

Introduction to Nexys 3 board - Detour Signal Lab

1. Synopsis:

This lab introduces the use of Field Programmable Gate Arrays (FPGA). This lab introduces the Digilent Nexys 3 board and demonstrate FPGA design flow through the design of a simple state machine that controls detour signal lights. The Nexys 3 FPGA test board integrates a **xilinx spartan-6 FPGA** with several input and output devices like push buttons, light-emitting diodes and seven segment displays. This lab introduces how each of those input and output devices will be used in EE 254L designs.

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). We provide a switch (**L/R**) to choose the detour left or detour right action. The complete 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 Idle state.

	GL	G1	G2	GR
Idle State	<○ ○>	○ ○	○ ○	○ ○
R1 State (G1 is ON)	<○ ○>	● ●	○ ○	○ ○
R12 State (G1, G2 are ON)	<○ ○>	● ●	● ●	○ ○
R123 State (G1, G2, GR are ON)	<○ ○>	● ●	● ●	● ●
Idle State	<○ ○>	○ ○	○ ○	○ ○
L1 State (G2 is ON)	<○ ○>	○ ○	● ●	○ ○
L12 State (G2, G1 are ON)	<○ ○>	● ●	● ●	○ ○
L123 State (G2, G1, GL are ON)	● ●	● ●	● ●	○ ○

Fig 1: Four groups of LEDs in different states

We will implement the 7-state controller using D-flip-flops using one-hot state assignment to the states. The schematic file `ee254l_detour.sch` (page 13) provides an incomplete implementation of the state machine.

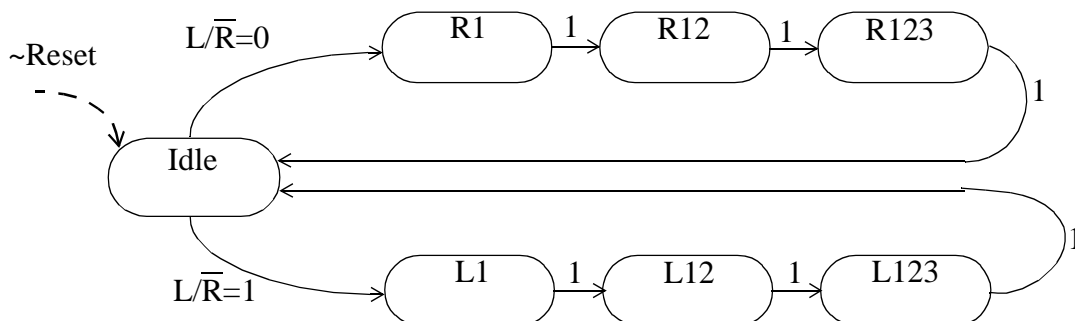


Fig 2: State diagram for the Detour Signal design

3. Introduction to the Nexys 3 FPGA Board:

Most of the information about the board and is taken from Nexys 3 Reference Manual http://digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf.

An FPGA consists of an array of logic blocks connected via programmable interconnect. A typical FPGA contains anywhere from 64 to tens of thousands of logic blocks and an even greater number of flip-flops. Each configurable logic block (CLB) has one or more D-flip-flops and combinational elements such as muxes, etc. Each CLB is capable of performing a semi-complex function such as implementing a full adder. Our objective is to understand how to use FPGAs and we assume that your TA will describe some of the important aspects of the FPGA architecture

XilinxTM Inc. (www.xilinx.com) is a major FPGA vendor. We will be using the **xc6slx16** FPGA from Xilinx's Spartan-6 family (<http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm>).

Spartan-6 FPGA Feature Summary For LX16 (used in EE201L and LX45 used in EE560)

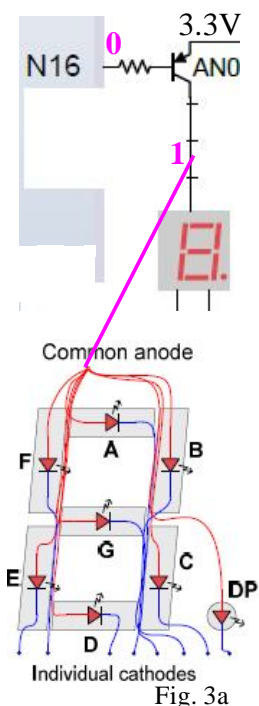
Device	Logic Cells ⁽¹⁾	Configurable Logic Blocks (CLBs)			DSP48A1 Slices ⁽³⁾	Block RAM Blocks		CMTs ⁽⁵⁾	Memory Controller Blocks (Max) ⁽⁶⁾	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices ⁽²⁾	Flip-Flops	Max Distributed RAM (Kb)		18 Kb ⁽⁴⁾	Max (Kb)						
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358

It has approximately 14,579 logic cells organized in 2,278 CLBs. Our **xc6slx16** device (Nexys 3 board) uses an CSG324C package.

