Dynamatic Tutorial: Exercise Instructions

Exercises 1, 2, and 3: Using Dynamatic

This part of the Dynamatic tutorial contains three exercises which illustrate the basic features and usage of Dynamatic on a toy example. The exercises are located in the following folders:

- Exercise 1: *tutorial/1-example-naive*
- Exercise 2: *tutorial/2-example-buffers*
- Exercise 3: *tutorial/3-example-lsq*

For all three exercises, follow the five steps below. Each exercise has additional questions in the following section.

1) Open a terminal and set the appropriate exercise folder as the working directory. For Exercise 1:
   `cd tutorial/1-example-naive`
2) Consider the code in `src/example.cpp` and the Dynamatic commands in `synthesis.tcl`. Use this script to synthesize the example function into a dataflow circuit:
   `dynamatic synthesis.tcl`
3) Study the DOT, PNG, and VHDL outputs of Dynamatic in folders *reports* and *hdl*.
4) Simulate the design using ModelSim:
   `vsim -do simulation.tcl`
   Study the waveforms and note the total execution time. Refer to the next page for tips on using ModelSim.
5) Check the correctness of the circuit by comparing the simulation results with the software results:
   `meld sim/C_OUT/output_a.dat sim/VHDL_OUT/output_a.dat`

For details on the compiler commands and the output files, please refer to the Dynamatic tutorial paper.

Exercises 1, 2, and 3: Additional Questions

Exercise 1.* All questions refer to the outputs of Exercise 1 in *tutorial/1-example-naive*.

a) In `example.png`, consider the buffer `buffI_0`. In the waveform, find the input and output signals of this buffer. What do the values stored in this buffer (i.e., values of `buffI_0_dataOutArray_0`) correspond to? Find a point in the simulation when the buffer is stalled by its successor (i.e., signal `buffI_0_nReadyArray_0` is set to zero while the buffer produces valid data and `buffI_0_validArray_0` is set to 1).

b) In `example.png`, consider component `icmp_15`. Find a point in the simulation where the loop condition (i.e., $i < 100$) evaluates to false (i.e., `icmp_15_dataOutArray_0` returns from value 1 to value 0).

c) In `example.png`, consider component `branch_1`. In the simulation, investigate how the change of the condition value from true to false affects the validity of the Branch outputs (i.e., signals `branch_1_validArray_0` and `branch_1_validArray_1`).

Exercise 2.* All questions refer to the outputs of Exercise 2 in *tutorial/2-example-buffers*.

a) In `example_optimized.png`, find all buffers and investigate their size and transparency (values provided in squared brackets for each buffer; for instance, `Buffer_1[2]` indicates a 2-slot nontransparent buffer and `Buffer_2[2t]` indicates a 2-slot transparent buffer).

b) In `example_optimized.png`, consider the buffer `Buffer_1`. In the waveform, find the input and output signals of this buffer. Check if, at any point in the simulation, the buffer is stalled by its successor (i.e., signal `Buffer_1_nReadyArray_0` is set to zero while the buffer produces valid data and `Buffer_1_validArray_0` is set to 1).

c) In the simulation, locate the read-after-write hazard between loop iterations 4 and 5. The load of iteration 5 occurs at 42 ns and the store of iteration 4 at 58 ns. The read and write address are named `a_address1` and `a_address0`, respectively.

Exercise 3.* All questions refer to the outputs of Exercise 3 in *tutorial/3-example-lsq*.

a) In `example_optimized.png`, locate the new LSQ component and study its inputs and outputs. Note that only the load (`load_7`) and store (`store_0`) to array `a` are connected to the LSQ, whereas the other loads connect to separate memories.

b) Check if the read-after-write dependency between iterations 4 and 5 is honoured. The load receives the read data (signal `load_7_dataInArray_0`) of iteration 5 at 82 ns. The store writes the data of iteration 4 to memory at 74 ns. Write address and data are named `a_address0` and `a_dout0`, respectively.

