Introduction to Nexys 4 board - Detour Signal Lab

1. Synopsis:

This lab introduces the use of Field Programmable Gate Arrays (FPGA). This lab introduces the Digilent Nexys 4 board and demonstrates FPGA design flow through the design of a simple state machine that controls a detour signal.
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 a 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 and the rightmost switch (SW0).

2. Description of the Circuit:

You all know the detour signal at road repair sites. The design in this lab is a simplified version of those light boards. It has four groups of lights -- GL (Group Left), G1, G2 and GR (Group Right) -- controlled in sequence to indicate *detour to the right* or *detour to the left* (Figure 1).

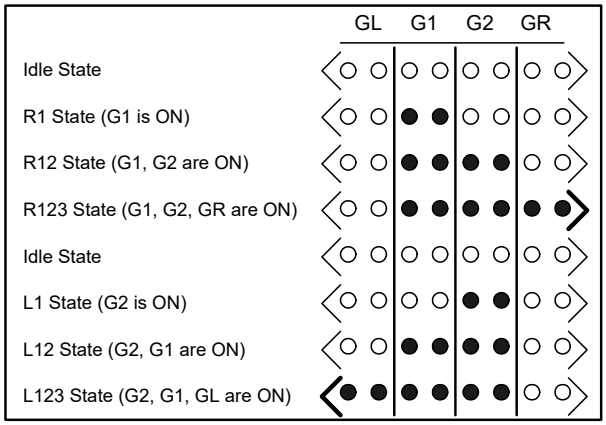


Fig 1: Four groups of LEDs in different states

We provide a switch (**L/$\overline{\mathbf{R}}$**) to choose the "detour-left" or "detour-right" action. The state machine (Figure 2) only looks at the switch during the Idle state. It then proceeds through a series of states (detour left or detour right) and then returns to the Idle state.
We will implement the 7-state control unit using D-flip-flops using one-hot state assignment to the 7 states. The schematic file `ee201l_detour.sch` (page 11) provides an incomplete implementation of the state machine.

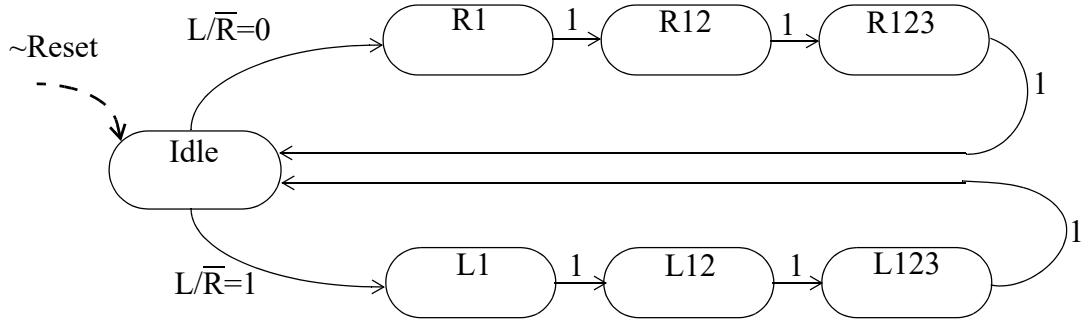


Fig 2: State diagram for the Detour Signal design

3. Introduction to the Nexys 4 FPGA Board:

Please review Nexys 4 details from your first week of lab/lecture material dir pdf pdf .avi .

4. FPGA Design Flow:

Please refer to your first week's lab on Nexys 4: Xilinx_project_synthesis_on_Vivado .pdf .mp4

5. Prelab: (Refer to the first paragraph of nexys4_rm.pdf for the first two questions)

Q 5. 1: Circle the family and device category of the FPGA used on Nexys4 board? (1pt)

7 Series		UltraScale		UltraScale+	
Spartan-7	Artix-7	Kintex UltraScale	Virtex UltraScale	Kintex UltraScale+	Virtex UltraScale+
Kintex-7	Virtex-7				

Q 5. 2: Google "CSG324 @Xilin.com" and find what is CSG324 in the part number of the device used on Nexys-4? (1pt)

❑ Speed grade ❑ Package ❑ Serial number ❑Model number

Q 5. 3: The **state memory** is usually implemented using: (2pts)

❑ And-Or gates
❑ RAM
❑ flip-flops
❑ latches

Q 5. 4: We will implement the detour signal controller using One-Hot state assignment method. How many D-flip-flops are needed to implement the controller? (2pts)

On power-on reset (i.e when power is first applied to the system), how many Flip-Flops are preset? _____ (1pt) and how many are cleared? _____ (1pt)

In the schematic `ee254l_detour.sch` (on page 11), next state logic for which state(s) is complete? (2pts)

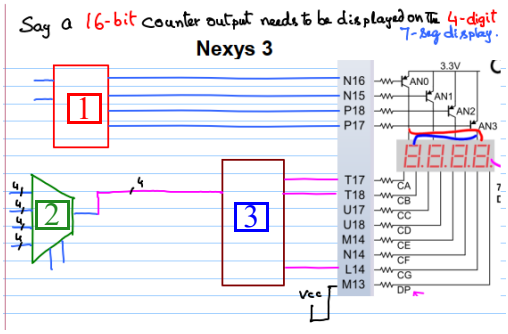
_____

Next State Logic (NSL) is purely a _____ (*combinational/sequential*) circuit. (2pts)

Output Function Logic (OSL) is purely a _____ (*combinational/sequential*) circuit. (2pts)

_____

Q 5. 5: Refer to the figure on the right, reproduced from this pdf, suggesting a way to display the output of a 16-bit binary number as a 4-digit hexadecimal number going from 0000 to FFFF on the 4 SSDs of a Nexys-3 board.
Name the three components and also state whether it is a combinational logic or a sequential logic.