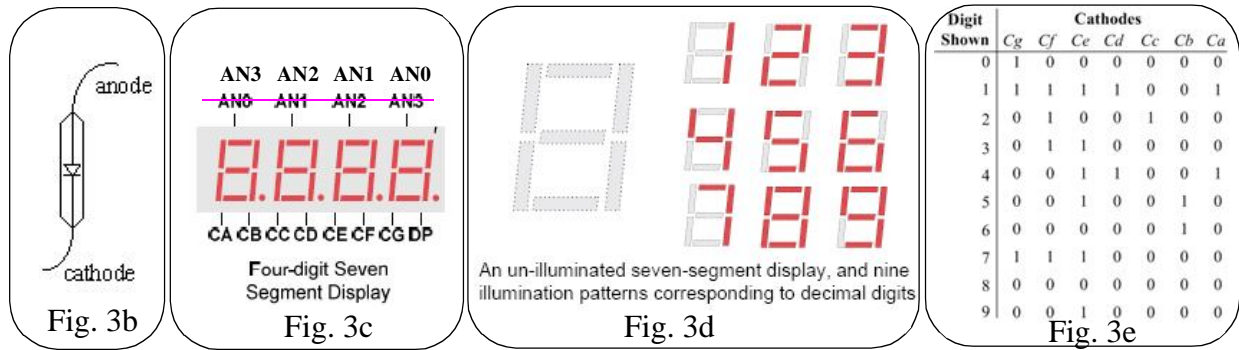
3.1 Seven Segment Display:

A seven segment display (SSD) is just seven LEDs arranged in a grid fashion. Our SSDs also have an eighth LED for the decimal point. The LEDs are labelled **a**, **b**, **c**, **d**, **e**, **f** and **g**. The decimal point is labelled as **dp**. The top LED is labeled **a** and we order the remaining labels clock-wise (Figure 3a). To make a segment glow requires sending a logical **0** to the cathode and a logical **1** to the anode. To reduce the wire-count for the SSD we connect all of the 8 anodes together to form a single anode control. We can turn on individual segments by controlling the cathode terminals (this is called the common anode configuration). In the common anode configuration each SSD requires only 9 pins instead of 16.

Notice that to present a **1** on the common anode we must actually present a **0** to the base of the pnp transistor (Figure 3a). So our **anode enables are active low!** To illuminate a single digit the anode enable (e.g. **AN0 pin**



AN16) is set to 0 and then the specific cathodes for that digit are driven to 0. Shown below (Figure 3b-3e) are the seven segment codes for the ten decimal digits.



3.2 Output Scanning Mechanism For SSDs

The Nexys 3 board has four SSDs and if they were individually connected to the FPGA, they would require 36 pins (9x4) of the FPGA. To lower the number of FPGA pins required to interface the four SSDs, the designers of the 4-digit SSD have connected the respective cathodes of the four SSDs together. For example, the **a** cathode of the four SSDs are connected together to form a single output called **Ca** (Cathode **a**). Thus there are 8 cathodes, 7 labeled as **Ca-Cg** and one labeled as **DP** for the decimal point. Therefore we should illuminate only one of the SSDs at any given time. To meet this constraint we select each SSD sequentially (setting **AN0=0**, then **AN1=0**, and so on) and synchronize the data sent on **Ca-Cg** pins (i.e., sending zeros for segments we want illuminated). Our designs do this “*fast enough*” to create the illusion (to the human eyes) that the SSDs are lit constantly. We call this the “output scanning mechanism for the SSDs”.

We use a 2-bit counter (00, 01, 10, 11) and a 2x4 decoder to generate the 4 SSD enables (00 => **AN0** active, 01 => **AN1** active, etc.). We use the same counter bits as select lines to a 4x1 mux to “steer” the proper **Ca-Cg** to the enabled SSD. For example, when the counter bits are 00 we have **AN0=0** (the others equal 1) and select **Ca-Cg** for the first SSD.

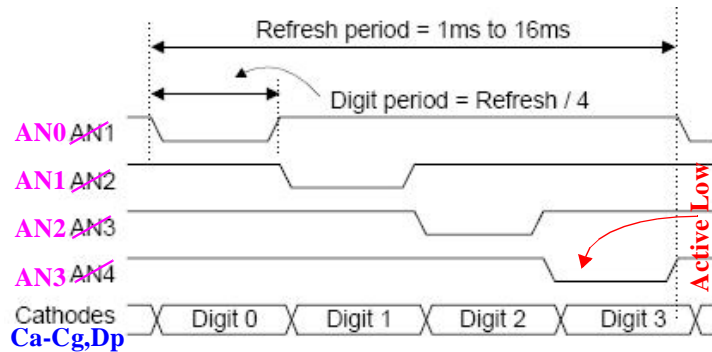


Fig. 4 Seven-segment display timing diagram

Note: We must sync (match) the order that we select the mux inputs and the corresponding anode control. If we overlook this then the digits will not appear in the correct order on the four SSDs. Figure 4 shows a timing waveform illustrating the output scanning mechanism.

You should also watch the following short Flash animation to help you understand how SSDs are connected on the Nexys 3 board:
http://www-classes.usc.edu/engr/ee-s/254/sev-en_segment_display.swf

4. FPGA Design Flow:

In this section we will discuss the work flow to implement a Xilinx Schematic design to the FPGA on your Nexys 3 board. Figure 5 summarizes the design flow that we will follow for this class.

