ECE 4156/6156 Hardware-Oriented Security and Trust

Spring 2026

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Lab 3, 100 pts.

Due Friday, March 13 prior to 11:55pm

(Please turn in homework electronically on Canvas)

Lab 3

This lab exercise involves evaluating the performance and resource utilization of three different configurations of the Keccak implementation. The first part of the lab focuses on simulating VHDL code in Model Sim and monitoring the throughput of each core. The second part of the lab involves experimenting with different configurations of the midcore and evaluating the impact on throughput. Finally, the lab requires you to perform timing analysis and resource utilization analysis on the three different cores using Quartus.

You will need to analyze and interpret the data they collect to draw conclusions about the performance and resource utilization of each core configuration.

Through the lab, you will become familiar with key concepts such as throughput, resource utilization, and timing analysis. Please type your answers to the questions in this lab into a lab report.

I. VHDL/ DE-10 Help

There is plenty of documentation available on how to write good VHDL. Some good simple examples can be found here. Some good YouTube videos introducing VHDL basics can be found here.

SHA3 documentation: https://keccak.team/obsolete/Keccak-implementation-3.1.pdf

II. SHA3 Code Hierarchy

The SHA3 code hierarchy is as follows:

- SHA3:
  - High_speed_core
    - Keccak.vhd
    - Keccak_buffer.vhd
    - Keccak_globals.vhd
    - Keccak_round.vhd
    - Keccack_round_constants_gen.vhd
    - Tb_keccak.vhd
    - Tb_keccack_permutation.vhd
  - Low_area_copro
    - Fsm.vhd
    - Keccack_copro.vhd
    - Keccack_globals.vhd
    - pe.vhd
    - System_mem.vhd
    - Tb_keccack_copro.vhd
  - Mid_range_core
    - Chi_iota_theta.vhd
    - Keccack.vhd
    - Keccack_globals.vhd
    - Keccack_round_constants_gen.vhd
    - Rho_pi.vhd
    - Tb_keccack.vhd
  - test_vectors
    - Keccack_in.txt
    - Keccack_ref_out.txt
    - Perm_in.txt
    - Perm_ref_out.txt
    - KeccackReference_vhdl.vcproj
    - Sources (C Folder)
  - Readme.txt

Inside the "high_speed_core" directory, you will find the VHDL files for the high-performance core and two test benches. The test bench files are prefixed with "tb_" and serve different purposes.

The first test bench is dedicated to testing the permutation function. It reads input data from "perm_in.txt" in the "test_vectors" directory and produces output data in the file "perm_out_high_speed_core_vhdl.txt". The output of this VHDL test bench is compared with the "perm_ref_out.txt" file generated by the reference code to ensure correct functionality.

The second test bench reads input data from the "keccak_in.txt" file and writes the resulting data to the "keccak_out_high_speed_vhdl.txt" file. This output file sis compared with the "keccak_ref_out.txt" file to ensure correct functionality.

The "low_area_copro" directory contains vhdl files for a low area implemenation and one test bench for verifying the correct functionality of the permutation function. Note that this coprocessor does not include the logic for performing the operations related to the XOR of the input message.

Finally, the "Mid_range_core" directory offers a medium area core along with customization options. You can modify the number of blocks and rounds by changing the "keccak_globals.vhd" file's lines 32 to 58 and appropriately commenting them. You can also adjust the code responsible for the core functionality by modifying the "keccak.vhd" file's lines 353 to 421 and properly commenting them.

The "test_vectors" directory contains a modified version of the reference code. Specifically, the "KeccakPermutationReference.c" and "KeccakPermutationReference.h" files have been modified to make it easier to produce test vectors that can be easily read in the VHDL test benches. This modification allows for more efficient testing of the hardware implementation of SHA-3.

