

Design of a Binary Number Lock (using schematic entry method)

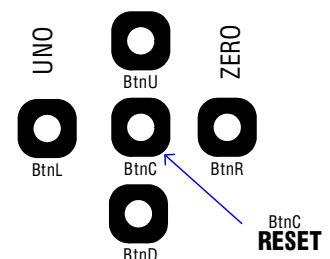
1. Synopsis:

This lab gives you more exercise in state machine design using the one-hot state method, further understanding of the Nexys 3/Nexys 4 board I/O resources (buttons, switches, LEDs, and SSDs), and extends your knowledge of top-designs to utilize these resources.

Until Fall 2019, we did this lab on Nexys 3 using schematic entry method. However the schematic entry tool in Xilinx ISE 14.7 is buggy. The current Xilinx Vivado tool does not support schematic entry. Hence we converted this lab as schematic-entry-on-paper lab. For easy porting and also to maintain resemblance to the earlier lab on Nexys 3, we use the same resources of the board that we used earlier, namely 8 singular LEDs (LD0 to LD7) on the right (instead of 16 LEDs), 8 switches on the right (instead of 16 switches) and 4 SSDs on the right (instead of the 8 SSDs).

2. Description of the Circuit:

In this design you will implement a slightly larger state machine than the simple Detour Signal state machine. In your design you will have two push buttons -- UNO and ZERO; *UNO in Spanish means ONE*. The two signals come out of the push button unit and into your state machine as inputs -- called **u** and **z**. The **u** signal goes high when UNO is pressed and the **z** signal goes high when the ZERO is pressed. Both signals remain low if neither is pressed. *Assume that your state machine is clocked by approximately 10Hz clock (0.1 second per clock cycle). Humans tend to press a push button usually anywhere between a one-eighth second to a quarter second. So once your state machine detects that a push button is pressed it should wait until the button is released -- your design should not interpret a long push as multiple pushes.*



The binary Number Lock secret code is **1 0 1 1**.

If the entered sequence is wrong, the state machine should return to the **INITIAL** state and start looking from the beginning. That is, if **1 0 1 0 1 1** is pressed, the number lock will not open even though the last four bits match with the code. This is because after **1 0 1 0** the machine returns to the **INITIAL**. We assume that the user will not press both the buttons together. This assumption simplifies the design. *However, one should not succeed in opening the lock by pressing both the buttons together every time!*

As in the earlier detour design we divide the design into a core design and a top design. We will provide you with about 60% of the core design and about 90% complete top design.

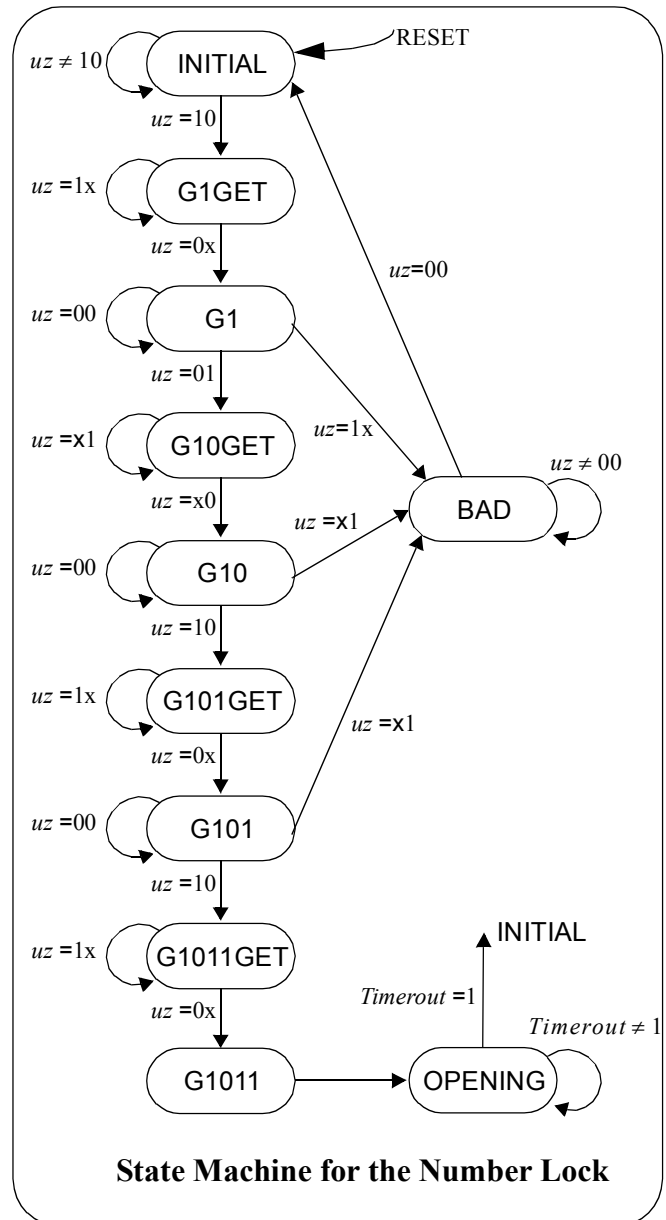
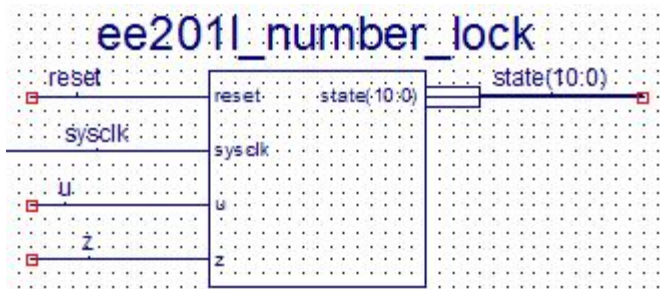
2.1 Core Design (ee254l_number_lock):

Your core design receives `sysclk`, `reset`, `u`, and `z`. It steps through 11 states based on the `u` and `z` inputs. It reports the current state with the 11-bit one-hot coded state vector `state(10:0)`.