4.1 Design Entry:

Some acronyms:

ISE = Integrated Software Environment

HDL = Hardware description Languages (Example: Verilog, VHDL).

XST = Xilinx Synthesis Tool

We will use Xilinx Schematic Editor to prepare the circuit schematic. In addition to the schematic for the core state machine we also need to create a “wrapper” or “top” design which interfaces signals from the core design to the input/output devices like the switches, push buttons and LEDs, etc. The Xilinx component library provides the following special components which we will need:

Input/Output Buffers (IBUF, OBUF): Xilinx FPGAs require buffers for each I/O marker. This ensures that signals entering or leaving the FPGA are of appropriate strength. The Xilinx component library provides two special buffers for this: input buffers (symbol: **IBUF**) and output buffers (symbol: **OBUF**). In schematic entry these IBUFs and OBUFs are used but if the design is a HDL-based design, then these are not required to be instantiated. The synthesis tool XST automatically infers these.

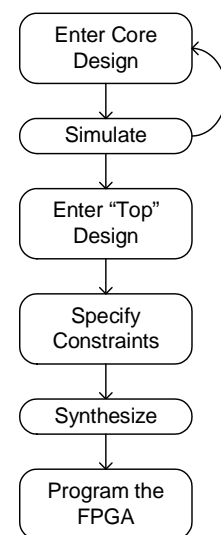
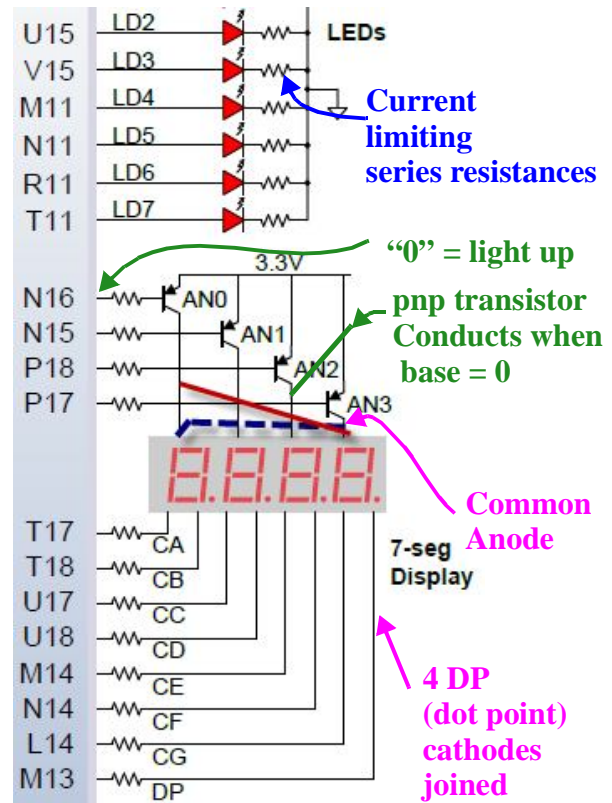
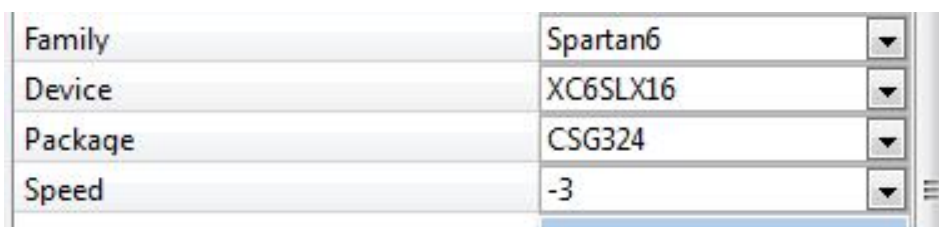


Fig 5: Design flow

Global Buffers (BUFGP): Xilinx FPGAs also have special buffers called “Global Primary Buffers” (symbol: **BUFGP**). We must use these buffer for certain incoming signals like **clock** which have “high fan-out”. For example, after the input marker for the clock pin you must place a global buffer (**BUFGP**). Only then can you send the clock to your circuit elements like flips-flops, counters, etc. You cannot use general purpose buffers (symbol: **BUF**) or input buffers (**IBUF**) for the clock signal!

4.2 Setting up the type of FPGA in XST (Xilinx Synthesis Tool):

We need to tell ISE what type of FPGA we are targeting the design for by setting the following properties of the FPGA in the Design Properties dialog box (which can be invoked by right clicking any of the items in the Hierarchy pane (top-left pane in the ISE window)) as follows:



4.3 Synthesis & Implementation:

Once you have specified the circuit (in a schematic or Verilog file) the Xilinx Synthesis Tool (XST) determines how to connect the CLBs (configurable logic blocks) and programmable interconnects to realize your design. The process of interpreting the design, simplifying the circuit, and creating an the implementation for a specified FPGA type is called *synthesis and implementation*. We will launch the XST from within the Project Navigator to synthesize your design and generate a configuration bit-stream for your FPGA.

4.4 Programming:



We program (or configure) the FPGA by sending the configuration bit-stream (generated by XST) to the FPGA through a USB connection. We use Digilent’s ADEPT configuration/programming software and connect the FPGA board to the computer with an USB provided in the Nexys 3 box.

