Tutorial 1

Introduction to MATLAB Simulink and HDL Coder

Introduction

In this tutorial exercise, you will learn how to use MATLAB Simulink and HDL Coder to build hardware designs from high-level dataflow models for both FPGA and ASIC targets.

Objectives

After completing this tutorial, you will be able to:

- Understand the design flow of MATLAB Simulink and HDL Coder.
- Model algorithms in Simulink as a dataflow system and perform bit-true cycle-accurate simulations for functional verification.
- Synthesize HDL codes from the Simulink model and implement hardware for both FPGA and ASIC targets.
- Estimate the performance and cost of the hardware implementation for both FPGA and ASIC targets.
- Understand some of the architectual optimization options available in HDL Coder.

Procedure

You will review and synthesize a design using fixed-point data types. This tutorial consists of four tasks:

1. Design a fixed-point complex multiplier using HDL coder blocks.
2. Simulate the complex multiplier design.
3. Generate VHDL/Verilog wrapper and create a hardware resource estimate for an FPGA target.
4. Generate VHDL/Verilog wrapper and create a hardware resource estimate for an ASIC Target.

Task 1: Design and Simulate a fixed-point Complex Multiplier

In this task, you will complete a design implemented with fixed-point data types.

1-1. Type "simulink" in the MATLAB command line to invoke Simulink.

1-2. Open the provided design file Tutorial1_1.slx. The Simulink design is shown in Figure 1. This design implements a complex multiplier with standard Simulink blocks (SL_Complex_Multiplier). Thus, it is a software model.

The multiplier model has two inputs X and Y, which receive data from two sources `data1` and `data2`. The input sources provide a sequence of fixed-point complex numbers from the MATLAB workspace generated by the provided MATLAB script (`data_sample_fp.m`). If you double-click on the block SL_Complex_Multiplier, you will be able to see the implementation of the complex multiplier. The model has one output Z connected to two different types of sink. All `To Workspace` sinks save the outputs into a single workspace variable `out`. The Scope helps us to see the output sequence in a graph.

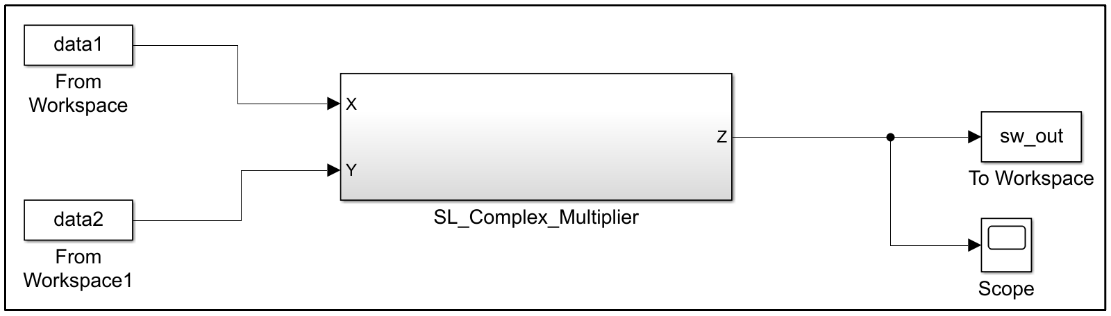


Figure 1: Multiplier model designed using standard Simulink blocks

1-3. Now, you will need to implement the same complex multiplier using HDL Coder blocks from the Simulink library. We will call this implementation HDL_Complex_Mutiplier. Note that a Simulink model implemented using HDL Coder blocks is a hardware model, which can be later on mapped into hardware targets. The SL_Complex_Multiplier and HDL_Complex_Multiplier blocks model exactly the same function.

1-4. Open the Simulink library browser using the button shown in Figure 2.

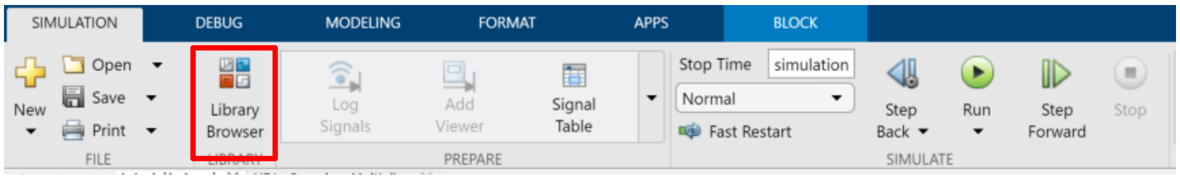


Figure 2: Simulink Library Browser Button

1-5. Expand the HDL Coder Blockset menu (Figure 3), select the Delay block from Commonly Used Blocks. A delay unit with n cycles of delay (Z^{-n}) in a dataflow model represents n stages of pipeline registers in hardware.

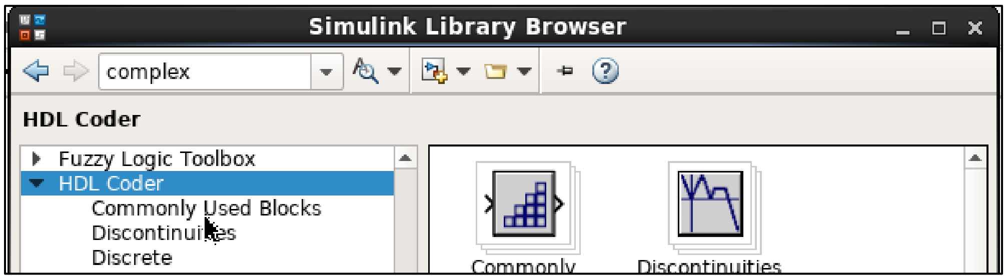


Figure 3: HDL Coder blocks in the Simulink Library Browser

1-6. Right-click the Delay block and select “Add block to model Tutorial1_1.slx”. If you need multiple instances of a block, you can use the Add block to model option from the library browser multiple times. Alternatively, you can go back to your design, right-click on a block, copy and paste. You can press Ctrl on the keyboard and drag a block using a mouse to make a copy as well.

1-7. Similarly, add the following blocks to your design:

Block Name	Library
`Delay`	HDL Coder -> Commonly Used Blocks
`Complex to Real-Imag`	HDL Coder -> Math Operations
`Real-Imag to Complex`	
`Product`	
`Add`	
`Scope`	Simulink -> Sinks
`To workspace`	

1-8. Figure 4 shows the completed design. Use this figure to make necessary copies of the blocks added from the library browser. Then connect the blocks according to Figure 4 by clicking on the ports.

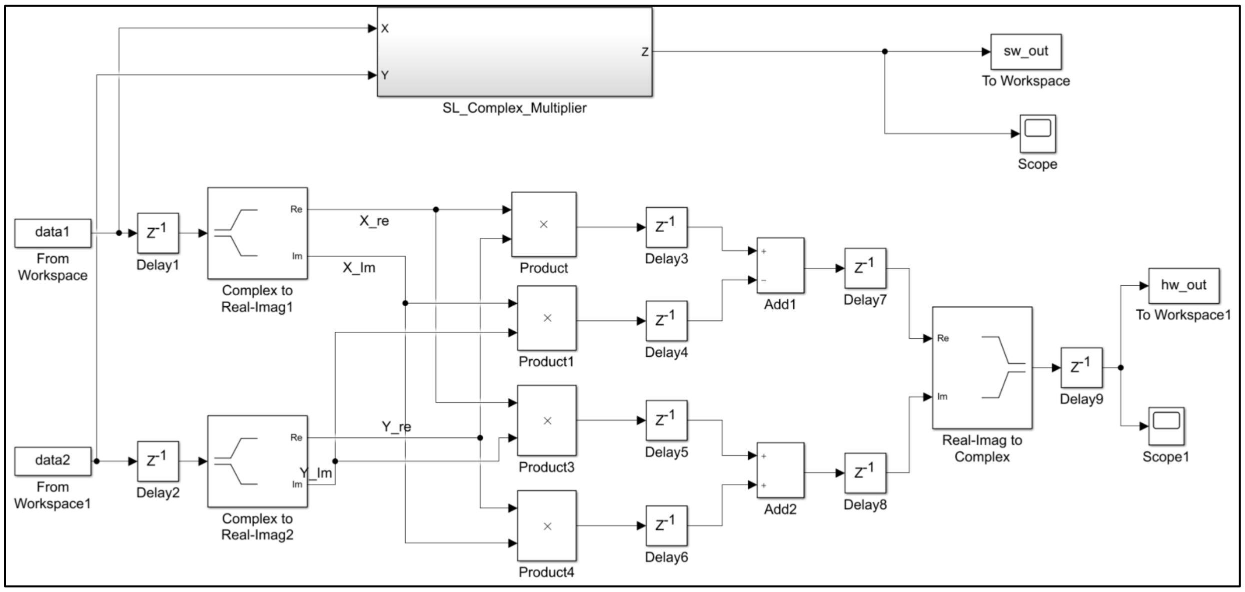
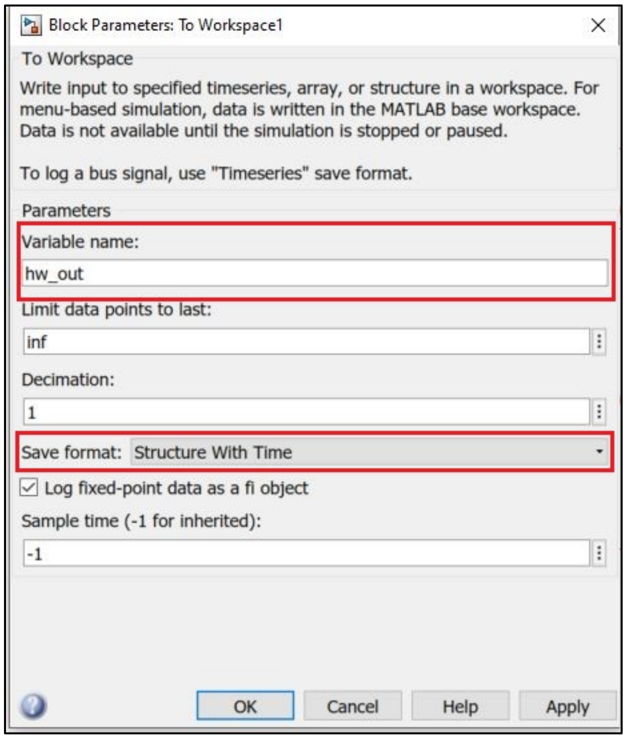


Figure 4: Complete Design

1-9. Double click on the `To Workspace` block added by you. Rename the block’s variable name from `simout` to `hw_out`. Change the Save format option to `Structure with Time` (Figure 5).



