

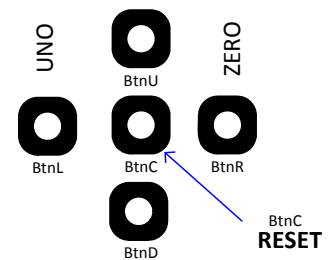
Design of a Binary Number Lock (using schematic entry method)

1. Synopsis:

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

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 quarter second to half a 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.



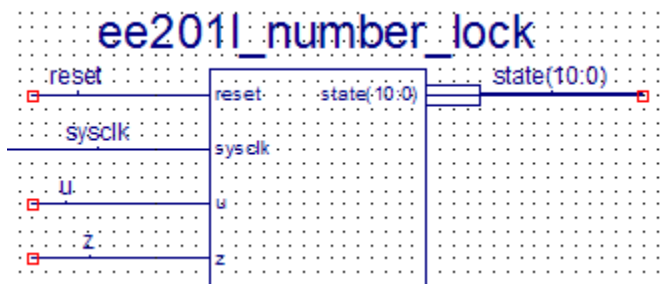
The binary Number Lock secret code is **1 0 1 1**.

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. *One should not succeed in opening the lock by pressing both the buttons together every time though.*

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 (ee201l_number_lock):

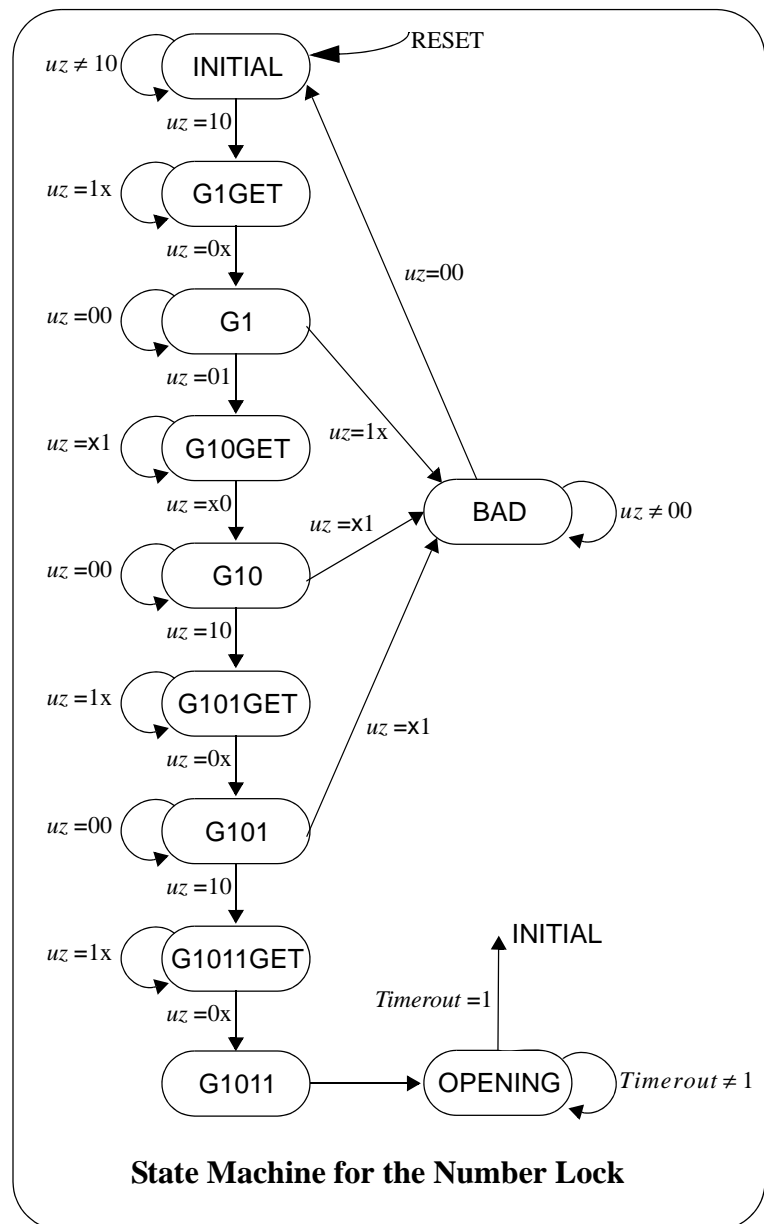
Your core design receives **sysclk**, **reset**, **u**, and **z**. It steps through 11 states based on the **u** and **z** inputs. It reports the current state with the 11-bit one-hot coded state vector **state(10:0)**.



