Vitis HLS Tool Command Line Interface

2022.1

Abstract

This lab introduces how to perform basic actions in the Vitis™ HLS tool command line interface (CLI).

This lab should take approximately 30 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 CLI
- Simulate a C design by using a self-checking test bench
- Synthesize the design
- Simulate an RTL design by using a C test bench
- Implement the design

Introduction

This lab provides an introduction to the major features of the Vitis High-Level Synthesis (HLS) tool Command Line Interface (CLI) flow. You will use the Vitis HLS tool in CLI 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.

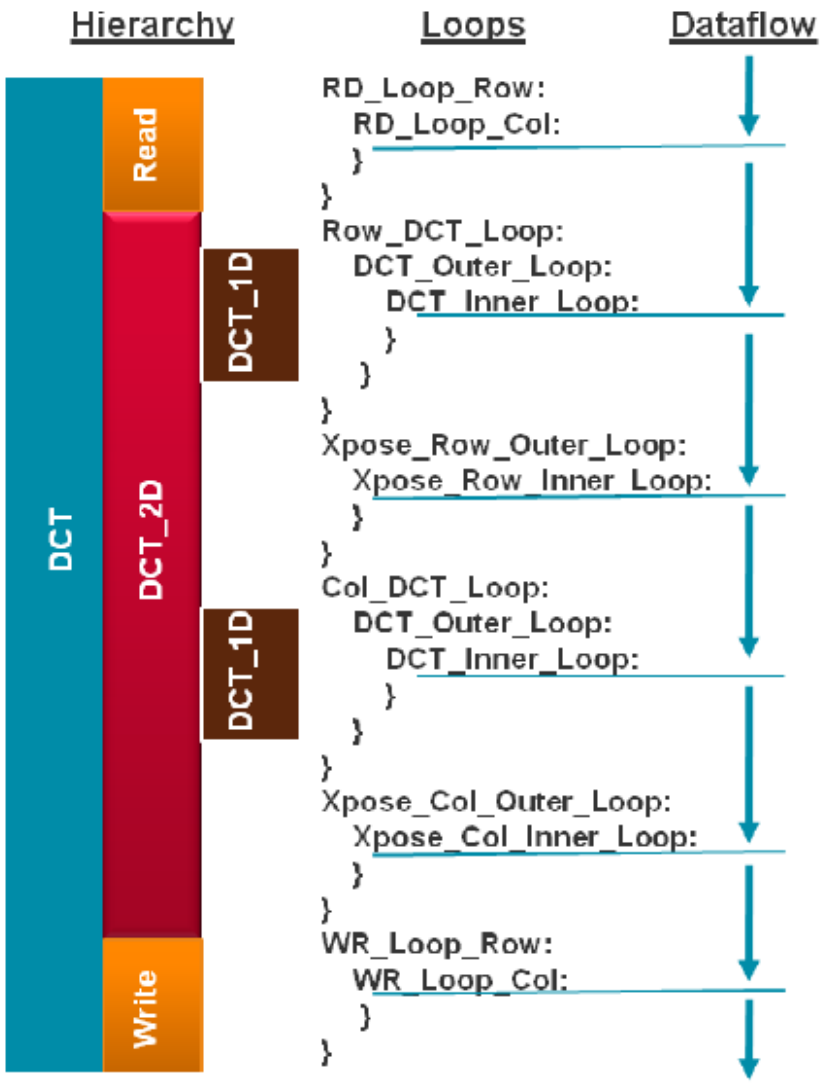


Figure 2-1: Design Hierarchy and Dataflow

Understanding the Lab Environment

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

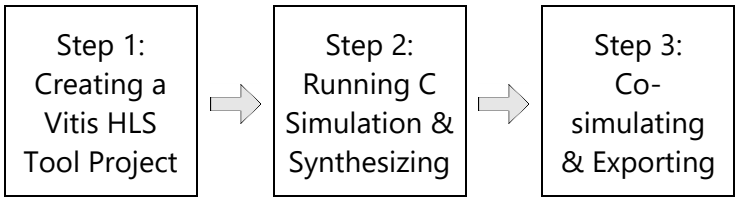
One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

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 create a new Vitis HLS tool project, add source files, and provide solution settings for the default solution using the Vitis HLS tool command prompt. Later, you will open the created project in the Vitis HLS tool GUI to have a quick review of the settings were applied.