Block Parameters: To Workspace1

To Workspace

Write input to specified timeseries, array, or structure in a workspace. For menu-based simulation, data is written in the MATLAB base workspace. Data is not available until the simulation is stopped or paused.

To log a bus signal, use "Timeseries" save format.

Parameters

Variable name: hw_out

Limit data points to last: inf

Decimation: 1

Save format: Structure With Time

☑ Log fixed-point data as a fi object

Sample time (-1 for inherited): -1

Figure 5: To Workspace Block Parameters

1-10. Define the design boundary that will be synthesized into HDL code. Click and drag using a mouse to select the HDL Coder blocks. Selected blocks will be marked with a blue boundary (Figure 5). Right-click on the selected block and select "Create Subsystem from Selection".

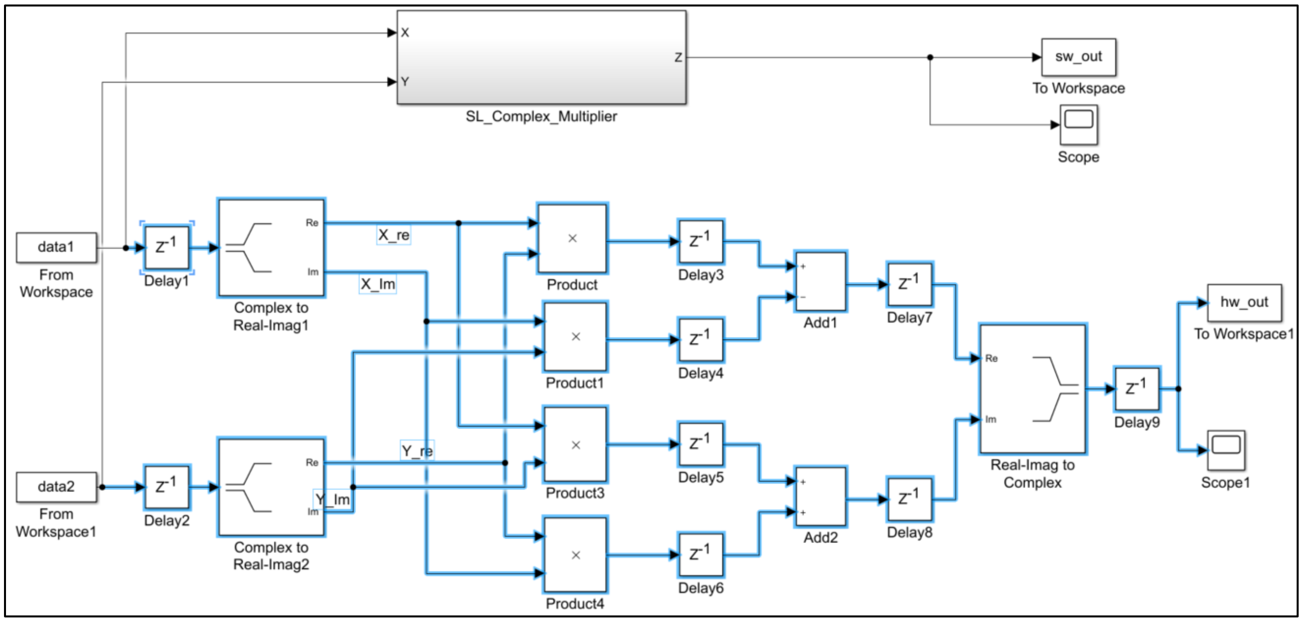


Figure 6: Select HDL Coder Blocks

1-11. Rename the subsystem created to HDL_Complex_Multiplier (Figure 6). Now, the design has both the software and the hardware models of the complex multiplier. The HDL_Complex_Multiplier block is the hardware model to be synthesized, and the SL_Complex_Multiplier block is the software reference model for functional verification purposes. It is important to note that HDL_Complex_Multiplier is built entirely by HDL coder blocks, as only HDL coder blocks can be synthesized into HDL codes. Differently, the design blocks in SL_Complex_Multiplier are general Simulink blocks that are not synthesizable.

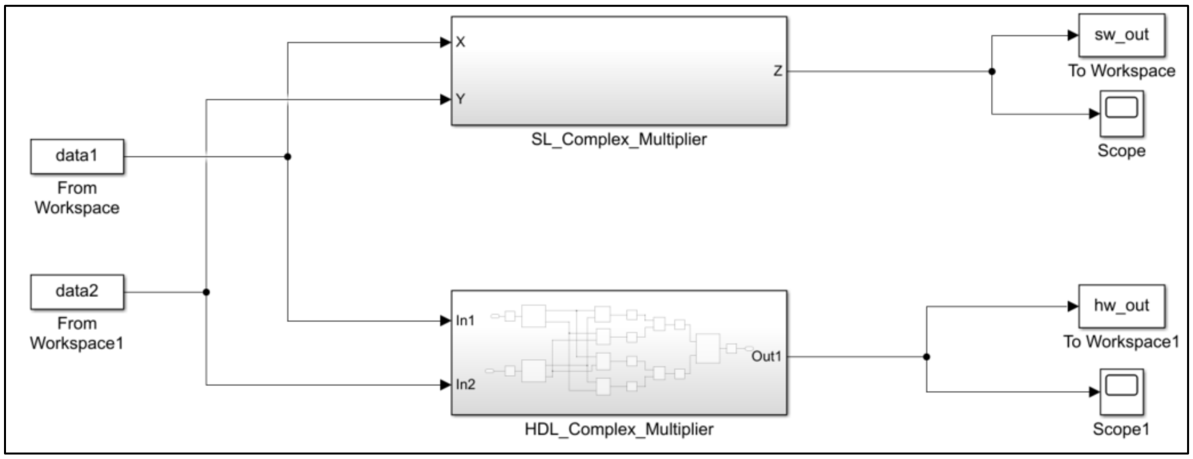


Figure 7: Subsystem HDL_Complex_Multiplier

For more HDL coder blocks, you can find them via Simulink Library Browser->HDL coder. For more guidelines about HDL coder, please refer to the HDL Coder Evaluation Reference Guide.

Task 2: Simulate the Complex Multiplier Design

In this task, you will simulate the complex multiplier design from task 1.

2-1. From the MATLAB window, open the provided script `data_sample_fp.m`. This script is used to generate a sequence of fixed-point complex numbers and send them to the complex multiplier blocks from MATLAB workspace. The script provided to you generates 50 complex numbers. If you want to generate more input samples, you can modify the `num_sample` variable. You can also change the `WordLength` and `FractionLength` of the fixed-point numbers generated to specify a different fixed-point data format. Note that the input sources' variable names (`data1` and `data2`) must match the `From Workspace` block's variable names in the Simulink model.

2-2. Run the 1st section of the MATLAB script using the button shown in Figure 7 to generate random input samples.

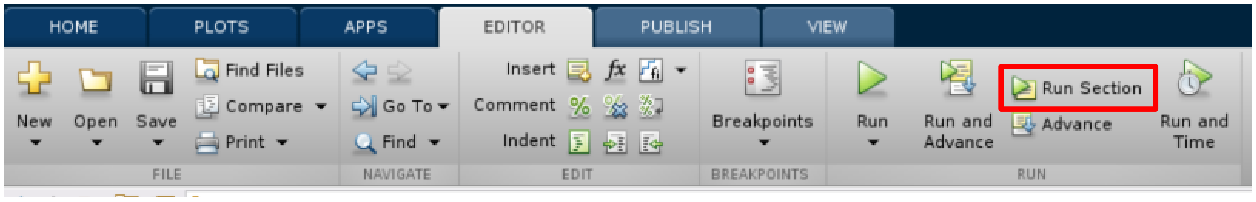
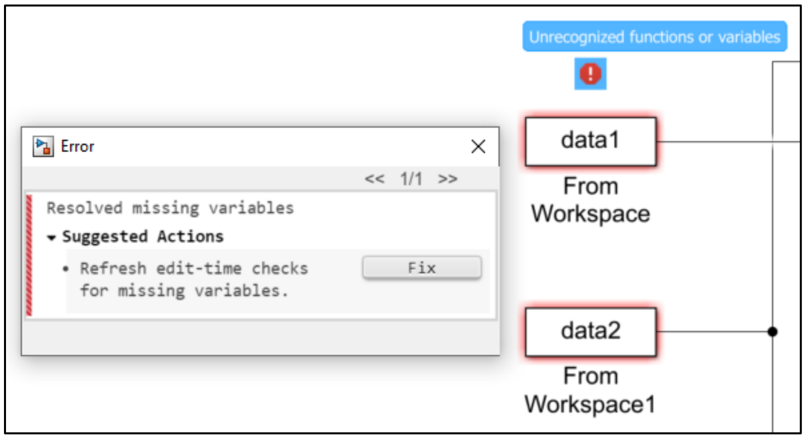


Figure 8: Run MATLAB Script

2-3. The `From Workspace` blocks will show an error "Unrecognized functions or variables" when there are no variables `data1` and `data2` defined in the MATLAB workspace. This error will be automatically fixed after the variables got defined by running the simulation or munually clicking the fix button (Figure 8).



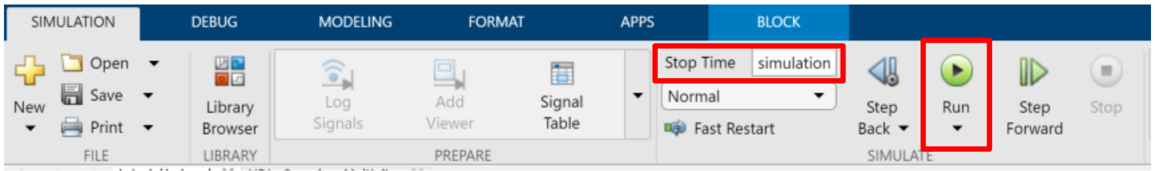
Resolved missing variables

Suggested Actions

- Refresh edit-time checks for missing variables.

Figure 9: Unrecognized functions or variables error

2-4. From your Simulink project worksheet, select **Simulation > Run** or click the **Run** simulation button. Double click on the data source blocks (`data1` or `data2`). See that the sample time we use to feed inputs into our design is specified by the `sample_time` variable, which is defined as 1µs (sampling frequency of 1MHz) by the MATLAB script. Also, see that the sample time of the input and output ports in HDL_Complex_Multiplier is specified as -1, which means to inherit the sample time from the previous block. Therefore, the system frequency of the HDL_Complex_Multiplier subsystem is equivalent to 1MHz. The end time of the simulation is specified by the `simulation_stop_time` variable, which is calculated to generate at least 50 samples at the outputs.