2.1.1 State Machine:

You are provided with a complete (and correct) state diagram. The state machine starts in the **INITIAL** state and as the user enters the Number Lock Code (by pressing UNO and ZERO buttons) 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 3 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) you remain in **G1GET**. When you release the UNO button then 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 button are ignored.



2.2 The Top-level schematic (ee201l_number_lock_top.sch)

The top-level schematic contains the connections between the I/O components on the Nexys 3 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 3 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 **ld2** and the **ld2** 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 seven-segment displays are to be activated in rotation but at slightly higher frequency. **div_clk18** and **div_clk19** bits of the clock counter are used for this. 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 $X = \underline{\hspace{1cm}}$

Frequency of $\text{div_clk}[23] = f/2^Y$ where $Y = \underline{\hspace{1cm}}$

Frequency of $\text{div_clk}[13] = f/2^Z$ where $Z = \underline{\hspace{1cm}}$

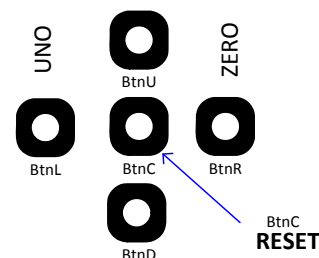
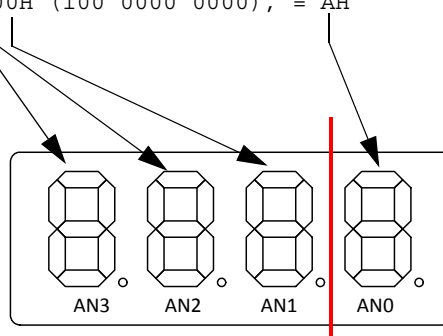
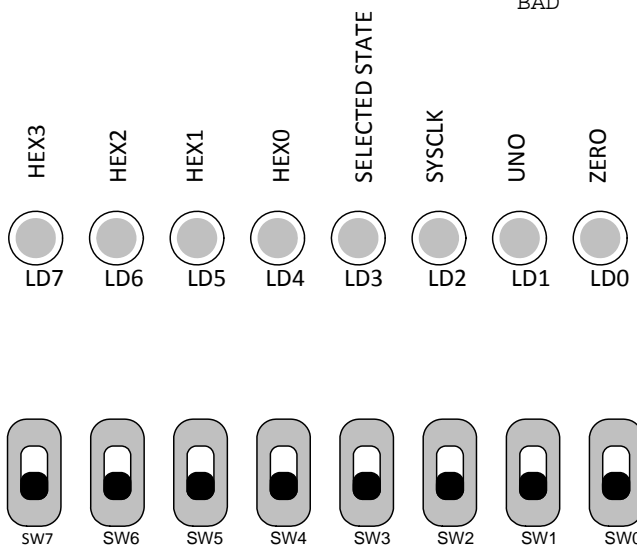
Frequency of $\text{div_clk}[1] = f/2^P$ where $P = \underline{\hspace{1cm}}$

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

Frequency of $\text{div_clk}[25] = \text{Frequency of sysclk} = f/2^R$ where $R = \underline{\hspace{1cm}}$

