Lab 3 (100 points): The light-sequence generator design

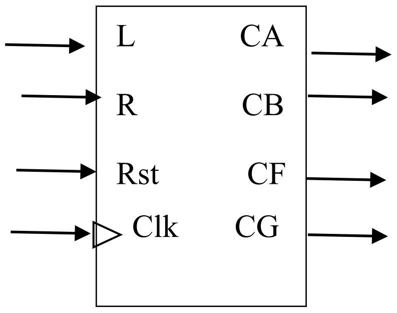
Demo: by the end of your scheduled lab session during the week of **Oct 2-6, 2023.**
Code submission: Submit your .v files and .xdc file on D2L under "Lab 3" **by 11.59 pm on Oct 6, 2023.**

Objectives:

- Design and implement **circuit of a light-sequence generator**
- Practice writing testbench for a sequential circuit
- Practice performing Behavioral and Post-synthesis functional simulation of your designed circuit.
- Program your designed circuit to Nexys 4 DDR board

Pre-lab 3 (10 points): Draw a state diagram of the light sequence generator. Show TA/ULA your group's state diagram within the first 30 minutes of your lab session during the week of Sep 25-29, 2023.

Problem Statement: Design the following light-sequence generator which has



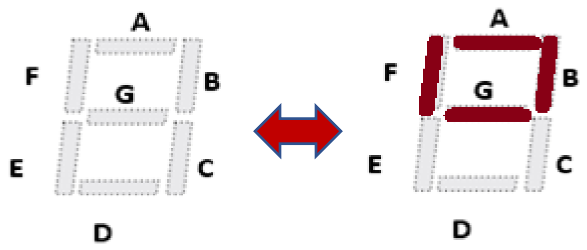
- 4 inputs: Clock (Clk), active-high synchronous Reset (Rst), L and R.
- 4 outputs: CA, CB, CF, CG. The segment A, B, F, G of the seven-segment display will be used in this assignment.
- If Rst = 1 at the rising edge of Clk, the machine is back at the initial state where all segments are OFF.
- The FSM starts from an initial state where all segments are OFF.

Reminder (discussed in lab 2): **To activate each segment (make it light up), logic 0 is used.**

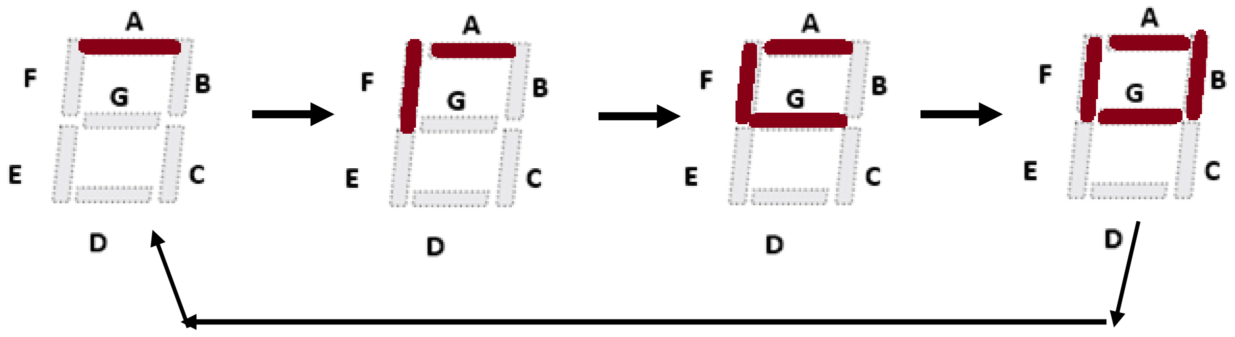
Requirement Specifications:

When both L = 0 and R = 0, all segments are OFF.

As long as L = 1 and R = 1 the light sequence (flashing) below is generated.
It starts from the initial state where all segments are OFF, then all segments are lit up, then go back to all segments are off, … This sequence is repeated as long as L = 1 and R = 1.