2.1.1 State Machine:

You are provided with a complete (and correct) state diagram. The state machine starts in an **INITIAL** state and as the user enters the Number Lock Code (by pressing UNO or ZERO, one at a time) the state machine moves through its states. Note the naming convention followed in the state machine: state **G1** means “got a 1”. Before this we have **G1GET** which means that we are in the process of “getting a 1”, meaning that the UNO button was pressed but has not been released yet.

At **RESET**, the state machine enters the **INITIAL** state and waits for valid input. If the UNO button (**BTNL** on our Nexys board) is pressed while the ZERO button (**BTNR**) is not, then the state machine enters **G1GET**. So the combination that takes you to **G1GET** is **UZ=10**. **G1GET** state means that you are still holding down the UNO button. Thus as long as **UZ=1x** (which means that UNO is pressed and you don't care about ZERO here) you remain in **G1GET**. When you release the UNO button, you go to state **G1** (releasing a button sends a 0 and hence **UZ** must be **0x** for the state machine to transit from **G1GET** to **G1**). This process continues if you keep entering the correct sequence, i.e. 1 0 1 1. Otherwise, the state machine moves to the **BAD** state and then back to **INITIAL**. If the entered sequence is correct then the state machine enters the **G1011** state which means that it got (received) 1011 -- the correct code. This can also be thought of as the “**DONE**” state for the system. The system remains in this state for only one clock cycle and then moves to the **OPENING** state. It stays in **OPENING** state until a counter times out. When the timer times out (i.e., **TIMEROUT = 1**), the state machine moves to the **INITIAL** state. Notice that while the machine is in the **OPENING** state the push buttons are ignored.



2.2 The Top-level schematic (ee254l_number_lock_top.sch)

The top-level schematic contains the connections between the I/O components on the Nexys board and our core design. The top-level design is nearly complete except for the SSD anode controls. To hide the details of the core design we use hierarchical design. Major components in the top-level design are described below. We provide a Verilog module to divide the main clock on the Nexys Board (**ClkPort**) and produce a slow clock (**sysclk**) for your core design. The board clock works at a high frequency “f” (100MHz). The derived slow clock **div_clk25** drives the **sysclk** which makes the core state machine run slow enough and effectively debounces the push-buttons. This **sysclk** is also tied to **1d2** and the **1d2** keeps flashing. Your TA may explain to you briefly about bouncing of mechanical switches and push buttons and how a slow clock helps here. A detailed discussion of bouncing is postponed to a later lab. Count the number of times it flashes in 30 sec. and verify your calculation of the **div_clk25** frequency below.

Also the four anodes of the four right-side seven-segment displays are to be activated in rotation but at slightly higher frequency. The pair of clock signals, **div_clk18** and **div_clk19**, produced by the clock counter, are used for this purpose. Complete the values for **x**, **y**, **z**, **p**, **q**, and **r** below.



Assume that the input clock (ClkPort) frequency is “f”.

Frequency of $\text{div_clk}[21] = f/2^X$ where $X = \underline{22}$

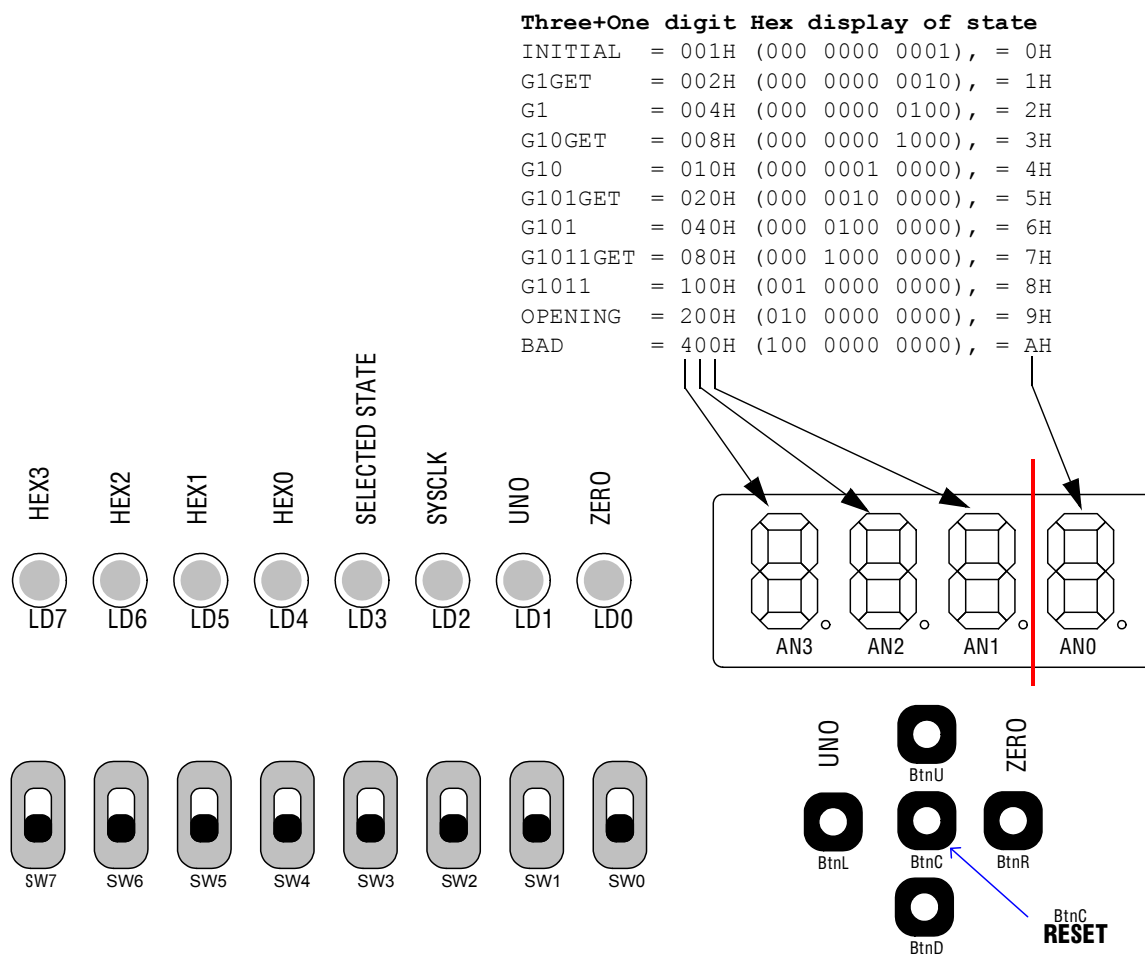
Frequency of $\text{div_clk}[23] = f/2^Y$ where $Y = \underline{24}$