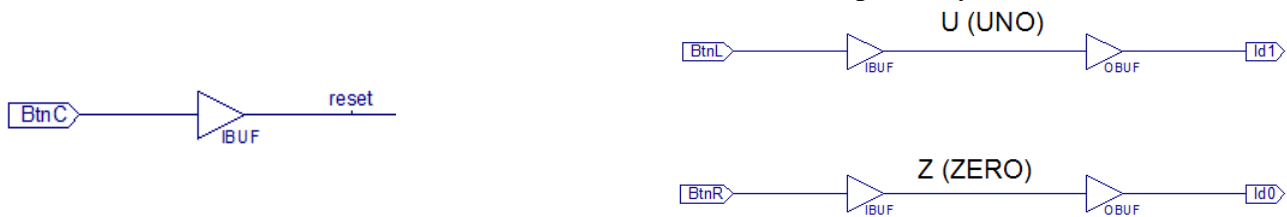
Three+One digit Hex display of state

INITIAL	=	001H	(000 0000 0001),	=	0H
G1GET	=	002H	(000 0000 0010),	=	1H
G1	=	004H	(000 0000 0100),	=	2H
G10GET	=	008H	(000 0000 1000),	=	3H
G10	=	010H	(000 0001 0000),	=	4H
G101GET	=	020H	(000 0010 0000),	=	5H
G101	=	040H	(000 0100 0000),	=	6H
G1011GET	=	080H	(000 1000 0000),	=	7H
G1011	=	100H	(001 0000 0000),	=	8H
OPENING	=	200H	(010 0000 0000),	=	9H
BAD	=	400H	(100 0000 0000),	=	AH



2.2.1 System Reset, UNO and ZERO buttons/LEDs:

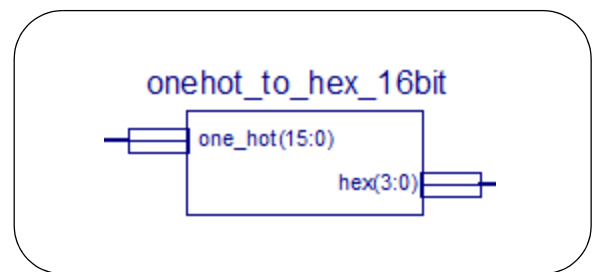
We use button **BtnC** on Nexys 3 board to reset the system. The two buttons **BtnL** and **BtnR** generate the signals UNO and ZERO respectively. So the sequence **BtnL** > **BtnR** > **BtnL** > **BtnL** should open the lock. LEDs **LD1** and **LD0** are connected to UNO and ZERO buttons, respectively.



2.2.2 Display of state output from the core design:

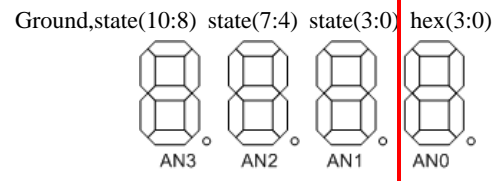
All eleven state bits are available as a state vector **state[10:0]**. The state information will be displayed in three ways. The 11-bit One-Hot code is prepended with five zeros and then converted to a 4-digit hexadecimal number (**hex[3:0]**) using a One Hot-to-hex converter (**onehot_to_hex_16bit**).

```
.....
Ground;Ground;Ground;Ground;Ground;state(10:0)
.....
```

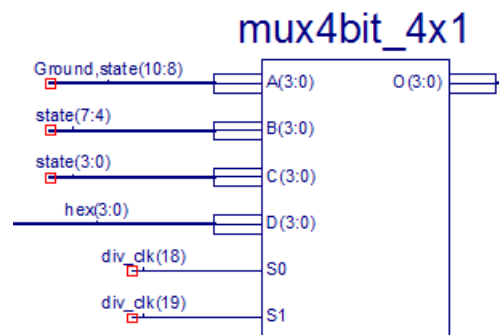


We also prepend a zero to the **state[10:0]** and treated it as a 12-bit binary number. We treat this as three hex digits **GND**, **state[10:8]**, **state[7:4]**, and **state[3:0]**. These three hex digits and the hex digit output by the converter are displayed on the four seven-segment displays as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

Seven segment display LEDs: We assume that you remember the 7-segment display scanning mechanism from the previous lab(s).