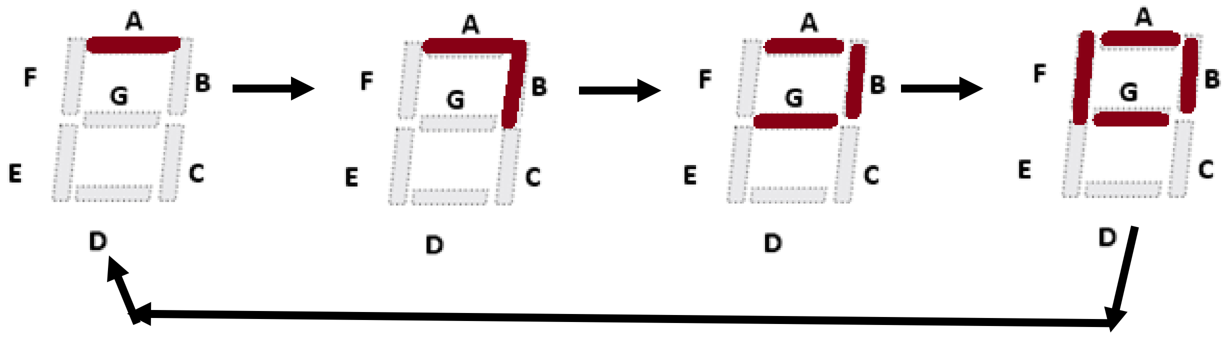


As long as L = 1 and R = 0 the light sequence (counterclockwise circle) below is generated.



When L = 1, R = 0, it goes to the state that seg_A is lit up, the next clock edge,
it goes to the state that seg_A and seg_F are lit up, then
the next clock edge, it goes to the state that seg_A, seg_F, seg_G are lit up, then
the next clock edge, it goes to the state that seg_A, seg_F, seg_G, seg_B are lit up, then
the next clock edge, it goes back to the state that the state that seg_A is lit up,
This sequence is repeated as long as L = 1 and R = 0.

As long as L = 0 and R = 1 the light sequence (clockwise circle) below is generated.



The sequence above is repeated as long as L = 0 and R = 1.

If the users change their mind about the input signals in the middle of any light sequence, make the FSM go back to the initial state.

Carefully read the requirement specification above and **design this sequential circuit** (draw its state diagram).

Part A:

Step 1: Design a finite state machine (**draw its state diagram**) of the light-sequence generator. See the requirement specifications on pages 1 and 2.

Step 2: Write **Verilog code** for your design in Step 1

Step 3: Write a **testbench** to test your designed FSM.

When writing the code for clock signal, use a **period of 200 ns** (not 20 ns). In other words, use #100 (not #10) when writing the code for clock signal.

Your testbench must show **all 5 different scenarios** when

- R = 0 and L = 0 (all lights are off)
- R = 0 and L = 1 for at least 6 clock cycles,
- R = 1 and L = 0 for at least 6 clock cycles,
- R = 1 and L = 1 for at least 6 clock cycles,
- the user changes his/her mind about the input signals in the middle of any light sequence.

The picture of waveform simulation on Page 6 shows all 5 scenarios. You do not have to recreate the exact waveform, however, all 5 scenarios must show up. You are expected to explain how it works to the TA.

When writing the testbench, the code below might be useful to make input(s) stay at that value for several clock cycles.

```
integer i;
#50 L <= 1'b0; R <= 1'b1;
for (i = 0; i < 6; i = i + 1) begin
    @(posedge Clk);
end
#50 L <= 1'b0; R <= 1'b0;
```

The code above is the same as writing the code shown below:

```
#50 L <= 1'b0; R <= 1'b1;
@(posedge Clk);
@(posedge Clk);
@(posedge Clk);
@(posedge Clk);
@(posedge Clk);
@(posedge Clk);
#50 L <= 1'b0; R <= 1'b0;
```

Step 4: Run Behavioral simulation. The output waveform should behave as stated in the requirement specifications. To run the simulation longer than 1000 ns, see How to use Vivado features, topic 1) on page 7 of this handout.

