Chapter 4

Seven Segment Controller

This lab will build upon the concepts learned in previous lab by designing a reusable seven-segment controller that can display a unique value on each of the four digits of the seven-segment display.

Learning Objectives

After completing this laboratory, you should:

- Understand how to create a reusable component and how to instance the component into a higher level design
- Create a sequential circuit using the on board clock and set its timing constraint
- Build a seven segment-controller that displays a unique value on each digit.

Exercises

Exercise 1 – Seven Segment Display Controller

In the previous laboratory assignment, you created a seven-segment decoder that could display a hexadecimal value on a single digit. In this lab, we are going to create a seven-segment controller that can display a unique value on each digit of the four-digit display. Displaying multiple digits on the display is accomplished by multiplexing the cathode signals and "driving" or displaying one digit at a time.

The example shown in Figure 4.1 demonstrates how the value "0123" is displayed on the display using time multiplexing. In this example, only one digit is being driven on the display at a given time – digit "0" is displayed on digit 3 (the left most digit) during the first time period, digit "1" is displayed on digit 2 during the second time period, digit "2" is displayed on digit 1 during the third time period, and digit "3" is displayed on digit 0 (the right most digit) during the fourth time period. This process of displaying one digit at a time repeats continuously to make it appear that all four digits are being displayed at the same time. More details on how this is done can be found on Page 6 of the Nexys2 reference manual.

Figure 4.1 demonstrates the signals that must be asserted during each time period to produce the desired display. During the first time period, AN3 is asserted (i.e. is low) and the other annode signals are deasserted. The following cathode signals are asserted (driven low) to display the character "0" on digit 3: CA, CB, CC, CD, CE, and CF. During the next digit period, AN2 is asserted (while AN2 is deasserted) and the following cathode signals are asserted to display the character "1": CB and CC. Next, AN1 is asserted and the following cathode signals are asserted to display the character "2": CA, CB, CC, CD, and CE. Finally, AN0 is asserted and the following cathode signals are asserted to display the character "3": CA, CB, CC, CD, and CG. This process repeats continuously to keep all four values actively highlighted on the four-digit display.