In this design we want to display four digits on the SSDs. This requires four different data values to be displayed on the four SSD *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**).



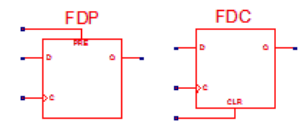
3. Procedure:


3.1 Download the zip file `ee2011_number_lock_sch.zip` containing the Xilinx ISE project from the class website. Extract the project folder `ee2011_number_lock_sch` into the projects folder (`C:\Xilinx_projects\`). Start the project in Xilinx Project Navigator and set up a project with name `ee2011_number_lock_sch` with the above `C:\Xilinx_projects\ee2011_number_lock_sch` as the project folder. We will follow the same procedure to simulate the core design in Xilinx and to implement the top design on our Nexys 3 board as you did in the detour signal lab.

3.2 The above zip file also contains a BIT (.bit) file `TAs_bit_file_ee2011_number_lock_top.bit`. Download this bit file to the board to observe the expected operation of the design. The dot points on the SSDs are constantly on in this provided bit file but in your design the dot points shall be constantly off. We constantly illuminate the dot points so our bit file does not get mixed up with yours!

3.3 Complete the core design. Open the core schematic `ee2011_number_lock.sch` for editing. 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 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 (**FDC**) for 10 of the 11 states. You must add an another flip-flop (**FDC** or **FDP**) 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 you schematic using .

