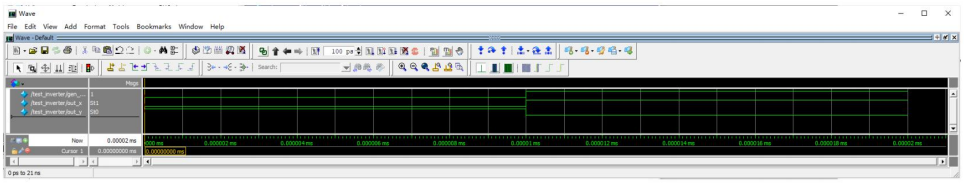
Workshop 1:

1. In Quartus, we must set the Top Level Entity Name.
2. We must give pins of the FPGA connect to the Top-Level module the name specifiied.
3. The FPGA on the DE1-SoC board is 5CSEMA5F31C6. It belongs to the Cyclone V family.
4. We need ModelSim-Altera to simulate our design.
5. Actual FPGA pins connection will be done later by using the Pin Assignment tool.
6. 'Assign' statements are to define logic gates. '!' means logical NOT.
7. A test bench is just another Verilog module for testing.
8. The $display function would show results in the console output of ModelSim.
9. press 'I' to zoom in, 'O' to zoom out and 'F' to zoom fitting.
10. https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=836&FID=ae336c1d5103cac046279ed1568a8bc3 (The DE1-SoC User Manual)
11. Exercise 1: inverter



a)

12. Selection can also be implemented as a combinational logic circuit.

Fill in the following truth table.

Front	Back	Switch	Display
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

13.

$$z = (A \wedge \overline{S_0}) \vee (S_0 \wedge B)$$

Derive the truth table for the above boolean expression.

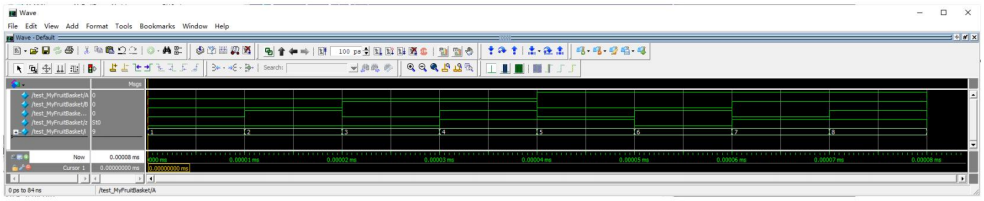
Input A	Input B	Input S_0	Output z
0	0	0	0 ∨ 0 = 0
0	0	1	0 ∨ 0 = 0
0	1	0	0 ∨ 0 = 0
0	1	1	0 ∨ 1 = 1
1	0	0	1 ∨ 0 = 1
1	0	1	0 ∨ 0 = 0
1	1	0	1 ∨ 0 = 1
1	1	1	0 ∨ 1 = 1

14.

15. Name the modules with meaningful names.
16. In Quartus, you can check the RTL (Register-transfer Level) view of your design after compiling.
17. The RTL Viewer, State Machine Viewer, and Technology Map Viewer allow you to view

schematic representations of the internal structure of your designs. Each viewer displays a unique view of the netlist, and allows you to view different internal structures.

18. Delays using '#' are used to generate real simulation signals.
19. Wires are like junctions to connect 'chips' together.
20. 'x = c ? y : z;' means x would be assigned the value of y if expression c is True otherwise it takes the value z.
21. '%' means modulo.
22. Exercise 2: Multiplexer



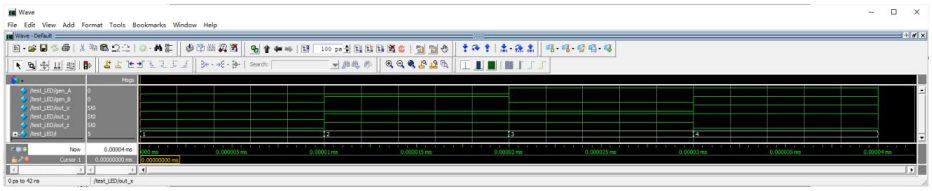
a)

23. We examine the difference between Boolean operators and Bitwise operators.

Type	Symbol	Description	Number of Operands
Bitwise	~	bitwise negation	1
	&	bitwise and	2
	|	bitwise or	2
	^	bitwise xor	2
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2