III. ModelSim Simulation of VHDL Code

Your first task is to run the provided files for each of the cores and understand the test bench. To simulate the code, follow these steps:

1. Open ModelSim.
2. Create a new project by clicking on File --> New --> Project...
3. Verify that the project location is set to an empty directory (ex: "lab3/Sha3/high_speed_core/") and give the project a name, such as "highspeed". Click OK.
4. Add all VHDL files to the project by clicking on Add Existing File --> Browse..., selecting all VHDL files in the directory, and clicking Open --> OK. Once done adding all VHDL files, click Close.
5. Compile the design files by clicking on Compile --> Compile Order, then clicking on Auto Generate and pressing OK. Check the transcript window to ensure that all files were successfully compiled.

6. Start the simulation by clicking on Simulate --> Start Simulation...
7. In the Start Simulation window, ensure that you are on the design tab, expand the work library, and choose the test bench named "keccak_tb" for this code, then click OK. ModelSim will change the view to simulation mode and a couple of other windows will show up.
8. Add signals of interest to the wave window to monitor their changes as the simulation proceeds. Be sure to change the radix format to hexadecimal.
9. Run the simulation.
10. Take a screenshot of one clear test case from your custom input of the simulation.
11. Calculate the time taken to complete one full computation cycle, i.e., from one start high to the other.
12. Find the clock speed.
13. Calculate the throughput in terms of computations completed per second (e.g., outputs generated per second). Helpful equation: T = clock period, C = number of clock cycles required to produce an output => Throughput = 1/(C*T)
14. Create a file named "/test_vectors/keccak_test.txt" and add a custom hexadecimal input string (preferably a random or at least nonzero value). Run the simulation on it and take a screenshot of the same.
15. Repeat steps 6-14 for both test benches for the High_speed_core, as well as the low_area_core and the mid_core with default configuration. You only need one screenshot for these cases compared to the High_speed_core case.
16. Note the throughput values for each test bench.

Note that the names will differ for all three test benches.

IV. Changing and Running in Mid Core

As described in section II of the instructions, the mid core of the Keccak implementation can be modified to run different blocks of slices and to parameterize the core with different numbers of blocks. To experiment with these different configurations, you can follow these steps:

1. Comment out the existing Keccak globals 32 (default) and Keccak 32 (default) configuration in the "keccak_globals.vhd" and “keccak.vhd” files, respectively, by placing two dashes (--) at the beginning of the line (the code contains comments for the sets of constants that should be uncommented in keccak_globals.vhd and keccak.vhd).
2. Uncomment the configuration for Keccak 16 and Keccak globals 32 by removing the two dashes at the beginning of the line.
3. Compile the design files in ModelSim by following the same steps outlined earlier in the instructions.
4. Run the simulation for the new configuration by selecting the appropriate testbench ("midcore_tb") and monitoring the throughput as before.

5. Repeat the process with the Keccak 16 and Keccak globals 16 configurations by uncommenting the configuration for Keccak 16 and Keccak globals 16 in the VHDL.
6. Repeat the process for the Keccak 2 and Keccak globals 2 configuration by commenting out the Keccak 16 and Keccak globals 16 configuration and uncommenting the Keccak 2 and Keccak globals 2 configuration in the VHDL.