Step 5: Run Synthesis and perform Post-synthesis functional simulation. This waveform should look the same as ones in Step 4.
Demo the output waveform of Step (4 and) 5 to TA/ULA

Part B:
To make the light-sequence generator work on the Nexys 4 DDR board, we have to use more components as shown in Figure 3, the top-level design, below. This **top-level design (.v file)** has

4 inputs:	CLK100MHZ (connected to pin E3 for 100 MHz signal),
	BTNC – (Center push button) is Reset
	SW[1] – Switch 1 is L input
	SW[0] – Switch 0 is R input
4 outputs:	CA, CB, CF, CG

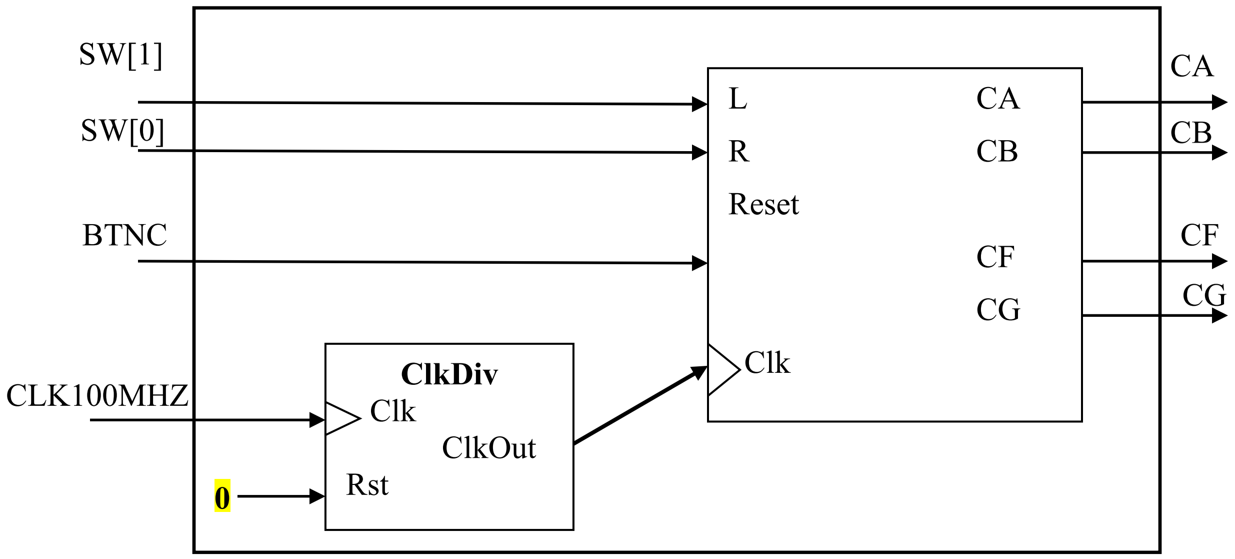


Figure 3: top-level design

Action items for Part B
1) Get the following .v files from *Lab3_PartB_Template* zipped folder on D2L lab 3 module and put them in the .srcs folder of your lab3 Vivado project

- ClkDiv.v file
- Top_Design .v file

=> On Vivado, click Add Sources. Add or create design sources to add the above .v files into your lab3 Vivado project.

2) Get the Nexys4DDR_Master (xdc) file from D2L (same zipped folder as above) and put them in the folder of your lab 3 Vivado project
=> On Vivado, click Add Sources. Add or create design constraints to add the above xdc file into your lab3 Vivado project.