1. _____
2. _____
3. _____

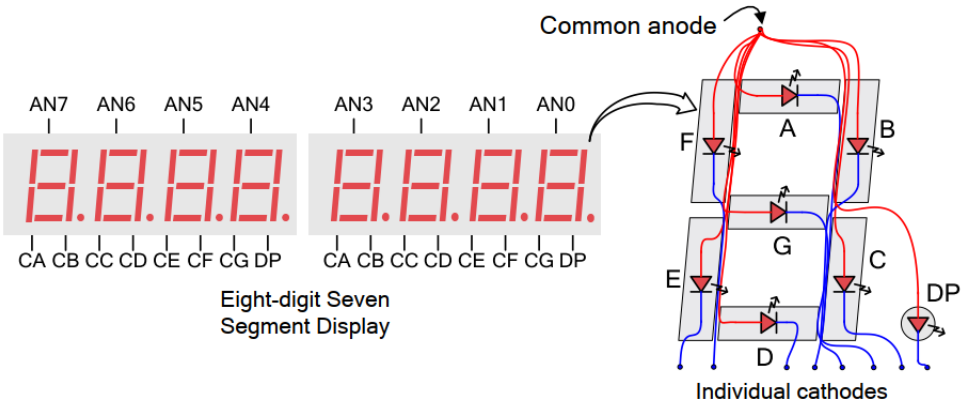
Now, if we want to display the output of a 32-bit binary counter as an 8-digit hexadecimal number going from 0000 0000 to FFFF FFFF on the 8 SSDs of a Nexys-4 board, what three components you intend to use?

1. _____
2. _____
3. _____

Do you believe that one of the above two displays is brighter. _____ Yes / No.
If you said "Yes", then which is brighter? ____ (A/B).
A= 16-bit counter display on the 4 digits of Nexys-3, B= 32-bit counter display on the 8 digits of Nexys-4
Why? _____

Q 5. 6: In the above diagram, VCC (= 3.3 volts = a constant logic 1) was tied inside the FPGA to the DP (via pin M13) to make _____ (one dot point/all dot points) continuously _____ (to be on/to be off).

Q 5. 7: Though we do not perform fractional arithmetic needing to display a dot point (such as the dot point in 23.42 or 1.234) in this course, it is possible to do so as the so called SSDs (Seven Segment Displays) contain actually 8 segments per digit including the dot point.



In a revised detour lab, suppose that you are given a Nexys-3 board with all singular LEDs removed. You are asked to use the four dot points on the 4 SSDs to represent the 4 groups of lights, GL, G1, G2, and GR. Show (on the side) **the needed additions** to control pin M13 in the figure in the above Q 5.5 to achieve this.

6. Procedure:

6.1 At the end of this handout, we have provided you 4 pages of incomplete schematic, which you will complete by hand on paper.
To help you, the ***incomplete/incorrect*** portions of the schematic are identified by dashed circles or rectangles .

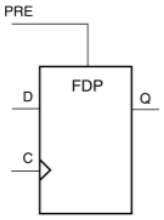
We divide almost all our designs to be implemented on the FPGA board into two major parts: the **CORE** design and the **TOP** design. The core design solves the particular problem/task (usually involving a DPU (data path unit) and a CU (control unit)). We generally simulate and debug the core design separately by using a core test bench. Once you are satisfied with your core design you can then incorporate your core by *instantiating* that in the top design. The top-design focuses on the logic needed in the FPGA to interface the board's I/O resources (push-buttons, switches, LEDs, and SSDs) to the core design. The scanning control needed to control the SSDs is the main part of this effort. You can then implement the design using synthesis tools (Vivado in our case) and program the bit file to the board and verify the operation of the system you built.
NOTE: **We do not usually write testbenches for the top-design!**

Part 1: Completing the core design (revised procedure from Spring 2020 onwards)

6.2 Read the `detour` schematic page 11 at the end of this lab handout. This is an incomplete implementation of the state machine. **Complete the design** by adding additional state memory (flip-flops), next state logic, and output function logic. Notice the two different types of flip-flops used -- FDP and FDC. Determine the difference between them. Please note that this schematic lab was originally designed for Nexys-3 which has Spartan 6 FPGA. So, let us continue to refer to the Spartan 6 library of components for completing this schematic by hand. Please refer to Spartan-6 Libraries Guide for Schematic Designs UG616. Your completed paper design must implement the state machine in Figure 2 and then produce the four output signals (**OFL**): **`GL`**, **`G1`**, **`G2`** and **`GR`**.

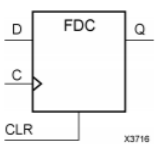
FDP

Primitive: D Flip-Flop with Asynchronous Preset



FDC

Primitive: D Flip-Flop with Asynchronous Clear



Part 2: Completing a simple "top" design

In part 2 our goal is to design a simple "top" schematic that wraps your Part 1 state machine and interfaces its inputs and outputs to the FPGA board. You should not change your core state machine in this Part. You will place all the necessary logic (called "glue logic") in the "top" file.