Frequency of $\text{div_clk}[13] = f/2^Z$ where $Z = \underline{14}$

Frequency of $\text{div_clk}[1] = f/2^P$ where $P = \underline{2}$

Frequency of $\text{div_clk}[2] = f/2^Q$ where $Q = \underline{3}$

Frequency of $\text{div_clk}[25] = \text{Frequency of sysclk} = f/2^R$ where $R = \underline{26}$



2.2.1 System Reset, UNO and ZERO buttons/LEDs:

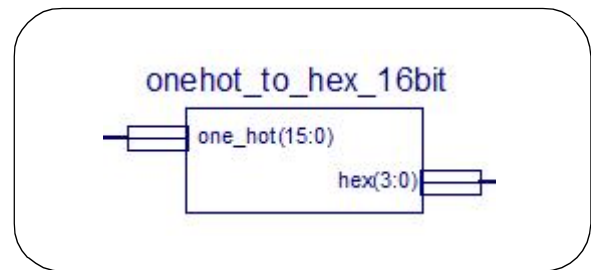
We use button **BtnC** on Nexys board to reset the system. The two buttons **BtnL** and **BtnR** generate the signals **UNO** and **ZERO** respectively. So the sequence **BtnL** > **BtnR** > **BtnL** > **BtnL** (which is 1011) should open the lock. LEDs **LD1** and **LD0** are connected to **UNO** and **ZERO** buttons, respectively.



2.2.2 Display of state output from the core design:

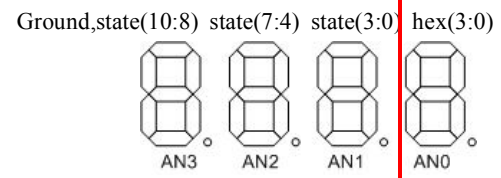
All eleven state bits are available as a state vector `state[10:0]`. The state information will be displayed in three ways. The 11-bit One-Hot code is prepended (appended on the left side) with five zeros and then the 16-bit value is converted to a 4-bit hexadecimal number (`hex[3:0]`) using a One Hot-to-hex converter (`one-hot_to_hex_16bit`).

```
Ground,Ground,Ground,Ground,Ground,state(10:0)
```

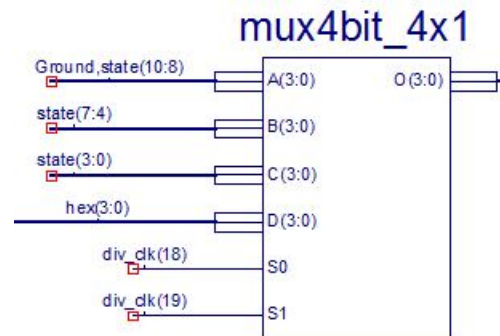


We also prepend a zero to the `state[10:0]` and treat it as a 12-bit binary number. We treat this as three hex digits `GND`, `state[10:8]`, `state[7:4]`, and `state[3:0]`. These three hex digits and the hex digit output by the converter are displayed on the four seven-segment displays as shown.

Seven segment display LEDs: Please recall the 7-segment display scanning mechanism from the previous lab(s).



In this design we want to display four digits on the four right-side SSDs. This requires four different data values to be displayed on the four SSDs *simultaneously*. -- at least so should it *appear* to human eye. We accomplish this by using the output scanning mechanism so that each of the four SSDs is sequentially selected and the corresponding data is sent to the common 7-bit bus. Carefully look at connections already made to the 4-bit wide 4 to 1 mux (`mux4bit_4x1`) and then arrive at the needed connections for the 2-to-4 decoder (`D2_4E`).



3. Procedure:

~~3.1 Download the zip file **ee2541_number_lock_sch.zip** containing the Xilinx ISE project.~~

3.1 Download Design files zip file (**number_lock_schematic_lab_design_files.zip**) containing the following 6 design files

1. **clock_divider.v** (It just divides the clock. This was the underlying Verilog file for the symbol clock_divider in the schematic)
2. **ee201_numlock_sm.v** (a Verilog Netlist file for the core state machine to allow you to simulate as well as synthesize)
3. **ee201_numlock_sm_tb.v** (testbench to test the above state machine)
4. **ee201_numlock_complete.v** (a Verilog Netlist file for the top design to allow you to synthesize and produce a .bit file)
5. **ee354_top.xdc** (a Xilinx design Constraints file for the top design to allow you to synthesize and produce a .bit file)
6. **TAs_ee201_numlock_top.bit** (for you to get acquainted with the expected behavior of the number lock).

Also download the **number_lock_schematic_lab_paper_submission.zip** containing the following 5 pdf files.

Number_Lock_state_digram_exercise_for_0100_code.pdf

ee2011_number_lock_core_3page_schematic.pdf (incomplete core schematic to be completed by you)

ee2011_number_lock_top_3page_schematic.pdf (incomplete top schematic to be completed by you)

number_lock_schematic_lab_Lab_Report.pdf (2 pages to be completed by you)

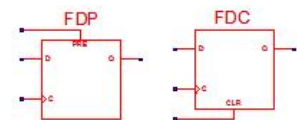
number_lock_schematic_lab_9page_paper_submission.pdf (one pdf combining all 9 pages, please try to use this)


3.2 Upload the TA's bit file (**TAs_ee201_numlock_top.bit**) to the board to observe the expected operation of the design.

3.3 Complete the **Number_Lock_state_digram_exercise_for_0100_code.pdf**.

3.4 Complete the core design **ee2011_number_lock_core_3page_schematic.pdf** on paper. Notice that there are three pages in the schematic. The first page consists of the state memory flip-flops and the next two pages consist mainly the next state logic for the 11 states besides the counter acting as timer for the opening state. The core design is about 60% complete.