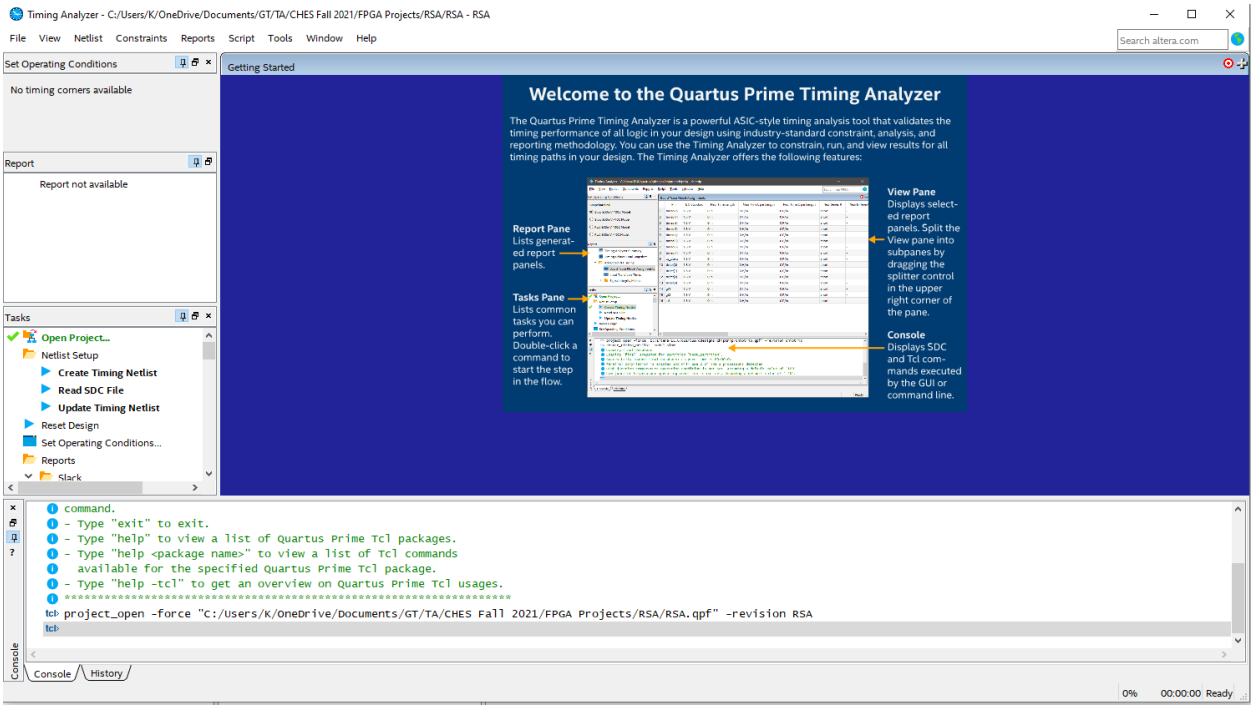
As you run each configuration and monitor the throughput (keeping the testbench clock period constant), you should take note of the changes you see in the simulation results. Compare the throughput of each configuration to the default configuration to see how the changes affect performance. In your report, you will be asked to explain your observations and provide insights into the impact of these parameterizations on the Keccak implementation.

V. Synthesizing SHA-3 in Quartus

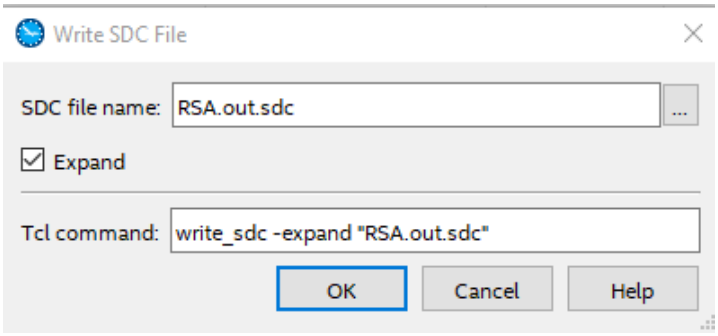
In this section we will obtain synthesis results for all three cores. We will not be actually loading the design onto the DE-10 Board.

For each core, create a project in Quartus as in the previous labs, with the appropraite keccak file as the top-level file.

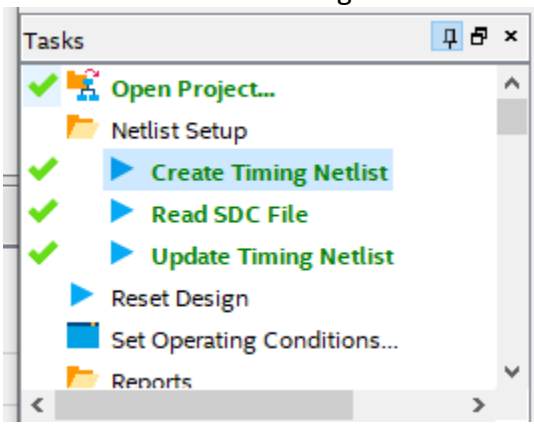
- Make sure you choose the correct FPGA as your target device.
- Next compile design with the default options and wait for the task to be completed.
- Now open **Tools -> Timing Analyzer**. You will see the below screen