Note **ClkDiv:** has two inputs: Clk and Rst, and one output: ClkOut. The relationship between frequency of Clk and ClkOut is

$$\text{Frequency of ClkOut (in Hz)} = \frac{\text{Frequency of Clk}}{2*(\text{DivVal}+1)}$$

The oscillator used on the FPGA board is 100MHz = 100*1e6 Hz. This is connected to pin E3.
If DivVal = 49999999 (in the code), then

$$\text{Frequency of ClkOut (in Hz)} = \frac{100*1e6}{2(49999999+1)} = \frac{100*1e6}{2*50*1e6} = 1\text{ Hz}$$

3) You may have to set Top_Design.v file as Top module (right-click on that file name under Design Sources and choose "Set as Top" Set as Top).
The file set as Top is one Vivado uses to generate the netlist file used for FPGA board.

4) In Top_Design.v file, write ***Structural*** Verilog code to connect the modules together as shown in Figure 3 (page 4)

The following code is already in Top_Design.v file

```
`timescale 1ns / 1ps
module Top_Design(CLK100MHZ, BTNC, SW, CA, CB, CF, CG);
   input CLK100MHZ, BTNC;
   input [1:0] SW; //SW[1] is L, SW[0] is R
   output CA, CB, CF, CG;

        //start writing your code below by instantiating the components
        //used for lab 3 top-level design which are ClkDiv and the light-sequence generator


endmodule
```

Note: You do not have to simulate the top-level code. If you want to, you can perform a simulation as follows: *Before* the simulation, first change DivVal value in ClkDiv to 1 (instead of 49999999). ***After*** the simulation, change DivVal back to its original value.

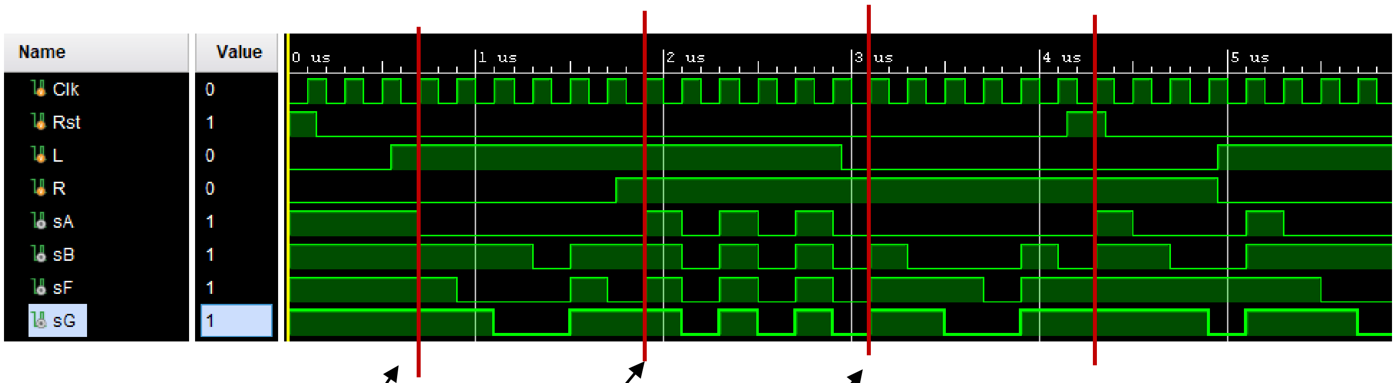
5) Pin assignments: ***in Nexys4DDR_Master.xdc file,*** Uncomment the pins that you will use for the 4 inputs and 4 outputs of the top-level code (Top_Design).

6) Run Synthesis, Run Implementation, Generate Bitstream and program the FPGA board. Demo to TA that your light sequence generator circuit work correctly on the FPGA board.