5. Prelab:

- Q 5. 1: What does “FPGA” stand for? (1pt)
- Q 5. 2: What is the name of the FPGA *family* that we are using in this lab? (1pt)
- Q 5. 3: Which FPGA of this family are we using? (1pt)
- Q 5. 4: What does CSG324 denote in the design properties dialog box? (1pt)
- ☐ Speed grade ☐ Package ☐ Serial number ☐ Model number
- Q 5. 5: The **state memory** is usually implemented using: (2pts)
- ☐ And-Or gates
☐ RAM
☐ flip-flops
☐ All of the above
- Q 5. 6: Next State Logic (NSL) is a purely _____
(*combinational/sequential*) circuit. (2pts)
- Q 5. 7: 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)
- Q 5. 8: In the schematic `ee254l_detour.sch` (on page 16), next state logic for which state(s) is complete? (2pts)
- Q 5. 9: Name the 3 components you need to construct the output scanning mechanism. Be sure to include bus-widths and input-output counts. (3pts)

6. Procedure:

6.1 Download the zip file `ee254l_detour_Nexys3_EXER.zip` containing the Xilinx ISE project from the Blackboard. Extract the zipped project folder `ee254l_detour_Nexys3_EXER` into the projects folder (`C:\Xilinx_projects\`). Open the project in Xilinx ISE. We provide incomplete schematic files which you will complete, simulate, synthesize/implement and finally download to your FPGA board.

To help you, the *incomplete/incorrect* portions of the schematic are identified by dashed circles  or rectangles  .


We divide the design into two major parts: the **CORE** design and the **TOP** design (refer to section 4.1). 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 use the symbol wizard and create a symbol for our core design. You can then incorporate your core by instantiating that in the top design. The top-design focuses on interfacing the FPGA's I/O resources (push-buttons, switches, LEDs, and SSDs) to the core design. You can then implement the design and download the bit file to the board and verify the operation of the system we built. **NOTE: We do not usually write test benches for the top-design!**

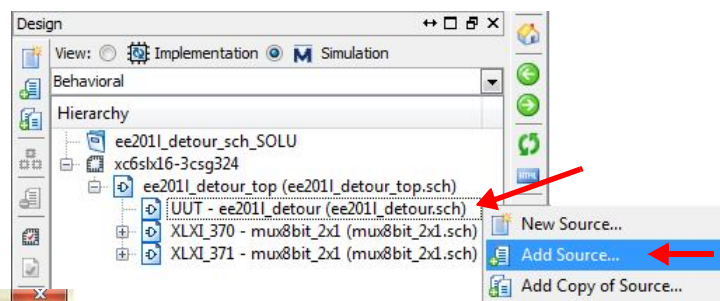
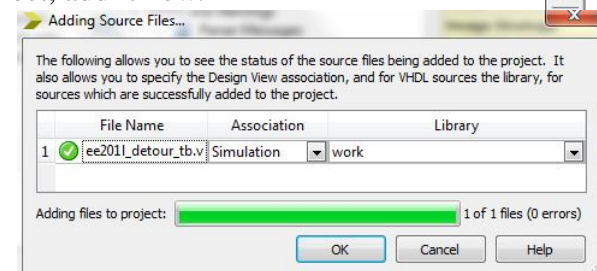
Part 1: Completing the core design




6.2 Open the `ee254l_detour.sch` schematic file. 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 using the "Symbol Info" button in the schematic editor's "Symbols" tab. Your completed design must implement the state machine in Figure 2 and then produce the four output signals (**OFL**): **GL**, **G1**, **G2** and **GR**.

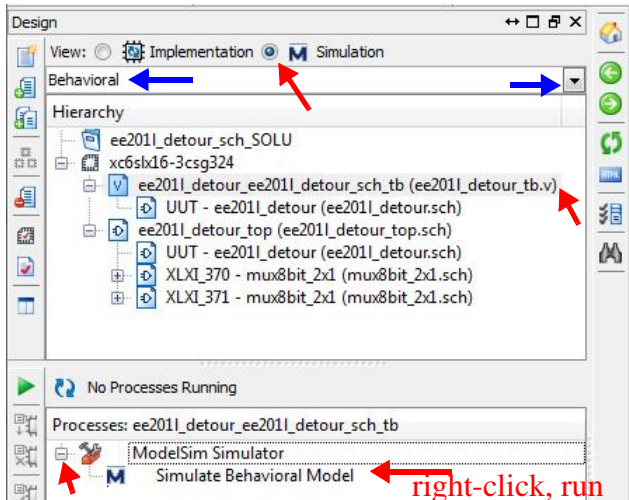
6.3 Use the provided Verilog test fixture (`ee254l_detour_tb.v`) to test your design.