Stop Time: simulation

Normal

Figure 10: Simulation Run Button and end time

2-5. When the simulation completes, you can see that the color of the blocks turns into red, and the port data types are displayed. If you cannot see the data types on the connections, you can enable it by right click, then Other Display->Signals & Ports->Port Data Types. Since the input sources are fixed-point numbers and the rest of the design are inherited from them, the entire design now is running with a fixed-point number.

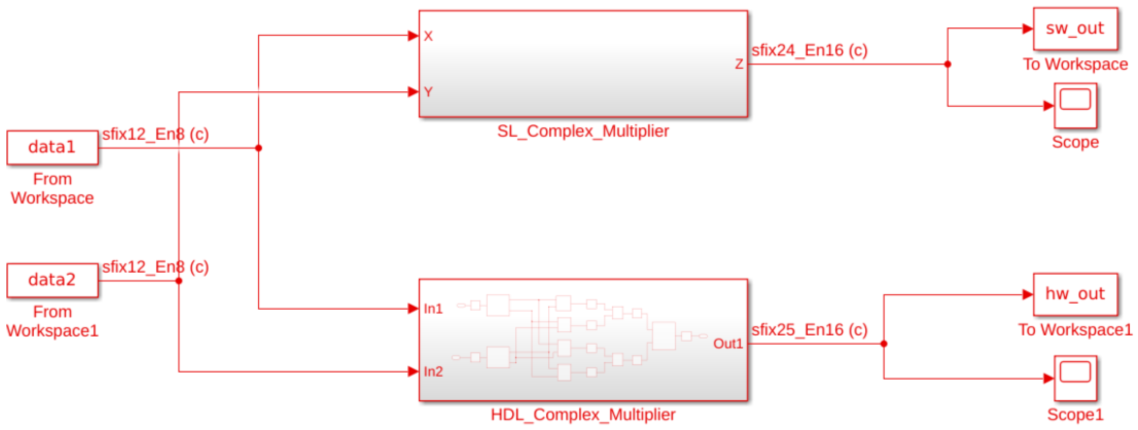
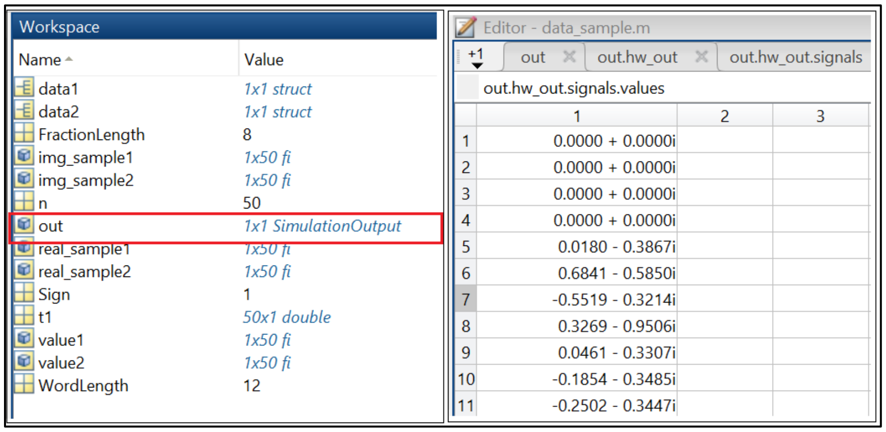


Figure 11: Model after Simulation

2-6. You can view the simulation result either from the workspace or using the Scope. The multiplication results are sent to the workspace by `To Workspace` blocks. You can check the results by checking the `out` variable in the workspace (Figure 12), which is a `SimulationOutput` object. The output from the SL_Complex_Multiplier block is at `out.sw_out.signals.values`, and the output from the HDL_Complex_Multiplier block is at `out.hw_out.signals.values`. Figure 12 shows some of the results from the HDL_Complex_Multiplier block.



Name	Value
data1	1x1 struct
data2	1x1 struct
FractionLength	8
img_sample1	1x50 fi
img_sample2	1x50 fi
n	50
out	1x1 SimulationOutput
real_sample1	1x50 fi
real_sample2	1x50 fi
Sign	1
t1	50x1 double
value1	1x50 fi
value2	1x50 fi
WordLength	12

	1	2	3
1	0.0000 + 0.0000i		
2	0.0000 + 0.0000i		
3	0.0000 + 0.0000i		
4	0.0000 + 0.0000i		
5	0.0180 - 0.3867i		
6	0.6841 - 0.5850i		
7	-0.5519 - 0.3214i		
8	0.3269 - 0.9506i		
9	0.0461 - 0.3307i		
10	-0.1854 - 0.3485i		
11	-0.2502 - 0.3447i		

Figure 12: Check simulation results from workspace

It should be noted that the hardware multiplier model has four stages of delay in the design. That is why the output results show a system latency of four clock cycles—the first four output samples are zero (Figure 12).

2-7. Double click on the scopes to view the simulation results as a graph. The results are shown in Figure 13. The Scope on the left-hand side of Figure 13 shows the multiplication result from the software model, and the one on the right-hand side shows the multiplication result from the hardware model. Since there are four stages of delay blocks in the data path, the output graph of the hardware model also shows a system latency of four clock cycles—the first valid result is generated after four clock cycles.

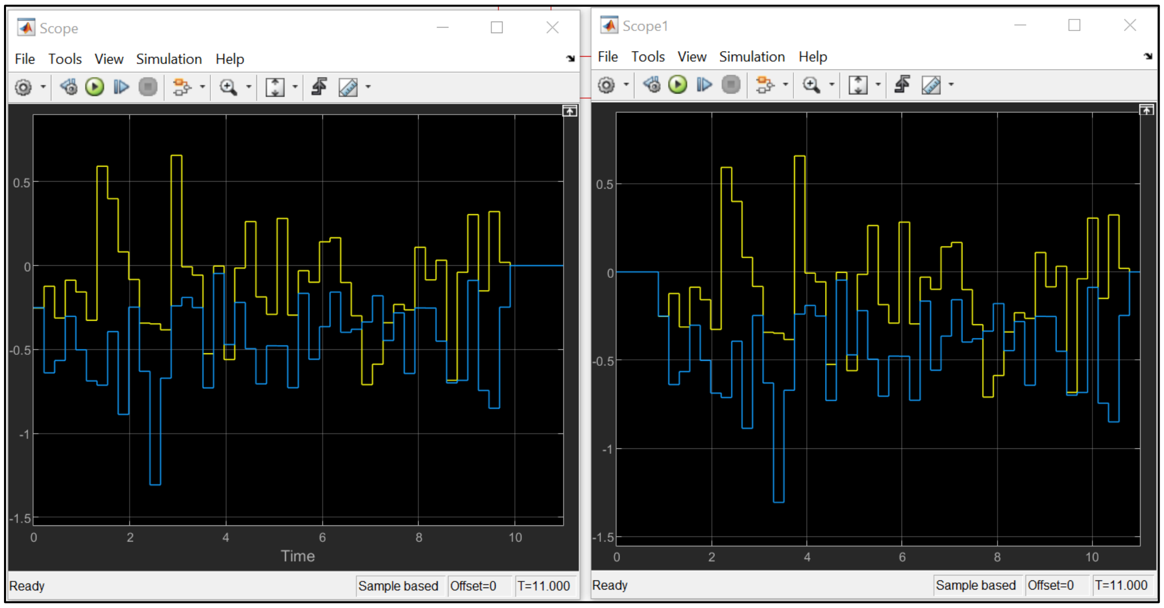


Figure 13: Simulation results as graphs

2-8. Next, we will perform functional verification by checking whether the multiplication results from software and hardware flows are identical. Double click on the SL_Complex_Multiplier block and add one unit of delay at the input and three units of delay at the output to match the system latency of the hardware model. The modified design is shown in Figure 14.

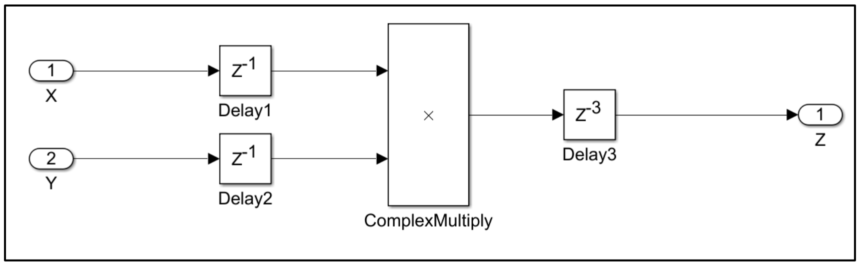
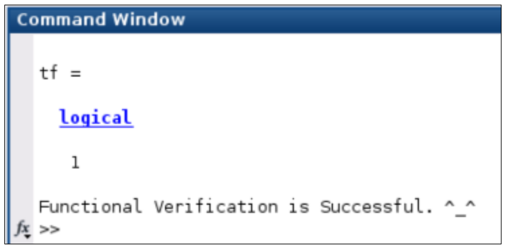


Figure 14: Modified SL_Complex_Multiplier

2-9. Run the 2nd section of the MATLAB script to compare the outputs of the hardware model against the software ones. If the outputs are identical, the functional verification is successful (Figure 15).



```
Command Window

tf =

  logical

   1

Functional Verification is Successful. ^_^
>>
```

Figure 15: Compare results from SW and HW flows

The 3rd section of the MATLAB script provides another example code to compare the outputs of the hardware Simulink model against the outputs of a MATLAB reference model for the same functional verification purpose.

You can also check the scopes and observe that the graphs are identical now. Both graphs should have a 4 unit time delay before the first valid output.