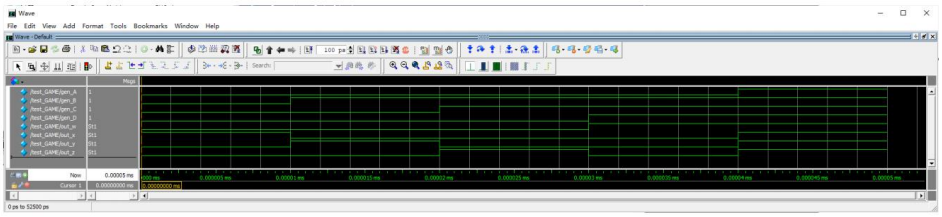
a)

24. Reg and always are used. The case statement itself is a bit like a bunch of "if" statements.
25. If the switches are all OFF, "0" will not be correctly displayed.
26. The objective is for you to be able to do these flfluently, without looking at the Workshop 1
27. slides. You will make a game for a (very) young child. It will use four switches and four LEDs.
28. Part 2: the game goes as follows: 1. Initially, all the switches should be in the OFF position, and there will be a single LED on, LED[2]. 2. The player must switch SW[2] on (because this corresponds to LED[2] which is the only LED that is on). 3. LED[2] will remain on, and a new LED will come on, say, LED[3]. 4. The player must keep SW[2] on and also switch SW[3] on. 5. A third LED will now come on, and the process repeats. 6. The game ends when all LEDs and all switches are ON.



a)

b)


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/test_GAME/pen_A
/test_GAME/pen_B
/test_GAME/pen_C
/test_GAME/pen_D
/test_GAME/out_x
/test_GAME/out_y
/test_GAME/out_z
Now
Cursor 1
0 ps to 52500 ps

Workshop 2:

1. Verilog is used for two very different purposes.
   a) Simulation Verilog's original purpose was to simulate digital hardware. Testing via simulation is usually cheaper and safer than testing the actual system.
   b) Synthesis Verilog code can be synthesised into physical hardware. This is an efficient way of implementing digital systems in the real world.
2.
   a) ModelSim This is a simulation tool. It interprets Verilog code and runs a simulation of the hardware described by Verilog on a regular computer.
   b) Quartus This is a synthesis tool specifically designed to generate digital logic for Intel FPGA devices from code written in a hardware description language.
3. A multiplexer (or Mux) is another word for a selector.
4. To find out how your circuit is being implemented on the FPGA, you must look at the Technology Map Viewer.
5. A LUT is a very small block of memory that works as a physical implementation of a Truth Table.
6. Use a 'case' to describe a truth table

```
module MyMUX (input A, input B, input S0, output reg z);
    always @(*) begin
        case({A, B, S0})
            3'b000: z = _FILL_THIS_IN; 0
            3'b001: z = _FILL_THIS_IN; 0
            3'b010: z = _FILL_THIS_IN; 0
            3'b011: z = _FILL_THIS_IN; 1
            3'b100: z = _FILL_THIS_IN; 1
            3'b101: z = _FILL_THIS_IN; 0
            3'b110: z = _FILL_THIS_IN; 1
            3'b111: z = _FILL_THIS_IN; 1
        endcase
    end
endmodule
```

a)

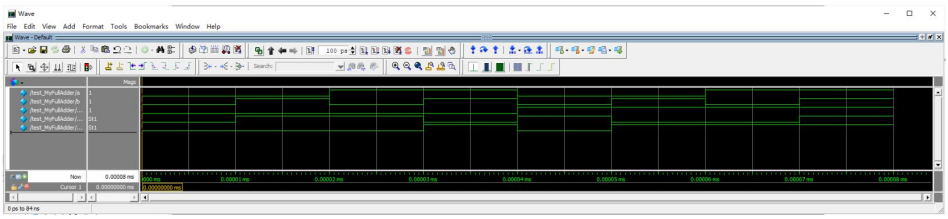
Look back at your lecture notes and make sure you understand MyMUX.

- What is an "always" block? run for more than 1 time
- What is a "sensitivity list"? * A, B, S0
- What does "*" mean in "always @(*)"? change when change
- Why do we declare "z" to be a reg? we want to change it
- What do the curly brackets {···} do?

7.

8. Exercise 1: Multiplexers Revisited

a)

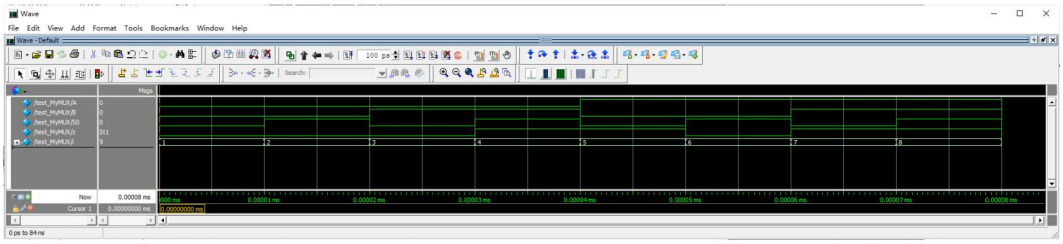


9.
$$\begin{array}{r} 1\,1\,\textcircled{1} \\ 1111 \\ +\quad 100\textcircled{1} \\ \hline 1\,1000 \end{array}$$