On the first sheet we provided 10 flip-flops (**FDP**) for 10 of the 11 states. You must add one more flip-flop (**FDC** or **FDP** you decide) for the initial state.



Complete the missing portions of the next state logic on the second and third sheets. Notice that there is no OFL (output function logic) in this trivial design since the states themselves act as outputs. ~~Save and check your schematic using~~ 

3.5 Complete the **ee2011_number_lock_top_3page_schematic.pdf**.

3.6 A Verilog netlist is Verilog file produced by the Vivado tool after synthesizing (or synthesizing and implementing). So it is a low-level design and difficult to interpret by humans. The TA is going to

provide such files for the completed core design and completed top design for you to learn simulation and synthesis. We are providing a netlist version of the number lock core design and the top design because you will be writing the actual Verilog code in a subsequent lab.

3.7 You are provided with the following Verilog files

1. `clock_divider.v`

(this divides the board clock; this was the underlying Verilog file for the symbol `clock_divider` in the schematic)

2. `ee201_numlock_sm.v`

(a Verilog Netlist file for the core state machine to allow you to simulate as well as synthesize)

3. `ee201_numlock_sm_tb.v`

(testbench to test the above state machine)

4. `ee201_numlock_complete.v`

(a Verilog Netlist file for the top design to allow you to synthesize and produce a .bit file)

3.8 Using the provided Verilog test fixture (`ee201_numlock_sm_tb.v`) simulate the core design, `ee201_numlock_sm.v`. Read the testbench in notepad++.

First read the testbench in notepad++.

Notice the timescale directive: ``timescale 1ns / 1ps` (format ``timescale unit /precision`).

Notice that the two `initial` procedural blocks are independently and simultaneously controlling the `u` and `z` signals (one controlling `u` and the other controlling `z`). The statement “`#520;`” means wait for 520 units (which is 520ns since the timescale directive above specifies units as 1ns) before executing the next statement. Hence these delays are incremental (additional) delays.

```
//test case 1
initial begin u = 0; #520; u = 1; #500; u = 0; #1000; u = 1; #500; u = 0; #500; u = 1; #500; u = 0; end
initial begin z = 0; #1200; z = 1; #500; z = 0; end
```

Figure out how long you need to simulate to exhaust the longest pattern you are applying to `u` and `z`. Arrive at these sums for test case #1:

$$520 + 500 + 1000 + 500 + 500 + 500 = \underline{3520} \text{ ns} \quad 1200 + 500 = \underline{1700} \text{ ns}$$

For each of the three test cases in the testbench, what sequence of states you expect the number lock to go through? Verify that the waveform you produced validates your prediction. For the case(s) where you actually enter the opening state use waveform cursors to measure the time period for which the system stays in the opening state. Also count the number of clocks for which the system stayed in the opening state. The `timerout` control signal is taken from the `CEO` output of the `CB4CE` (4-bit counter). Look at the data sheet on page 98/681 of [pdf](#) (in ISE14.7 Help => Software Manuals => Libraries Guide => Spartan-6 Libraries Guide for Schematic Designs => scroll down and locate the symbol) and *see if the `CEO` signal goes active when the count is showing 15 or after the count just finished 15*. Does the state machine move from the opening state to initial state *the moment* the `timerout` signal goes active *or* when `timerout` is active *and* the (next) positive edge of `sysclk` occurs?

$$z = \text{bit width} - 1$$

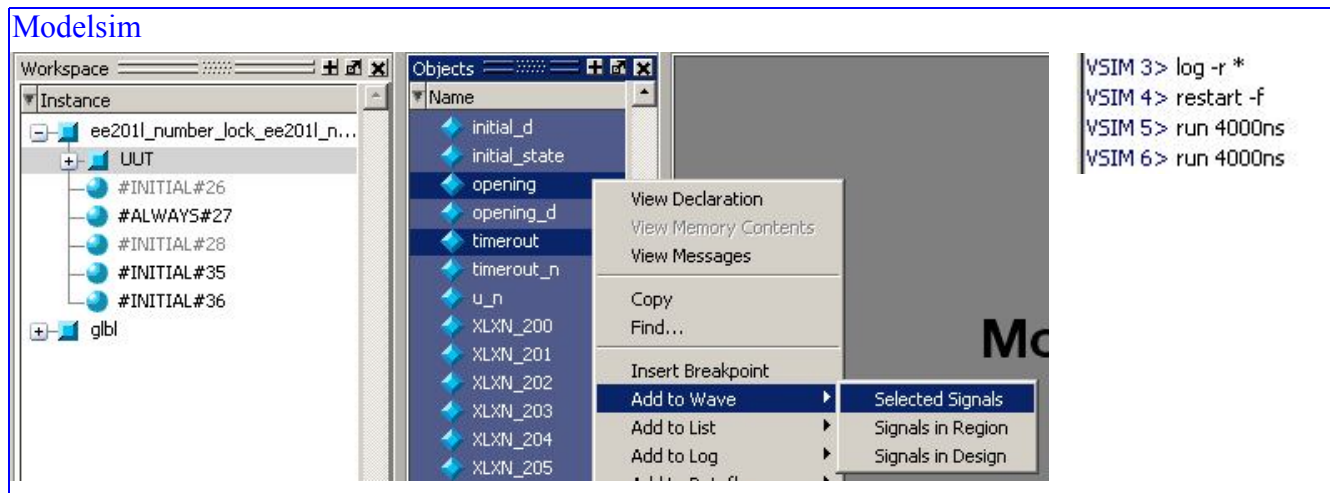
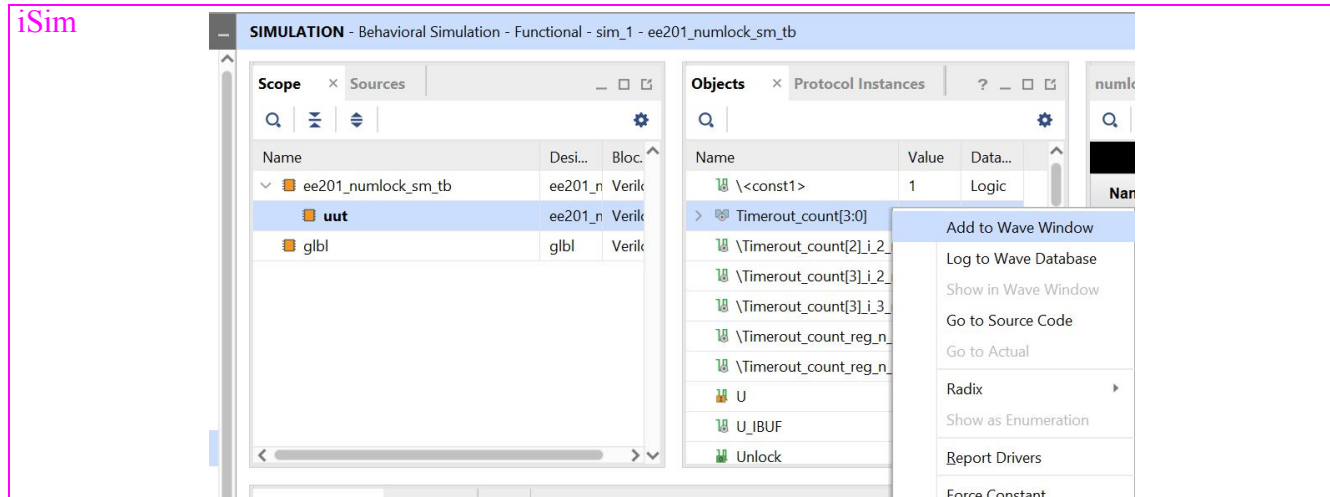
$$TC = Qz \cdot Q(z-1) \cdot Q(z-2) \cdot \dots \cdot Q0$$

