Lab 2: Counter and Traffic Light Controller

Submission Due Dates:

Demo: 2021/10/05 17:20
Source Code: 2021/10/05 18:30
Report: 2021/10/10 23:59

Objective

Getting familiar with counter and FSM designs in Verilog.

Action Items

1 Counter (60%)

A. lab2_1.v

Design a 6-bit counter which counts upward from 0 to 63 and then downward to 0 repeatedly.

a. IO list:
   - ✓ Inputs: clk, rst
   - ✓ Outputs: out

b. The counter follows the following rules:
   - The counting is split into two sequences: sequence ***a*** and sequence ***b***.
     - ♦ When one sequence is over, the other will begin.
     - ♦ The last number of the current sequence is the first number of the following sequence.
     - ♦ The index of both sequence starts at 0.
   - When counting upward from 0 to 63, the generated sequence $a_0, a_1, a_2 \ldots$ is defined as:

$$a_n = \begin{cases} 0 & if\ n = 0 \\ a_{n-1} - n & if\ a_{n-1} - n > 0 \\ a_{n-1} + n & otherwise \end{cases}$$

   - When counting downward, the n^{th} number in downward sequence decreases by 2^{n-1}:

Value	$X \rightarrow (X-1) \rightarrow ((X-1)-2) \rightarrow ((X-1-2)-4) \rightarrow \cdots \rightarrow 0$				
Sequence *b*	b_0	b_1	b_2	b_3	$\cdots$

c. You should use the following template for your design:

```
`timescale 1ns/100ps

module lab2_1 (
  input clk,
  input rst,
  output reg [5:0] out);
  // add your design here
endmodule
```

   - **clk**: the counter is triggered by positive clock edges

- **rst**: the positive-edge-triggered reset to set the counter value (out) to 0.
- **out**: the output of the counter.

Design Constraint:

- It is **NOT allowed** to use a **lookup table** or a **fixed, hand-coded sequence** to generate the counter values. Otherwise, your design will get **a zero score**.

Hint 1: The sequence of a 3-bit counter with the maximum value of 7 shows below:

$0 \to 1 \to 3 \to 6 \to 2 \to 7$ (from 0 to 7)
$\to 6 \to 4 \to 0$ (from 7 to 0)
$\to 1 \to 3 \to \cdots$ (from 0 to 7 again)

Hint 2: The sequence of a 6-bit counter with the maximum value of 63 shows below:

$0 \to 1 \to 3 \to 6 \to 2 \to 7 \to 1 \to \cdots \to 62 \to 6 \to 63$ (from 0 to 63)
$\to 62 \to 60 \to \cdots \to 32 \to 0$ (from 63 to 0)
$\to 1 \to 3 \to 6 \to \cdots$ (from 0 to 63 again)

B. lab2_1_t.v

Write a testbench (lab2_1_t.v) by yourself to verify the design. Remember to test **boundary conditions**. During the demo, TA will check the **result/waveform** with questions about how you built your testbench.

2 Traffic Light Controller (40%)

A. lab2_2.v

Consider the problem of controlling a traffic light at the intersection of two busy streets, A Street and B Street. Design a traffic light controller and test your design with the **given testbench file lab2_2_t.v**.

a. IO list:
   - ✓ Inputs: clk, rst, carA, carB
   - ✓ Output: lightA, lightB

b. The signals of **carA** and **carB** represent whether there is a car before the intersection in either direction on A St. and B St. respectively.

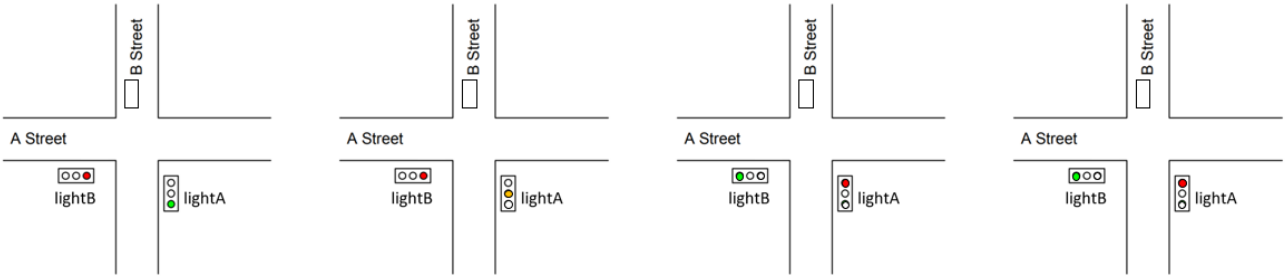
Signal	Representation	Signal	Representation
carA = 0	No car on A St.	carB = 0	No car on B St.
carA = 1	A car on A St.	carB = 1	A car on B St.

c. The signals of **lightA** and **lightB** represent the traffic light controller on A St. and B St. respectively.

Signal	Representation	Signal	Representation
lightA = 3'b001	Green light on A St.	lightB = 3'b001	Green light on B St.
lightA = 3'b010	Yellow light on A St.	lightB = 3'b010	Yellow light on B St.
lightA = 3'b100	Red light on A St.	lightB = 3'b100	Red light on B St.

d. The design must obey the following rules:
   I. According to the input signal **carA** and **carB**, there are four conditions when the light is green on A St. and red on B St.,

(1) If there is a car on A St. and another car waiting on B St., keep the lights unchanged.
(2) If there is a car on A St. but no car waiting on B St., keep the lights unchanged.
(3) If there is no car on A St. but a car waiting on B St., consider two cases.
Case one: If the light on A St. has kept green for at least two cycles, then give A St. a yellow light for one clock cycle, and then give A St. a red light and B St. a green light for at least two cycles. (Show in the picture below)
Case two: If the light on A St. has not kept green for two cycles, keep the lights unchanged.
(4) There is no car on both streets, keep the lights unchanged.



II. When the light is red on A St. and green on B St., follow the rule I but replacing A St. with B St. and B St. with A St in the statements.
III. When reset, the light should be initially green on A St. and red on B St.
IV. Note that no matter what input signals **carA** and **carB** are, the yellow light turns red in the next cycle.

e. You should use the following template for your design:

```
`timescale 1ns/100ps

module lab2_2 (
  input clk,
  input rst,
  input carA,
  input carB,
  output reg [2:0] lightA,
  output reg [2:0] lightB);
  // add your design here
endmodule
```

- **clk**: the traffic light controller is triggered by positive clock edges
- **rst**: the positive-edge-triggered reset

Design Constraint:

- You must design one **finite state machine (FSM)**. Draw the state diagram(s) before coding.
- In the report, you have to explain **FSM and state diagram(s)**.

Attention

- DO NOT copy-and-paste code segments from the PDF materials. Occasionally, it will also paste invisible non-ASCII characters and lead to hard-to-debug syntax errors.
- You have to hand in **three** source files lab2_1.v, lab2_1_t.v, lab2_2.v.
  Upload each source file separately! DO NOT hand in any compressed ZIP file!
- You should also hand in your report as **lab02_report_StudentID.pdf** (i.e., lab02_report_108456789.pdf).
- You should be able to answer questions of this lab from TA during the demo.
- You may also add a **$monitor** in the testbench to show all the inputs and outputs during the simulation.
- You may have to change your runtime (ns) in “Simulation Settings” to fit your testbench settings before you run the simulation.