1-1. Launch the Vitis HLS tool command prompt and change the directory to the `$TRAINING_PATH/hls_cli_flow/lab/[zcu104 | vck190]` working directory.

1-1-1. [Windows 10 users]: Select **Start** > **Xilinx Design Tools** > **Vitis HLS 2022.1 Command Prompt** to launch the Vitis HLS tool command prompt.

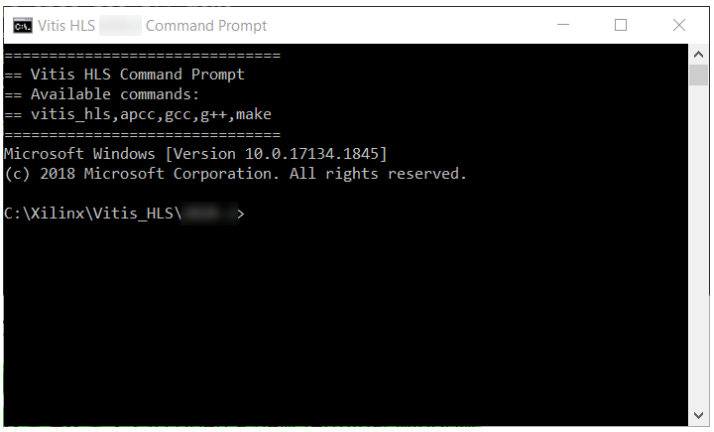


Figure 2-2: Vitis HLS Command Prompt

[Linux users]: Set up the Vitis HLS environment.

- Press <**Ctrl** + **Alt** + **T**> to open a new terminal window.
- Change the directory to the Vitis HLS tool installation path:
  [host] $ cd $TRAINING_PATH/hls_cli_flow/lab/[zcu104 | vck190]
- Enter the following command to set up the Vitis HLS environment appropriately:
  [host] $ source /opt/Xilinx/Vitis/2022.1/settings64.sh
- Type **vitis_hls -i** in the terminal and press <**Enter**> to launch the Vitis HLS tool command line interface.

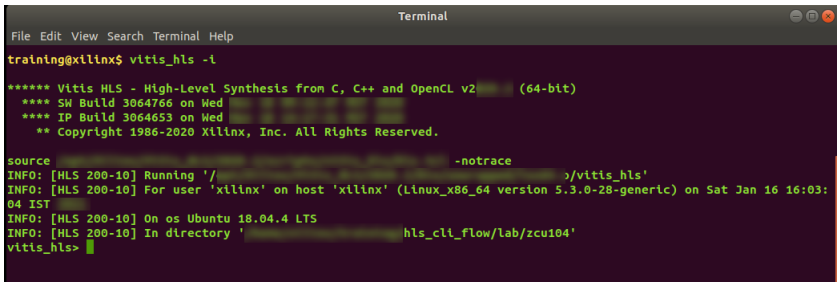


Figure 2-3: Vitis HLS Tool Command Prompt in Linux

1-2. Write the commands to create a new project, associate source files, and configure solution settings.

1-2-1. Browse to the following location and open **dct.tcl** with your preferred text editor.

`$TRAINING_PATH/hls_cli_flow/lab/[zcu104 | vck190]`

1-2-2. Insert the following command at line no. 2 of *dct.tcl* to create a new project named *dct_prj*:

```
open_project -reset dct_prj
```

Question 1

What does the –reset switch do in the previous command?

_____

_____

_____

1-2-3. Enter the following command at line no. 4 to specify *dct* as a top-level function to be synthesized in the design:

```
set_top dct
```

1-2-4. Enter the following command at line no. 6 to add *dct.c* as a design source file to the project:

```
add_files dct.c
```

Question 2

How does the above command differ from adding test bench files to the project?