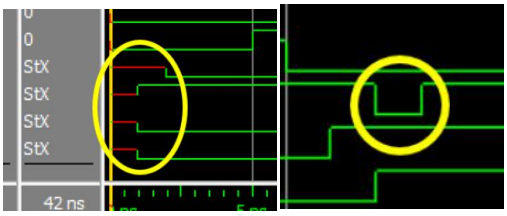
10. A ‘full adder’ contains ‘half add’s.

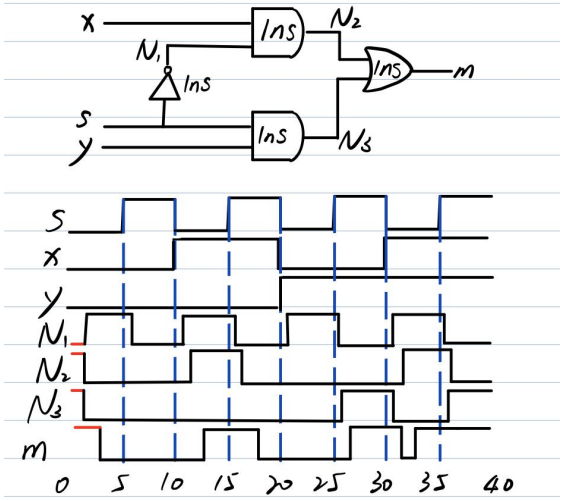
11. $$sum = (a \oplus b) \oplus c_{\text{in}}$$
$$carry = (a \wedge b) \vee (c_{\text{in}} \wedge (a \oplus b))$$

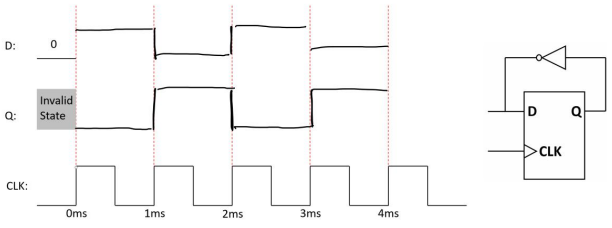
12. Instantiate ‘half adder’ to get a ‘full adder’
13. Exercise 2: Adding Numbers

a) 

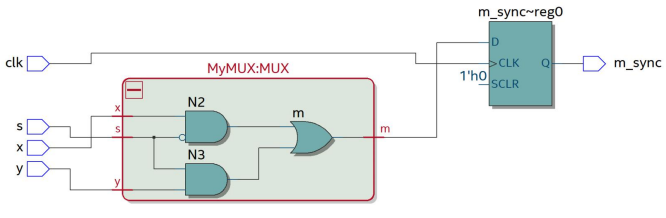
Workshop 3:

1. An RTL Simulation assumes ideally: there is no delay. It is fast.
2. Propagation Delay: The time taken for the output to be correct, and remain correct, from the time the inputs have stopped changing, is called the propagation delay.
3. Glitches: 

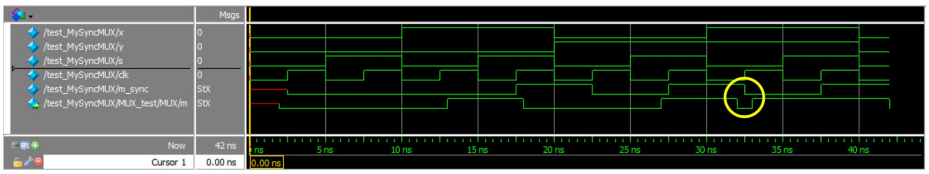
4. 

5. In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information – a bistable multivibrator.
6. FF can eliminate the glitches.
7. Learn how to add waves in modelsim
8. An ideal flip-flop has an “infinitesimal delay”.
9. 