6.3.1 Open `ee254l_detour_tb.v` in

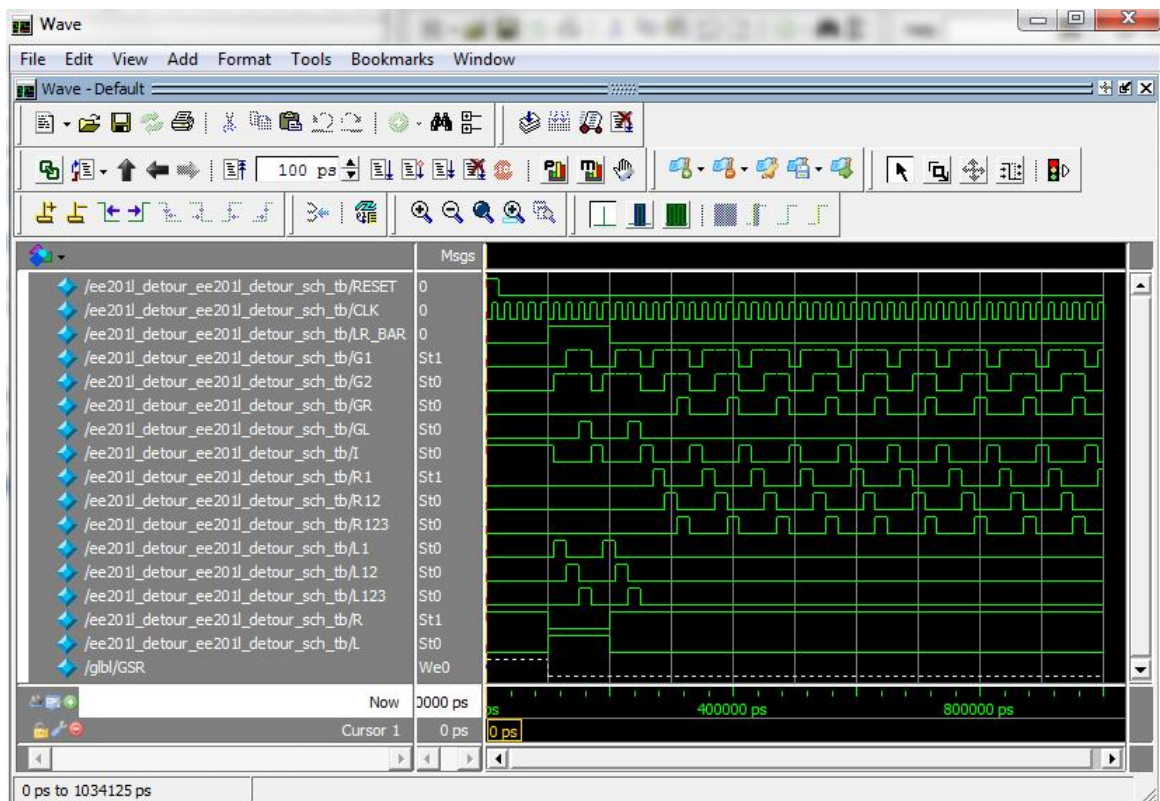
 Notepad++ and inspect the same. Notice that the test bench instantiates the unit under test (UUT, here it is `ee254l_detour.sch`). If you have not added this testbench file to your project, add it now.




6.3.2 Start simulation. As shown on the side, you need to select  **Simulation** in the top panel (Design panel), then select **Behavioral** using the pull-down options, then select  **ee201l_detour_ee201l_detour_sch_tb (ee201l_detour_tb.v)**. In the Processes panel below, right-click on  **Simulate Behavioral Model**



6.3.3 The ModelSim 10.1 kicks up, simulates and displays waveform.



Note: If you do not see the signals in the waveform window, you probably made a mistake in the schematic. It is a good-practice to click the “Check Schematic” button  before simulating. Also, make sure that your input/output (wires with I/O markers) signal names match those used in the test bench. Primary inputs for the design are: **CLK**, **RESET**, and **LR_BAR**. The primary outputs are: **GL**, **G1**, **G2**, **GR** and the one-hot coded states **I**, **L1**, **L12**, **L123**, **R1**, **R12** and **R123**. **Verify and show the waveform of the completed and functionally correct design to your TA.**

Part 2: Completing a simple “top” design

In part 2 our goal is to implement your detour design onto the FPGA board. You will 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.