1-2-5. Enter the following commands at line numbers 8, 9, 10, respectively, to associate *dct_test.c*, *in.dat*, and *out.golden.dat* as test bench source files to the project:

```
add_files -tb dct_test.c
add_files -tb in.dat
add_files -tb out.golden.dat
```

1-2-6. Enter the following command at line no. 12 to create the solution to the project named *solution1*:

```
open_solution solution1 -reset
```

1-2-7. Insert the command at line no. 14 to associate the device to the solution:

For the ZCU104 board:

```
set_part {xczu7ev-ffvc1156-2-e}
```

For the VCK190 board:

```
set_part {xcvc1902-vsva2197-2MP-e-S}
```

1-2-8. Insert the command at line no. 16 to associate the clock with a 10ns period to solution1:

```
create_clock -period 10ns
```

After writing the above commands, the *dct.tcl* file should look like the figure below.

```
dct.tcl
1  # Insert the command to create new project
2  open_project -reset dct_prj
3  # Insert the command to specify the top-level function
4  set_top dct
5  # Insert the command to add design file named dct.c
6  add_files dct.c
7  # Insert the command to add testbench files named dct_test.c, in.dat and out.golden.dat
8  add_files -tb dct_test.c
9  add_files -tb in.dat
10 add_files -tb out.golden.dat
11 # Insert the command to create the solution named solution1
12 open_solution solution1 -reset
13 # Insert the command to associate xczu7ev-ffvc1156-2-e device to the solution1
14 set_part {xczu7ev-ffvc1156-2-e}
15 # Insert the command to associate clock with 10ns period to the solution1
16 create_clock -period 10ns
17 # Insert the comamnd to run C simulaiton
18
19 # Insert the comamnd to Synthesize the design
20
21 # Insert the comamnd to perform C/RTL Cosimmulation
22
23 # Insert the comamnd to Export RTL as IP
```

Figure 2-4: dct.tcl with Project Information Example (ZCU104)

1-2-9. Save the **dct.tcl** file.

1-2-10. Exit the text editor to close the Tcl file.

1-3. Run the *dct.tcl* file to create the project and open the created project in the Vitis HLS tool GUI.

1-3-1. Enter the following command in the Vitis HLS tool command prompt to run the *dct.tcl* file:

```
vitis_hls -f dct.tcl
```

Note: If you get any error saying "Too many positional arguments specified", then go back to the Tcl file and replace '–' with '-'. This error occurs because when you copy the command from the Windows environment to the VM, the Linux VM treats the command differently.

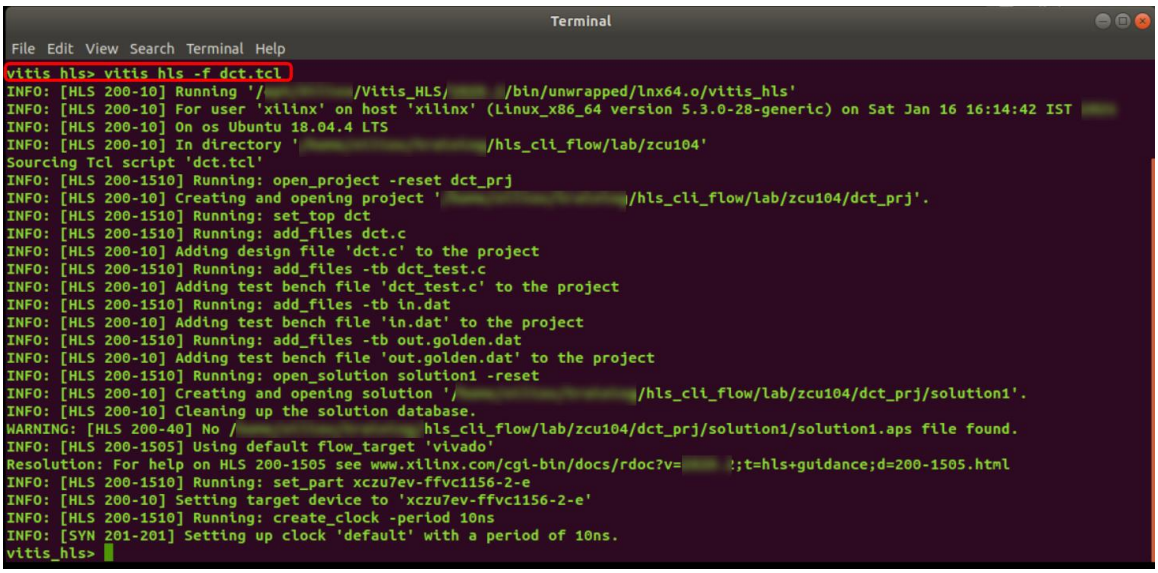


Figure 2-5: Creating the Project

1-3-2. Enter the following command to launch the GUI with the recently created project:

```
vitis_hls -p dct_prj
```