$$CEO = TC \cdot CE$$

Well, doesn't the `timerout` go active on a positive edge of `sysclk`? If that is the case, the above questions seems to be a moot (or dumb) question. Or if one of the two (`sysclk` and `timerout`) is a cause and the other is the effect, since effect takes place *after* the cause, there may be an issue here to think!

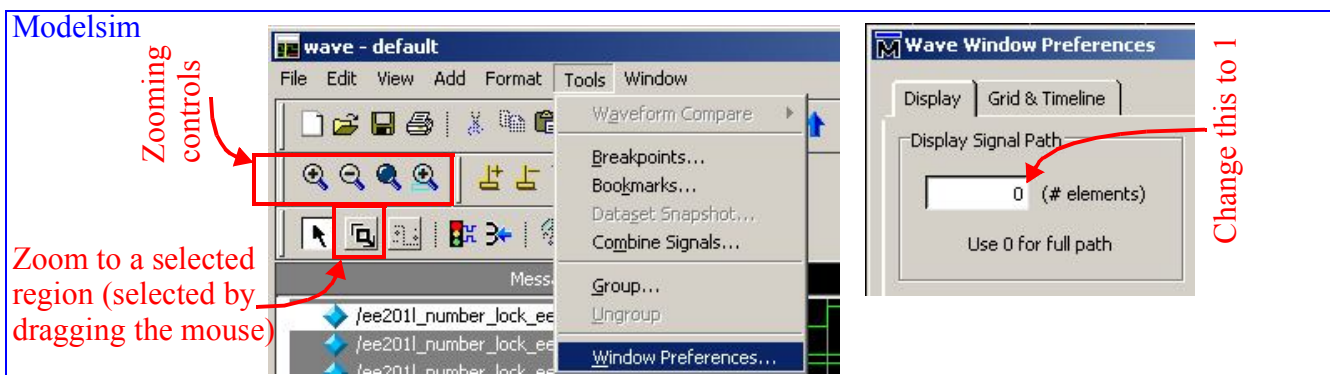
Show the waveform for each of the three test cases to your TA.

3.9 Your TA can help you with **iSim/Modelsim**. He/she can show you how to add signals from deeper in the schematic hierarchy. After adding new signals we need to restart and resimulate because the newly added signals may not show past activity (the simulation time before the addition of these new signals). The “**log -r ***” simulation command tells the tool to keep logging signal activity on *all* signals whether or not they are displayed on the waveform or not. This helps if you decide to add some more signals after simulating for a while.



If your waveform is displaying full path of signals, you can change it to display just the signal name (without the complete hierarchical path) by doing the following in Modelsim (Tools => Window Preferences => Display Signal Path = 1.