10. Exercise 2: A Synchronised MUX

a) 

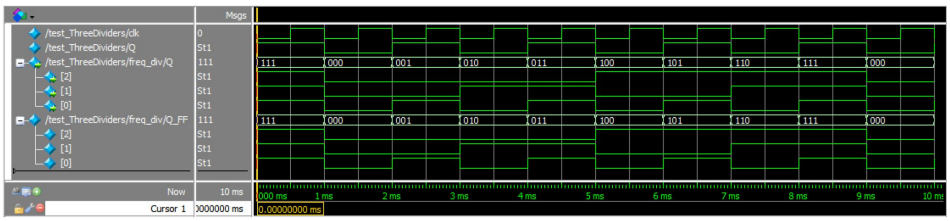
b)



11. Quartus initialises regs to zero by default but ModelSim does not. We must explicitly initialise regs to zero whenever the code relies on this initialisation if ModelSim is to behave the same as Quartus.
12. Warning: Use only a single clock to clock all your flip-flops!

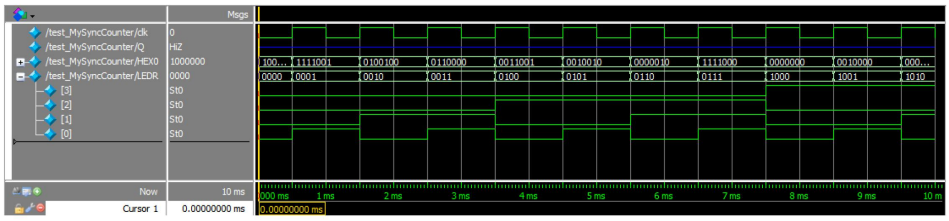
> **Note**
> The push buttons have been debounced in hardware. Otherwise this design would not work as intended.

13.
14. Exercise 3: Let's Count



a)

15. Having a combinational logic circuit (in this case, an adder) in the feedback loop of a bank of flip-flops is a powerful design paradigm
16. Exercise 4: Synchronous Counter



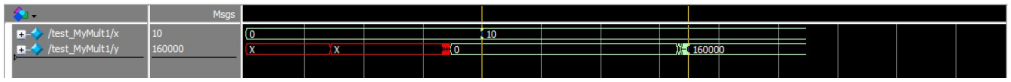
a)

Propagation delay is the length of time starting from the inputs become stable to the outputs reach statable.
Contamination delay is the minimum time from when an input changes until any output starts to change.
Setup time is the minimum time that the inputs must be stable before the clock edge.
Hold time is the minimum time that the inputs must remain stable after the clock.

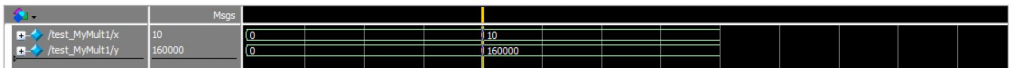
17.

Workshop 4:

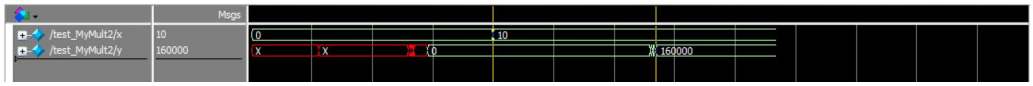
1. What delay are we measuring? Because MyMult1 was declared a top-level module, the propagation delay includes all of the following.
    a) The time it takes the signal to travel from the physical pins of the FPGA that were assigned to x to the fifirst multiplier block.
    b) The time it takes the signal to pass through all the multiplier blocks, and come out as valid.
    c) The time it takes the valid output of the multipliers to travel to the physical pins of the FPGA that were assigned to y.
2. Use SDC file to define the clock.
3.
    - Compile again and go to **Timing Analyzer → Slow 1200mV 85C Model → Fmax Summary**. 146.41 MHz
    - You can see the Fmax calculated by Quartus, which is the maximum frequency your design can operate at.
    - Now look at **Slow 1200mV 0C Model → Fmax Summary**.
        - Is Fmax larger or smaller at lower temperatures? 158.43MHz
4.
    - Compile and go to **Timing Analyzer → Slow 1200mV 85C Model → Fmax Summary**.
        - What is the maximum frequency for the new design?
        - Is it higher than for the old design? 213.81 MHz
5. Exercise 1: Pipelined Arithmetic
    a) Gatelevel:



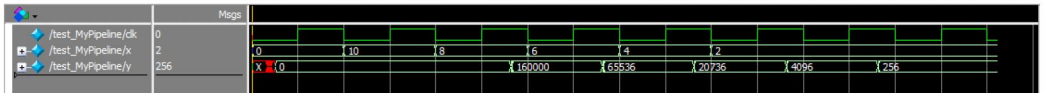
    b) RTL:



6.
    a)