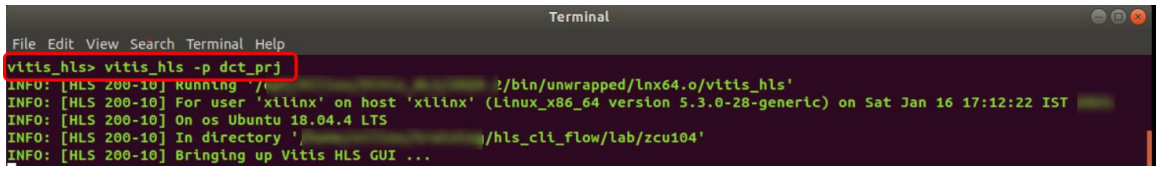


Figure 2-6: Launching the GUI from the CLI

You should see the created project in the Explorer view.

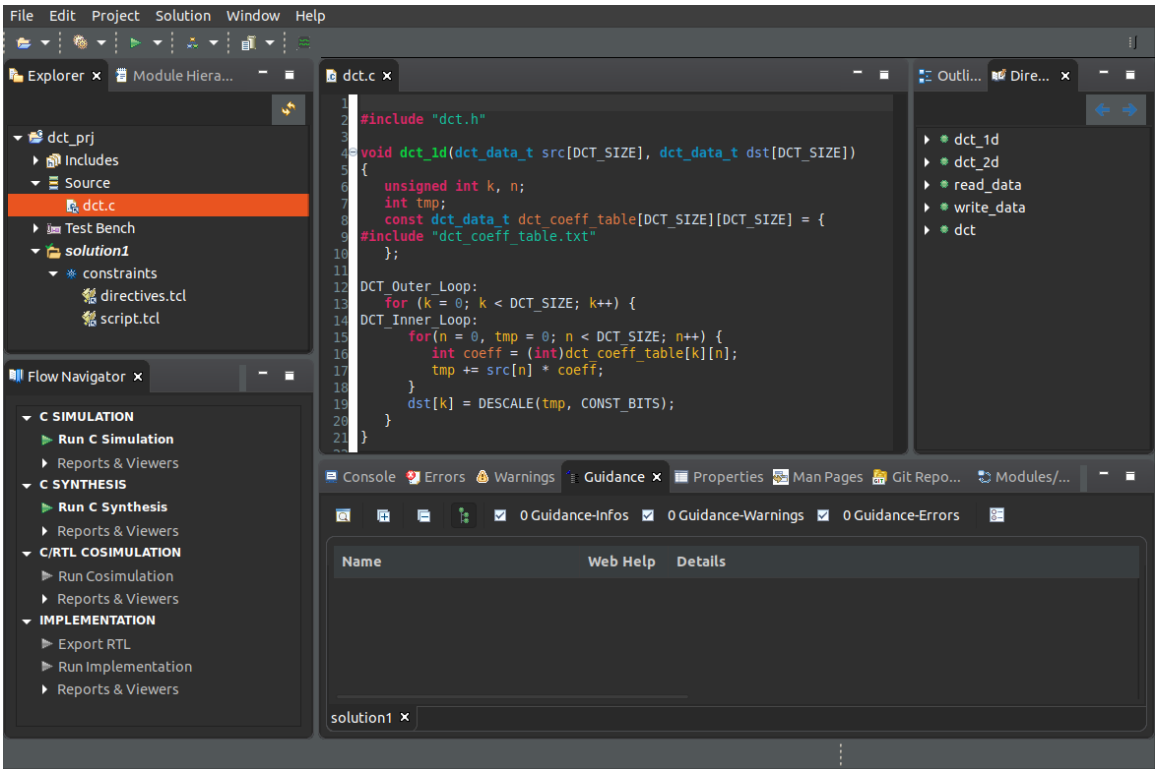


Figure 2-7: Vitis HLS Tool GUI with Recently Created Project

1-3-3. Expand **dct_prj** > **Source**.