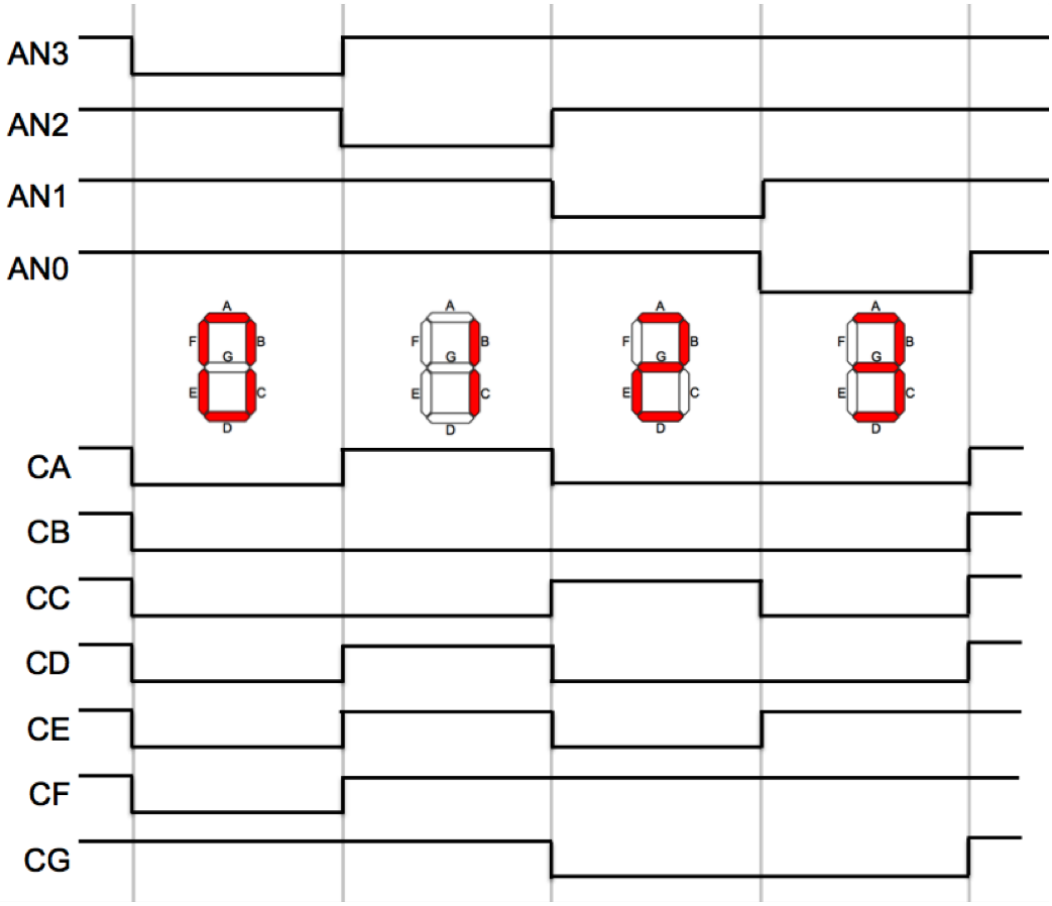

Figure 4.1: Displaying the value “0123” on the Four-Digit Display.

Even though only one digit is actively displayed at a time, our persistence of vision and slow brain response cannot perceive this single digit display strategy. Instead, we view the display as if all four digits are driven continuously. If you were to take a high-speed video of the display, you would see this sequential digit display approach in action.

The challenge we have is deciding how long to drive each anode signal. If the “on” time of the anode signals is too long, the eye and brain will detect that only one digit is being displayed at a time and we will see the digits turn on and then off in sequential order. If the “on” time is just right, human brain will perceive all four characters as being displayed at the same time through persistence of vision.

Question: What do you think the display will look like if the digits are sequenced too fast?

Anode Sequence Counter

An essential component of your 4x7 segment display controller is a sequential binary counter to individually ”turn on” one of the four digits in a regular repeating sequence. The counter will be used to make sure that each digit on the display is turned on for a fixed amount of time and that each digit is turned on in a consistent, repeating order. This counter also determines how much time the a digit is turned on before moving on to the next digit in the display.

Figure 4.2 demonstrates a simple 8-bit binary counter that could be used to sequence the digit anode signals. This binary counter consists of an 8-bit register and next state logic that performs the operation of “+1”. Note that this counter is “free-running” meaning that the counter will continue the binary count sequence without stopping. An easy way to determine which anode signal to drive is by using the top two bits of this counter (bits 7 and 6) to select which anode signal to assert. When the top two bits of the counter are “00”, the AN0 signal (corresponding to digit 0) should be asserted, when the top bits are “01” assert the AN1, and so on. Using the top two bits of the counter to select the anode signal will insure that the anode signal corresponding to each digit is asserted for the same amount of time and in a repeating manner.

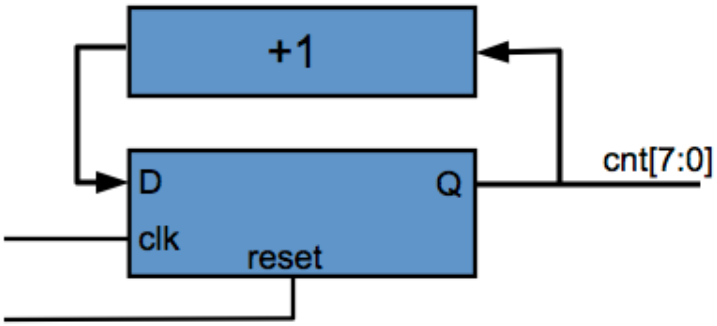

Figure 4.2: Binary Counter for Anode Selection.

With eight bits, this free-running counter will sequence through $2^8 = 256$ different binary numbers before repeating. If the clock that drives this counter is operating at 50 MHz (20 ns clock period) then this counter will repeat the binary sequence every 256 x 20 ns = 5.12 us. The anode corresponding to digit 0 will be asserted for the count values of "00000000" through "00111111", digit 1 will be asserted for the count values of "01000000" through "01111111", and so on. Each digit will be asserted for 1/4th of this time (64 clock cycles) or 1.28 us. This time is far too short for the display to be effective and longer anode assertion times are needed to generate an effective display. Answer the following questions to test your understanding of the sequencing counter.