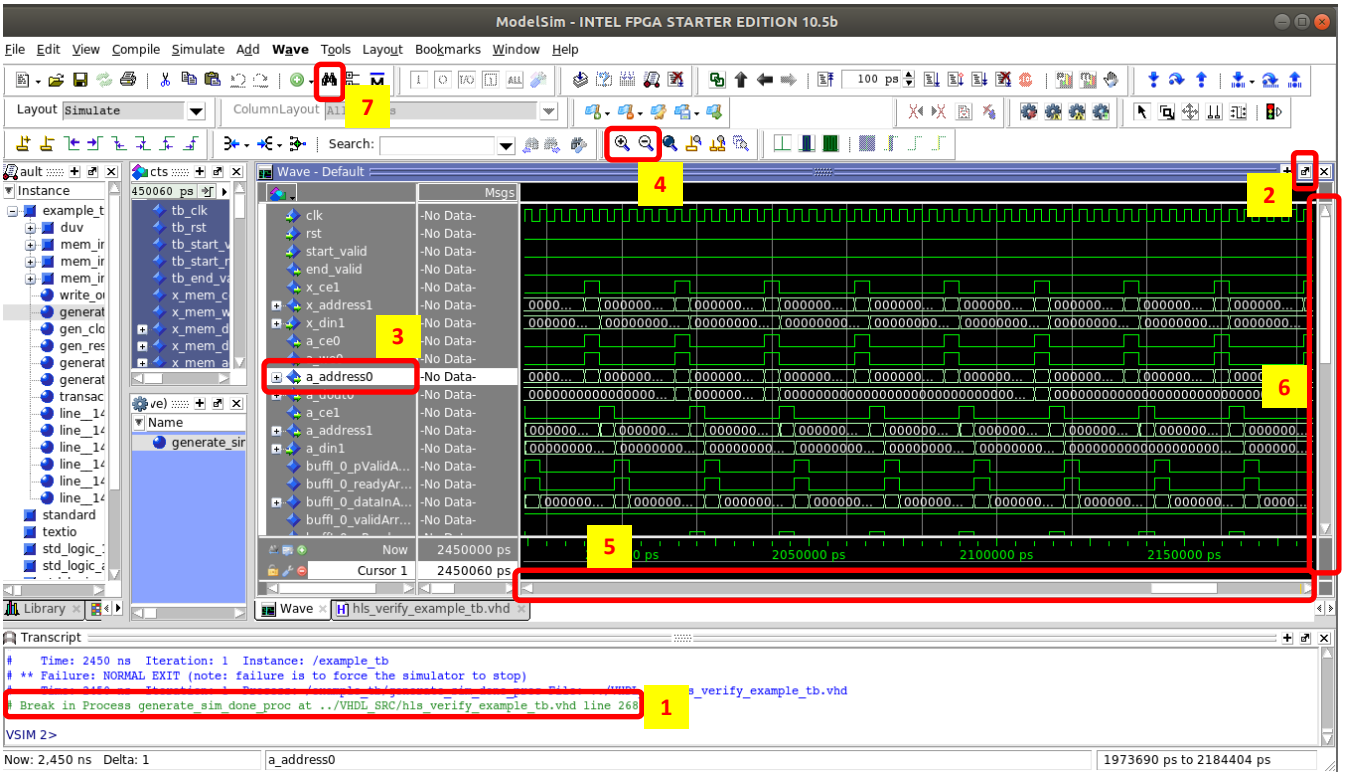

Figure 1: ModelSim.

Exercise 4: Modifying the Dynamatic Flow

This part of the tutorial illustrates some possibilities for modifying the Dynamatic flow (i.e., integrating new VHDL components and changing the intermediate representation). All questions refer to the files in *tutorial/4-example-mul*.

a) Synthesize and simulate `example_mul.cpp` as in the previous exercises. Note the simulation execution time.

b) In `hdl/mul_wrapper.vhd`, replace the 4-stage multiplier with an 8-stage multiplier by changing the multiplier entity from `work.mul_4_stage` to `work.mul_8_stage`. In the same file, change the `LATENCY` constant to reflect the component latency change. Simulate the design and compare the execution time to the prior one.

c) Integrate the information about the new multiplier into the Dynamatic flow:

- Invoke the Dynamatic shell using the command *dynamatic* and type manually the commands from `synthesis.tcl` to set up the project and synthesize the design (i.e., all commands up to *optimize*).
- Before invoking the *optimize* command, in `reports/example_mul.dot`, update the latency field of multiplier `mul_8` to reflect the latency change.
- Run the remaining commands from `synthesis.tcl` to place buffers and produce the VHDL netlist.
- Repeat the steps from question 4b to replace the multiplier in `hdl/mul_wrapper.vhd`.

Simulate the design, note the total execution time, and compare it from the one from the previous questions.

Using ModelSim

To complete the exercises, use the ModelSim features described in this section and depicted in Figure 1.

- After running the `simulation.tcl` script, wait for the simulation to complete; a green text line will appear in the *Transcript* window (marked with number 1 in the figure), indicating simulation break.
- Extend the waveform across the entire screen by clicking on the undock button (number 2).
- To change the number format of a signal from binary to unsigned, right-click on the signal name (e.g., number 3) and choose *Radix* > *Unsigned*.
- Zoom in/out of the waveforms using the zoom in/out buttons (number 4).
- Scroll through simulation time using the horizontal bar (number 5) and through signals using the vertical bar (number 6).
- Search for a signal by name using the find button (number 7).