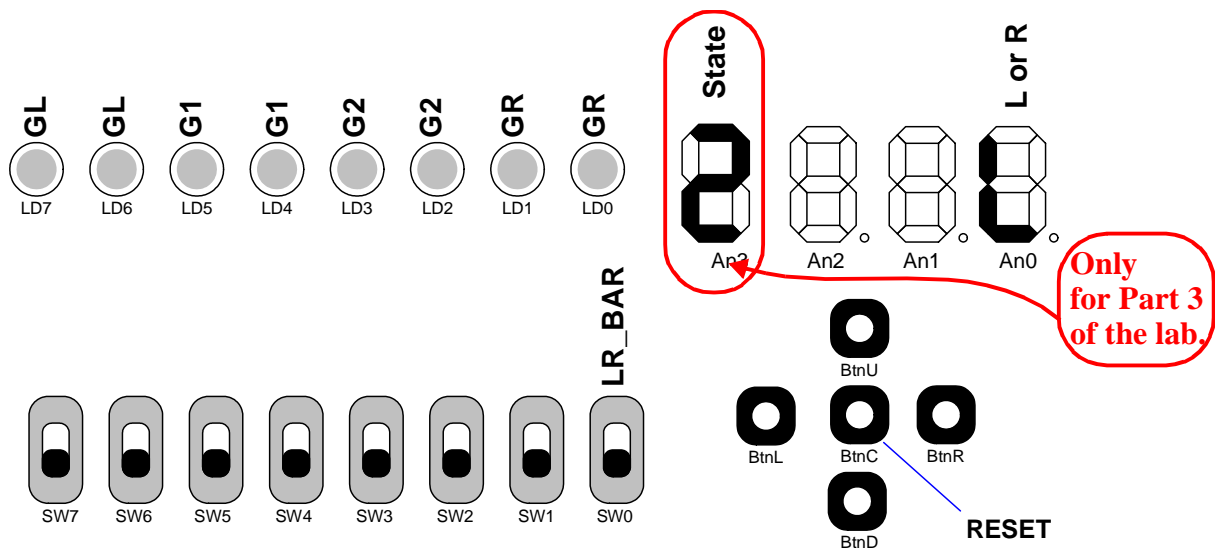


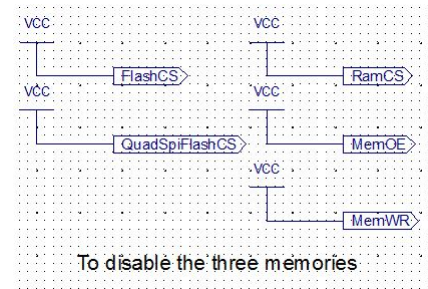
Fig 6: Diagram of the I/O resources on the Nexys 3 board for the simple “top” design (Parts 2 &3)

Figure 6 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 pairs of two) to indicate detour arrows growing left or growing right. Also one of the seven segment LEDs will show “R” or “L” depending upon the position of the **LR_BAR** switch. Since “R” can not be displayed on a SSD we instead show “A” (A) which is close to “R” in shape. Push button **BtnC** generates **RESET**.

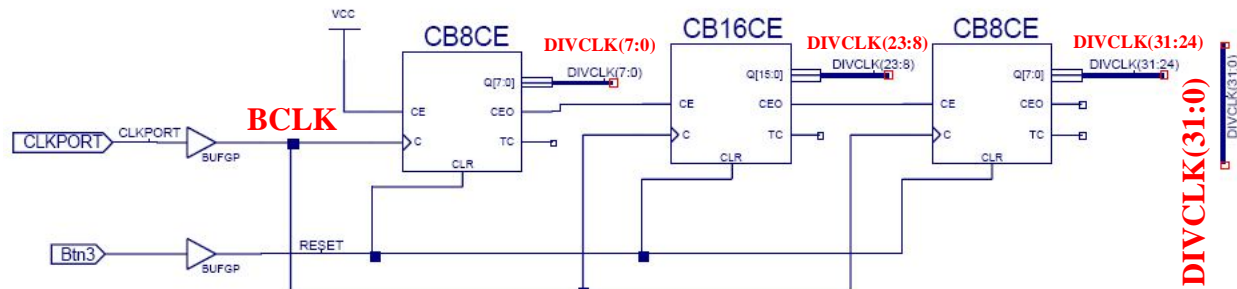
6.4 Switch the Sources window view from *Simulation* to *Implementation*. You should see that the top-most schematic file in the hierarchy is `ee254l_detour_top.sch`. This is an incomplete “top” design. Your TA will discuss its structure. You must understand how the provided (incomplete) design works because you will see it through the entire term.

6.4.1 There are three memories on the Nexys 3 board which we will not use in most of our labs. So we need to disable them so that they do not interfere with the rest of our design. The hook-up shown on the side achieves this.

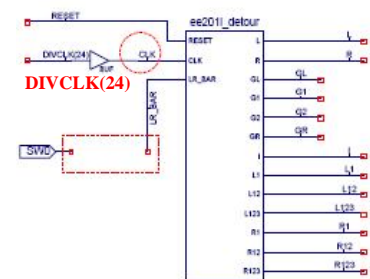
We must include this hook-up in all our TOP designs.





6.4.2 Clock division: The board clock is at 100MHz. We produce the slower clocks **DIVCLK(31:0)** with a counter cascade (below). **DIVCLK(0)** is 50MHz (half of 100MHz), **DIVCLK(1)** is 25MHz, and so on. You will use **DIVCLK(25)** to clock your detour signal CU.





6.4.3 Core instantiation in the top file: We (the TAs) created a symbol using symbol wizard in Xilinx ISE for the **core** design. We already instantiated it 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.

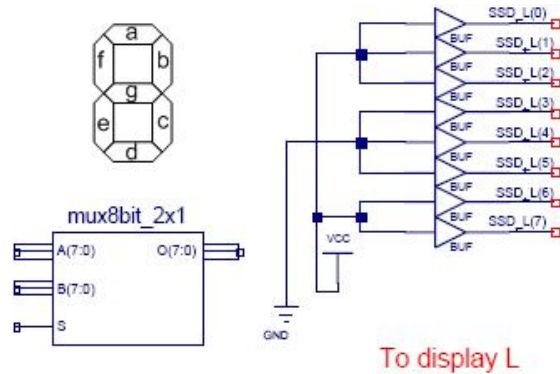


6.5 On sheet 1 of ee254l_detour_top.sch, complete the connection for **LR_BAR (SW0)**. Note: You can not connect these two with a wire as you can not have a wire with two names (**LR_BAR** and

SW0). So use a **BUF** component (). If a signal (i.e. ) is driven by multiple sources in this lab you probably made a labeling error (multiple source designs require more advanced techniques such as open-collector or tri-state). You should use the “Check Schematic” button (see 6.3) to warn you if you placed more than one source driving a signal. Leave the items associated with the Part 3 of this assignment as is for the time.