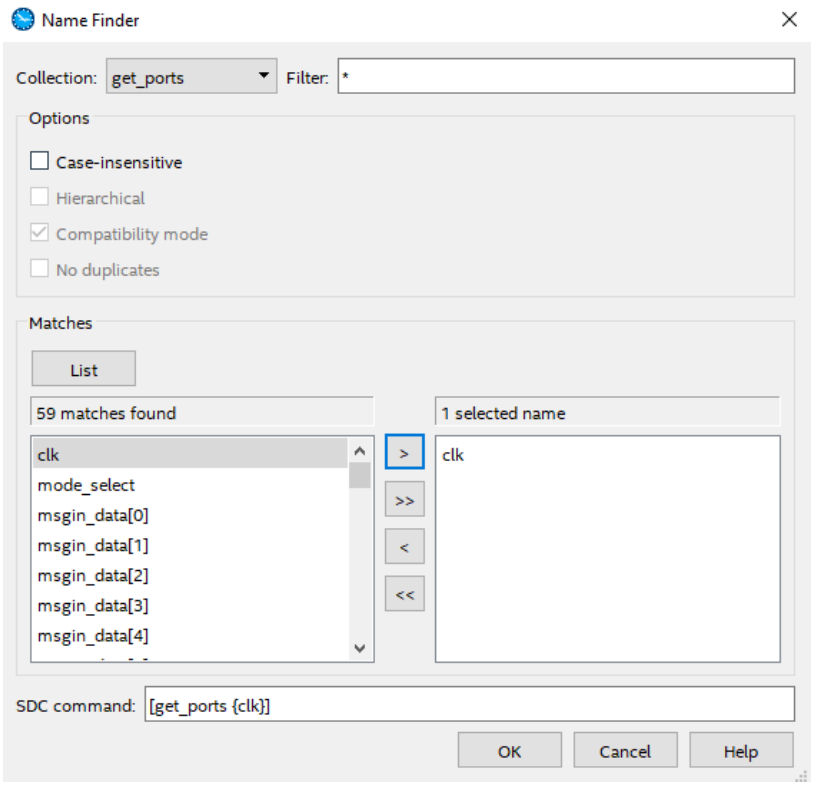
- Double click “**Create Timing Netlist**” on the left under the task bar. It should turn green.
- Run **Constraints -> Write SDC File**. Press “OK” on the opened dialogue box.



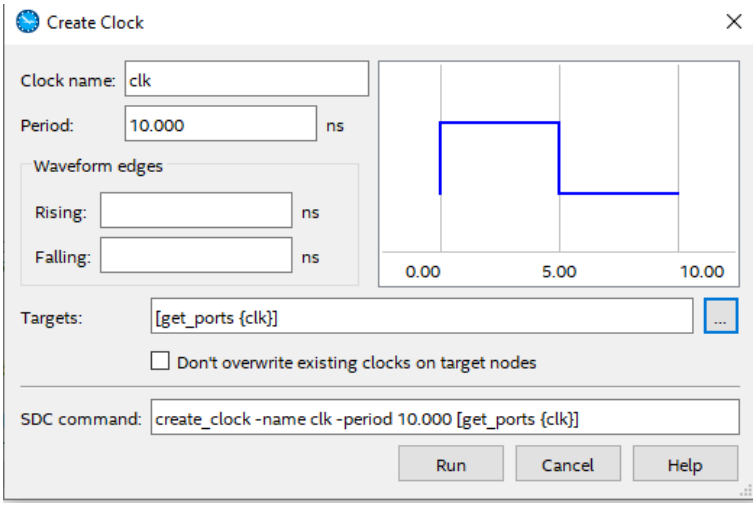
- You should now see the following in the Tasks window



- Now run **Constraints -> Create Clock.** In the pop-up window, put “clk” for the Clock Name and hit the “…” next to “Targets”. On the new pop up window click “list” and move “clk” to the right as in the example shown below. Hit OK.