Grading Criteria

Part A	
- State diagram of light-sequence generator	10 pts
- Verilog code for the light-sequence generator	30 pts
- Testbench for the light-sequence generator (5 cases) and correct waveform from Post synthesis functional simulation	20 pts
Part B	
- Structural Verilog code for Top-level design	20 pts
- Correctly uncomment pins used in Nexys4DDR_Master.xdc file	5 pts
- Circuit working on the Nexys 4 DDR board	15 pts

Example of simulation waveform:



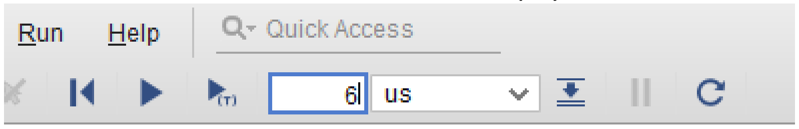
- @ the 1st edge of Clk signal, reset (Rst) = 1, FSM starts from the initial state where all lights are off (observe that all outputs are 1)
- For the next 2 clock edges, all lights are off since both L and R inputs are 0.
- Then at 0.7 us (microseconds), only the input L = 1, the light sequence works as follows
  sA which is CA = 0 then the next clock cycle (L is still 1),
  sA, sF = 0, then the next clock cycle (L is still 1),
  sA, sF, sG = 0, the next clock cycle (L is still 1),
  sA, sF, sG, sB = 0, the next clock cycle (L is still 1),
  sA = 0, the next clock cycle (L is still 1), sA, sF = 0
- At 1.9 us(microsecond), the FSM sees that L is 1 and R is 1, it first goes to the initial state (all outputs are 1), then it goes to the next state where all outputs are 0.
- At the clock edge at 3.1 us, L is 0 and R is 1, the FSM goes from the state where all outputs are on to the the state(s) outputting the clockwise light sequence where First, sA = 0, then sA, sB = 0, then sA, sB, sG = 0, , then sA, sB, sG, sF = 0, then sA = 0.
- At 4.3 us, $\longrightarrow$
  it shows how reset works, when the reset input is 1 (it does not matter whether other inputs are 1 (in this example, R = 1 at that time), all outputs are off since the machine is in the initial state.
- The last part of the waveform after the reset is back to 0 shows the scenario when the users change their mind during the middle of the sequence.

How to use Vivado features:

1) To run a simulation waveform longer than a default 1000 ns

If you use #100 to generate the clock signal in the testbench, this means that one clock cycle/period is 200 ns. If you want to see the simulation for every scenarios (5 scenarios and each scenarios uses 6 clock cycles) as discussed in Step 3 of part A, you will need at least 5*6*200 = 6000 ns = 6 us (microsecond).

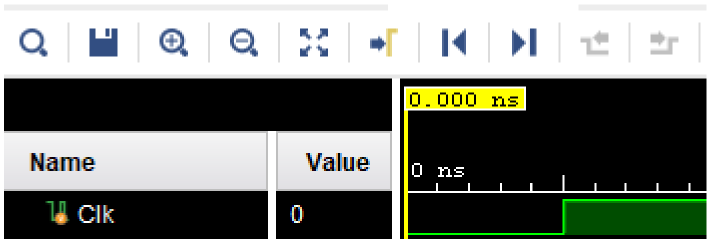
When the simulation waveform show up, you will see below:



1) Click on the box above to change the value to any number, the above shows 6 and use the dropdown menu to change the unit to us (microsecond). 6 us is 6 ns. You can change it to any number and unit that you want until you can see the simulation that you desire.
2) Click to clear the waveform.
3) Click to run the simulation for 6 us. Note: if you click again, it will run for another 6 us.

   To put the entire simulation in the simulation window, click to zoom fit (see the explanation below).

Menu bar of the waveform simulation



- Click to bring the yellow vertical bar to the beginning of the waveform (at time 0.00 ns).

Note: When you click zoom in or zoom out , it does the zoom at the location where the yellow vertical bar is at.