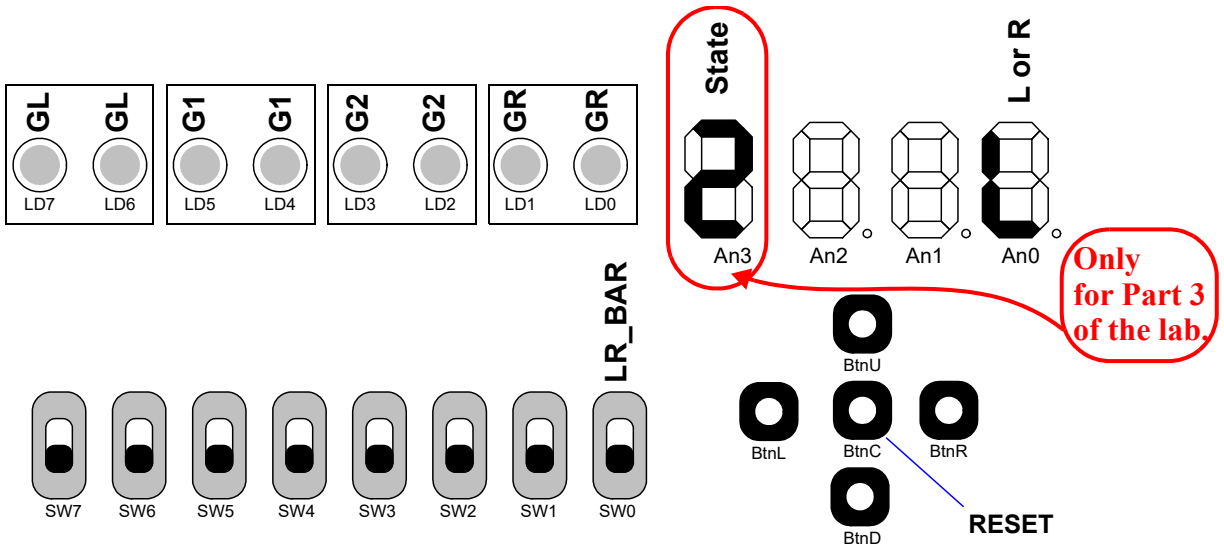


Fig 3: Diagram of the I/O resources on the Nexys 3 board for the simple "top" design (Parts 2 &3)

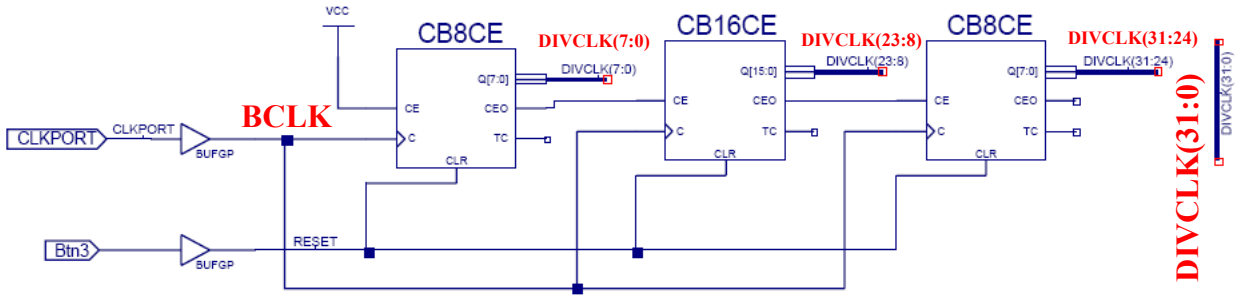
Figure 3 shows the I/O resources you will use for this lab. For the simple top we will use one of the switches (SW0) to exercise the left/right detour selection (i.e., to produce the **LR_BAR** signal) and the eight LEDs (LD7 through LD0, in 4 pairs) to indicate detour arrows growing left or growing right. Also one of the seven segment LEDs will show "**R**" or "**L**" depending on the position of the **LR_BAR** switch. Since "**R**" can not be displayed on a SSD we show "**A**" ( )which is close to "**R**" in shape. Push button **`BtnC`** generates **`RESET`**.

6.3 Since you are doing paper design now you cannot simulate your paper design using a testbench and verify its functionality. But we do so in a later lab in Verilog.

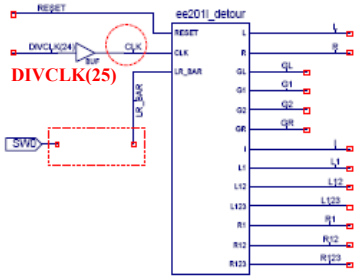
6.4 You are given an incomplete "top" design. Your TA will discuss its structure. You must understand how the provided (incomplete) design needs to be completed because you will come across a top design through out the entire semester.

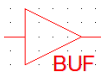
6.5 Details of the top design

6.5.1 Clock division: The board clock is at 100MHz. We produce slower clocks **`DIVCLK(31:0)`** with a counter cascade. **`DIVCLK(0)`** is 50MHz (half of 100MHz), **`DIVCLK(1)`** is 25MHz, and so on. We will use **`DIVCLK(25)`** to clock our detour signal Control Unit.

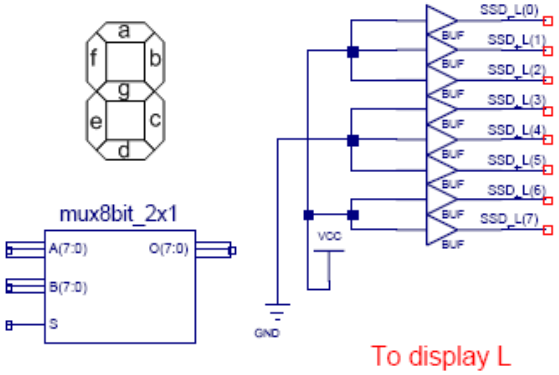


6.5.2Core instantiation in the top file: We (the TAs) created a symbol for the **core** design as shown on the right. The core is instantiated in the **top** as shown in the figure to right. Note that even though we used identical signal names in the top design (e.g. **`L`** and **`R`**) it is not necessary. For example, we could use signal names such as **`LEFT`** and **`RIGHT`** in the top and connect to the **`L`** and **`R`** pins of the core design instance as shown on the left side.

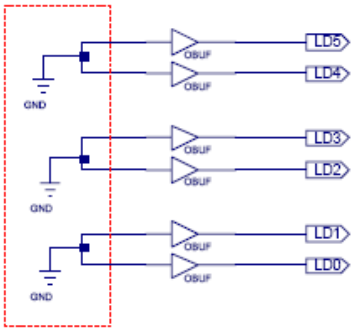


6.6 On sheet 1 (page 12) of the top schematic `ee201l_detour_top.sch`, complete the connection for **`LR_BAR`** (**`SW0`**). Note: If you use an actual schematic tool, you can not connect these two with a wire as you can not have a wire with two names (**`LR_BAR`** and **`SW0`**). So use a dummy buffer (a **`BUF`** component ()). If a signal (i.e. ) is driven by multiple sources in this lab you probably made a labeling error. Designs, needing multiple sources to drive a line, require more advanced techniques such as open-collector output gates or tri-state buffers. and will be covered later in the course. Leave the items associated with the Part 3 of this assignment *as is* for the time.