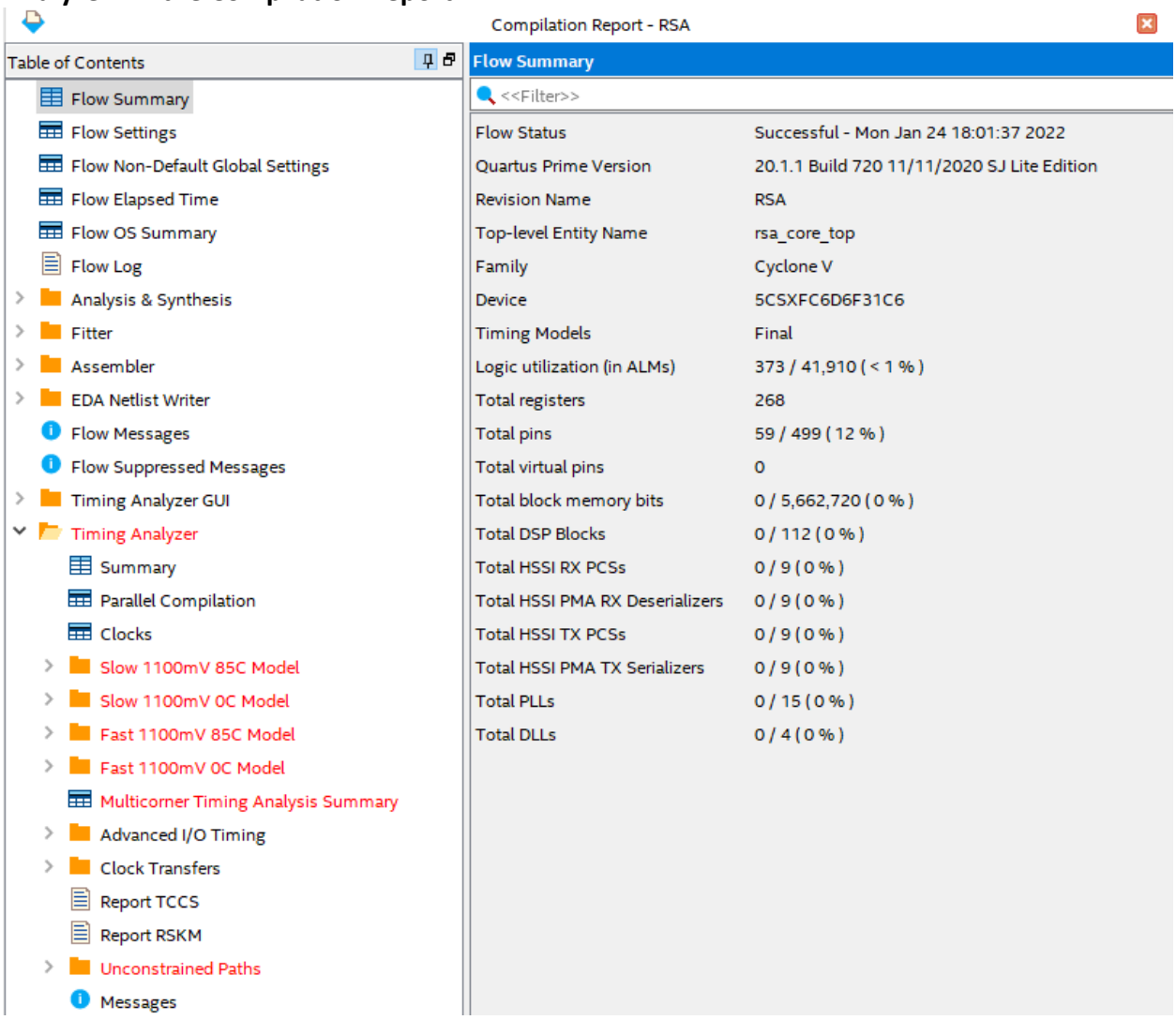


- Now hit run in the below window



- Double click “Update Timing Netlist” in the Tasks window of the Timing Analyzer.
- Back in the main Quartus screen, click the arrow next to Timing Analysis in the Tasks window and open “Edit Settings”. In the Settings window, click the “…” and add the .out.sdc file you just created to the project.
- Now re-run the full Compilation task in Quartus. After the compilation is complete look for the Timing Analysis results. The results can be seen in the “Timing Analyzer” section of the Compilation Report