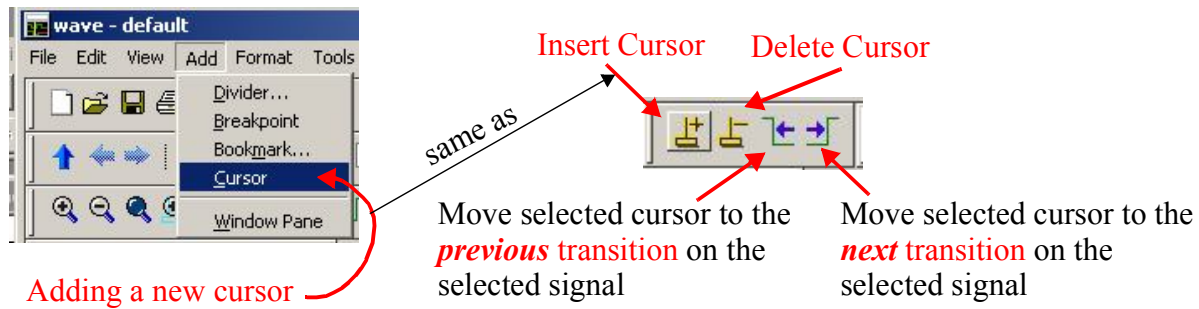
Some useful cursor controls



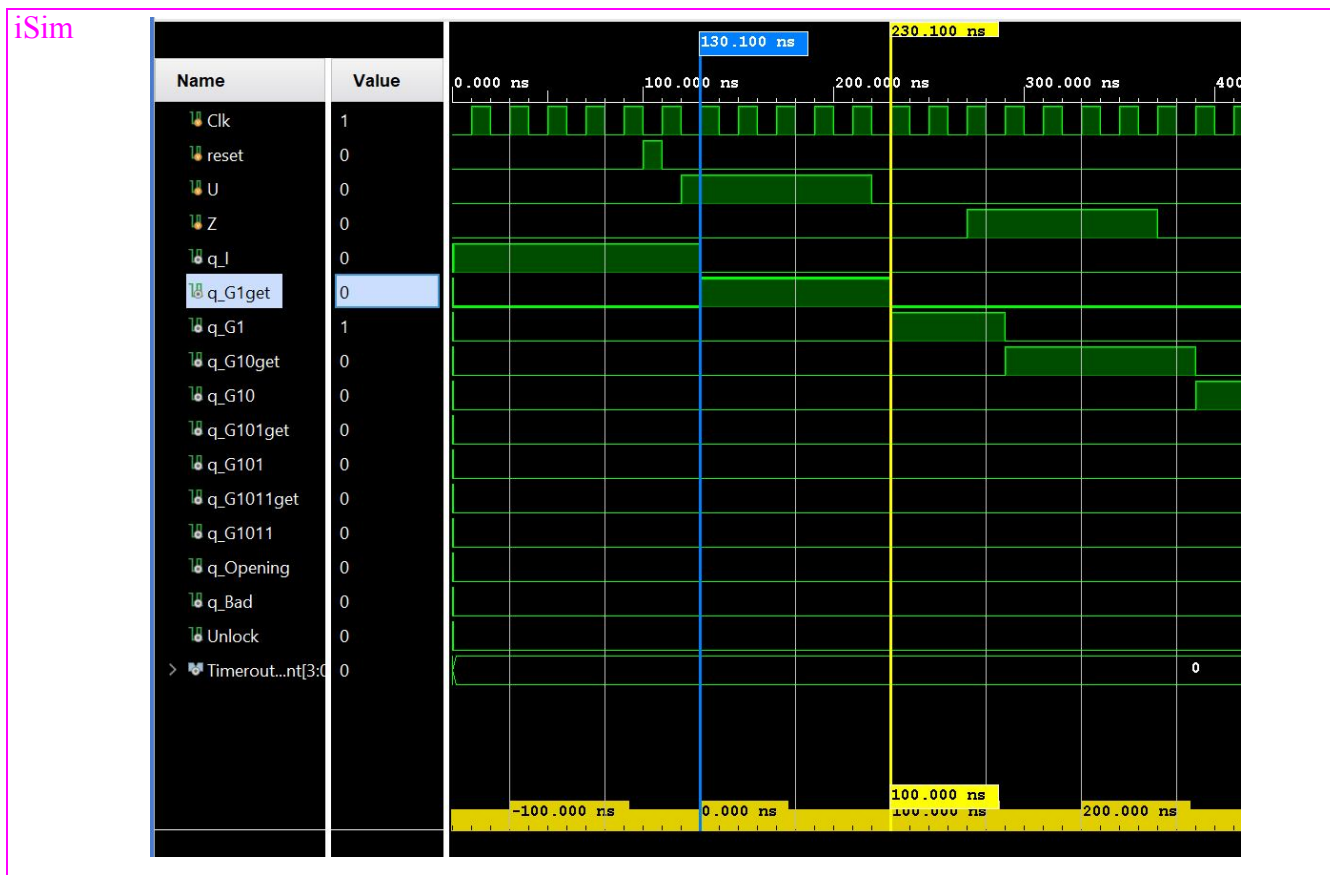
iSim



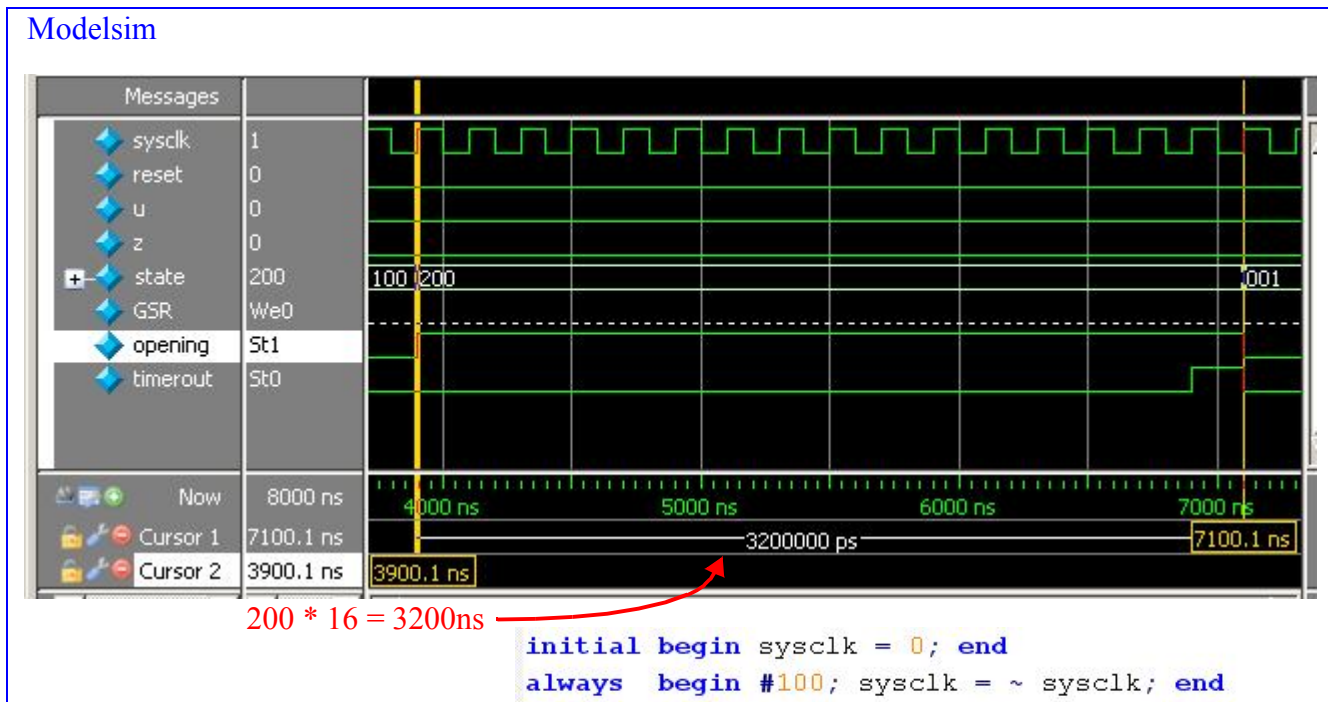
Modelsim



Practice displaying waveforms and working with the waveforms.