6.7 On sheet 2 (page 13) of `ee201l_detour_top.sch`, we have provided the signals you need to drive the cathodes to display **L** () . You must arrive at a similar layout to generate **R** (). It looks like an "A" but for this lab, let us pretend that it is a "R"!.
Then you need a mux controlled by the **`LR_BAR`** to steer **L** (**`SSD_L(7:0)`**) or **R** (**`SSD_R(7:0)`**) to **`SSD_Out(7:0)`**. A custom-made 8-bit wide 2x1 mux is provided on the right for you to use for this purpose.



6.8 Complete the connections to the LEDs.



G1 (**LD5** & **LD4**), **G2** (**LD3** & **LD2**) and **GR** (**LD1** & **LD0**). Notice the special buffers connected to the I/O markers for **SW0** (**IBUF = Input Buffer**) and **LD7**/**LD6** (**OBUF = Output buffer**). Be sure to use appropriate buffers for each signal. **CLKPORT** (on sheet 1) is connected using a third kind of special buffer (**BUFGP** = Global Buffer Primary). This buffer is only needed for global signals like the clock which go on clock tree.

In the top schematic (`ee201l_detour_top.sch`) we have already added some glue logic between the I/O markers and your core design symbol. Each of these input/output devices (switches, buttons & LEDs) is connected to a unique pin of the FPGA. By associating each I/O marker in the top design with a pin number we specify that the I/O signals in our design will be tied to correct FPGA pins. For instance, pin **V10** of the Nexys 3 FPGA is connected to the on-board clock generator. So, in order to connect the on-board FPGA clock generator to the **CLKPORT** input in our top design, we must associate the I/O marker labeled **CLKPORT** to pin **V10**. This is done in the User Constraint File (or UCF file in short), using the line:
`NET ClkPort LOC = V10;` for Nexys 3 in the old ISE14.7 synthesis tool. For Nexys 4, the clock pin number is E3. And this information is specified in a file called .xdc file (XDC = Xilinx Design Constraint) in the new (and current) Vivado synthesis tool.
The following line is from the `test_nexys4_verilog.xdc`. We do not need to learn the grammar of writing .xdc file. The .xdc grammar is similar to tcl (tcl = tool control language).

```
# Clock signal
#Bank = 35, Pin name = IO_L12P_T1_MRCC_35,      Sch name = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports ClkPort]
      set_property IOSTANDARD LVCMOS33 [get_ports ClkPort]
      create_clock -add -name ClkPort -period 10.00 [get_ports ClkPort]
```

6.9 You will work with the .xdc files in future projects. Basically you will comment/uncomment lines from the master `.xdc` file (`Nexys4_Master.xdc`).

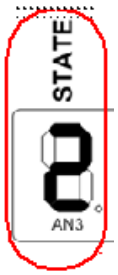
6.10 Dummy paragraph to maintain paragraph numbers with the old Nexys 3 based lab involving schematic entry.

6.11 Dummy paragraph

6.12 Connect the Nexys 4 board to your PC using the provided USB cable. Your TA will demonstrate how to program the Nexys 4 board using either the standalone Digilent's ADEPT tool or the Hardware manager in the Vivado tool. Download the TA provided `ee201l_detour_top.bit` to your Nexys 4 board and understand the operation of the design.

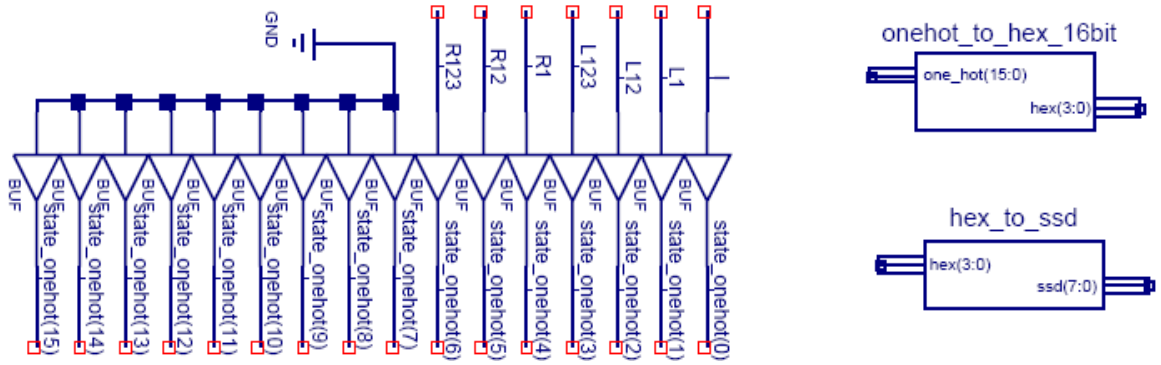
6.13 Verify the TA's design by operating **SW0** and confirming the correct sequence of LEDs and the correct direction indicator on **SSD0** -- **L** (8) or **R** (8). **Show your paper design to your TA and relate your paper design to the behavior displayed on the board**.

Part 3: Implementing a more complex top



In this part, you will modify the top to also display the current state number on the fourth SSD (**SSD3**) (left-most SSD in Nexys-3) (in addition to "**L**" or "**R**" on the right-most SSD **SSD0**). You will need to employ the output **scanning mechanism** to accomplish this.
Your TA will explain output scanning (if it is not covered in your lecture). Once you understand how output scanning works you can proceed with the following procedure steps.