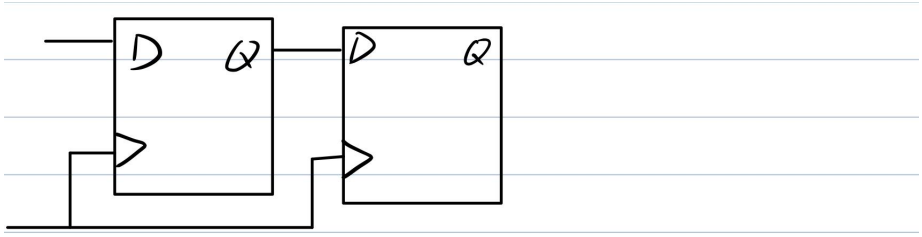
    b)



    c)

7. A synchroniser is a chain of 2 or more flflip-flflops used to trade input latency for the reduced probability of metastability at its output.
8. '#105', '# 30', '# 85'

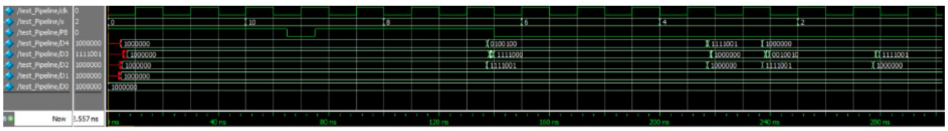
9.



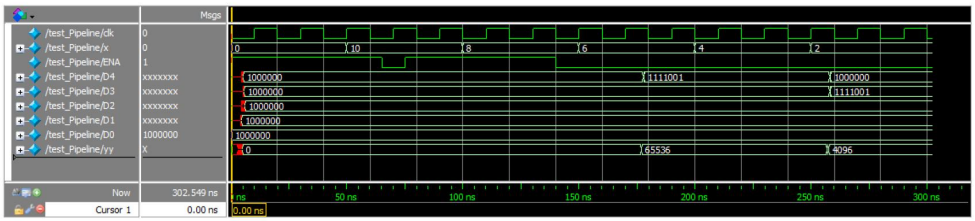
The output of the first FF would have a valid logic level before next rising edge

10. Exercise 2: Pipelined Arithmetic

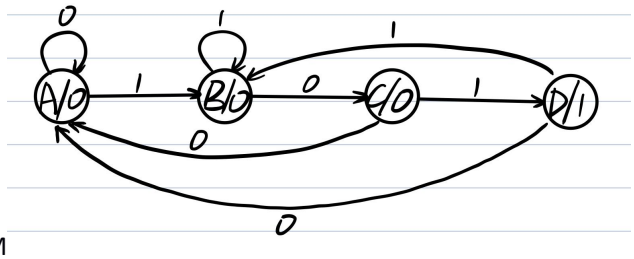
a)



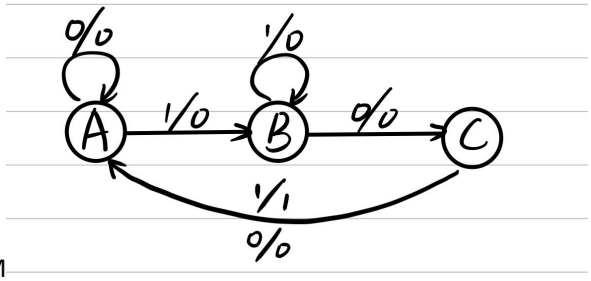
b)



Workshop 5:

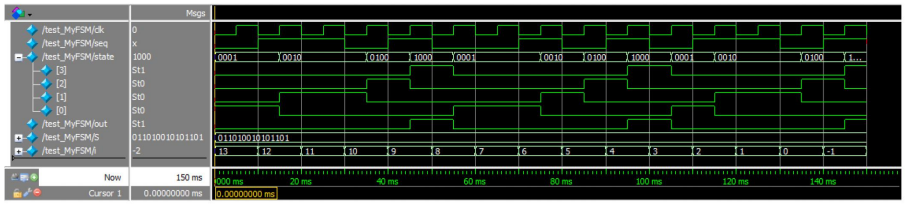


1. Moore FSM

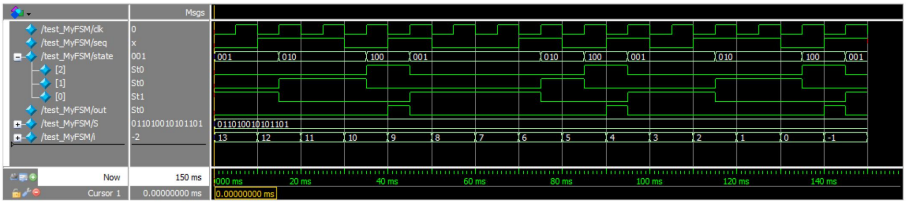


2. Mealy FSM
3. Exercise 1: A Sequence Detector

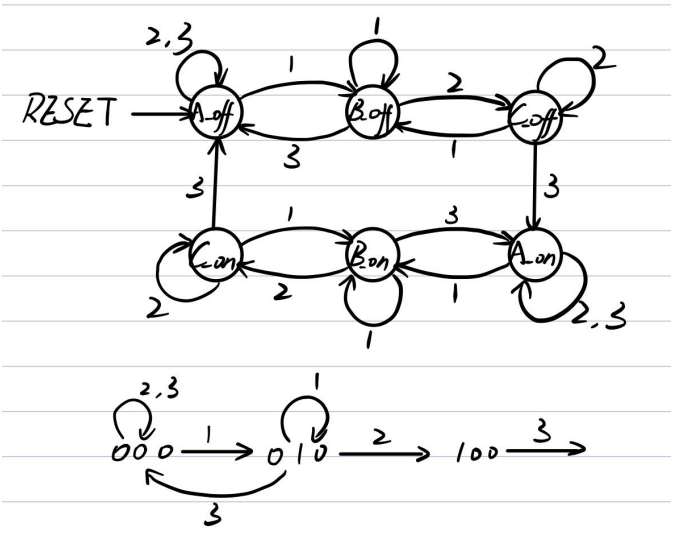
a)



b)



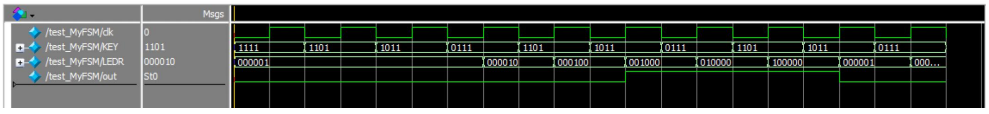
4.

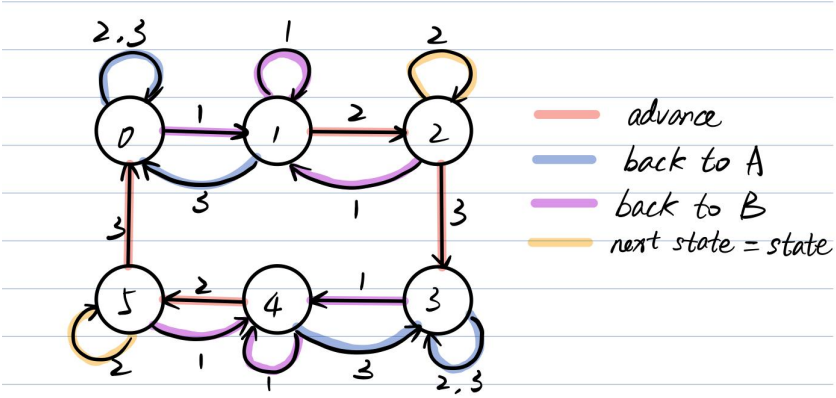


5. State is {nkeys, out}

6. What should happen, according to the state diagram? At least in the version we gave, we did not consider this; we assumed the inputs were mutually exclusive, meaning only one key could be pressed at a time.
7. What should happen, according to the Verilog code? It depends on how you wrote your code.
8. Why do we allow someone to press as many 1s as they like before pressing a 2? Since the keys are sampled at 50 MHz, “pressing” a key for even just 1 ms will cause 50,000 1s to be input to the FSM.
9. Since we are only dealing with single-bit inputs, we simply put each input through a double flip-flop synchroniser to solve metastability.
10. Exercise 2: Basic Combination Lock

a)

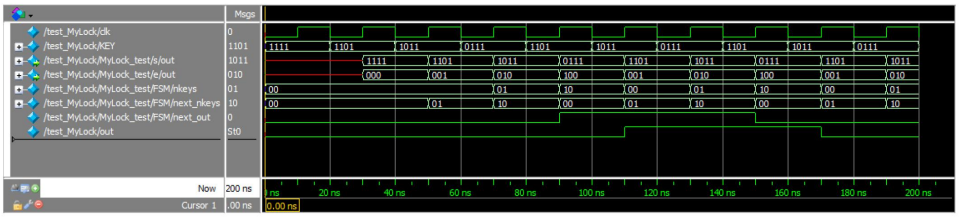




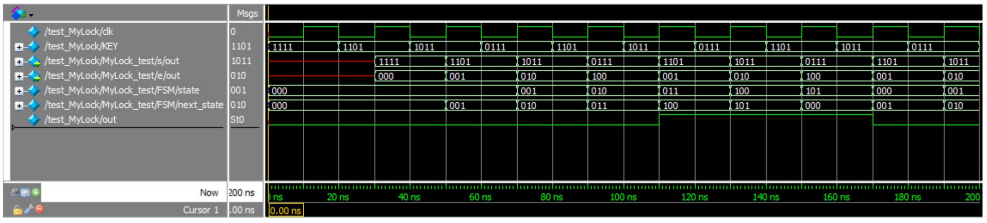
11.

