Lab 3: Timing Optimization

EEL 6935 – Reconfigurable Computing 2

Objective:

The objective of this lab is to take a circuit with some significant timing bottlenecks and optimize it to run as fast as possible.

Required tools and parts:

You must use the lab servers for this lab. When you log into one of the servers, run the following:

module load quartus_std/20.1.0.711
module load modelsim

The TA will grade the assignment using this version of Quartus so, it is important that you don't use a different version.

Application:

You are provided with an unoptimized module called timing_example (see timing_example.sv). This file contains 3 modules: bit_diff, fifo, timing_example.

Bit_diff is an unoptimized version of the FSMD from my SystemVerilog tutorial. It has the same functionality, but is not an efficient implementation.

Similarly, the fifo module is a functional FIFO, but is can be significantly improved

The timing_example module is a largely artificial application, but ultimately that doesn't matter since you will often be required to optimize a circuit you don't fully understand. The timing_example module takes the data_in input and computes the bit difference every time data_in_valid is asserted. This part is no different than a normal FSMD, where the go signal is connected to data_in_valid. There is some additional logic that tracks the total number of executions of the bit_diff module using a 64-bit counter. This counter is output on the count output.

Every time there is an output from the bit_diff module, timing_example writes that output into a FIFO. Whenever the FIFO isn't empty, the rest of timing_example reads data from the FIFO and passes the data to a pipeline that multiplies the data with 16 different input values (from pipe_in), accumulates those products, and outputs the multiply-add result on data_out. There is also a data_out_valid that specifies when data_out represents a valid output from the pipeline.

Timing_example also has a ready signal that specifies when it is able to accept new inputs. For example, data_in_valid will be ignored when ready = 0.

You can assume the pipe_in inputs will be set before execution and held constant forever. This functionality is representative of many DSP applications, where a set of coefficients might be initialized after reset, and then only changed infrequently.

Instructions:

1. Open the provided timing_example.qpf in the required version of Quartus. This project targets a MAX10 FPGA, which I selected to minimize compilation times.
2. Compile the design through timing analysis and verify the clock frequency. Use the Slow 1200mV 80C mode to determine the clock frequency, since that is the slowest version. The provided design should achieve a clock frequency of just over 100 MHz (106 MHz in my test).
3. Open the timing analyzer and determine the critical path.
4. For the critical path, apply an optimization to eliminate the bottleneck.

5. Re-execute the provided testbench (timing_example_tb.sv) to verify that the design still works. You are allowed to increase the total number of cycles to improve clock frequency, but only by 5%, which the testbench will check.
6. Repeat from step 2 until you can no longer improve the clock frequency.
7. The provided sdc file has a constraing for 250 MHz. You are not required to achieve this frequency for full credit, but if you do, you will get full credit assuming the testbench still passes.
8. I do not have a rubric yet to decide what frequency corresponds to what grade, so it largely depends on the distribution of frequencies for the entire class. Without too much effort, I was able to achieve 242 MHz, so that will likely be full credit.

Restrictions:

For a normal design, you would be able to make whatever changes you want to achieve a clock goal, as long as the design was functional and met other constraints. Because this design is synthetic, there is a good chance you could optimize away most of the functionality with a different architecture, which isn't the point of the lab. Instead, I want you to apply the timing optimizations we have learned in class. Therefore, make sure to follow these restrictions when changing your code:

1. You cannot remove a module. For example, you might notice that the FIFO will usually be empty. Normally, that would be a good reason to get rid of the FIFO, but you must use the FIFO module, and all other modules.
2. You cannot change the default parameters to timing_example.
3. You cannot shrink any of the variables unless doing so guarantees the correct output as the larger width. For example, a 32-bit state register can be shrunk to the appropriate number of bits, but an adder can't be shrunk just to reduce delays.
4. You can increase total cycles, but only up to 5%, otherwise the testbench will throw an error.
5. You cannot modify the testbench, unless you want to improve it, in which case send it to me first for approval. The provided testbench is not very thorough, so if you spot an error that it doesn't catch, feel free to improve it. I'll offer extra credit for significant improvements.

Hints:

1. You can assume that the bit_diff implementation will never output data faster than every other cycle.
2. See the optimizations I applied to bottlenecks on individual examples. Every bottleneck you encounter is likely to be an instance of what I already covered.
3. You can't remove the FIFO, but you can reduce its potential timing bottlenecks with changes to its parameters.
4. If you add registers, you will have likely to increase the latency of the data_out_valid logic.

Report:

Document the series of optimizations you made to improve the design. Show the critical path and corresponding frequency at each step, then describe the optimization you applied to improve that path.

For your final design, include a screenshot of the achieved frequency, and a screenshot of the success simulation in Modlesim.

Turn in instructions:

Submit the following to Canvas:

- timing_example.sv – Your modified timing_example.sv file with all optimizations.
- Any other changed project files (e.g., timing_example.sdc).
- report.pdf
- README.txt with your group member names. If any problems occurred that I should be aware of for grading, include them here.