6.14 In ee201l_detour_top.sch (sheet 1) (page 12) you are provided with a bus named **state_onehot[15:0]**. state_onehot(15:0)



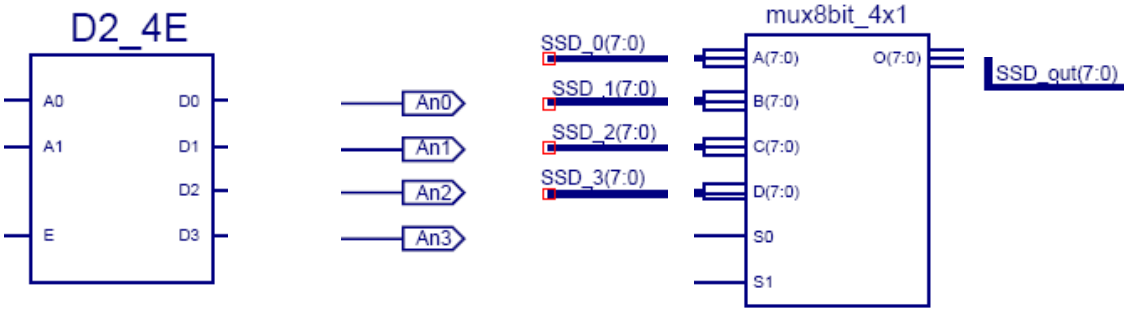
This 16-bit one-hot coded vector has to be converted to a 4-bit hexadecimal number, which can then be displayed on the SSD (after converting the 4-bit hex to 7-segment **Ca-Cg**). The necessary (custom) components are provided with this project.

6.14.1 The goal is to show the state number ('**0**' for **Idle** state, '**1**' for **L1**, '**2**' for **L12**, and so on) on the fourth SSD (**SSD3**) (controlled by **AN3**) and the direction indicator ('**L**' or '**R**') on the right most SSD (controlled by **AN0**) and nothing on the middle two SSDs (so blank them off).

6.14.2 There are two ways to achieve the above.

6.14.2.1 **Method #1**: Permanently disable **AN1** and **AN2**. Then activate **AN0** and **AN3** alternately at a reasonable speed *(neither too fast nor too slow,* your TA will help*)* while sending the correct 7-segment (actually 8-segment) signals (**Ca-Cg**) to the SSDs. You need a 8-bit wide 2-to-1 mux for this. You might use `DIVCLK(15)` to alternately enable one of **SSD0** or **SSD3** to accomplish this.

6.14.2.2 **Method #2**: Instead of permanently disabling **AN1** and **AN2** (as in Method#1), let us send **Ca-Cg,Dp=11111111** to the 8 cathodes (remember these are active low so this will blank the displays). Now we need to activate the 4 anodes, one at a time in sequence, while sending the corresponding 8-segment information to the cathodes with a 8-bit wide 4-to-1 mux.
You can use perhaps `DIVCLK(15:14)` for this purpose.



You can use a 2x4 decoder such as the above **`D2_4E`** . Look up the Xilinx library UG616 for the D2_4E decoder to check if the enable input is active high or low and the outputs are active high or active low. We also provided you with the 8-bit wide 4x1 mux shown above. The diagrams on pages 18/29 to 20/29 (section 9.1 on Seven Segment Display) of the Nexys 4 reference manual will help you in understanding the scanning operation.

6.15 Modify the top schematic following Method #1 above. S**how your paper design to your TA**.

6.16 Print a copy of the design from 6.15 above (and also hold a copy of the file) so you can submit that. Then modify the schematic according to Method #2.
Again show your paper design to your TA.

7. Lab Report:

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

For TAs: Pre-lab (10): _____ Schematic completion (70): _____ Report (out of 20): ___
Comments:

Q 7. 1: Refer to Xilinx UG616.What are the two different D-flip-flops that are used in this lab? What is the difference between them? (2pts)

Q 7. 2: Name 2 more D-flip-flops that are available in the Xilinx library UG616 (and have enable pins) and briefly explain how are they different from the ones that are used in this lab.(8pts)

Q 7. 3: Which pin of the FPGA is connected to the **`BtnC`** on the Nexys 4 board? Refer to the reference manual for the Nexys 4 or the .xdc file of Nexys 4 (see files in the dir) . (4pts)

Q 7. 4: Which three special buffers are needed for signals that connect to the input and output devices (including the on-board clock generator) ? Give their names and whether they are needed to connect inputs or outputs or both. (6pts)

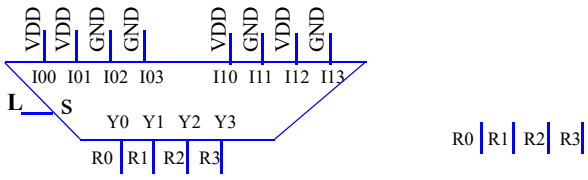
_____

_____

_____

Q 7. 5: The frequency of the clock entering the FPGA is 100MHz. Notice that we are dividing the input clock by using counters. Calculate the frequency of the divided clock (`DIVCLK[25]`) that triggers the detour signal state machine. (5pts)