Question: How long will each digit of the display be driven by the controller if the binary counter used for this sequencing is 15 bits? Assume a 50 MHz input clock.

Question: Assuming a 15 bit counter, how long will each digit be blanked?

4x7 Segment Controller Architecture

To begin your design of the seven-segment display controller, create a new VHDL file and create an empty entity with the name "`seven_segment_control`". Add the ports and generic listed in Table 4.1. It is important to name the ports as listed in the table so you can use your controller within the laboratory testbench. Although the logic in this laboratory will be similar to that of your previous laboratory, this entity and architecture should be created in a new, separate file from your previous laboratory (avoid modifying the VHDL file from your previous laboratory).

The suggested architecture of the 4x7 segment controller is shown below in Figure 4.3. You should create this circuit in a *single* VHDL architecture and use several concurrent statements to implement the proposed logic shown in this figure. Begin the design of your controller by creating the free running binary counter to sequence each of the four seven segment digits. Declare a signal for your counter in which the range is specified by the COUNTER_BITS generic (see the declaration of the signals r_reg and r_next on page 482 of the text to demonstrate how to use the generics in your declaration). Add a sequential process in your architecture that implements the counter (i.e., the counter increments every clock cycle). Because this counter will not use an asynchronous reset, you MUST provide a default value for the counter when you declare the counter signal. This default value will initialize the counter for simulation and synthesis (if you do not have a default value, the counter will initialize to all 'U' values and your counter will not count).

Question: What is the purpose of the COUNTER_BITS generic used in this design? Why would someone want to change the value of this generic?

As described earlier, the top-two bits of this counter are used to indicate which anode should be asserted. These two signals, called "anode_select" in the diagram, will be used in several places of your design. You may want to create a new two-bit signal called "anode_select" and assign the top two bits of your counter to "anode_select" for use in your design.

Port Name	Direction	Width	Purpose
clk	Input	1	Clock for segment counter
data_in	Input	16	Data to display (data_in[3:0] = digit 0, data_in[7:4] = digit 1, data_in[11:8] = digit 2, data_in[15:12] = digit 3)
dp_in	Input	4	Value of four "digit" points (high asserted) (dp_in(0) = digit 0, dp_in(1) = digit 1, dp_in(2) = digit 2, dp_in(3) = digit 3)
blank	Input	4	When asserted, blank individual segment of display (blank(0) = blank digit 0, blank(1) = blank digit1 blank(2) = blank digit 2, blank(3) = blank digit 3)
seg	Output	7	Segment control signals (i.e., seg(0) = CA, seg(6) = CG)
dp	Output	1	Digit point control signal
an	Output	4	Segment anode control signals (i.e., AN0, AN1, AN2, and AN3)
Generic	Type		Purpose
COUNTER_BITS	NATURAL		Indicates the number of bits on the segment counter (DEFAULT=15)

Table 4.1: Interface for the Sevent Segment Controller.