Observe that the source file has been added.

1-3-4. Double-click the **dct.c** file to open and review the file.

1-3-5. Select **Project** > **Project Settings** in the Vitis HLS tool GUI.

The Project Settings dialog box opens.

1-3-6. Click **Simulation** and **Synthesis** to ensure that the design and test bench sources were added to the project as entered in the Tcl file.

1-3-7. Close the Project Settings dialog box once the source files are verified.

1-3-8. Select **Solution** > **Solution Settings** in the Vitis HLS tool GUI.

The Solution Settings dialog box for the active solution, i.e., *Solution1*, the only solution that exists in this current project, opens.

1-3-9. Select **Synthesis** to ensure that the clock frequency and part number are the ones described in the Tcl file.

1-3-10. Once the clock frequency and device settings are verified, click **Cancel** to close the Solution Settings dialog box.

1-3-11. Select **File** > **Exit** in the Vitis HLS tool GUI to exit the GUI.

Running C Simulation and Synthesizing the Design Step 2

In this step, you will run C simulation and synthesize the design from the Vitis HLS tool command prompt and then open the Vitis HLS tool GUI to review the project status and Synthesis report.

2-1. Simulate the design.

2-1-1. Enter the following command in the Vitis HLS tool command prompt to run C simulation:

```
csim_design
```

This command compiles and runs pre-synthesis C simulation of the design using the provided C test bench.

2-1-2. Observe the *Results are good* message in the command prompt.

