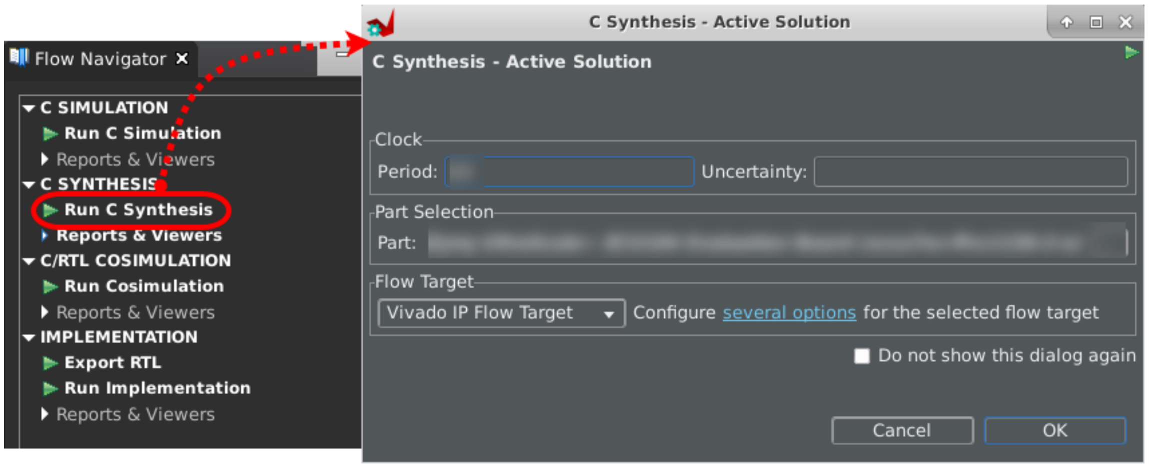
Figure 1-15: C Synthesis Dialog Box

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

This synthesizes the currently selected solution.

All solutions (or selected solutions) can be synthesized by using the drop-down menu next to the synthesis icon. You can synthesize all solutions or synthesize selected solutions in addition to the default.

You can also run C synthesis from the Flow Navigator by clicking **Run C Synthesis** under C SYNTHESIS. You can then select the desired part and flow target to run the synthesis.



C Synthesis - Active Solution

Clock
Period: [redacted] Uncertainty:

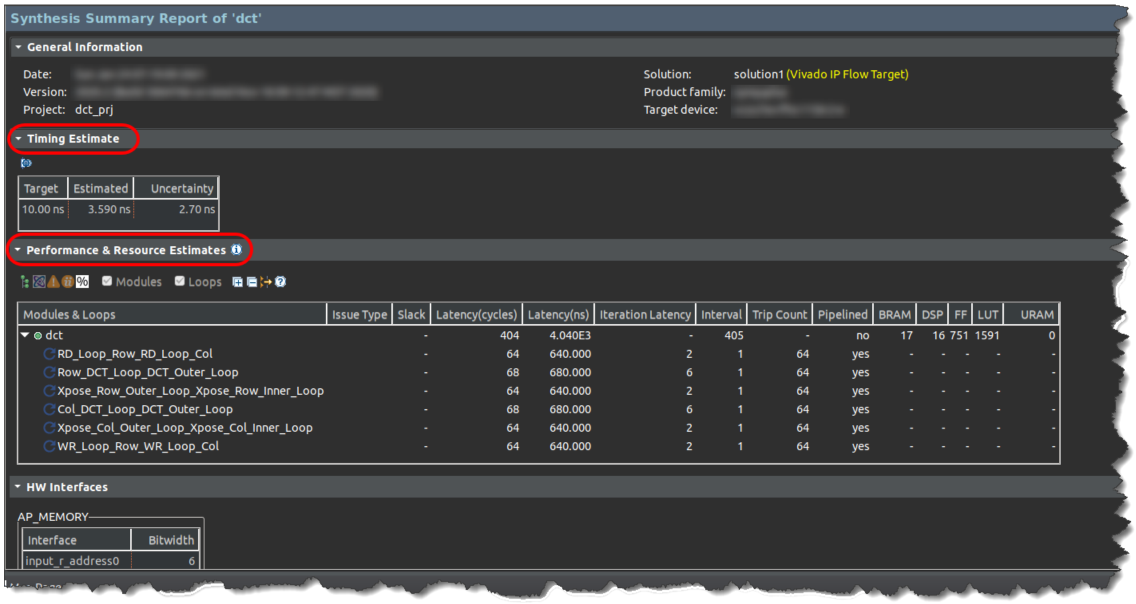
Part Selection
Part: [redacted]

Flow Target
Vivado IP Flow Target ▾ Configure several options for the selected flow target

☐ Do not show this dialog again

Figure 1-16: Launching Synthesis Using the Flow Navigator

When synthesis completes, the **Synthesis Summary Report** will be displayed in the Information pane.



