EE201L
Divider RTL design in Verilog

1. **Objective**: To introduce students
   --- RTL coding style for state machine and datapath coding
   --- Understand basic Verilog syntax (case, if-else, always, assign, etc.)
   --- Run testbench in Modelsim

2. **Files provided**:

A zip file is provided containing a Modelsim project, two source files and a test bench file. *Please read the notes at the top of each file to get to know important aspects of the design to note.*

1. `divider_combined_cu_dpu.v`
2. `divider_separate_cu_dpu.v`
3. `divider_tb_str.v`

`* ee201l_divider_lab.mpf --> Modelsim project`

A short description of each of the above 3 files follows.

3. **divider_combined_cu_dpu.v**

A Verilog module for a simple divider. It takes two 4-bit inputs `Xin` and `Yin` (dividend and divisor) and produce 4-bit `Quotient` and `Remainder`, and three state bits, `Qi, Qc` and `Qd`, as outputs. There are inputs that control the state machine as well (`Start, Ack, Clk, Reset` and `Done`)

```
module divider (Xin, Yin, Start, Ack, Clk, Reset, Done, Quotient,
Remainder, Qi, Qc, Qd)
```

Note that the name of the module is `divider`, and it is because we're going to use a single test bench to test two different divider designs `divider_combined_cu_dpu.v` and `divider_separate_cu_dpu.v`. Their functionalities should be identical.

There are three states – `INITIAL`, `COMPUTE` and `DONE_S` - in the state machine and they are one-hot coded as below.

```
localparam
INITIAL = 3'b001,
COMPUTE   = 3'b010,
DONE_S    = 3'b100;
```

In this file, there is a single `always` block where you can locate NSL and SM for three states listed above.

4. divider_separate_cu_dpu.v

Another Verilog module for a simple divider. In this design, there are two separate `always` blocks, where one is for CU and the other is for DPU. There are some blank indicated as ‘`TODO`’ in this file, for students to complete.

5. divider_tb_str.v

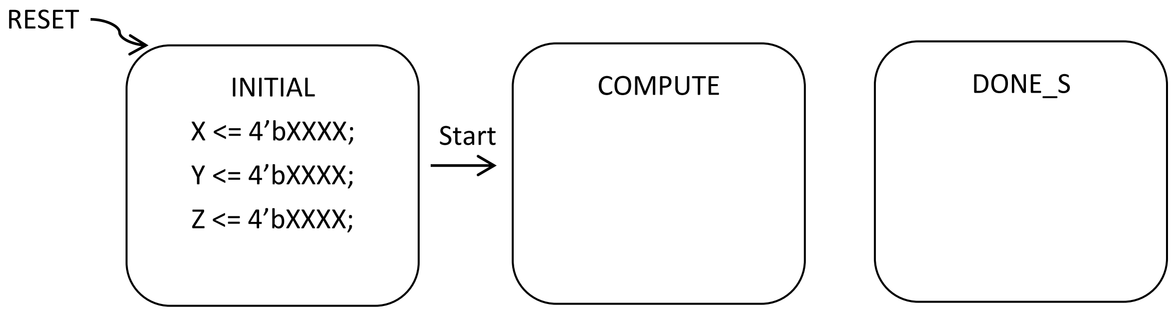
A test bench file that performs three divider operations. These are $15 \div 7, 5 \div 8$ and $11 \div 3$. Run Modelsim simulation and see messages in the transcript panel for results of the above three operations.

6. Procedure

a) Read file **divider_combined_cu_dpu.v** and answer to exercise question 7-a.
b) Open `ee201l_divider_lab.mpf` and type `do divider.do` in the transcript panel. See messages in the transcript panel and check results of three divider operations.
c) Type `quit -sim` to escape from simulation.
d) Read file **divider_separate_cu_dpu.v** and fill-in every ‘`TODO`’ blank.
e) In the same project `ee201l_divider_lab`, type `do divider_exercise.do` in the transcript panel. See messages in the transcript panel and compare results of three divider operations with those obtained from 6-b.
f) Answer to question 7-c. If necessary, change the line in **divider_combined_cu_dpu.v** and repeat procedure 6-b again to see the difference (or they could be the same)

7. Questions

a) Complete a state machine of the divider module based on **divider_combined_cu_dpu.v**



b) Fill-in every blank named ‘TODO’ in **divider_separate_cu_dpu.v**
c) In **divider_combined_cu_dpu.v,** what will happen if line 73

```
if (!(X < Y))
```

is changed to

```
if (X > Y)
```

? Can we still obtain correct results from the modified divider design? How about three test cases in our test bench? Write your answer.