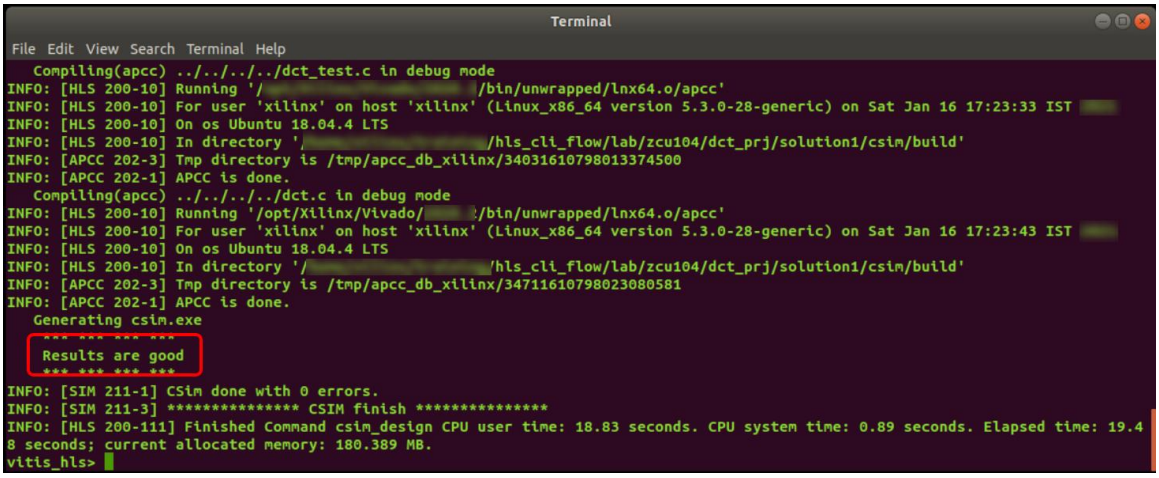


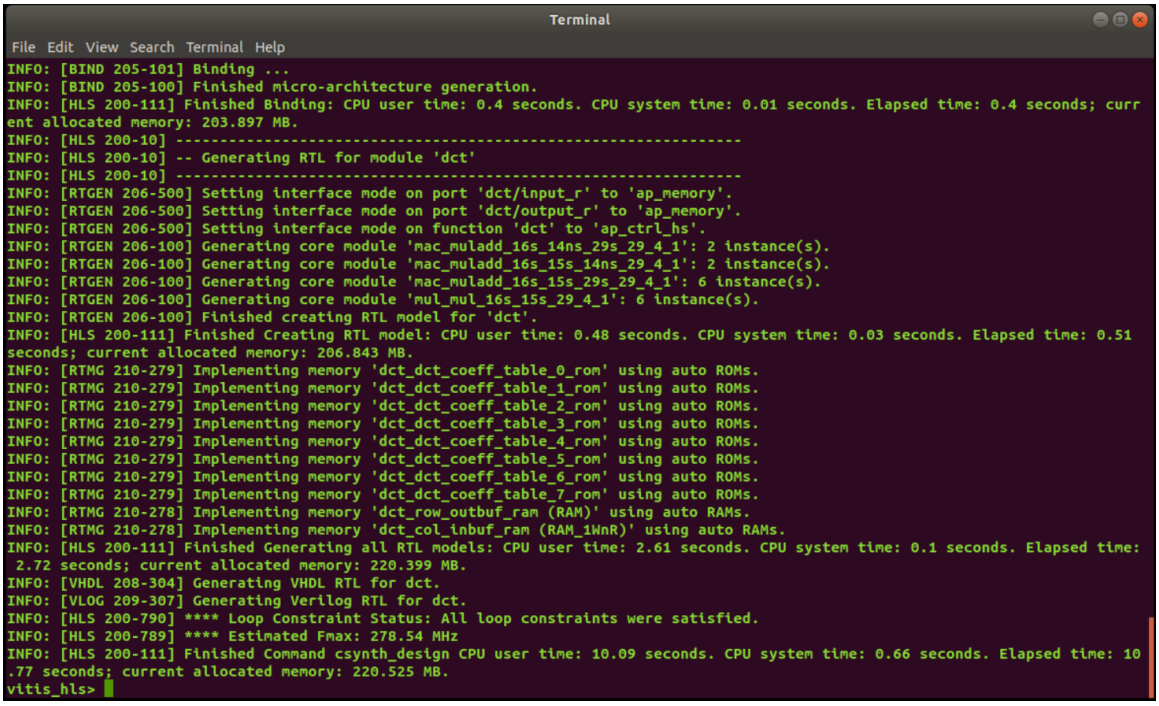
Figure 2-8: Running C Simulation from the Command Prompt

2-2. Synthesize the design.

2-2-1. Enter following command in the Vitis HLS tool command prompt to synthesize the design:

```
csynth_design
```

This will elaborate and perform high-level synthesis on the source files added to the project. This also analyzes the sources files, validates the directives if they are present, and performs some initial code transformations.



```
Terminal
File  Edit  View  Search  Terminal  Help
INFO: [BIND 205-101] Binding ...
INFO: [BIND 205-100] Finished micro-architecture generation.
INFO: [HLS 200-111] Finished Binding: CPU user time: 0.4 seconds. CPU system time: 0.01 seconds. Elapsed time: 0.4 seconds; curr
ent allocated memory: 203.897 MB.
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [HLS 200-10] -- Generating RTL for module 'dct'
INFO: [HLS 200-10] ----------------------------------------------------------------
INFO: [RTGEN 206-500] Setting interface mode on port 'dct/input_r' to 'ap_memory'.
INFO: [RTGEN 206-500] Setting interface mode on port 'dct/output_r' to 'ap_memory'.
INFO: [RTGEN 206-500] Setting interface mode on function 'dct' to 'ap_ctrl_hs'.
INFO: [RTGEN 206-100] Generating core module 'mac_muladd_16s_14ns_29s_29_4_1': 2 instance(s).
INFO: [RTGEN 206-100] Generating core module 'mac_muladd_16s_15s_14ns_29_4_1': 2 instance(s).
INFO: [RTGEN 206-100] Generating core module 'mac_muladd_16s_15s_29s_29_4_1': 6 instance(s).
INFO: [RTGEN 206-100] Generating core module 'mul_mul_16s_15s_29_4_1': 6 instance(s).
INFO: [RTGEN 206-100] Finished creating RTL model for 'dct'.
INFO: [HLS 200-111] Finished Creating RTL model: CPU user time: 0.48 seconds. CPU system time: 0.03 seconds. Elapsed time: 0.51
seconds; current allocated memory: 206.843 MB.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_0_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_1_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_2_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_3_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_4_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_5_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_6_rom' using auto ROMs.
INFO: [RTMG 210-279] Implementing memory 'dct_dct_coeff_table_7_rom' using auto ROMs.
INFO: [RTMG 210-278] Implementing memory 'dct_row_outbuf_ram (RAM)' using auto RAMs.
INFO: [RTMG 210-278] Implementing memory 'dct_col_inbuf_ram (RAM_1WnR)' using auto RAMs.
INFO: [HLS 200-111] Finished Generating all RTL models: CPU user time: 2.61 seconds. CPU system time: 0.1 seconds. Elapsed time:
 2.72 seconds; current allocated memory: 220.399 MB.
INFO: [VHDL 208-304] Generating VHDL RTL for dct.
INFO: [VLOG 209-307] Generating Verilog RTL for dct.
INFO: [HLS 200-790] **** Loop Constraint Status: All loop constraints were satisfied.
INFO: [HLS 200-789] **** Estimated Fmax: 278.54 MHz
INFO: [HLS 200-111] Finished Command csynth_design CPU user time: 10.09 seconds. CPU system time: 0.66 seconds. Elapsed time: 10
.77 seconds; current allocated memory: 220.525 MB.
vitis_hls>
```