Synthesis Summary Report of 'dct'

General Information

Date:	[redacted]	Solution:	solution1 (Vivado IP Flow Target)
Version:	[redacted]	Product family:	[redacted]
Project:	dct_prj	Target device:	[redacted]

Timing Estimate

Target	Estimated	Uncertainty
10.00 ns	3.590 ns	2.70 ns

Performance & Resource Estimates

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dct		-	404	4.040E3	-	405	-	no	17	16	751	1591	0
RD_Loop_Row_RD_Loop_Col		-	64	640.000	2	1	64	yes	-	-	-	-	-
Row_DCT_Loop_DCT_Outer_Loop		-	68	680.000	6	1	64	yes	-	-	-	-	-
Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop		-	64	640.000	2	1	64	yes	-	-	-	-	-
Col_DCT_Loop_DCT_Outer_Loop		-	68	680.000	6	1	64	yes	-	-	-	-	-
Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop		-	64	640.000	2	1	64	yes	-	-	-	-	-
WR_Loop_Row_WR_Loop_Col		-	64	640.000	2	1	64	yes	-	-	-	-	-

HW Interfaces

AP_MEMORY

Interface	Bitwidth
input_r_address0	6

Figure 1-17: Synthesis Report [Example]

The **Synthesis Summary Report** report shows the timing, performance, and area estimates as well as estimated latency in the design.

3-1-6. Expand the **dct_prj** > **solution1** > **syn** > **report** folder in the Explorer pane.

3-1-7. Double-click the **dct_csynth.rpt** file to view the **Synthesis Details Report**.

3-1-8. Scroll to **Performance Estimates** and **Utilization Estimates** in the **Synthesis Details Report** to answer the following question.

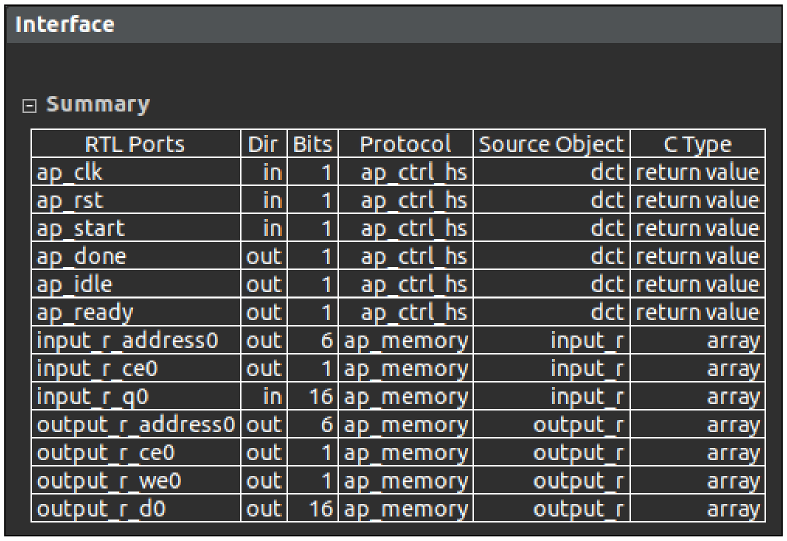
Question 1

Write down the following details from the Synthesis report:

- Target\Estimated clock periods:
- Worst case latency cycles/absolute (max):
- Number of BRAM_18K:
- Number of DSP48E used:
- Number of FFs used:
- Number of LUTs used:

3-1-9. Select **Interface** > **Summary** in the Synthesis report.

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



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 1-18: Generated Interface Signals

You can see that *ap_clk* and *ap_rst* and ap_... are automatically added.

ap_start, *ap_done*, and *ap_idle* are top-level signals used as handshaking signals to indicate

- when the design is able to accept the next computation command (*ap_idle*),
- when the next computation is started (*ap_start*),
- when the computation is completed (*ap_done*).

Other signals are generated based on the design itself.

3-1-10. 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.

Also, note that the target design has hierarchical functions, and reports corresponding to lower-level functions are also 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

After synthesis completes, the Vitis HLS tool automatically creates another synthesis reports to help you understand and analyze the performance of the implementation.

*These reports include the **Schedule Viewer**, **Function Call Graph** and **Dataflow Viewer**.*

*You can view these reports from the **Flow Navigator** in the Vitis HLS tool IDE.*

- ***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 understand the design behavior.

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

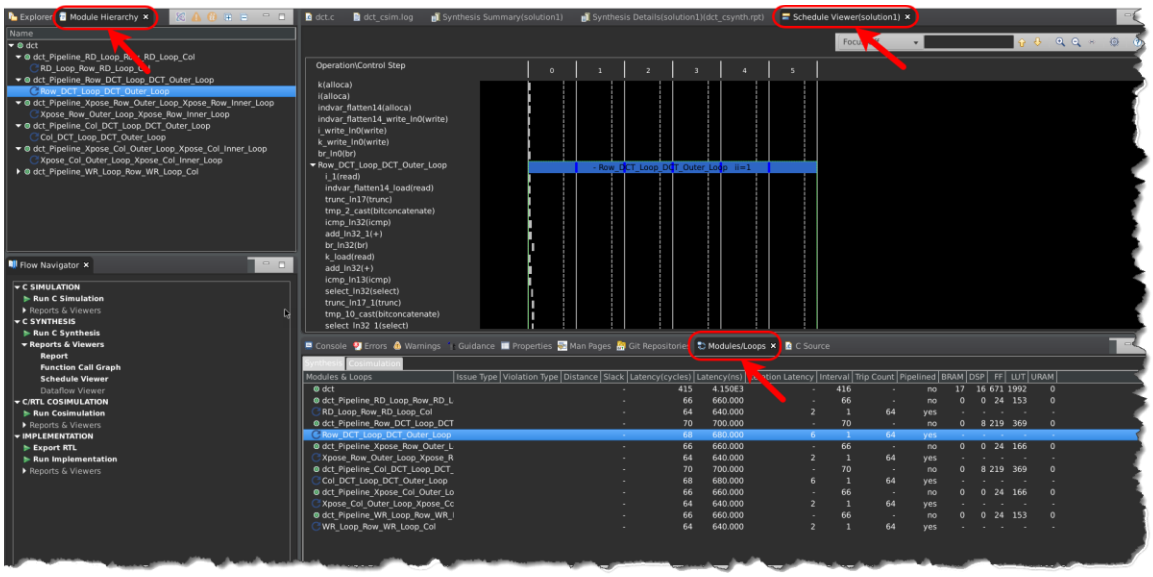
You can also open the Schedule Viewer from the Flow Navigator by expanding the Reports & Viewers section and clicking **Schedule Viewer**.