12. Exercise 3: A Press is a Press

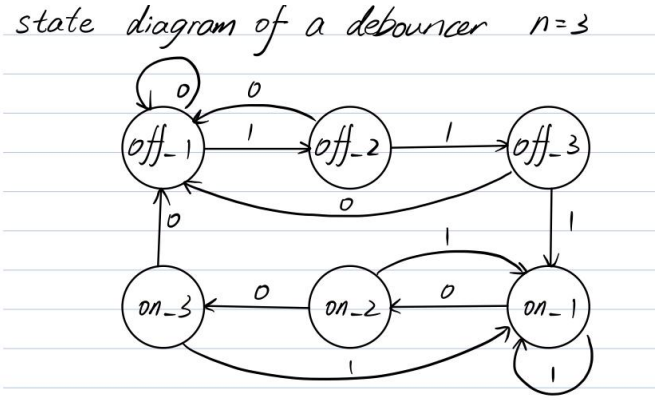
a)

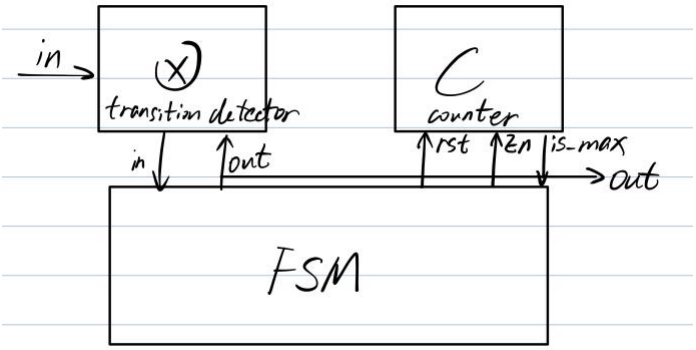


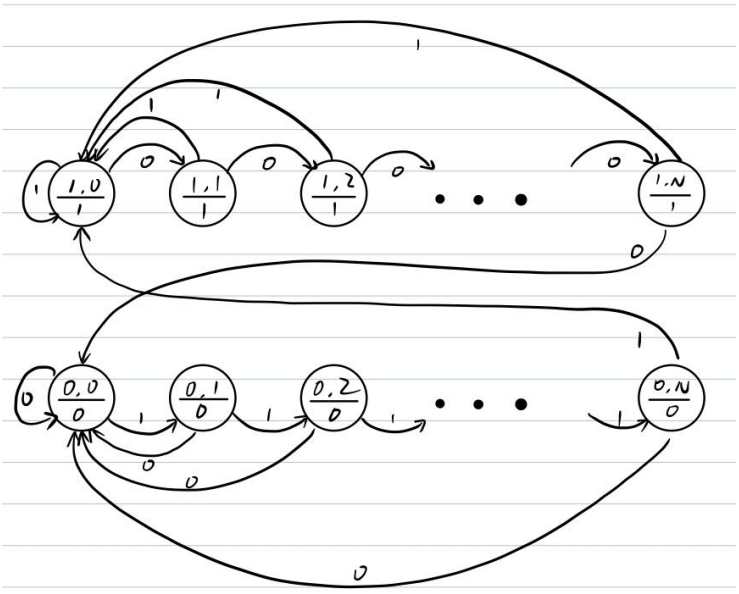
b)



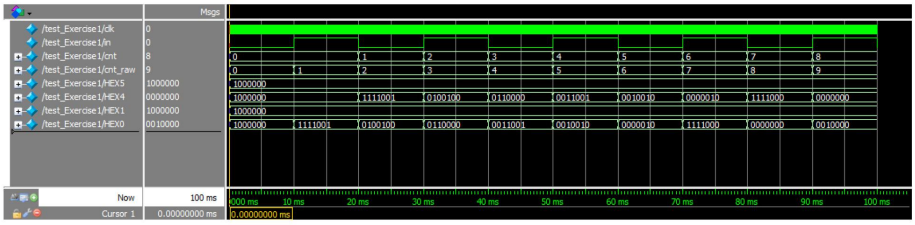
Workshop 6:

1. 

2. 