3.4 Use the provided Verilog test fixture (`ee2011_number_lock_tb.v`) to test your design. 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).

```
//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:

$$520 + 500 + 1000 + 500 + 500 + 500 = \underline{\hspace{2cm}} \text{ ns} \quad 1200 + 500 = \underline{\hspace{2cm}} \text{ ns}$$

For each of the three test cases 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 `timeout` control signal is taken from the `CEO` output of the `CB4CE` (4-bit counter). Look at the data sheet (right-click on symbol => Symbol => Symbol info) 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 `tim-`

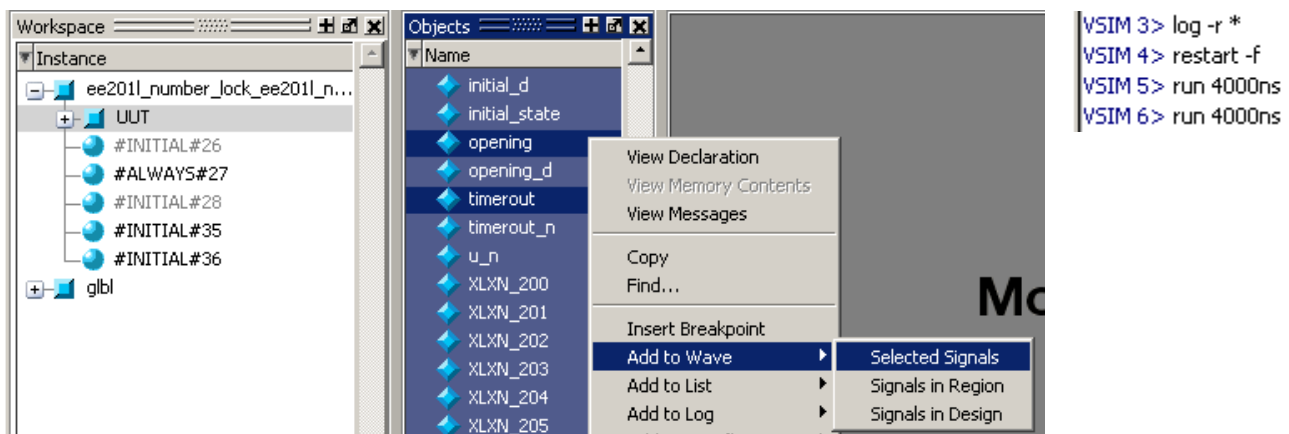
$$\begin{aligned} z &= \text{bit width} - 1 \\ TC &= Qz \bullet Q(z-1) \bullet Q(z-2) \bullet \dots \bullet Q0 \\ CEO &= TC \bullet CE \end{aligned}$$

erout signal goes active *or* when **timerout** is active *and* the positive edge of **sysclk** occurs?

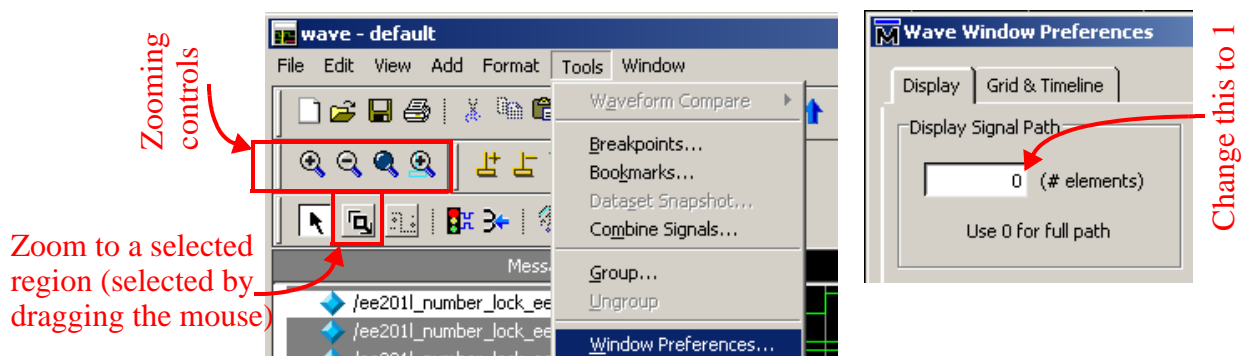
Well doesn't the **timerout** go active on a positive edge of **sysclk**? If that is the case, the above questions seems to be a moot (or dumb) question. Or if one of the two (**sysclk** and **timerout**) is a cause and the other is the effect, since effect takes place after the cause, there may be an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

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

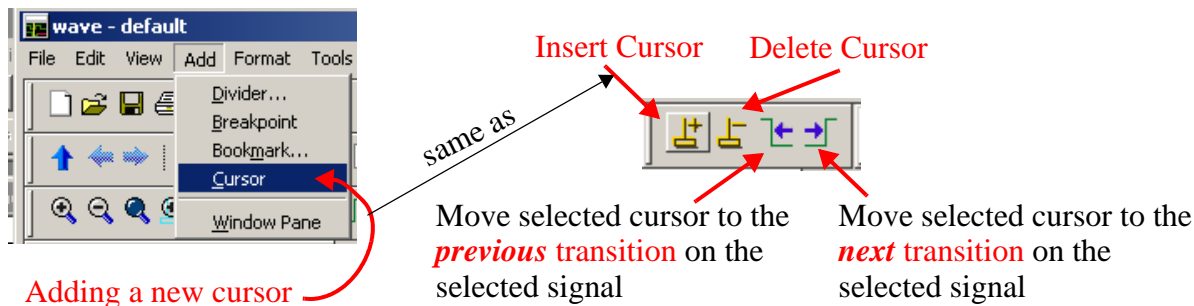