Along with Schedule Viewer, you also get the following windows:

- **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.



```
Operation\Control Step
k(alloca)
i(alloca)
indvar_flatten14(alloca)
indvar_flatten14_write_ln0(write)
i_write_ln0(write)
k_write_ln0(write)
br_ln0(br)
Row_DCT_Loop_DCT_Outer_Loop
  i_1(read)
  indvar_flatten14_load(read)
  trunc_ln17(trunc)
  tmp_2_cast(bitconcatenate)
  icmp_ln32(icmp)
  add_ln32_1(+)
  br_ln32(br)
  k_load(read)
  add_ln32(+)
  icmp_ln13(icmp)
  select_ln32(select)
  trunc_ln17_1(trunc)
  tmp_10_cast(bitconcatenate)
  select_ln32_1(select)
```

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_l				-	66	660.000	-	66	-	no	0	0	24	153	0
WR_Loop_Row_WR_Loop_Col				-	64	640.000	2	1	64	yes	-	-	-	-	-

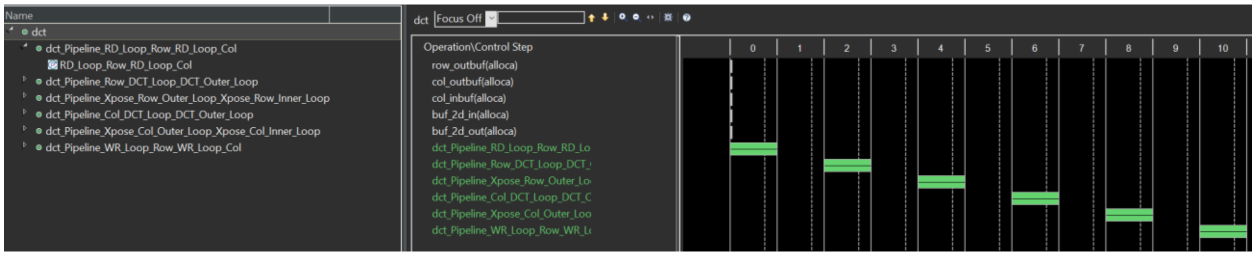
The **Schedule Viewer** provides a detailed view of the synthesized RTL, showing each operation and control step of the function, and the clock cycle that it executes in. It helps you to identify any loop dependencies that are preventing parallelism, timing violations, and data dependencies.

- The left vertical axis shows the names of operations and loops in the RTL hierarchy in topological order, implying that an operation on line **n** can only be driven by operations from a previous line and will only drive an operation in a later line.
- 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 Vivado Design Suite back-end processes, like place and route.
- Each operation is shown as a gray 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 case of function calls, the provided cycle information is equivalent to the operation latency.
- Multi-cycle operations are shown as gray 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.
- In addition, lines of source code are associated with each operation in the Schedule Viewer report. Right-click the operation to use the Goto Source command to open the input source code associated with the operation.

4-2. Analyze the performance and resources of the dct module.

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

4-2-1. Select **dct** in the Module Hierarchy.


Name
- dct
  - dct_Pipeline_RD_Loop_Row_RD_Loop_Col
    - RD_Loop_Row_RD_Loop_Col
  - dct_Pipeline_Row_DCT_Loop_DCT_Outer_Loop
  - dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop
  - dct_Pipeline_Col_DCT_Loop_DCT_Outer_Loop
  - dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop
  - dct_Pipeline_WR_Loop_Row_WR_Loop_Col

dct Focus Off

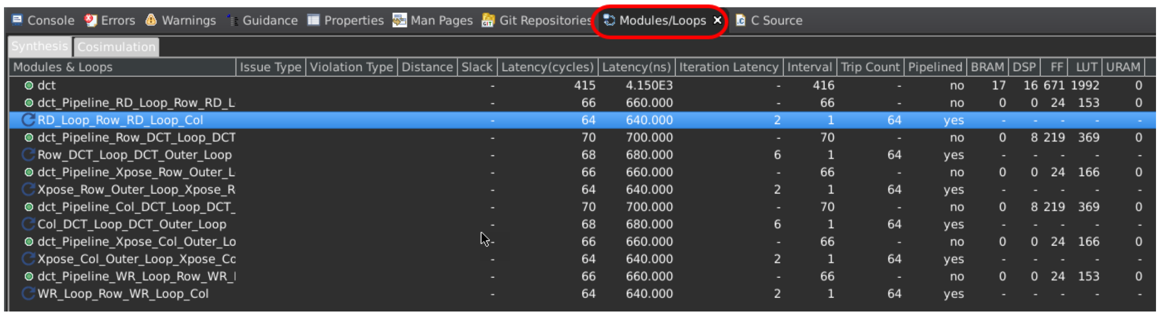
Operation\Control Step
row_outbuf(alloca)
col_outbuf(alloca)
col_inbuf(alloca)
buf_2d_in(alloca)
buf_2d_out(alloca)
dct_Pipeline_RD_Loop_Row_RD_Lo
dct_Pipeline_Row_DCT_Loop_DCT_
dct_Pipeline_Xpose_Row_Outer_Lo
dct_Pipeline_Col_DCT_Loop_DCT_C
dct_Pipeline_Xpose_Col_Outer_Loo
dct_Pipeline_WR_Loop_Row_WR_L

This pane 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 are able to view Schedules and the resources used by that function.