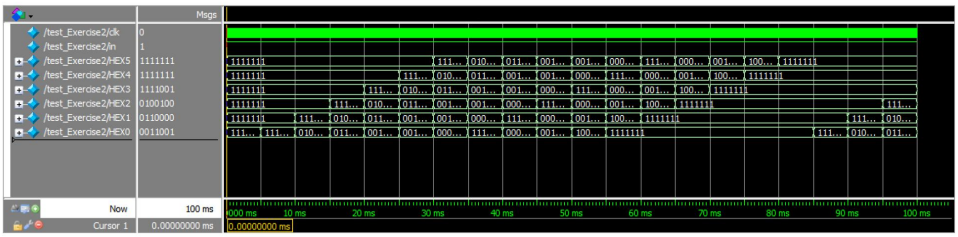
3. 

4. Exercise 1: Debounce the Bounce



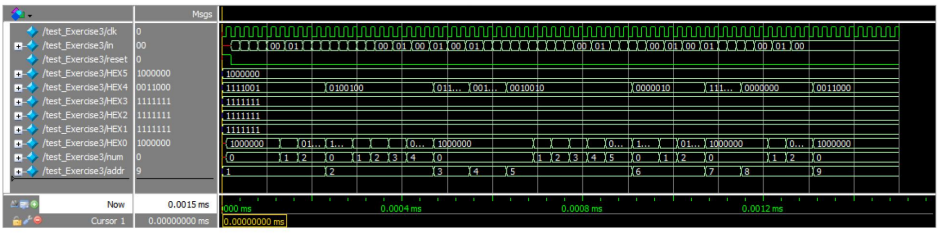
a)

5. Exercise 2: Rolling Fast and Slow



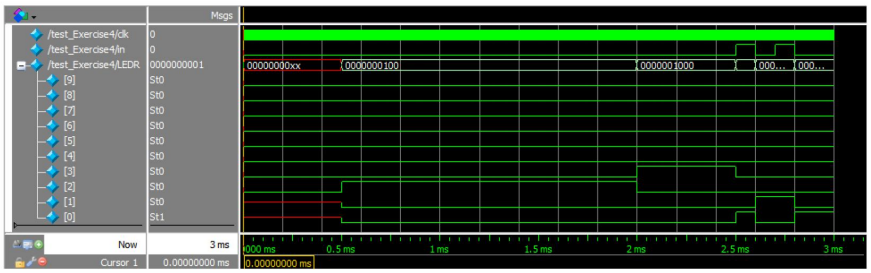
a)

6. Exercise 3: Editing a 10-digit Number



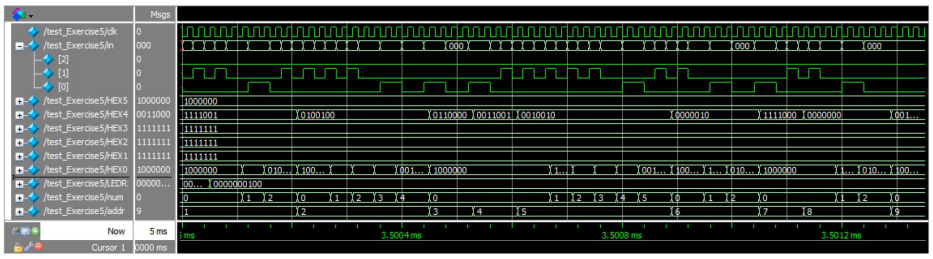
a)

7. Exercise 4: Mode Selection



a)

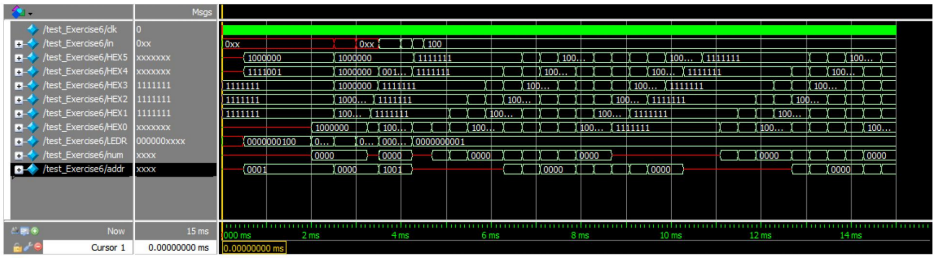
8. Exercise 5: Adding Reset to Exercise 3



a)

9. Exercise 6: The Finished Product

a)


Msgs
/test_Exercise6/clk
/test_Exercise6/in
/test_Exercise6/HEX5
/test_Exercise6/HEX4
/test_Exercise6/HEX3
/test_Exercise6/HEX2
/test_Exercise6/HEX1
/test_Exercise6/HEX0
/test_Exercise6/LEDR
/test_Exercise6/num
/test_Exercise6/addr
Now
Cursor 1
0.00000000 ms
2 ms
4 ms
6 ms
8 ms
10 ms
12 ms
14 ms