3.5 Your TA can help you with ModelSim. He/she can show you how to add signals from deeper in the schematic hierarchy. After adding new signals we need to restart and resimulate because the newly added signal may not show past activity (the simulation time before the addition of these new signals). The “`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.



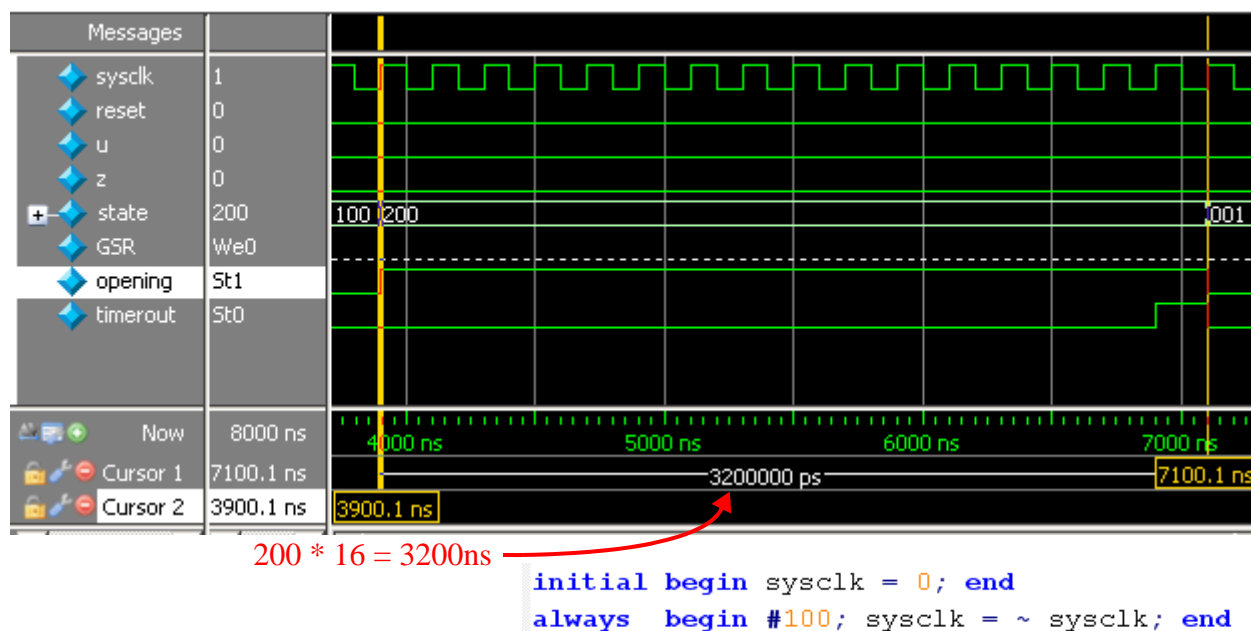
If your waveform is displaying full path of signals, you can change it to display just the signal name (without the complete hierarchical path) by doing the following (Tools => Window Preferences => Display Signal Path = 1).



Some useful cursor controls





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

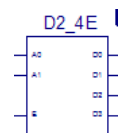



3.6 Now complete the “top” design (ee2011_number_lock_top.sch).

Go through the 3-sheet schematic. Sheets 1 and 3 are fully complete! Sheet 2 is 70% complete. You need to complete:

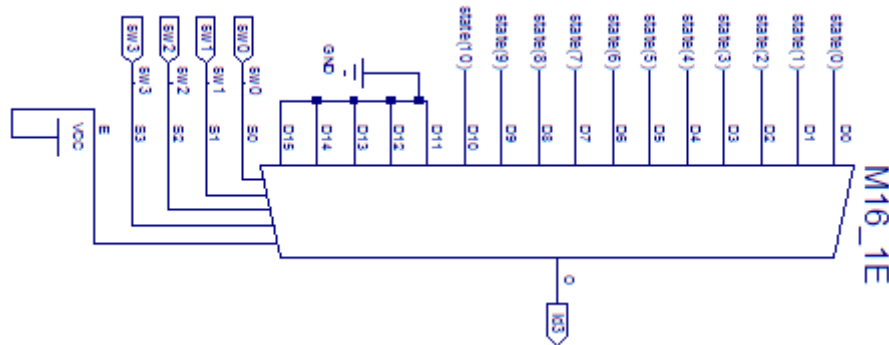
- the logic driving the anodes (using a D2_4E decoder).
- I/O markers for the anodes and naming the connected nets.

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 you schematic using 

3.7 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.



3.8 Synthesize and implement the design using the same procedure steps you followed in the previous labs and **show the working design to your TA.**

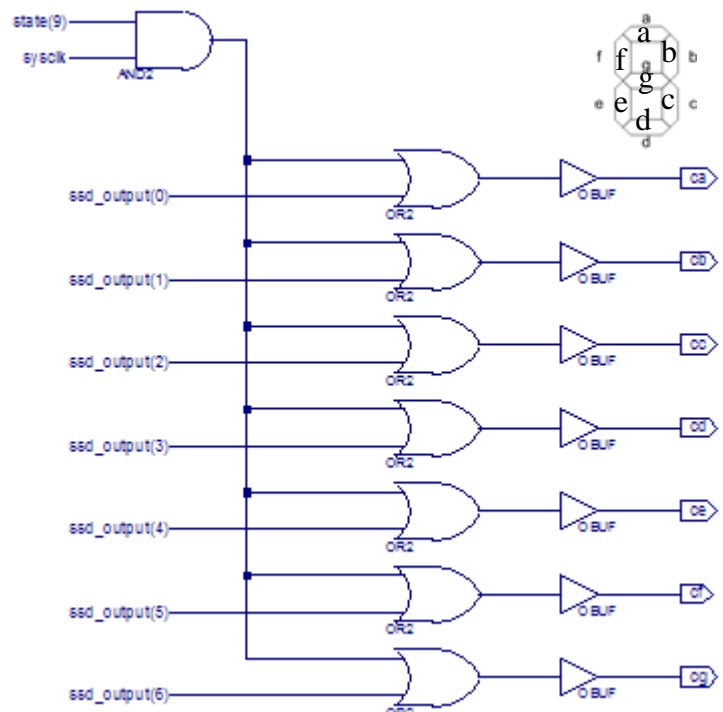
3.9 Flashing of the 7-segment displays during the **OPENING** state is achieved by the following circuit. The cathodes are active-low. So during the **OPENING** state (**state[9]=1**) **sysclk** sends a 1 to the cathodes to turn them off for half-clock of every clock. Analyze and understand the circuit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