4-2-2. Select **RD_Loop_Row_RD_Loop_Col** in the **Module/Loops** pane and review the performance and resources used in this pane information.



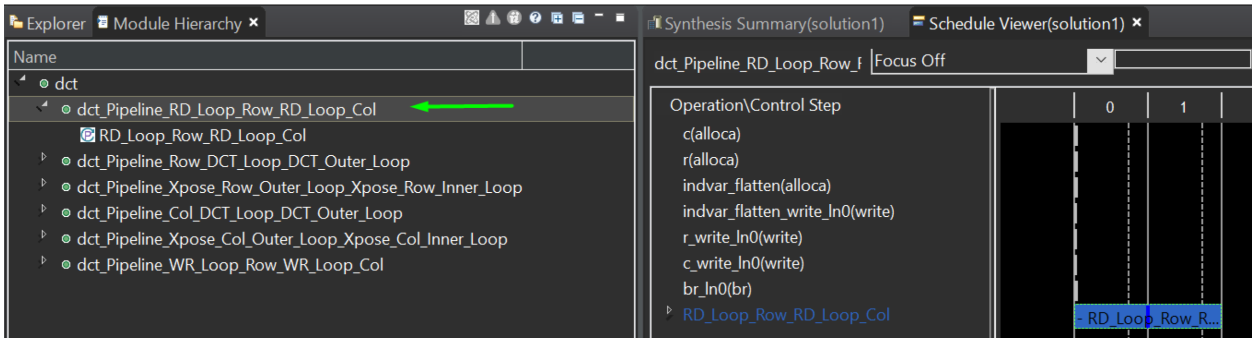
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
● dct				-	415	4.150E3	-	416	-	no	17	16	671	1992	0
● dct_Pipeline_RD_Loop_Row_RD_L				-	66	660.000	-	66	-	no	0	0	24	153	0
RD_Loop_Row_RD_Loop_Col				-	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 1-19: Modules/Loops Pane (ZCU104)

This pane shows that this loop has a trip count of 64.

It also shows that the loop RD_Loop_Row takes 64 clock cycles to execute.

4-2-3. Select **RD_Loop_Row_RD_Loop_Col** in the Module Hierarchy and expand the loop **RD_Loop_Row_RD_Loop_Col** in the Schedule Viewer (solution1).



Name
- dct
  - dct_Pipeline_RD_Loop_Row_RD_Loop_Col
    - RD_Loop_Row_RD_Loop_Col
  - dct_Pipeline_Row_DCT_Loop_DCT_Outer_Loop
  - dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop
  - dct_Pipeline_Col_DCT_Loop_DCT_Outer_Loop
  - dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop
  - dct_Pipeline_WR_Loop_Row_WR_Loop_Col

dct_Pipeline_RD_Loop_Row_f Focus Off

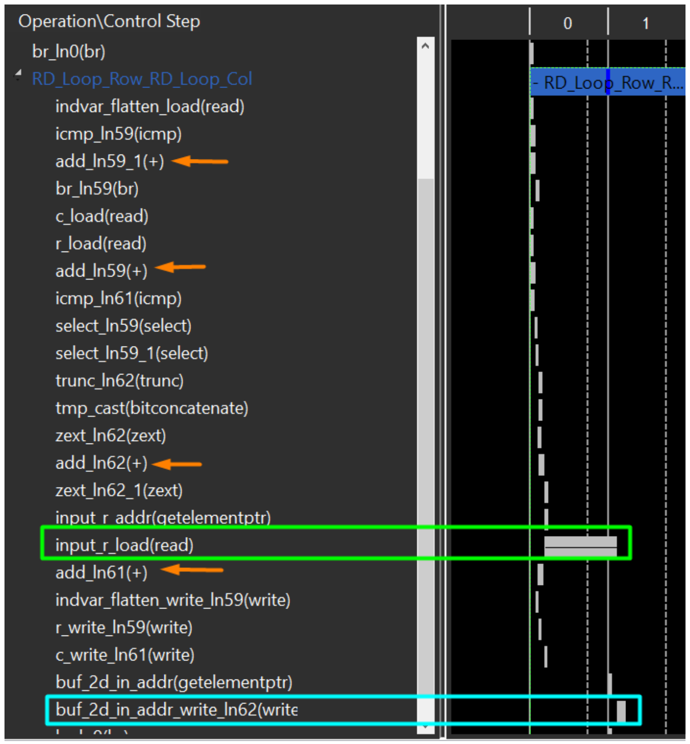
Operation\Control Step
c(alloca)
r(alloca)
indvar_flatten(alloca)
indvar_flatten_write_ln0(write)
r_write_ln0(write)
c_write_ln0(write)
br_ln0(br)
RD_Loop_Row_RD_Loop_Col

The information presented in the Schedule Viewer:

- *The design starts in the 0 state.*

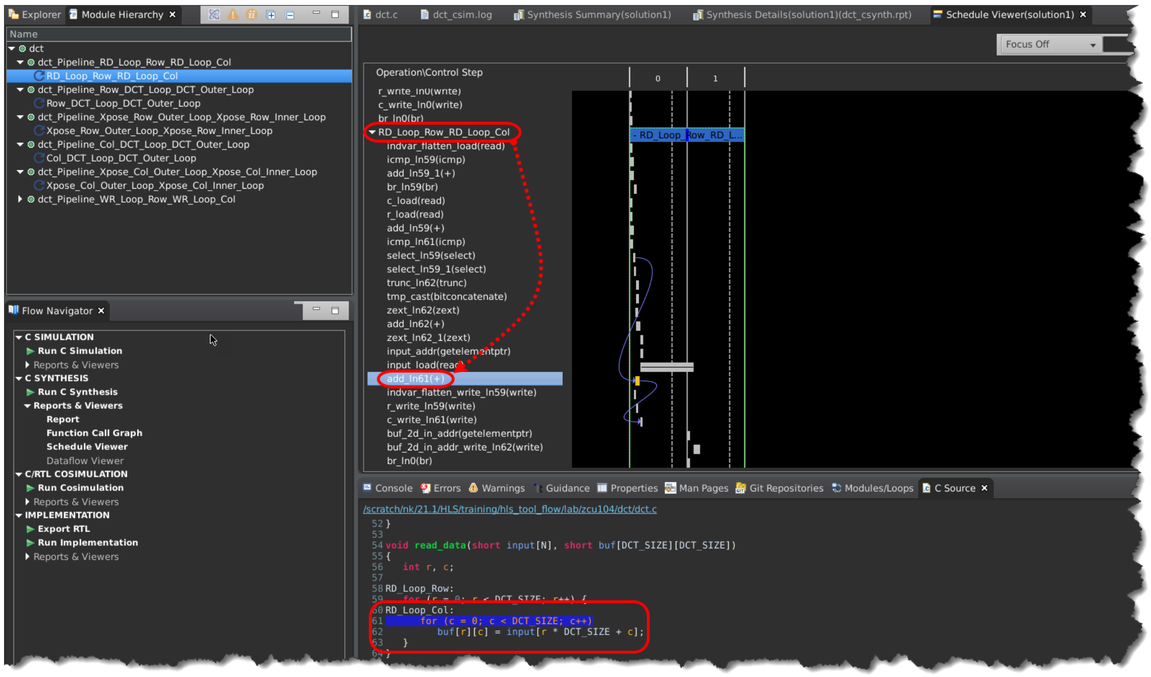
- *It then starts to execute the logic in the loop **RD_Loop_Row_RD_Loop_Col**.*
- *The loop executes over two states: 0, 1.*

*Within the loop **RD_Loop_Row_RD_Loop_Col**, you can see that there are some adders; a two-cycle read operation, and a write operation.*



4-2-4. Select **add_ln61(+)**.

4-2-5. Right-click the highlighted box and select **Goto Source**.



/scratch/nk/21.1/HLS/training/hls_tool_flow/lab/zcu104/dct/dct.c

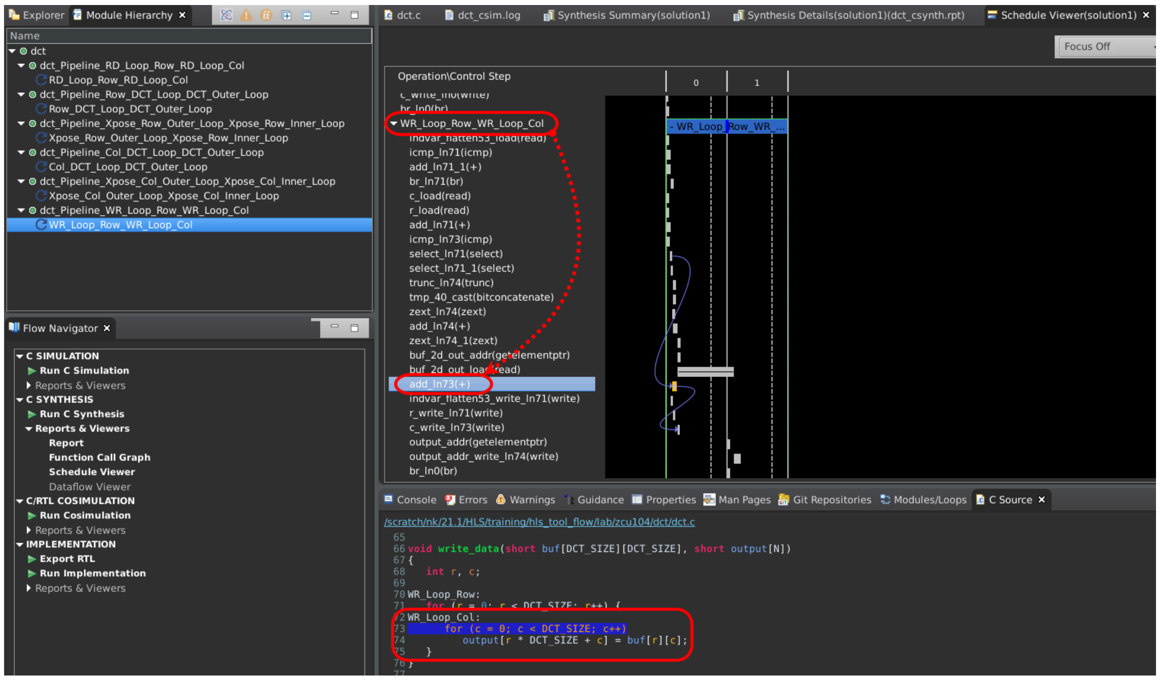
```
}

void read_data(short input[N], short buf[DCT_SIZE][DCT_SIZE])
{
   int r, c;

RD_Loop_Row:
   for (r = 0; r < DCT_SIZE; r++) {
RD_Loop_Col:
      for (c = 0; c < DCT_SIZE; c++)
         buf[r][c] = input[r * DCT_SIZE + c];
   }
}
```

Figure 1-20: Performance of the dct Loop Operation

You can see the source code for the operation.