Aligning cursors to the beginning and ending of the “opening” period and measuring time period:



3.10 Now complete the “top” design (`ee2541_number_lock_top.sch`) on paper. Go through the 3-sheet schematic. Sheets 1 and 3 are fully complete! Sheet 2 is 70% complete. You need to complete:

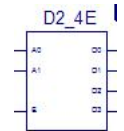
(a) the logic driving the anodes (using a **D2_4E** decoder).

(b) I/O markers for the anodes and naming the connected nets.

Pay attention to the order of anode labels. Determine if you want inverting or non-inverting buffers (

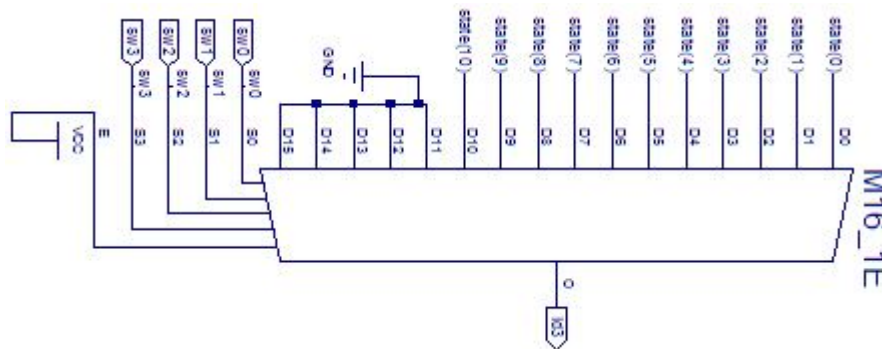


) at the input or the output of the 2-to-4 decoder **D2_4E**. Check to see if the decoder has active high or active low outputs. Also check whether the enable input is active high or active low by reading the symbol info.



Does it matter in which order you cause the 4 SSDs display their respective information as long as you synchronize the display of these 4 in some (fixed) order? What matters is that the anode control design should match with the mux control (`mux4bit_4x1`) so that there is coordination between activation of anodes and conveyance of the respective SSD data. Notice that the provided bit file actually makes the right most digit glow **5 times brighter** than the other three displays. You do not have to do this but if you have time, you can discuss with your TA how to do this. ~~Save and check your schematic using~~

3.11 Notice the 16-to-1 mux on page 1 of the schematic. In some designs, the number of I/O pins on the package is limited and we can not bring out a large number of signals (such as all the 11 state outputs). So we must implement a method to monitor any one state at a time using less number of pins. Using the four switches, **SW[3:0]**, and the 16-to-1 mux we are able to select and display any one of the states signals on LED #3 (**LD3**).



Note: If you take EE658 “Diagnosis and Design of Reliable Digital Systems” you will be introduced to more exotic test methods.

3.12 Flashing of the 7-segment displays during the **OPENING** state is achieved by the following circuit. The cathodes are active-low. So, during the **OPENING** state (**state[9]=1**), **sysclk** sends a 1 to the cathodes to turn them off for half-clock of every clock.

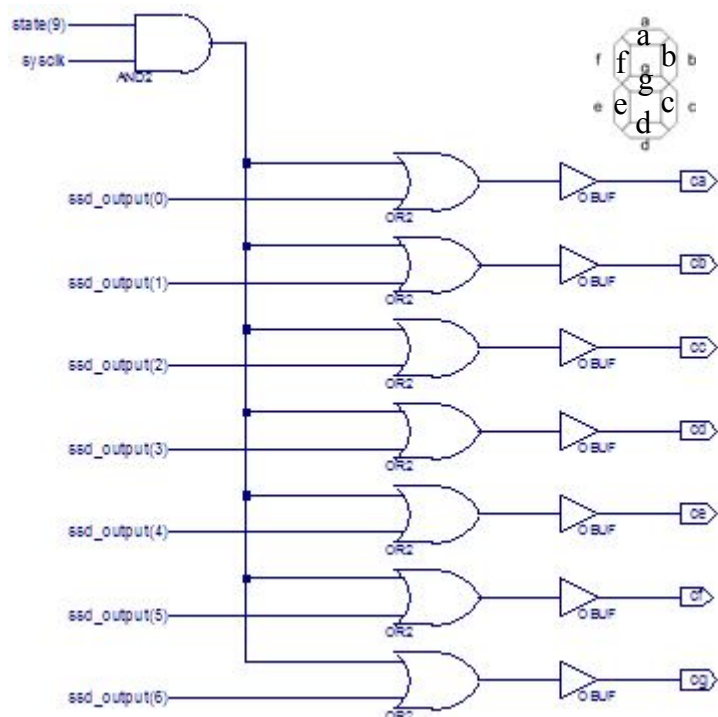
Analyze and understand the circuit on the side.

Suppose we replace the seven 2-input **OR** gates with seven 2-input **AND** gates. Do you think you can make the circuit work by either changing the single **AND** gate or by inverting the **ssd_output[6:0]**, or do you think it is just not possible?



Discuss your ideas with your TA.

It is worthwhile to look at the Quiz exam questions

[ee354L_selected_Lab_Questions_from_past_exams.pdf](#)



3.13 You know that a combinational logic can be implemented using a ROM. And you also know that a bigger ROM (a 16x8 ROM) can be built using several smaller ROMs (8 of 16x1 ROMs) (i.e. an 8-bit wide memory can be built using 8 of single-bit wide memories, all are 16 locations deep).