3.10 You know that a combinational logic can be implemented using a ROM. And you also know that a bigger ROM can be built using several smaller ROMs.

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	0	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	8412	D004	D860	2812	FFFF

4. Lab Report:

Name: _____	Date: _____
Lab Session: _____	TA's Signature: _____

For TAs: Simulation and Implementation (60): _____ Report (40): _____
Comments:

Q 4. 1: Simulate your design using the three test patterns provided in the Verilog test fixture Attach the simulation waveforms to the report.



Q 4. 2: Attach Sheet 2 of the completed (and implemented) top schematic to the report.
What is the reason for commenting out the **BtnU** and **BtnD** lines in the ucf file (2 pts)
(ee2011_number_lock_top.ucf) ?

```
Net "BtnC" LOC = B8;
# Net "BtnU" LOC = A8;
Net "BtnL" LOC = C4;
# Net "BtnD" LOC = C9;
Net "BtnR" LOC = D9;
```

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

Q 4. 3: Fill-up the frequencies of the clock divider (6 pts)

Q 4. 4: Flashing effect created in the OPENING state: Refer to section 3.9 above. Suppose we replace the seven 2-input OR gates with seven 2-input NAND gates (Note NAND here, unlike AND in section 3.9). What other modifications you need to make to your circuit in order to make it work? (6 pts)

Q 4. 5: Refer to section 3.10: What would you do to change the display of nine to  from  (6 pts)

Q 4. 6: 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. (20 pts)