4-2-6. Select **RD_Loop_Row_RD_Loop_Col** in the Module Hierarchy and expand the loop **WR_Loop_Row_WR_Loop_Col** loop in the Schedule Viewer (solution1)/. Then select **add_ln73(+)**.

4-2-7. Right-click the highlighted box and select **Goto Source**.



```
Explorer | Module Hierarchy
Name
dct
  dct_Pipeline_RD_Loop_Row_RD_Loop_Col
    RD_Loop_Row_RD_Loop_Col
  dct_Pipeline_Row_DCT_Loop_DCT_Outer_Loop
    Row_DCT_Loop_DCT_Outer_Loop
  dct_Pipeline_Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop
    Xpose_Row_Outer_Loop_Xpose_Row_Inner_Loop
  dct_Pipeline_Col_DCT_Loop_DCT_Outer_Loop
    Col_DCT_Loop_DCT_Outer_Loop
  dct_Pipeline_Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop
    Xpose_Col_Outer_Loop_Xpose_Col_Inner_Loop
  dct_Pipeline_WR_Loop_Row_WR_Loop_Col
    WR_Loop_Row_WR_Loop_Col
```

```
Flow Navigator
C SIMULATION
  Run C Simulation
  Reports & Viewers
C SYNTHESIS
  Run C Synthesis
  Reports & Viewers
    Report
    Function Call Graph
    Schedule Viewer
    Dataflow Viewer
C/RTL COSIMULATION
  Run Cosimulation
  Reports & Viewers
IMPLEMENTATION
  Export RTL
  Run Implementation
  Reports & Viewers
```

```
dct.c | dct_csim.log | Synthesis Summary(solution1) | Synthesis Details(solution1)(dct_csynth.rpt) | Schedule Viewer(solution1)
Focus Off
Operation\Control Step                       0    1
  c_write_ln0(write)
  br_ln0(br)
WR_Loop_Row_WR_Loop_Col                      - WR_Loop_Row_WR_...
  indvar_flatten53_load(read)
  icmp_ln71(icmp)
  add_ln71_1(+)
  br_ln71(br)
  c_load(read)
  r_load(read)
  add_ln71(+)
  icmp_ln73(icmp)
  select_ln71(select)
  select_ln71_1(select)
  trunc_ln74(trunc)
  tmp_40_cast(bitconcatenate)
  zext_ln74(zext)
  add_ln74(+)
  zext_ln74_1(zext)
  buf_2d_out_addr(getelementptr)
  buf_2d_out_load(read)
  add_ln73(+)
  indvar_flatten53_write_ln71(write)
  r_write_ln71(write)
  c_write_ln73(write)
  output_addr(getelementptr)
  output_addr_write_ln74(write)
  br_ln0(br)
```

```
Console | Errors | Warnings | Guidance | Properties | Man Pages | Git Repositories | Modules/Loops | C Source
/scratch/nk/21.1/HLS/training/hls_tool_flow/lab/zcu104/dct/dct.c

void write_data(short buf[DCT_SIZE][DCT_SIZE], short output[N])
{
   int r, c;

WR_Loop_Row:
   for (r = 0; r < DCT_SIZE; r++) {
WR_Loop_Col:
      for (c = 0; c < DCT_SIZE; c++)
         output[r * DCT_SIZE + c] = buf[r][c];
   }
}
```

Figure 1-21: Performance of WR_Loop_Row_WR_Loop_Col

You can see that the write operation is implementing the writing of data into the buf array from the input array variable.

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.

Run C/RTL co-simulation, selecting Verilog and skipping VHDL. Verify that the simulation passes.

5-1. Cosimulate the Vitis HLS tool design.

5-1-1. Select **Solution** > **Run C/RTL Cosimulation**.

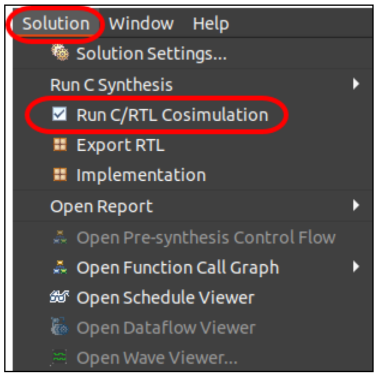


Figure 1-22: Launching from the Menu

You can also launch C/RTL Co-simulation from the Flow Navigator.