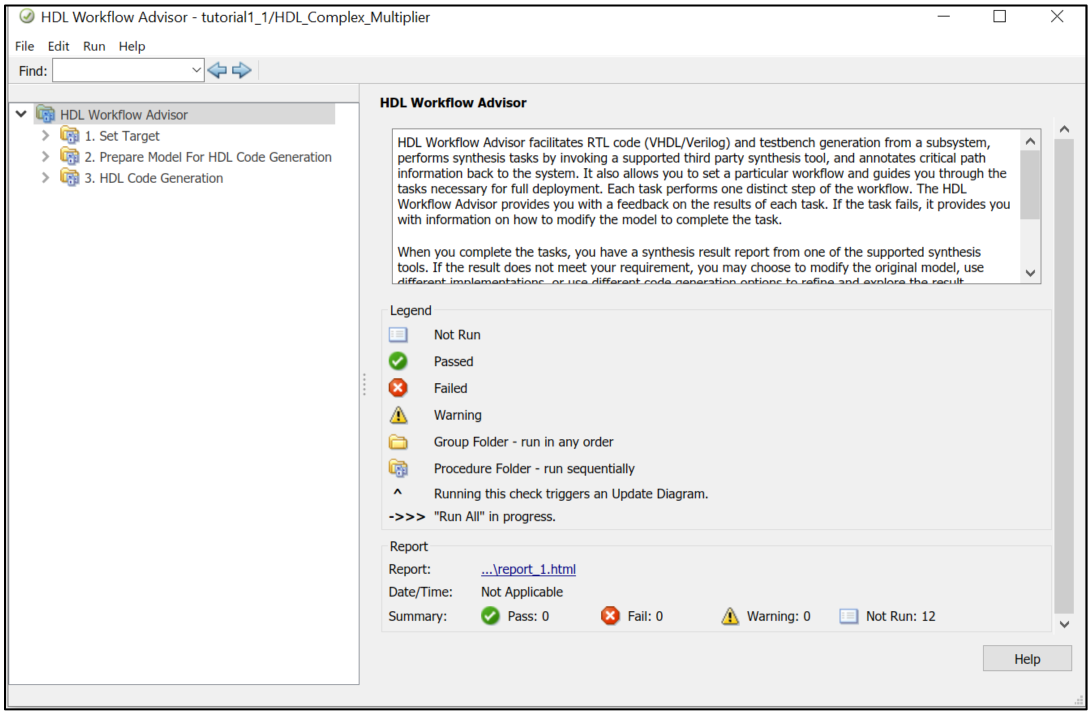
Task 3: Generate VHDL/Verilog Wrapper and Create An Hardware Resource Estimate for an FPGA Target

The task is to implement this design in hardware. This process will synthesize the HDL_Complex_Multiplier subsystem into hardware description language (HDL) codes. The HDL codes can be in either Verilog or VHDL. This process is controlled by the **HDL Workflow Advisor**. With HDL Workflow Advisor, you can synthesize your design without launching the synthesis tool manually.

3-1. First, use `hdlsetuptoolpath` command to add the path for Xilinx Vivado to MATLAB. The command needs to be run once per MATLAB session. You can save it in a MATLAB script.