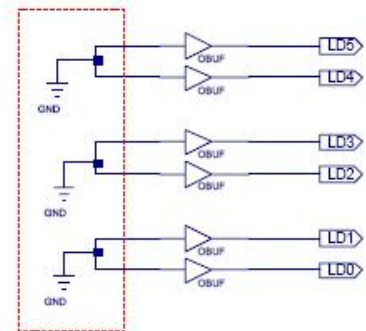
6.6 On sheet 2 of `ee254l_detour_top.sch`, we have provided the signals you need to drive the cathodes to display **L** (). You must arrive at a similar layout to generate **R** ().

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 in the project directory for you to use for this.



6.7 Complete the connections to the LEDs.

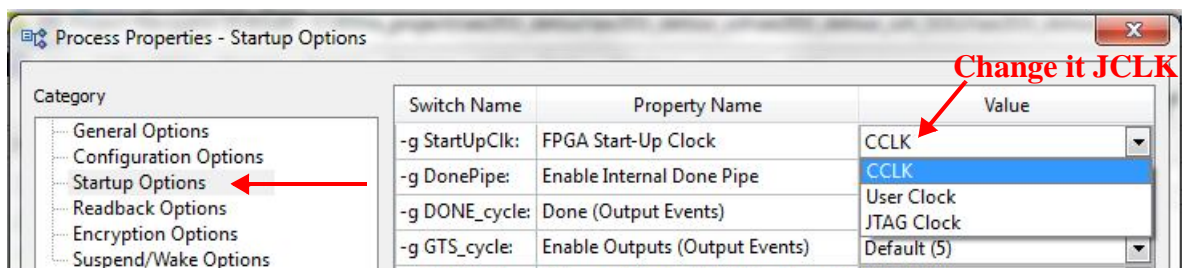
G1 (**LD5 & LD4**), **G2** (**LD3 & LD2**) and **GR** (**LD1 & LD0**). Notice the special buffers connected to the I/O markers for **SW0** (**IBUF**) and **LD7/LD6** (**OBUF**). Look up the details for these buffers (using the “Symbol Info” button). Be sure to use appropriate buffers for each signal. **CLKPORT** (on sheet 1) is connected using a third kind of special buffer (**BUFGP**). This buffer is only needed for global signals like the clock.



In the top schematic (`ee254l_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 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;`



6.8 Go to the project directory (`C:\Modelsim_projects\ee254l_detour_Nexys3`) and find the UCF file (`ee254l_detour.ucf`). Open this file in a text editor (such as NotePad++). Comment out lines associated with unused resources such as **sw1-sw7** and **Btn0-Btn2**. Use a “#” character at the start of a line to comment it from the .ucf file. Example: `#NET sw1 LOC = H18;` Save the file and return to the Xilinx ISE Project Navigator. You are now ready to synthesize the design.

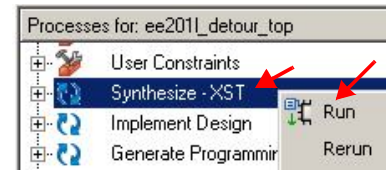
6.9 In the Project Navigator window select the top schematic. In the Processes window, right-click on “Generate Programming File” and select “Properties”. Click the “Startup options” tab and select **JTAG** clock for FPGA Start-Up Clock. Click OK.



6.10 “**Synthesize**” your top design. Then “**Implement**” your design. Then “**Generate Programming File**” for your design.

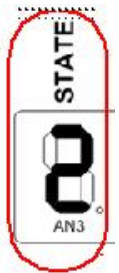
6.11 Connect the Nexys 3 board to your PC using the provided USB cable. Your TA will demonstrate how to program the Nexys 3 board using Digilent’s ADEPT tool. Download `ee254l_detour_top.bit` to your Nexys 3 board

6.12 Verify your design by operating **SW0** and confirming the correct sequence of LEDs and the correct direction indicator on **SSD0** -- **L** () or **R** (). **Show the correct implementation to your TA.**