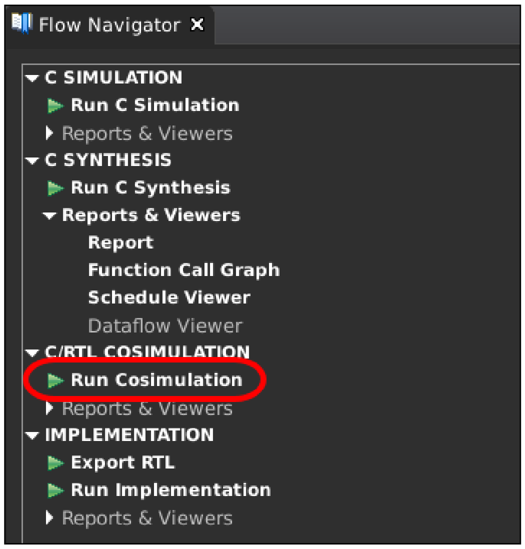


Figure 1-23: Launching Cosimulation from Flow Navigator

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

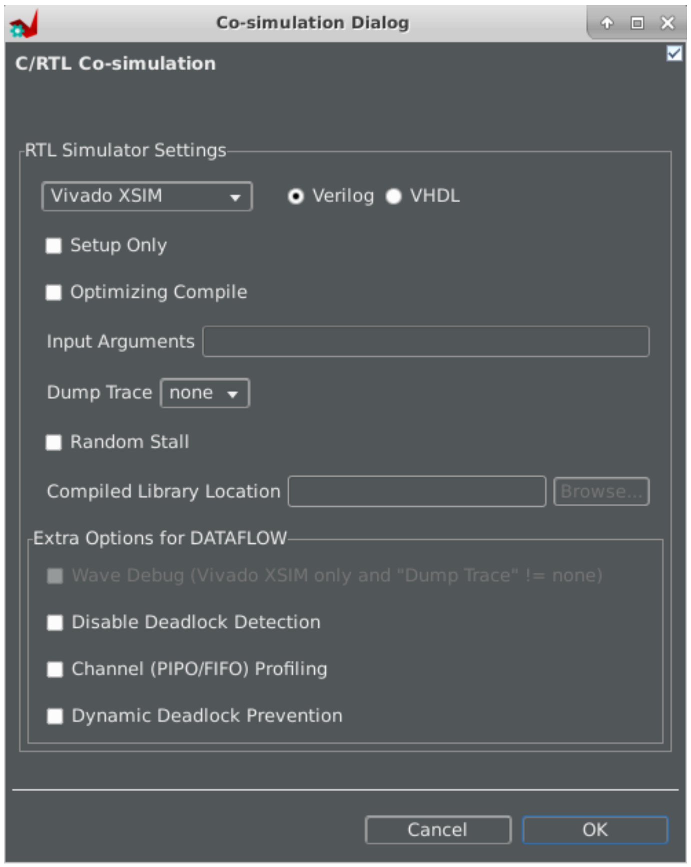

Co-simulation Dialog
C/RTL Co-simulation
RTL Simulator Settings
Vivado XSIM
Verilog
VHDL
Setup Only
Optimizing Compile
Input Arguments
Dump Trace
none
Random Stall
Compiled Library Location
Browse...
Extra Options for DATAFLOW
Wave Debug (Vivado XSIM only and "Dump Trace" != none)
Disable Deadlock Detection
Channel (PIPO/FIFO) Profiling
Dynamic Deadlock Prevention
Cancel
OK

Figure 1-24: 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 is the default simulator.
- Language: Specify the use of Verilog or VHDL as the output language for simulation.
- Setup Only: Create the required simulation files, but do not run the simulation. The simulation executable can be run from a command shell at a later time.
- Optimizing Compile: Enable optimization to improve the runtime performance, if possible, at the expense of compilation time.
- Input Arguments: Specify any command-line arguments to 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. Leave **the default options (i.e. select nothing)**.

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

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.

Also, in the Console tab, notice the "Results are Good " *message that is displayed.*

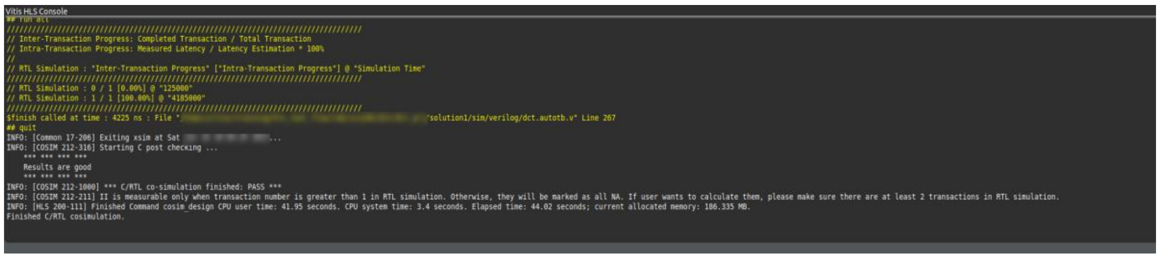
5-2. View the Cosimulation report.

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 of the simulation engine messages as well. The simulation log provides only a few simulation engine messages and the simulated code output.

5-2-1. Select the **Console** tab in the lower portion of the Vitis HLS tool GUI.

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



```
## run all
////////////////////////////////////////////////////////////////////////////////////
// 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%] @ "125000"
// RTL Simulation : 1 / 1 [100.00%] @ "4185000"
////////////////////////////////////////////////////////////////////////////////////
$finish called at time : 4225 ns : File "[redacted]/solution1/sim/verilog/dct.autotb.v" Line 267
## quit
INFO: [Common 17-206] Exiting xsim at Sat [redacted]...
INFO: [COSIM 212-316] Starting C post checking ...
    *** *** *** ***
    Results are good
    *** *** *** ***
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulation. Otherwise, they will be marked as all NA. If user wants to calculate them, please make sure there are at least 2 transactions in RTL simulation.
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 41.95 seconds. CPU system time: 3.4 seconds. Elapsed time: 44.02 seconds; current allocated memory: 186.335 MB.
Finished C/RTL cosimulation.
```

Figure 1-25: Example Output After a C/RTL Co-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 is how to access the simulation report.