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado',
'ToolPath','/usr/local/Xilinx-2019.2/Vivado/2019.2/bin');
```

3-2. Right-click the HDL_Complex_Multiplier block, then select HDL Code->HDL Workflow Advisor.



HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier

HDL Workflow Advisor

HDL Workflow Advisor facilitates RTL code (VHDL/Verilog) and testbench generation from a subsystem, performs synthesis tasks by invoking a supported third party synthesis tool, and annotates critical path information back to the system. It also allows you to set a particular workflow and guides you through the tasks necessary for full deployment. Each task performs one distinct step of the workflow. The HDL Workflow Advisor provides you with a feedback on the results of each task. If the task fails, it provides you with information on how to modify the model to complete the task.

When you complete the tasks, you have a synthesis result report from one of the supported synthesis tools. If the result does not meet your requirement, you may choose to modify the original model, use

Legend

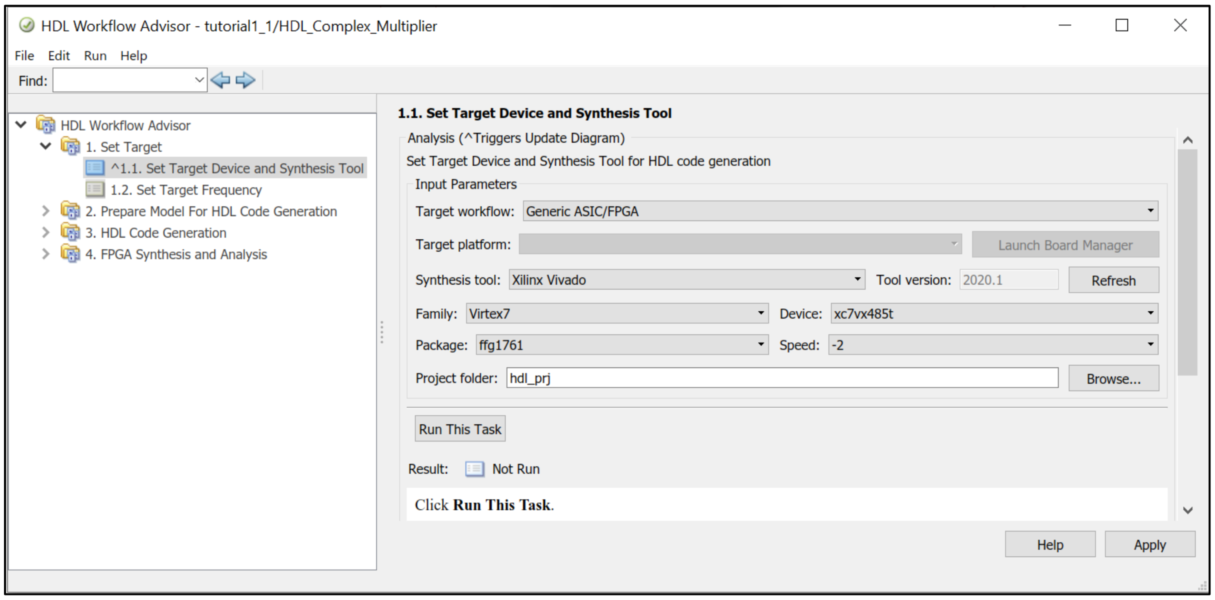
▭	Not Run
✓	Passed
✗	Failed
⚠	Warning
🗀	Group Folder - run in any order
🗀	Procedure Folder - run sequentially
^	Running this check triggers an Update Diagram.
->>>	"Run All" in progress.

Report

Report: ...\report_1.html
Date/Time: Not Applicable
Summary: Pass: 0 Fail: 0 Warning: 0 Not Run: 12

Figure 16: HDL Workflow Advisor

3-3. In 1.1 Set Target Device and Synthesis Tool, set your desired workflow, hardware target, and synthesis tool. For design iterations through synthesis, use the default Generic ASIC/FPGA in Target workflow.



HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier

1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: Generic ASIC/FPGA
Target platform:
Synthesis tool: Xilinx Vivado Tool version: 2020.1
Family: Virtex7 Device: xc7vx485t
Package: ffg1761 Speed: -2
Project folder: hdl_prj

Result: Not Run

Click **Run This Task**.

File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
3. HDL Code Generation
4. FPGA Synthesis and Analysis
Launch Board Manager
Refresh
Browse...
Run This Task
Help
Apply

Figure 17: Set target device and tool

3-4. For FPGA targets: Specify FPGA Device details for which you want to do resource estimation. You can use the FPGA target in Figure 17 as an example.

3-5. After configuration, click “Run This Task”. This button will be clicked at the end of each workflow step.

3-6. Set Target Frequency. The target frequency here can be different from the sample time you use in the Simulink block. This is the FPGA target frequency (HDL clock) that will be used in synthesis. Please use 100 MHz for this example.

In Simulink block, for a single-rate model, 1 sample time in Simulink maps to 1 clock cycle in HDL. You can use a relative mapping (e.g., 1 second in Simulink = one HDL clock cycle) or an absolute mapping (e.g., 10e-9 second in Simulink = one HDL clock cycle), depending on your preference and design requirement.

For more information, please refer to Section 2.1 of the HDL Coder Evaluation Reference Guide.

3-7. In step 2 of the workflow (Prepare Model for HDL Code Generation), select Run All. This step performs some basic checks on the design like checking algebraic loop, sample time, and compatibility of the blocks for HDL generation.

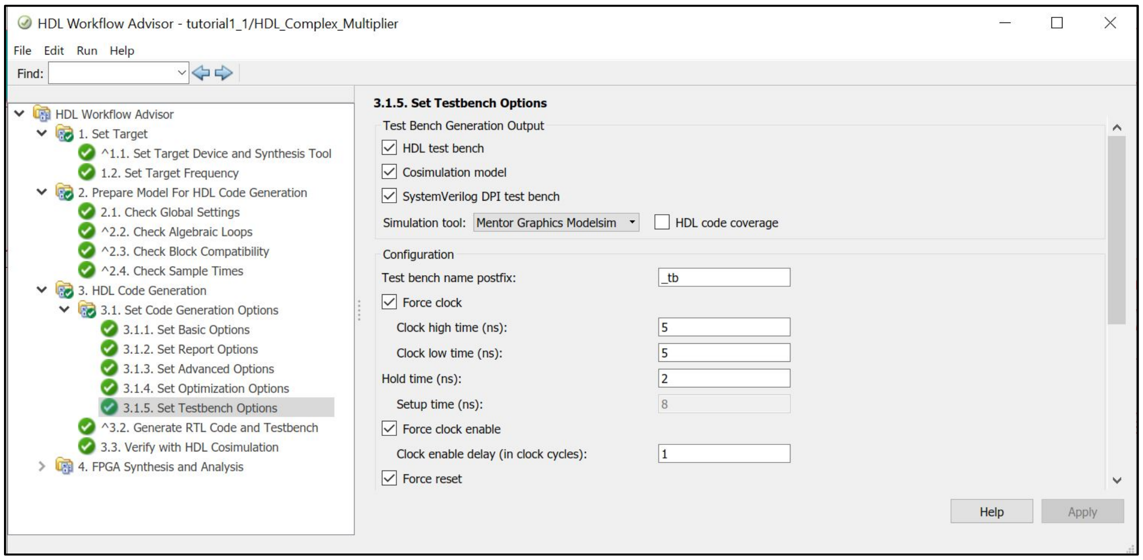
3-8. In step 3.1.1, you can choose the desired HDL for your hardware description. Here, we choose Verilog.

3-9. In step 3.1.2, you can choose to generate optional reports for the HDL code generation. Refer to Section 3.4.1 of the HDL Coder Evaluation Reference Guide for more details.

3-10. In step 3.1.3, you can set advanced options on details of the HDL code generation, such as synchronous or asynchronous, active-low or active-high reset, clock settings, naming conventions, coding style, etc. Here, we use the default.

3-11. In step 3.1.4, you can choose to apply several architectural optimizations, including pipelining and resource sharing, to optimize for performance or area cost. Here, we use the default. Refer to Section 3.3.4 of the HDL Coder Evaluation Reference Guide for more details about the pipelining options.

3-12. In step 3.1.5, Set the Testbench Options, as shown in Figure 18. Then, select 3.1 and click Run All.


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier
File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
3.3. Verify with HDL Cosimulation
4. FPGA Synthesis and Analysis
Help
Apply

3.1.5. Set Testbench Options

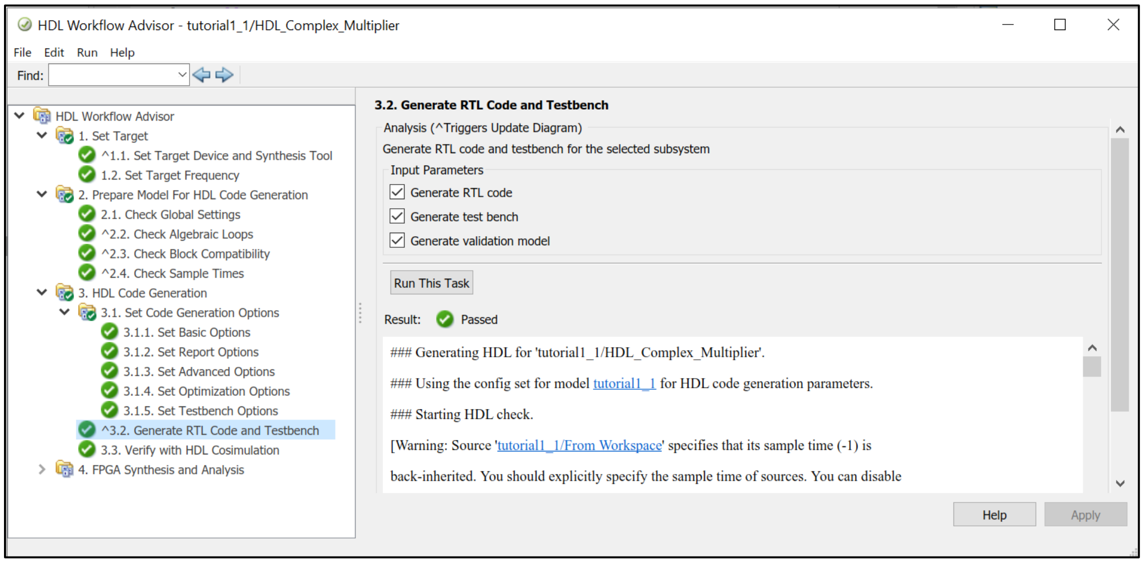
Test Bench Generation Output
- ☑ HDL test bench
- ☑ Cosimulation model
- ☑ SystemVerilog DPI test bench

Simulation tool: Mentor Graphics Modelsim ☐ HDL code coverage

Configuration
- Test bench name postfix: _tb
- ☑ Force clock
- Clock high time (ns): 5
- Clock low time (ns): 5
- Hold time (ns): 2
- Setup time (ns): 8
- ☑ Force clock enable
- Clock enable delay (in clock cycles): 1
- ☑ Force reset

Figure 18: Set testbench option

3-13. In step 3.2 Generate RTL code and Testbench, check the options related to testbench generation, as shown in Figure 19. Click Run this task. Note that a new Simulink model for HDL cosimulation is created and pops up upon the completion of this task.