- Next, find the best clock period at which your new design runs by changing your clock constraint. **Report the new timing values shown in the "Clocks" tab inside "Timing Analyzer" in the Compilation Report.**



- In the above example, you can see that the four Models (Slow 85C, Slow 0C, Fast 85C, Fast 0C) are in the red. This is due to failing to meet timing requirements. They will be in black when the timing constraints are met.
- Adjust the clock frequency and rerun the Timing Analysis to try to find the maximum clock frequency. The clock frequency for the Timing Analysis can be adjusted by editing the SDC file created by the Timing Analyzer. This file is located in your project folder with an .out.sdc extension. In the file you will see the following line.

```
38  #**************************************************************
39  # Create Clock
40  #**************************************************************
41
42  create_clock -name {clk} -period 1.000 -waveform { 0.000 0.500 } [get_ports {clk}]
43
44
```

- The 1.000 after period is the clock period in ns. The numbers in the braces define the duty cycle of the clock. We want a clock with equal high and low times per cycle, so the numbers in the braces should be {0, N/2} for a clock period of N. To change this value to something else, such as 5.000, the new line will be "..... -period 5.000 -waveform { 0.000 2.5000 } …."
- o Save the file. You can now rerun just the "Timing Analysis" task without needing to redo full compilation.
- Report the fastest clock achieved and take a screenshot showing the contents of the Messages tab of the "Timing Analyzer" under the compilation report.
- Take a screenshot of the resource usage summary

Repeat this for all the three cores, and the mid-range core should have the default configuration I.e. 32, 32.

Then, complete synthesis for the mid-range core with the keccak and keccak_global parameters as shown in the second table in Section VI ({16, 32}, {16, 16}, and {2, 2}).

With the fastest clock frequency, find the throughput for each core.

VI. Comparing outputs from simulation and synthesis

In this section, we will be comparing the throughput results obtained from the previous ModelSim simulations and the timing analysis generated by Quartus.

To facilitate the comparison, create a similar table in the report.

Synthesis/Simulation	Core	Clk_speed	Clk Cycles per Computation	Throughput (Computations per second)
Simulation	High_Speed_perm			
Synthesis	High_Speed			
Simulation	High_Speed			
Synthesis	Low_area_copro			
Simulation	Low_area_copro			
Synthesis	Mid_range (Default)			
Simulation	Mid_range (Default)			

Mid_range different Synthesis configurations

Keccak	Keccak_global	Clk_speed	Clk Cycles per Computation	Throughput (Computations per second)
32	32			
16	32			
16	16			
2	2			

VII. Lab Report

Please provide your answers in typed format and include a cover sheet with your first and last name, GT ID number, and GT username.

For the ModelSim simulation of VHDL code, please provide the following:

1. keccak_test.txt file
2. Explain briefly how the testbench is designed for all the three cores.
3. Screenshot of keccak_test.txt data simulation on ModelSim
4. Screenshots of a clear test case while running the three cores
5. Throughput of all three cores (outputs generated per second)
6. Which core is the fastest and why? Discuss possible differences in the way the VHDL for the fastest core is written that may be contributing to its speed.

For the MidCore evaluation, please provide the following:

1. Throughput of the different configurations
2. Explanation of how changing the values in Keccak and Keccak_global results in a change in output. Specifically, does the 16/32 configuration have a different output, and if so, why?
3. Explanation of the significance of these values

For the Timing Analysis on Quartus, please provide the following:

1. What is the fastest core in terms of clock frequency, and what is the speed?
2. Which core has the highest throughput, and what is the best throughput achieved?
3. Is there a significant difference between the synthesis and simulation outputs? Please explain.
4. Give your throughput and fastest clock achieved for the three configurations.
5. Submit tables as instructed in section VI.

For the Resource Utilization on Quartus, please provide the following:

1. Resource utilization of all three cores
2. Which core has the lowest resource utilization?

For the Hardware Implementation Question, please provide the following:

1. Based on all the tests conducted, which core would you prefer to run your SHA3 algorithm and why? Also discuss possible scenarios where you may favor one core over another.