5-2-2. Expand **dct_prj** > **solution1** > **sim** > **report** in the Explorer pane.

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

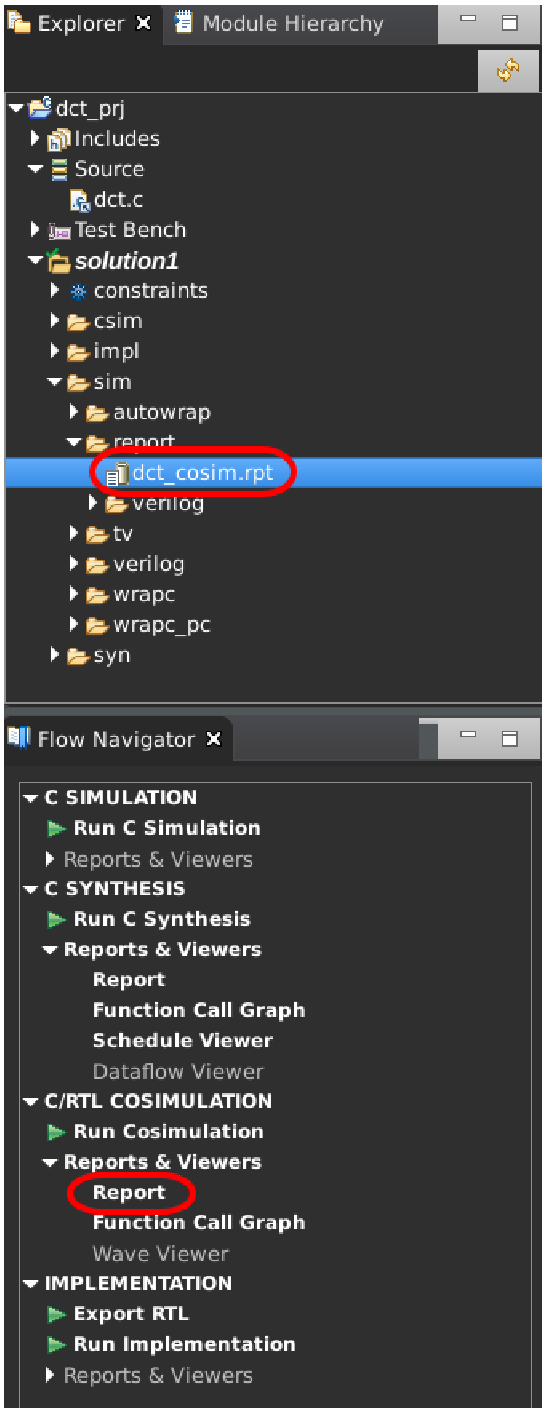
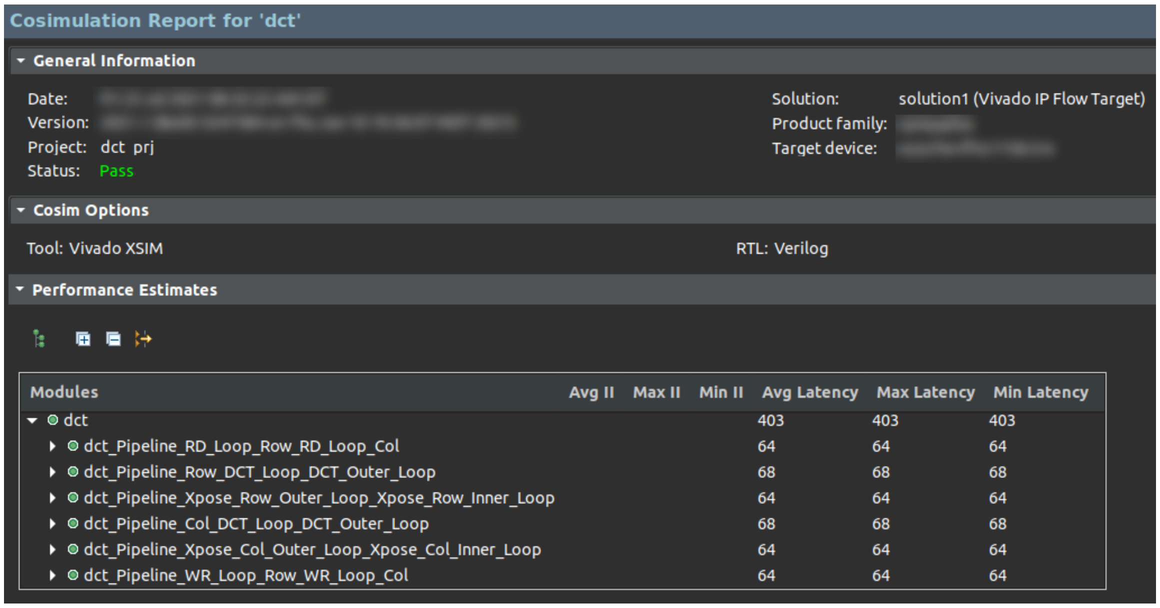


Figure 1-26: Locating the Co-simulation Log File

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: solution1 (Vivado IP Flow Target)
Product family:
Target device:

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 1-27: Cosimulation Report – HTML