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier
File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
3.3. Verify with HDL Cosimulation
4. FPGA Synthesis and Analysis
Run This Task
Help
Apply

3.2. Generate RTL Code and Testbench

Analysis (^Triggers Update Diagram)

Generate RTL code and testbench for the selected subsystem

Input Parameters

- [x] Generate RTL code
- [x] Generate test bench
- [x] Generate validation model

Result: Passed

Generating HDL for 'tutorial1_1/HDL_Complex_Multiplier'.

Using the config set for model tutorial1_1 for HDL code generation parameters.

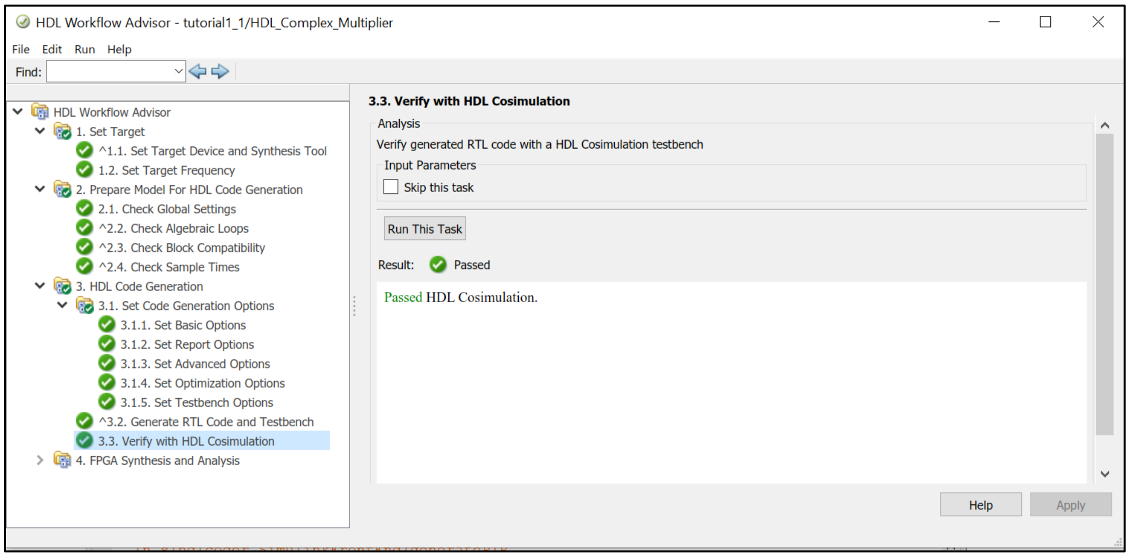
Starting HDL check.

[Warning: Source 'tutorial1_1/From Workspace' specifies that its sample time (-1) is

back-inherited. You should explicitly specify the sample time of sources. You can disable

Figure 19: Generate RTL and Testbench options

3-14. Select step 3.3 and click on Run this task. This step uses the Simulink testbench generated in the previous step to perform HDL cosimulation using ModelSim to verify the functionality of the generated HDL codes. After the cosimulation is finished, a message Passed HDL Cosimulation is displayed (Figure 20). Two graphs also pop up, which show the comparison between the real and imaginary outputs from the Simulink and the HDL simulations, respectively. (Figure 21).


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier
File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
3.3. Verify with HDL Cosimulation
4. FPGA Synthesis and Analysis
Run This Task
Help
Apply

3.3. Verify with HDL Cosimulation

Analysis

Verify generated RTL code with a HDL Cosimulation testbench

Input Parameters

- [ ] Skip this task

Result: Passed

Passed HDL Cosimulation.

Figure 20: HDL Cosimulation

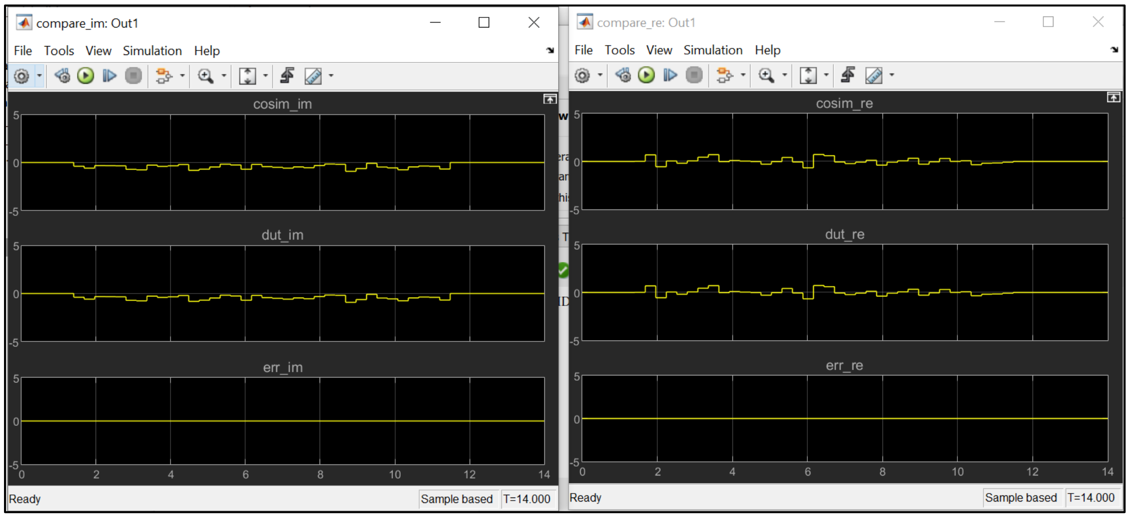
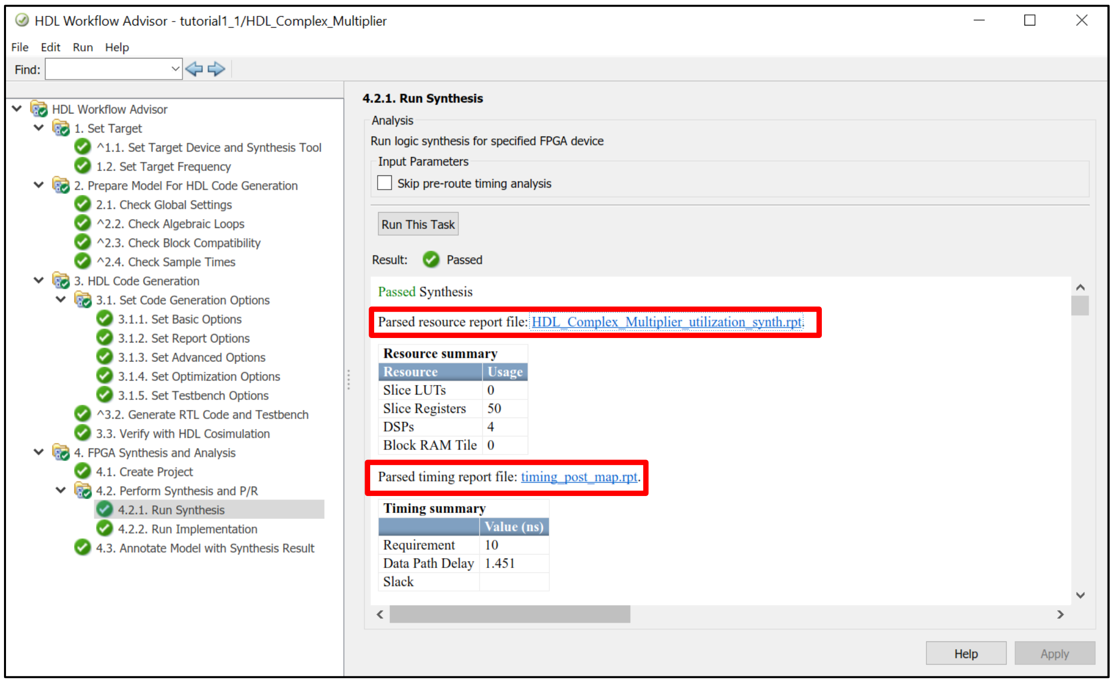


Figure 21: Comparison of cosimulation results

3-15. Click "run all" in step 4 FPGA Synthesis and Analysis. This step creates a Vivado project and performs the synthesis in the background. **Note that Vivado cannot work with a project folder that contains any space in its path.** So make sure you are running your HDL Workflow Advisor under a work folder with no space in its path. Otherwise, the project creation will fail and Vivado run will fail.

When the synthesis run is completed, you can check the post-synthesis resource utilization and timing summaries from the synthesis results (Figure 22). By default, the implementation stage (placement and routing) is skipped. If you want to run the implementation stage after synthesis, uncheck the "Skip this task" option in step 4.2.2.


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier
File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
3.3. Verify with HDL Cosimulation
4. FPGA Synthesis and Analysis
4.1. Create Project
4.2. Perform Synthesis and P/R
4.2.1. Run Synthesis
4.2.2. Run Implementation
4.3. Annotate Model with Synthesis Result
Run This Task
Help
Apply

4.2.1. Run Synthesis

Analysis

Run logic synthesis for specified FPGA device

Input Parameters

☐ Skip pre-route timing analysis

Result: Passed

Passed Synthesis

Parsed resource report file: HDL_Complex_Multiplier_utilization_synth.rpt.

Resource summary

Resource	Usage
Slice LUTs	0
Slice Registers	50
DSPs	4
Block RAM Tile	0

Parsed timing report file: timing_post_map.rpt.

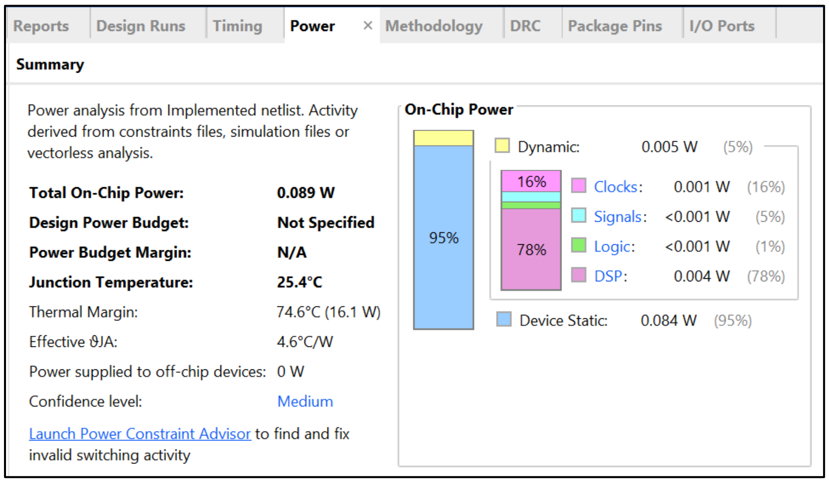
Timing summary

	Value (ns)
Requirement	10
Data Path Delay	1.451
Slack	

Figure 22: Result Summary

3-16. You can click on the parsed resource and timing report files for detailed reports (Figure 22). In the resource report, you can see a list of used and available resources on the FPGA, as well as the percentage of resource utilization. In the timing report, you can check the worst-case setup and hold time slacks and details of the maximum and minimum delay paths. Under the Design Timing Summary section of the timing report, the WNS (ns) value is the worst-case setup time slack, and the WHS (ns) value is the worst-case hold time slack.

3-17. If you need to run FPGA synthesis and implementation manually in Vivado and/or want to see other details of synthesized and/or implemented design and/or perform bit file generation with valid pin-mapping constraints, you can open the created Vivado Project inside the `hdl_prj` folder using the Xilinx Vivado tool to proceed further. You can also check the power report of the design after running the implementation in Vivado (Figure 23).



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.089 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	74.6°C (16.1 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

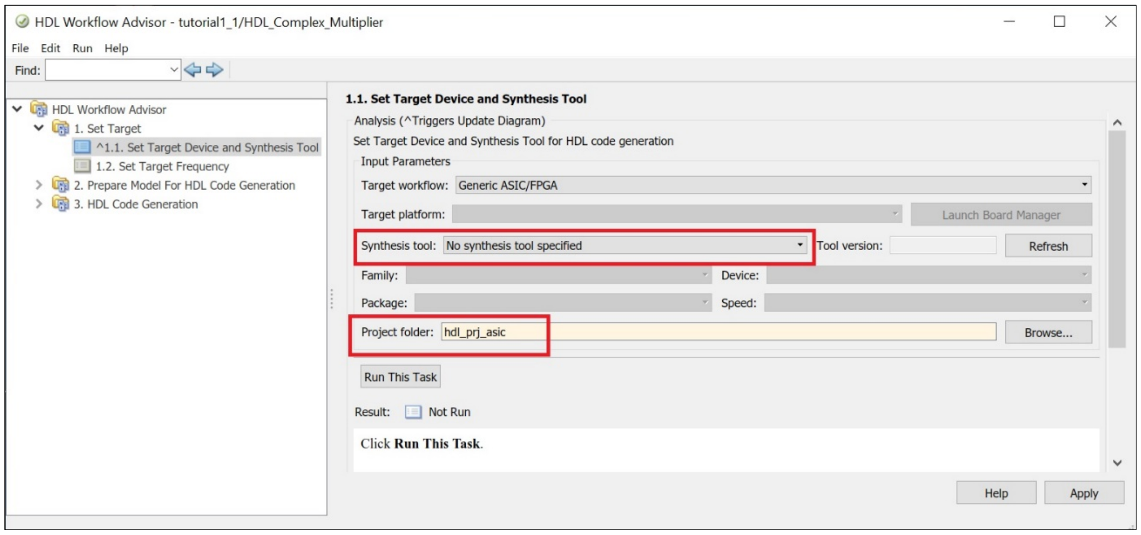
Launch Power Constraint Advisor to find and fix invalid switching activity

Dynamic:	0.005 W	(5%)
Clocks:	0.001 W	(16%)
Signals:	<0.001 W	(5%)
Logic:	<0.001 W	(1%)
DSP:	0.004 W	(78%)
Device Static:	0.084 W	(95%)

Figure 23: Power report from Vivado Project

Task 4: Generate VHDL/Verilog Wrapper and Create Hardware Resource Estimate for an ASIC Target