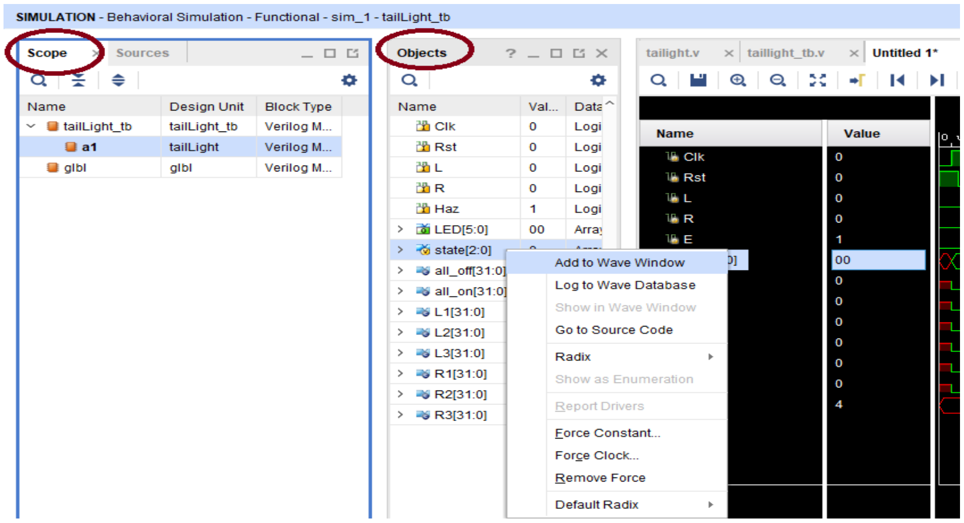
- is to zoom fit. It will fit the entire simulation in the simulation window.

One way to debug your circuit, when you run Behavioral simulation, is to add ***Internal*** signals (signals that are NOT inputs and outputs of the module) such as state of the sequential circuit.

To add internal signal (such as state) to the waveform in Behavioral simulation

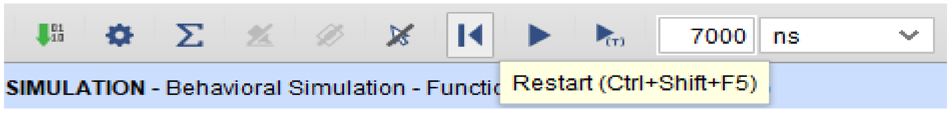
On "Scope" window, click > to expand so that the module that you want to add the signal is shown

On "Objects" window, Right-click on the signal that you want to add to the waveform and choose ***Add to Wave Window***.



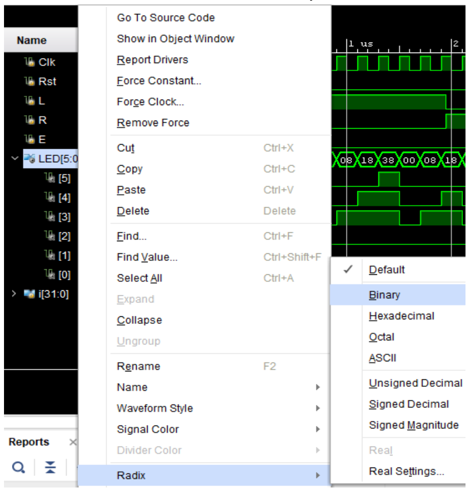
After you add the internal signal such as state, if its waveform does not show up, click on ***Restart*** button (the arrow pointing to the right).

Then type the time to the duration that you want to run the simulation (in the picture below, it shows 7000 ns), then click the ***play button with (T)***. You will see your simulation waveform runs from 0 to 7000ns



To change the base number (Radix) for the vector signal

By default, the vector signal shows its value in hexadecimal. Right-click on the vector signal name on the waveform, choose Radix, then choose the base-number that you want to use.


Go To Source Code
Show in Object Window
Report Drivers
Force Constant...
Force Clock...
Remove Force
Cut
Copy
Paste
Delete
Find...
Find Value...
Select All
Expand
Collapse
Ungroup
Rename
Name
Waveform Style
Signal Color
Divider Color
Radix
Default
Binary
Hexadecimal
Octal
ASCII
Unsigned Decimal
Signed Decimal
Signed Magnitude
Real
Real Settings...