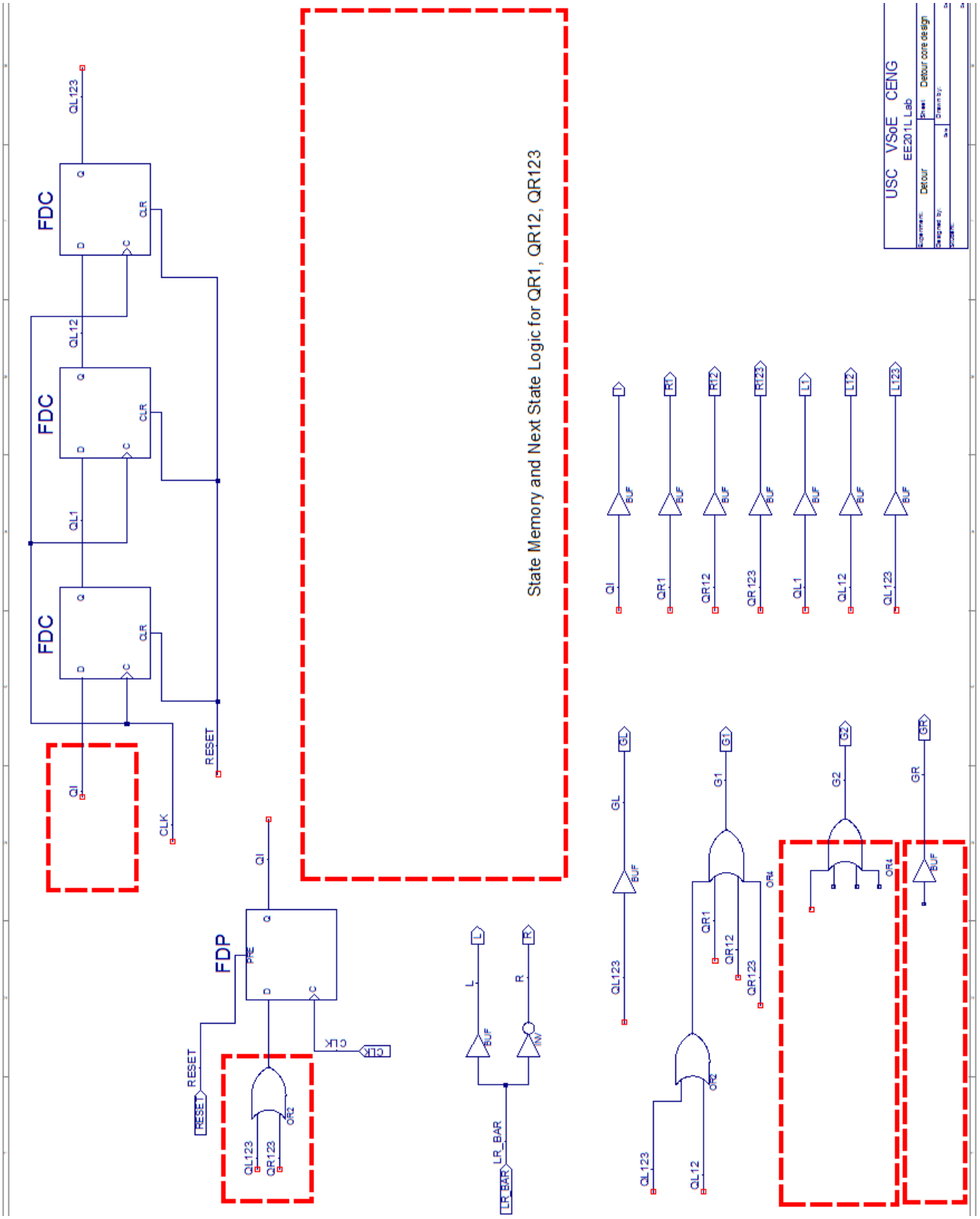
Q 7. 6: Here, L (for Left) and R (for Right) are always opposite to each other. Can you optimize (simplify) the 4-bit output R0 R1 R2 R3 generating combinational logic on the side? Try to avoid the expensive 4-bit wide 2-to-1 mux by tying either **VDD** or **GND** or **L** or **R** as appropriate to each of the four outputs.Note: Since **R** = ~**L**, instead of generating **L_Bar** by inverting **L**, you can use **R** wherever you need **L_Bar**.