Given below is the truth table for the `hex_to_ssd` converter. For the 8 outputs, we have 8 ROMs as shown in the `hex_to_ssd.sch`. Suppose you want the display for the number 9 as  (with bottom segment not glowing) instead of . What modifications do you need to make to the truth table and which ROM(s) do you need to modify? Discuss your ideas with your TA.

HexDigit	hex(3)	hex(2)	hex(1)	hex(0)	Cg	Cf	Ce	Cd	Cc	Cb	Ca	dp
0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	1	1
2	0	0	1	0	0	1	0	0	1	0	0	1
3	0	0	1	1	0	1	1	0	0	0	0	1
4	0	1	0	0	0	0	1	1	0	0	1	1
5	0	1	0	1	0	0	1	0	0	1	0	1
6	0	1	1	0	0	0	0	0	0	1	0	1
7	0	1	1	1	1	1	1	1	0	0	0	1
8	1	0	0	0	0	0	0	0	0	0	0	1
9	1	0	0	1	0	0	1	0	0	0	0	1
A	1	0	1	0	0	0	0	1	0	0	0	1
b	1	0	1	1	0	0	0	0	0	1	1	1
C	1	1	0	0	1	0	0	0	1	1	0	1
d	1	1	0	1	0	1	0	0	0	0	1	1
E	1	1	1	0	0	0	0	0	1	1	0	1
F	1	1	1	1	0	0	0	1	1	1	0	1
ROM (Look Up Table) Values =====>					1083	208E	02BA	8492	D004	D860	2812	FFFF

3.14 Using Vivado tool, please produce you own .bit file from the following provided files:

1. `ee201_numlock_sm.v`

(a Verilog Netlist file for the core state machine to allow you to simulate as well as synthesize)

2. `ee201_numlock_complete.v`

(a Verilog Netlist file for the top design to allow you to synthesize and produce a .bit file)

3. `ee354_top.xdc`

Show to your TA that you were able to successfully produce a .bit file, and upload the same on to the Nexys-4 board.

4. Lab Report:

Name: <u>Scott Susanto</u>	Date: <u>09/13/2022</u>
Lab Session: <u>31309</u>	TA's Signature: _____

For TAs: Paper design, Simulation of a compiled core, and analysis, synthesis and generation of .bit file, and demo (70 points): _____ Report (30 points): _____
Comments:

Q 4. 1: You have simulated the core design, `ee201_numlock_sm.v` using `ee201_numlock_sm_tb.v`. Attach (to the report) the simulation waveform for a/the test case which led you open the lock.
checked off in Lab

Q 4. 2: Complete the following three items and attach them to this report.

`Number_Lock_state_digram_exercise_for_0100_code.pdf` ✓
`ee2011_number_lock_core_3page_schematic.pdf` ✓
`ee2011_number_lock_top_3page_schematic.pdf` ✓

Q 4. 3: What is the reason for commenting out the `BtnU` and `BtnD` lines in the `.xdc` file (`ee354_top.xdc`) (2 pts)?

they are not used

```
#Bank = 15, Pin name = IO_L14P_T2_SRCC_15,      Sch name = BTNU
#set_property PACKAGE_PIN F15 [get_ports BtnU]
#set_property IOSTANDARD LVCMOS33 [get_ports BtnU]
```



```
#Bank = 14, Pin name = IO_L21P_T3_DQS_14,      Sch name = BTND
#set_property PACKAGE_PIN V10 [get_ports BtnD]
#set_property IOSTANDARD LVCMOS33 [get_ports BtnD]
```

Q 4. 4: Fill-up the frequencies of the clock divider (6 pts)

Q 4. 5: Flashing effect created in the OPENING state:
Refer to section 3.12 above. Suppose we replace the seven 2-input OR gates with seven 2-input NAND gates. What other modifications you need to make to your circuit in order to make it work? Think about DeMorgan's theorem! (6 pts)

*replacing the OR with NAND gates would
require us to invert both inputs*

Assume that the input clock (ClkPort) frequency is f .
Frequency of `div_clk[21]` = $f/2^X$ where $X = \underline{22}$
Frequency of `div_clk[23]` = $f/2^Y$ where $Y = \underline{24}$
Frequency of `div_clk[13]` = $f/2^Z$ where $Z = \underline{14}$
Frequency of `div_clk[1]` = $f/2^P$ where $P = \underline{2}$
Frequency of `div_clk[2]` = $f/2^Q$ where $Q = \underline{3}$
Frequency of `div_clk[24]` = Frequency of `sysclk` = $f/2^R$ where $R = \underline{25}$

Q 4. 6: Refer to section 3.11: What would you do to change the display of nine to  from  (6 pts)

HexDigit	hex(3)	hex(2)	hex(1)	hex(0)	Cg	Cf	Ce	Cd	Cc	Cb	Ca	dp
0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	1	1	1	1	0	0	1	1
2	0	0	1	0	0	1	0	0	1	0	0	1
3	0	0	1	1	0	1	1	0	0	0	0	1
4	0	1	0	0	0	0	1	1	0	0	1	1
5	0	1	0	1	0	0	1	0	0	1	0	1
6	0	1	1	0	0	0	0	0	0	1	0	1
7	0	1	1	1	1	1	1	1	0	0	0	1
8	1	0	0	0	0	0	0	0	0	0	0	1
9	1	0	0	1	0	0	1	0	0	0	0	1
A	1	0	1	0	0	0	0	1	0	0	0	1
b	1	0	1	1	0	0	0	0	0	1	1	1
C	1	1	0	0	1	0	0	0	1	1	0	1
d	1	1	0	1	0	1	0	0	0	0	1	1
E	1	1	1	0	0	0	0	0	1	1	0	1
F	1	1	1	1	0	0	0	1	1	1	0	1
ROM (Look Up Table) Values =====>					1083	208E	02BA	8492	D004	D860	2812	FFFF

deactivate Ce (0→1)

Q 4. 7: Suppose that you could press a button exactly for one clock period. State why the following state diagram does not work. Fix the state diagram. Include a “BAD” state. (10 pts)

It would not account for the button release