Figure 2-9: Performing Synthesis from the Command Prompt

2-3. Verify the Synthesis report in the Vitis HLS tool GUI.

2-3-1. Enter the following command to launch the GUI:

```
vitis_hls -p dct_prj
```

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

2-3-3. Double-click the **dct_csynth.rpt** file to view the Synthesis report.

2-3-4. Analyze the report file.

Question 3

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:

2-3-5. Scroll to **Interface** > **Summary** in the synthesis report.

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

2-3-6. Select **File** > **Exit** in the Vitis HLS tool GUI to exit the GUI.

Co-simulating and Exporting the RTL Step 3

In this step, you will perform C/RTL co-simulation on the generated RTL files by using the C test bench from the Vitis HLS tool command prompt. Also, you will generate the IP from the Vitis HLS tool command prompt.

3-1. Perform C/RTL co-simulation.

3-1-1. Enter the following command in the Vitis HLS tool command prompt to execute post-synthesis co-simulation:

```
cosim_design
```

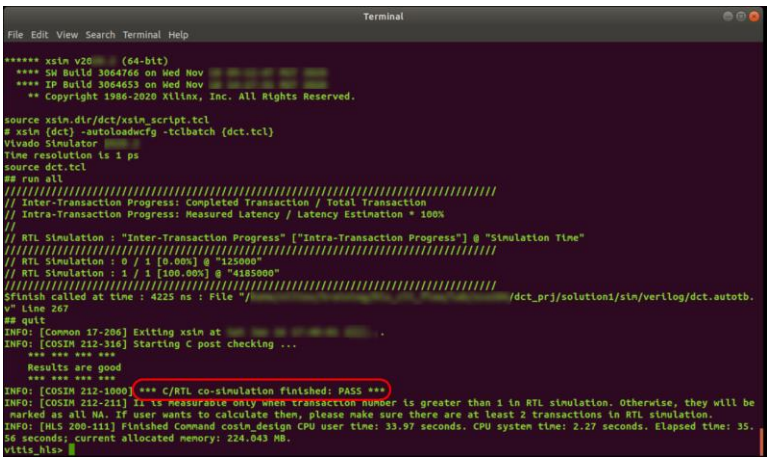


Figure 2-10: C/RTL Co-Simulation

Notice that the message ***** C/RTL co-simulation finished: PASS ***** is displayed.

3-2. Export the design to IP.