4-1. For ASIC targets: Launch a new HDL Workflow Advisor. In 1.1 Set Target Device and Synthesis tool, apply the settings as shown in the figure below. Change the folder name to `hdl_prj_asic` so that you don't overwrite the FPGA synthesis results.


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier

File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
3. HDL Code Generation

1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: Generic ASIC/FPGA

Target platform:

Synthesis tool: No synthesis tool specified

Tool version:

Family:

Device:

Package:

Speed:

Project folder: hdl_prj_asic

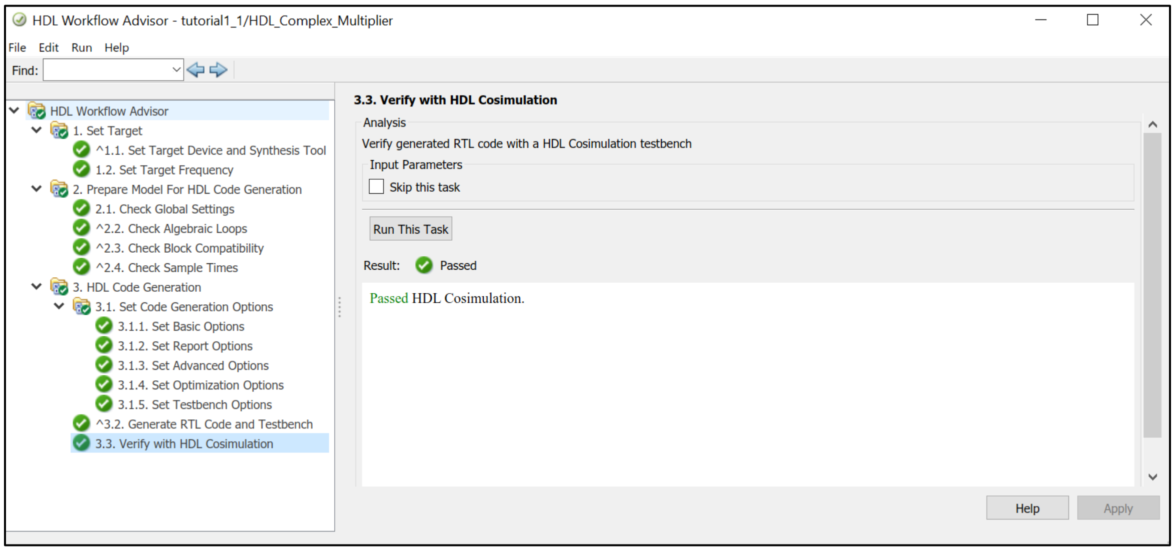
Result: Not Run

Click **Run This Task**.

Figure 24: ASIC Target

4-2. In step 1.2, set the target frequency to zero. Follow the rest steps as in the FPGA target flow. The HDL cosimulation shall pass (Figure 25).

Unlike the FPGA target flow, there will be no synthesis step in the HDL workflow advisor since the ASIC synthesis will be performed using a separate tool—Synopsys Design Compiler. Sine no synthesis tool is specified, the target frequency setting in this step is useless. We will specify the target frequency for synthesis later on using a Synopsys Design Compiler constraint file (.sdc) instead.


HDL Workflow Advisor - tutorial1_1/HDL_Complex_Multiplier

File Edit Run Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis Tool
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
3.3. Verify with HDL Cosimulation

3.3. Verify with HDL Cosimulation

Analysis

Verify generated RTL code with a HDL Cosimulation testbench

Input Parameters

☐ Skip this task

Result: Passed

Passed HDL Cosimulation.

Figure 25: HDL Workflow Advisor for the ASIC flow

4-3. Download the provided template_dc.tcl and template_dc.sdc and place them under the same folder where the generated VHDL/Verilog files are (under hdl_prj_asic/hdlsrc/tutorial1_1_sol). And rename the two files with the name of the HDL Simulink model as <HDL_Name>_dc.tcl and <HDL_Name>_dc.sdc, where the <HDL_Name> is `HDL_Complex_Multiplier` in this Tutorial.

4-4. Modify following lines in the <HDL_Name>_dc.tcl.

set DESIGN_NAME top_level_module_name -> set DESIGN_NAME <HDL_Name>

source “/path/to/a_dc_setup_file.setup” -> source “<specify the full path to the chosen DC setup file>”

4-5. The design compiler (DC) setup file defines the standard cell libraries for timing and power analysis and the process, voltage, and temperature (PVT) corner for setup and hold time analysis, respectively. To get you started with ASIC synthesis, we have provided three different DC setup files (DC_setup_28nm_1v.setup, DC_setup_28nm_0p85v.setup and DC_setup_28nm_0p75v.setup). These setup files are configured to load the Synopsys 28nm generic library that includes standard VT (SVT), low VT (LVT), and high VT (HVT) cells. This will allow you to run leakage optimization in DC, which first synthesizes your design using HVT cells only and then insert SVT/LVT cells into the critical paths to improve timing (all with sizing optimization). To allow you to explore voltage scaling on your design, the three setup file is configured for a supply voltage of 1V (typical voltage), 0.85V (middle voltage), and 0.75V (low voltage), respectively. So, the choice of the setup file depends on the supply voltage that you want to run your design under. For this tutorial, we choose the 1V setup file DC_setup_28nm_1v.setup. Specify the setup file in the last line of Step **4-4**.

You can change or modify the setup file to run synthesis again and observe the difference in results.

4-6. The provided .sdc file defines a 800MHz clock as the system clock. To synthesize the design for a different clock frequency, you will need to modify the clock period defined in the .sdc file. Synthesize the design at 800MHz, 400 MHz, and 200MHz, respectively, and notice the differene in results due to circuit-level optimizations.

4-7. Type following command in the terminal to do ASIC synthesis.

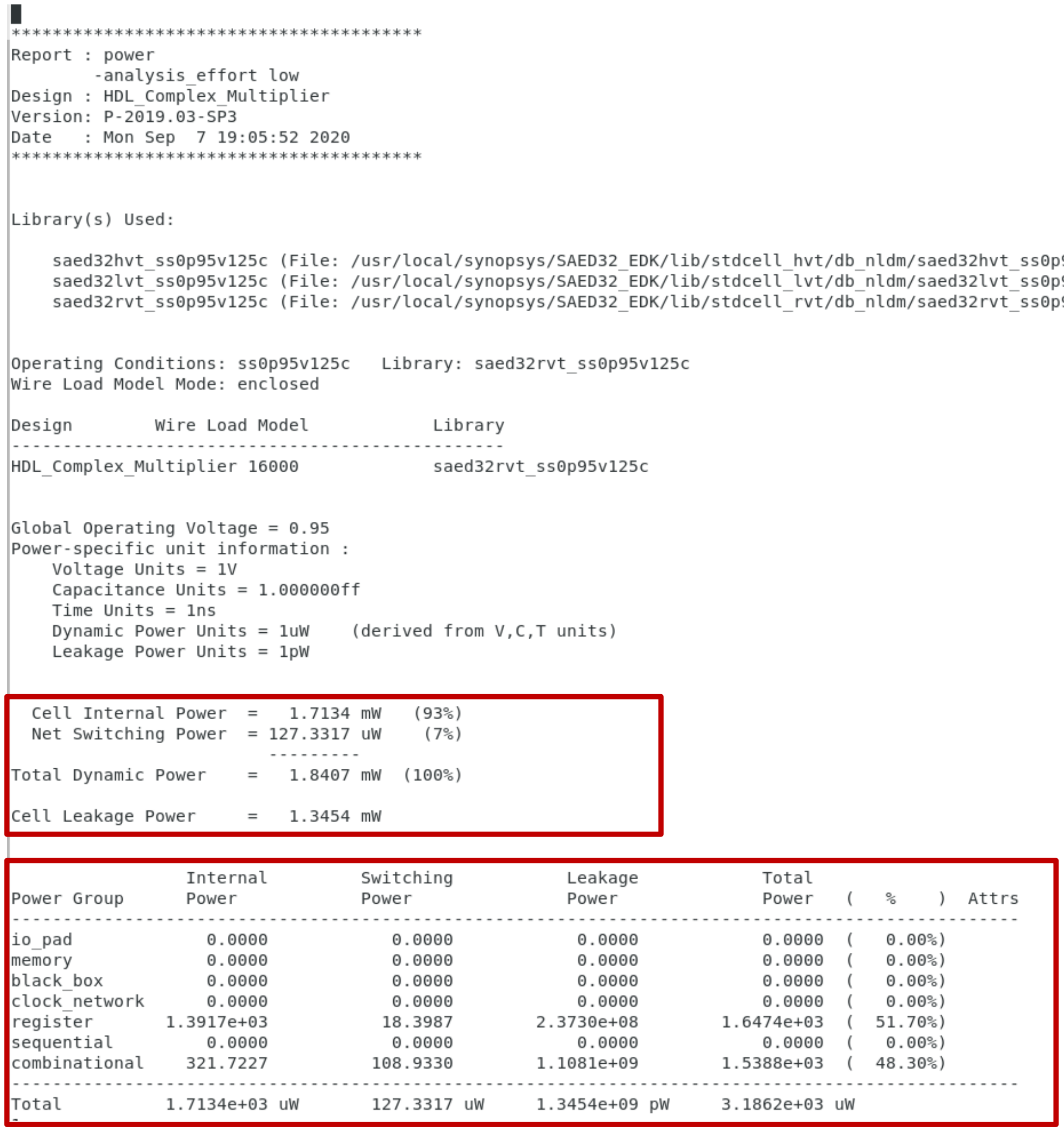
```
$ dc_shell –f <Project_Name>_dc.tcl
```

4-8. The reports on Power, Area, and Timing can be found under the “Report” folder created in the project folder.

4-9. Figure 26 shows an example power report (.mapped.power.rpt). You can observe the total dynamic power and cell leakage power from the report. The breakdown of dynamic power into cell internal power and switching power and their percentages are shown. The contribution of the sequential and combinational logic to the total power is shown in the table. The unit of dynamic and leakage power is in micro-watts (mW) and picowatts (pW), respectively.

The switching power of a driving cell is the power dissipated by the charging and discharging of the load capacitance at the output of the cell. The total load capacitance at the output of a driving cell is the sum of the net and gate capacitances on the driving output.

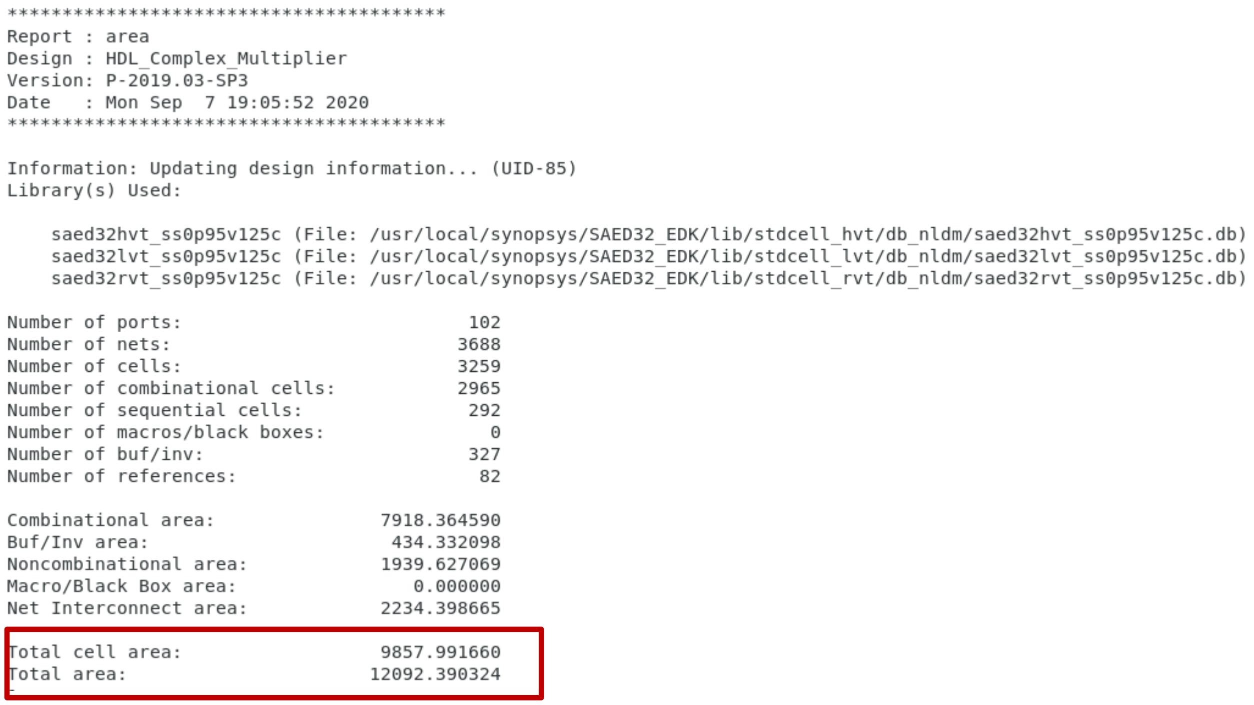
Internal power is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances internal to the cell. Internal power also includes power dissipated by a momentary short circuit between the P and N transistors of a gate, called short-circuit power.