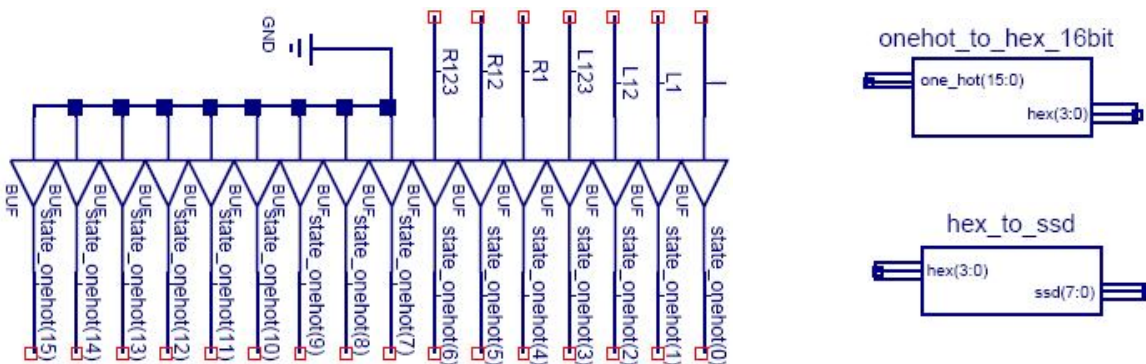


Part 3: Implementing a more complex top

In this part, you will modify the top to also display the current state number on the left-most SSD (**SSD3**) (in addition to “**L**” or “**R**” on the right-most SSD **SSD0**). You will need to employ the output scanning mechanism described in Section 3.2 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.



6.13 In ee254l_detour_top.sch (sheet 1) you are provided with a bus named **state_one-hot[15:0]**. `state_onehot(15:0)`

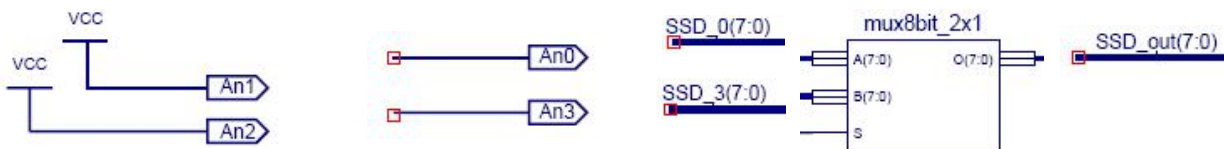


This 16-bit one-hot coded vector has to be converted to a 4-bit hexadecimal number, which can then be displayed on the SSD (after converting the 4-bit hex to 7-segment **Ca-Cg**). The necessary (custom) components are provided with this project. You can use these components or design your own and use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

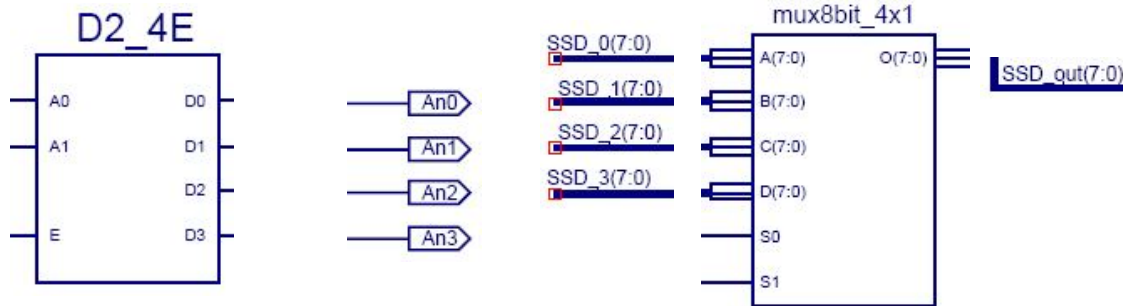
6.13.1 The goal is to show the state number (‘0’ for **Idle** state, ‘1’ for **L1**, ‘2’ for **L12**, and so on) on the left most SSD (controlled by **AN3**) and the direction indicator (‘**L**’ or ‘**R**’) on the right most SSD (controlled by **AN0**) and nothing (blank) on the middle two SSDs.

6.13.2 There are two ways to achieve the above. First review sections 3.1 and 3.2 on SSDs.

6.13.2.1 **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.



6.13.2.2 **Method #2:** Instead of permanently disabling **AN1** and **AN2** (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 use **DIVCLK(15:14)** for this purpose.



You can use a 2x4 decoder such as the **D2_4E** available in Spartan 6 library. We also provided you with the 8-bit wide 4x1 mux shown above. The diagrams in Section 3 from the Nexys 3 reference manual will help you in understand the scanning operation.

6.14 Modify the top schematic following Method #1 above. Synthesize and implement the design and **show the working design to your TA**.

6.15 Copy the design from 6.14 so you can submit that. Then modify the schematic according to Method #2. Synthesize and implement the design and **show the working design to your TA**.

7. Lab Report:

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

For TAs: Pre-lab (15): _____ Implementation (50): _____ Report (out of 35): _____
--

Comments:

- Q 7. 1: What are the two different D-flip-flops that we are using 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 and briefly explain how are they different from the ones that we are using in this lab.(8pts)
- Q 7. 3: Which pin of the FPGA is connected to the **Btnc** on the Nexys 3 board?. (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 that triggers the detour signal state machine. (5pts)
- Q 7. 6: Why are we dividing the clock? (5pts)
- Q 7. 7: Do we need to maintain any relation between the output scanning frequency and the divided system clock used in the core state machine? Does one need to be faster than the other? Can one be a multiple of the other (say 4 times, 27 times)? Note that in real-life designs, the system clock frequency is much higher than the rate at which the detour arrow grows or traffic lights change. Timers are used to control timing of such events. (5 pts)