3-2-1. Enter the following command to perform an RTL implementation for Verilog HDL:

```
export_design -flow impl -format ip_catalog
```

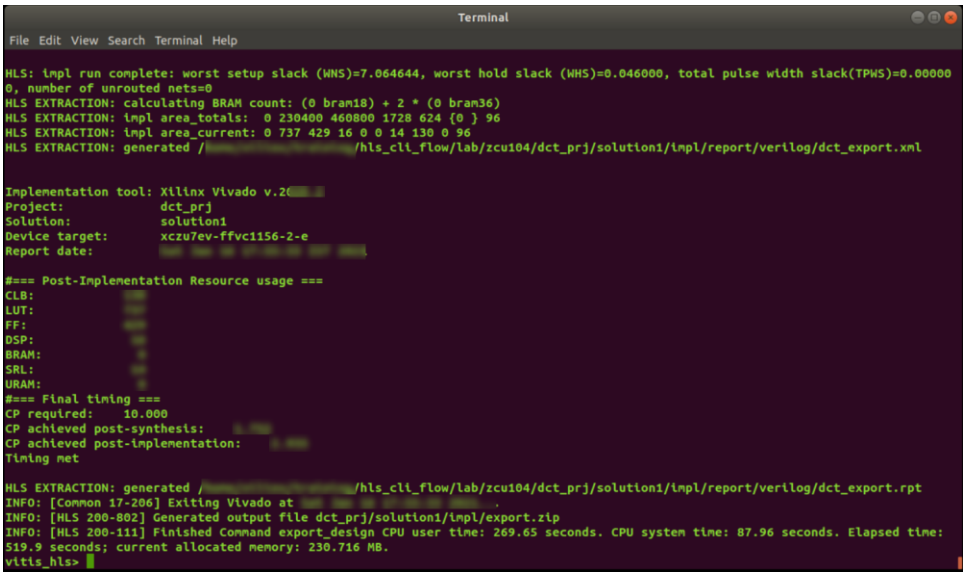


Figure 2-11: Exporting Synthesized RTL as an IP Example (ZCU104)

Note: Ignore the warnings.

The `-flow` option will perform RTL synthesis or RTL synthesis and implementation on the generated IP based on the option given (syn | impl). Implementation is run to evaluate and provide confidence that the RTL will meet its estimated timing and area goals. These results are not included as part of the exported package.

The `-format` option specifies the format to package the IP. The supported formats are:

- `ip_catalog`: A format suitable for adding to the Xilinx IP catalog.
- `xo`: A format accepted by the Vitis compiler for linking in the Vitis environment application acceleration flow.
- `syn_dcp`: Synthesized checkpoint file for the Vivado Design Suite. If this option is used, RTL synthesis is automatically executed. Vivado implementation can be optionally added.
- `sysgen`: Generates Vivado Design Suite IP and .zip archive for use in System Generator.

3-2-2. Observe the **Timing Met** message.

3-3. Verify the results in the Vitis HLS tool GUI.

3-3-1. Enter the following command to launch the GUI:

```
vitis_hls -p dct_prj
```

3-3-2. Select **Solution** > **Open Report** > **Cosimulation** in the Vitis HLS tool GUI to open the RTL Simulation report.

3-3-3. Select **Solution** > **Open Report** > **Implementation** > **Place & Route** to open the implementation report.

Note: You can open these reports from the Flow Navigator as well by expanding C/RTL COSIMULATION and IMPLEMENTATION > **Reports & Viewer**.

3-3-4. Analyze the reports and then close the Vitis HLS tool GUI.

3-3-5. Close the Vitis HLS tool command prompt.

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_cli_flow` directory.

3-4. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

3-4-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/hls_cli_flow`.

3-4-2. Select **`hls_cli_flow`**.

3-4-3. Press <**Delete**>.

-- OR --

Using the command line:

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

3-4-5. Enter the following command to delete the contents of the workspace:

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

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

Summary

In this lab, you learned how to use the Vitis HLS tool command prompt to:

- Create the Vitis HLS tool project
- Create a solution with the desired settings
- Execute major tasks in the Vitis HLS design flow such as simulating, synthesizing, co-simulating, and exporting the RTL of the design

Answers

1. What does the –reset switch do in the previous command?

   This option resets the project by removing any project data that already exists. Later if you wish to run the project again, this option helps reset the project data and creates a fresh one.

2. How does the above command differ from adding test bench files to the project?

   To add the test bench source, the "`-tb`" switch should be added to the `add_files` command.

3. 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