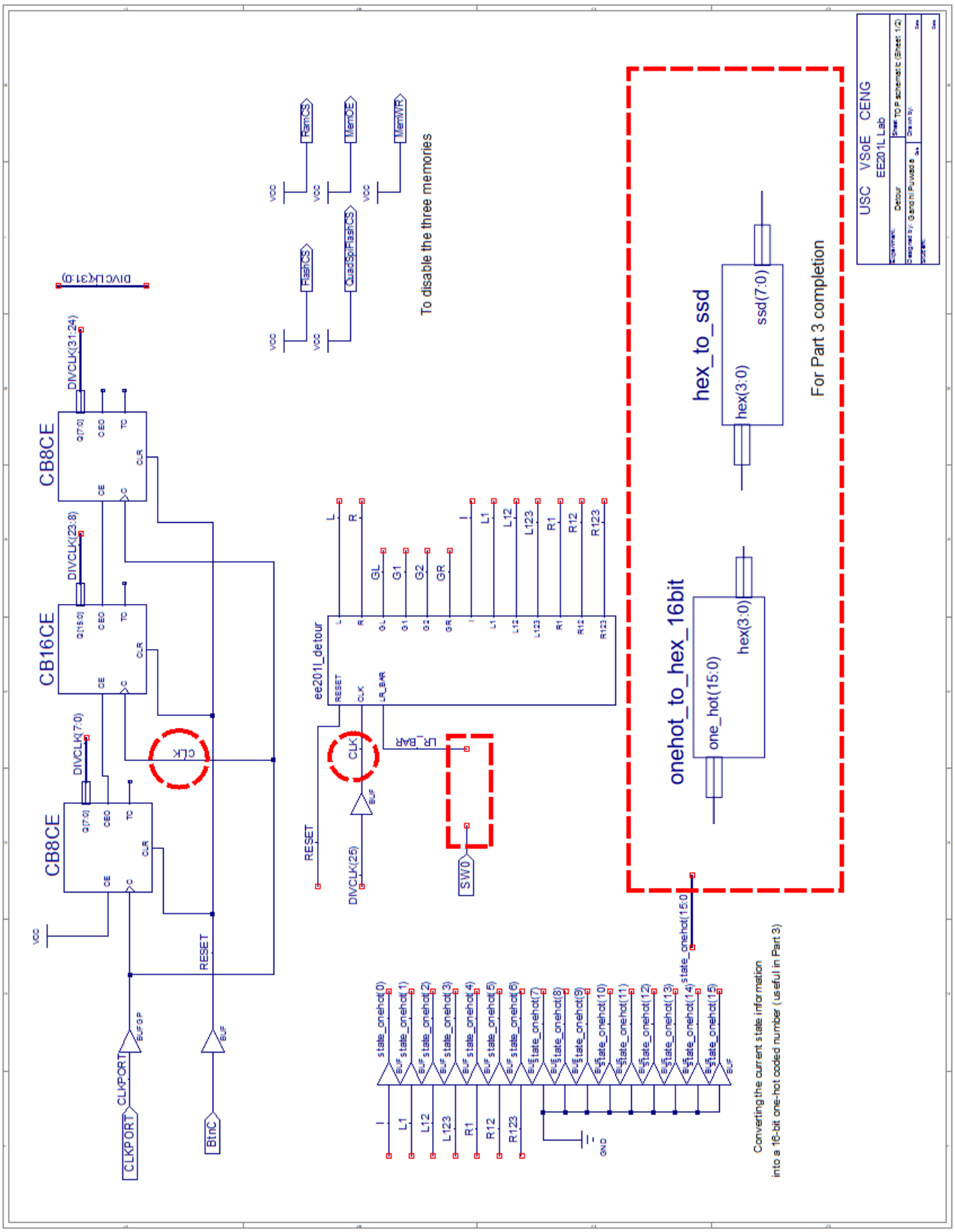


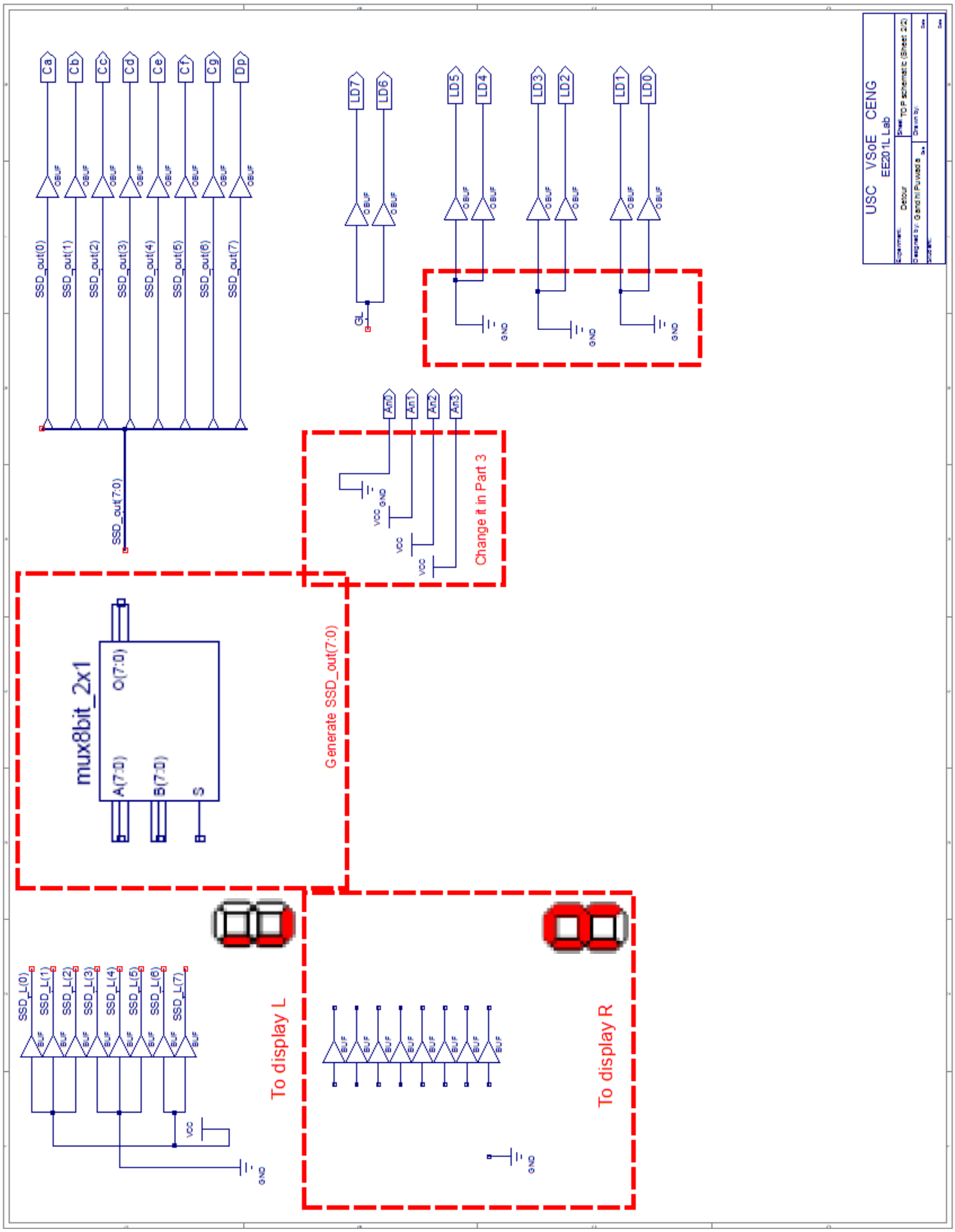
Based on the above, you suggest that you could have avoided _____

_____

in the second page of the top schematic.

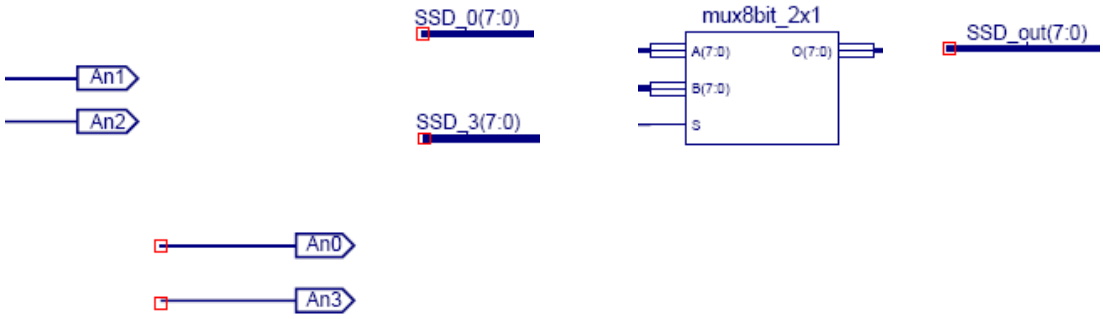

State Memory and Next State Logic for QR1, QR12, QR123
USC VSoE CENG
EE201L Lab
Detour
Detour core design
FDC
FDP
QL123
QL12
QL1
QI
RESET
CLK
QR123
QR12
QR1
QL
QR
G1
G2
GL
GR
L
R
LR_BAR
BUF
INV
OR2
OR4