```
****************************************
Report : power
        -analysis_effort low
Design : HDL_Complex_Multiplier
Version: P-2019.03-SP3
Date   : Mon Sep  7 19:05:52 2020
****************************************


Library(s) Used:

    saed32hvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_hvt/db_nldm/saed32hvt_ss0p95v125c.db)
    saed32lvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_lvt/db_nldm/saed32lvt_ss0p95v125c.db)
    saed32rvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_rvt/db_nldm/saed32rvt_ss0p95v125c.db)


Operating Conditions: ss0p95v125c   Library: saed32rvt_ss0p95v125c
Wire Load Model Mode: enclosed

Design        Wire Load Model            Library
------------------------------------------------
HDL_Complex_Multiplier 16000             saed32rvt_ss0p95v125c


Global Operating Voltage = 0.95
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =   1.7134 mW   (93%)
  Net Switching Power  = 127.3317 uW    (7%)
                         ---------
Total Dynamic Power    =   1.8407 mW  (100%)

Cell Leakage Power     =   1.3454 mW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad              0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory              0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box           0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network       0.0000            0.0000            0.0000            0.0000  (   0.00%)
register         1.3917e+03           18.3987        2.3730e+08        1.6474e+03  (  51.70%)
sequential          0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational     321.7227          108.9330        1.1081e+09        1.5388e+03  (  48.30%)
--------------------------------------------------------------------------------------------------
Total            1.7134e+03 uW       127.3317 uW     1.3454e+09 pW     3.1862e+03 uW
```

Figure 26: Example Power Report

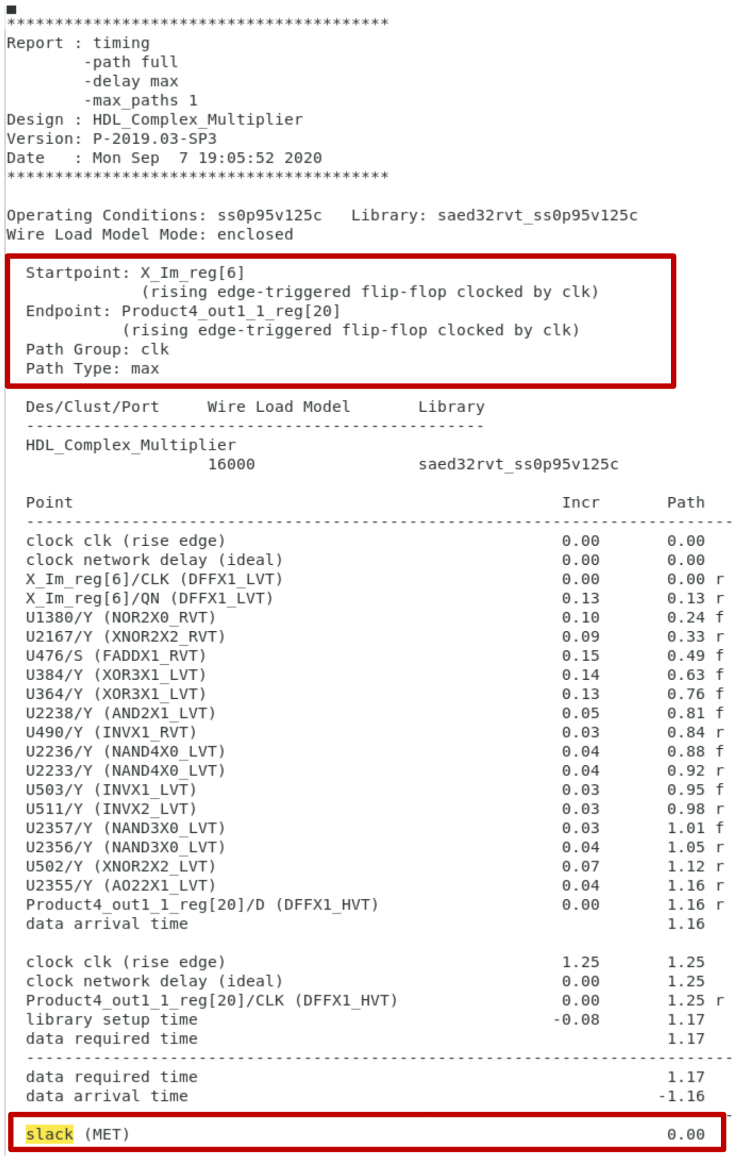
4-10. Figure 27 shows an example area report (.mapped.area.rpt). From the area report, you can see the number of ports, nets, cells, and buffers in your design. You can also observe the total area and the breakdown of the area into different components of the circuits. The unit of area is square micronmeter (μm^2).



```
****************************************
Report : area
Design : HDL_Complex_Multiplier
Version: P-2019.03-SP3
Date   : Mon Sep  7 19:05:52 2020
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    saed32hvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_hvt/db_nldm/saed32hvt_ss0p95v125c.db)
    saed32lvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_lvt/db_nldm/saed32lvt_ss0p95v125c.db)
    saed32rvt_ss0p95v125c (File: /usr/local/synopsys/SAED32_EDK/lib/stdcell_rvt/db_nldm/saed32rvt_ss0p95v125c.db)

Number of ports:                          102
Number of nets:                          3688
Number of cells:                         3259
Number of combinational cells:           2965
Number of sequential cells:               292
Number of macros/black boxes:               0
Number of buf/inv:                        327
Number of references:                      82

Combinational area:               7918.364590
Buf/Inv area:                      434.332098
Noncombinational area:            1939.627069
Macro/Black Box area:                0.000000
Net Interconnect area:            2234.398665

Total cell area:                  9857.991660
Total area:                      12092.390324
```

Figure 27: Example area report

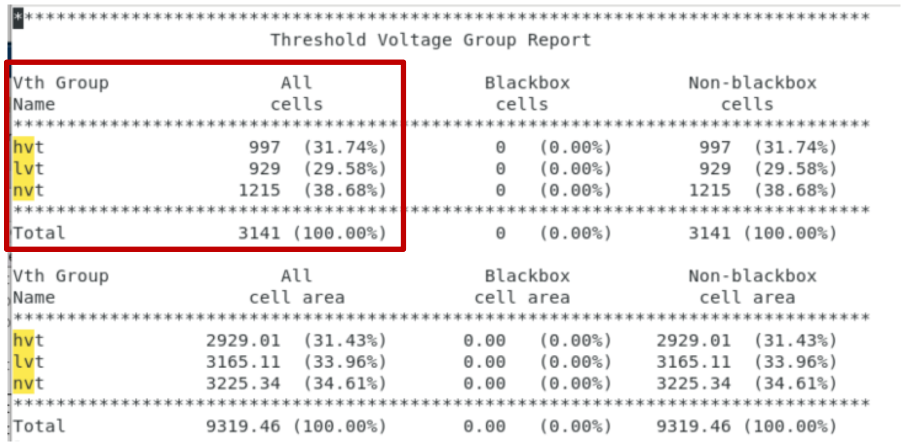
4-11. Figure 28 shows an example timing report (.mapped.timing.rpt). You can observe the critical path of your design and whether the timing requirement of the critical path is satisfied with a positive or zero slack.



```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : HDL_Complex_Multiplier
Version: P-2019.03-SP3
Date   : Mon Sep  7 19:05:52 2020
****************************************

Operating Conditions: ss0p95v125c   Library: saed32rvt_ss0p95v125c
Wire Load Model Mode: enclosed

  Startpoint: X_Im_reg[6]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: Product4_out1_1_reg[20]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  HDL_Complex_Multiplier
                     16000                 saed32rvt_ss0p95v125c

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock clk (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  X_Im_reg[6]/CLK (DFFX1_LVT)                             0.00       0.00 r
  X_Im_reg[6]/QN (DFFX1_LVT)                              0.13       0.13 r
  U1380/Y (NOR2X0_RVT)                                    0.10       0.24 f
  U2167/Y (XNOR2X2_RVT)                                   0.09       0.33 r
  U476/S (FADDX1_RVT)                                     0.15       0.49 f
  U384/Y (XOR3X1_LVT)                                     0.14       0.63 f
  U364/Y (XOR3X1_LVT)                                     0.13       0.76 f
  U2238/Y (AND2X1_LVT)                                    0.05       0.81 f
  U490/Y (INVX1_RVT)                                      0.03       0.84 r
  U2236/Y (NAND4X0_LVT)                                   0.04       0.88 f
  U2233/Y (NAND4X0_LVT)                                   0.04       0.92 r
  U503/Y (INVX1_LVT)                                      0.03       0.95 f
  U511/Y (INVX2_LVT)                                      0.03       0.98 r
  U2357/Y (NAND3X0_LVT)                                   0.03       1.01 f
  U2356/Y (NAND3X0_LVT)                                   0.04       1.05 r
  U502/Y (XNOR2X2_LVT)                                    0.07       1.12 r
  U2355/Y (AO22X1_LVT)                                    0.04       1.16 r
  Product4_out1_1_reg[20]/D (DFFX1_HVT)                   0.00       1.16 r
  data arrival time                                                  1.16

  clock clk (rise edge)                                   1.25       1.25
  clock network delay (ideal)                             0.00       1.25
  Product4_out1_1_reg[20]/CLK (DFFX1_HVT)                 0.00       1.25 r
  library setup time                                     -0.08       1.17
  data required time                                                 1.17
  --------------------------------------------------------------------------
  data required time                                                 1.17
  data arrival time                                                 -1.16
  --------------------------------------------------------------------------
  slack (MET)                                                        0.00
```

Figure 28: Example Timing Report

4-12. Figure 29 shows an example threshold voltage group report (.mapped.vth.rpt), which shows the percentage breakdown of the logic gates (standard cells) in each threshold flavor used in the synthesized design.



```
*****************************************************************************
                      Threshold Voltage Group Report

Vth Group                 All                Blackbox             Non-blackbox
Name                     cells                 cells                 cells
*****************************************************************************
hvt                   997   (31.74%)        0   (0.00%)          997   (31.74%)
lvt                   929   (29.58%)        0   (0.00%)          929   (29.58%)
nvt                  1215   (38.68%)        0   (0.00%)         1215   (38.68%)
*****************************************************************************
Total                3141 (100.00%)         0   (0.00%)         3141 (100.00%)

Vth Group                 All                Blackbox             Non-blackbox
Name                   cell area            cell area              cell area
*****************************************************************************
hvt               2929.01   (31.43%)     0.00   (0.00%)      2929.01   (31.43%)
lvt               3165.11   (33.96%)     0.00   (0.00%)      3165.11   (33.96%)
nvt               3225.34   (34.61%)     0.00   (0.00%)      3225.34   (34.61%)
*****************************************************************************
Total             9319.46 (100.00%)      0.00   (0.00%)      9319.46 (100.00%)
```

Figure 29: Example Timing Report

Grading Rubrics

Criteria	Inspection Item	Point
Design Correctness	Completed model of complex multiplier using HDL Coder blocks.	20
	Successful verification of HW model against the SW or MATLAB model.	20
FPGA Synthesis	HDL code generation for FPGA is successful.	10
	FPGA design synthesis is successful.	20
ASIC Synthesis	HDL code generation for ASIC is successful.	10
	Correct timing, power and area reports are generated.	20
	Total	**100**

Due Date

Refer to Canvas.

Instructions on Submission

1. A submission should include the MATLAB project folder containing the following items:
   - The completed Simulink model containing the HDL_Complex_Multiplier design.
   - The MATLAB script `data_sample_fp.m`.
   - The folder (hdl_prj) containing the Vivado Project and the HDL codes generated for FPGA synthesis and FPGA synthesis results.
   - The folder (hdl_prj_asic) containing the Design Compiler run files and the HDL codes generated for the ASIC Synthesis and all ASIC synthesis reports.
2. Compress the files and folders into a single zip archive file named **cen571-firstname_lastname-tut1.zip**. Note that any other codes or any temporary build files should not be included in the submission.
3. Submit the zip archive to Canvas by the due date and time.
4. Failure to follow these instructions may cause the TA or instructor to deduct points while grading your assignment.