USC VSoE CENG
EE201L Lab
To disable the three memories
FlashCS
QuadSpiFlashCS
RamCS
MemOE
MemWR
CB8CE
CB16CE
CB8CE
DIVCLK(31:24)
DIVCLK(23:8)
DIVCLK(7:0)
DIVCLK(31:0)
CLK
RESET
CLKPORT
BtnC
ee201l_detour
LR_BAR
SW0
DIVCLK(25)
hex_to_ssd
hex(3:0)
ssd(7:0)
onehot_to_hex_16bit
one_hot(15:0)
hex(3:0)
For Part 3 completion
state_onehot(15:0)
Converting the current state information
into a 16-bit one-hot coded number (useful in Part 3)

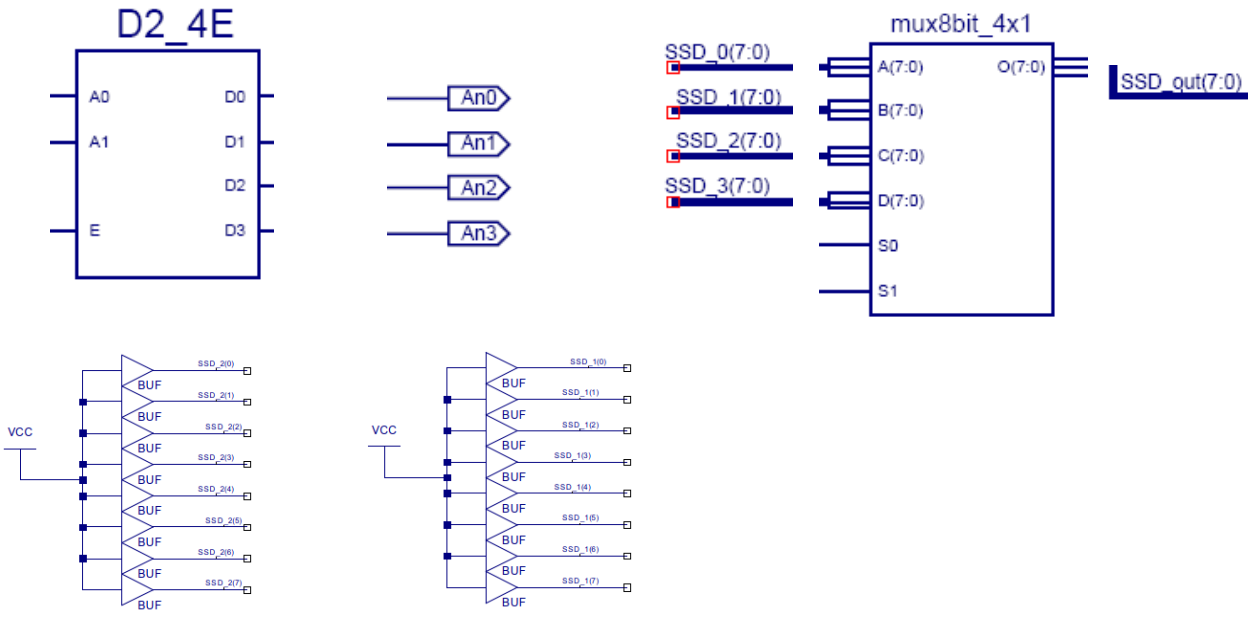

USC VSoE CENG
EE201L Lab
Ca
Cb
Cc
Cd
Ce
Cf
Cg
Dp
LD7
LD6
LD5
LD4
LD3
LD2
LD1
LD0
SSD_out(0)
SSD_out(1)
SSD_out(2)
SSD_out(3)
SSD_out(4)
SSD_out(5)
SSD_out(6)
SSD_out(7)
SSD_out(7:0)
GL
GND
VCC
An0
An1
An2
An3
Change it in Part 3
mux8bit_2x1
O(7:0)
A(7:0)
B(7:0)
S
Generate SSD_out(7:0)
SSD_L(0)
SSD_L(1)
SSD_L(2)
SSD_L(3)
SSD_L(4)
SSD_L(5)
SSD_L(6)
SSD_L(7)
To display L
To display R

Part 3 On this page generate only Anode and Cathode controls.

Method #1: Permanently disable **AN1** and **AN2**. Then activate **AN0** and **AN3** alternately at a reasonable speed *(neither too fast nor too slow,* your TA will help*)* while sending the correct 7-segment (actually 8-segment) signals (**Ca-Cg**) to the SSDs. You need a 8-bit wide 2-to-1 mux for this. You might use `DIVCLK(15)` to alternately enable one of **SSD0** or **SSD3** to accomplish this.



Method #2: Instead of permanently disabling **AN1** and **AN2** (as in Method#1), let us send **Ca-Cg,Dp=11111111** to the 8 cathodes (remember these are active low so this will blank the displays). Now we need to activate the 4 anodes, one at a time in sequence, while sending the corresponding 8-segment information to the cathodes with a 8-bit wide 4-to-1 mux.
You can use perhaps `DIVCLK(15:14)` for this purpose.



You can use a 2x4 decoder such as the above **D2_4E** . Look up the Xilinx library UG616 for the D2_4E decoder to check if the enable input is active high or low and the outputs are active high or active low. We also provided you with the 8-bit wide 4x1 mux shown above. The diagrams on pages 18/29 to 20/29 (section 9.1 on Seven Segment Display) of the Nexys 4 reference manual will help you in understanding the scanning operation. **Celebrate the completion of your lab!**