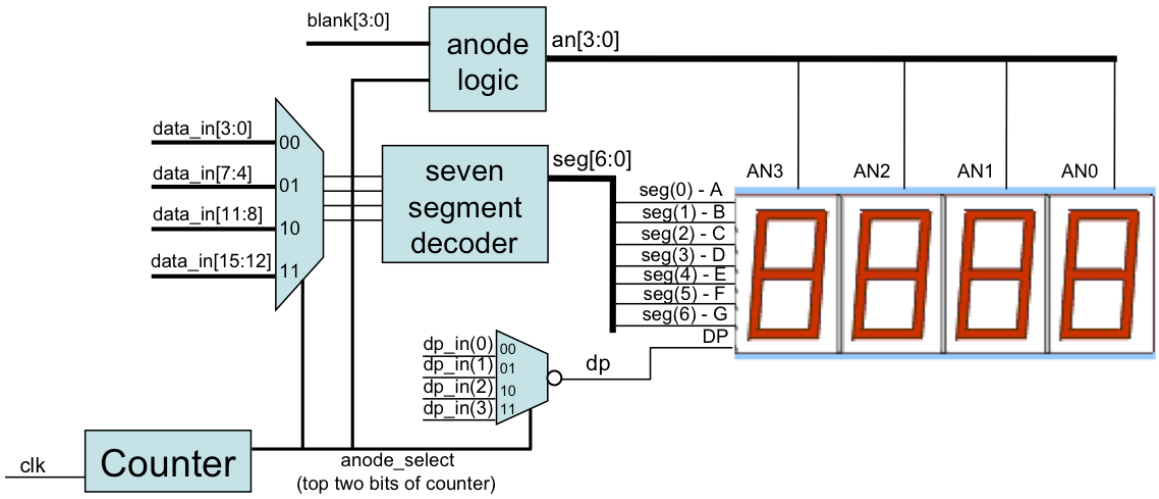

Figure 4.3: Seven Segment Controller Architecture.

This controller contains a four-bit input signal "blank" that allows an external circuit to "blank" or turn off any of the four digits as needed. This is convenient if you want to display only two of the four digits. Each bit of the blanking signal corresponds to one of the digits. If the input `blank(0)` is asserted, then digit 0 should be blanked (i.e., the digit associated with AN0). Blanking a digit is accomplished by preventing the corresponding anode signal from being asserted low during its cycle. Create the logic for the four anode signals from the anode_select and the blank inputs.

The controller architecture shown in Figure 4.3 contains two multiplexers. One multiplexer is used to select the appropriate 4-bit segment input to display. This is a 4-bit, 4 to 1 multiplexer. The second multiplexer is used to select the appropriate digit point to display. The proposed architecture also contains two decoders. The first decoder is used to decode the four-bit value into the seven segment signals. This decoder will be the same logic you created in the previous lab. The second decoder is used to decode the two-bit anode_select signal and generate the four independent anode control signals.

Exercise 2 – Simulation

After creating your component and removing all compilation errors, create a few simple tests with a .tcl simulation script. For this .tcl script, you will need to create an oscillating clock signal to drive the clock of your circuit. ISim supports the ability to create an oscillating signal input using the Isim force command. The following command can be used to create a 50 MHz oscillating clock:

```
isim force add clk 1 -value 0 -time 10 ns -repeat 20 ns
```

Question: What command would you use to create a 100 MHz oscillating clock signal with a 50% duty cycle (i.e., the '0' time and the '1' time of the clock are the same)?

After adding an oscillating signal in your .tcl file, add additional lines to properly test your seven-segment display controller. When testing this seven-segment display controller, you will need to simulate for a large number of clock cycles so you can see the all four digits being properly asserted. Do not proceed with this exercise until you have tested your display controller with a few test cases.

Upload: Upload your .tcl simulation script for your seven-segment display controller.

The seven-segment controller will be used in several labs throughout the semester. It is essential that this controller operate properly as you do not want to debug this in a future lab. After testing your controller with a few sample test cases, proceed to simulate your controller with a predefined testbench. This testbench provides many test cases such as measuring the time that each anode signal is asserted, checks the segment outputs, and the blanking behavior.

Download: tb_seven_segment_control.vhd

After using the testbench to debug your circuit, answer the following questions:

Question: What time does the testbench simulation end?

Upload: Copy and paste your working seven-segment controller VHDL after it passes the testbench.

Exercise 3 – Top-Level Design file

The seven-segment controller you created and tested in the previous two exercises is not intended to be a top-level design that you place on an FPGA and downloaded onto the FPGA board. Instead, it is intended to be a reusable component that you will use in multiple labs throughout the semester. The next exercise of the lab is to create a top-level design (i.e., new VHDL file with entity and architecture) that *instances* your seven-segment display controller. As shown in Figure 4.4, this top-level design is a separate VHDL file that instances your seven-segment display controller and adds additional logic to provide design-specific functionality. In this lab, and in all future labs, you will create top-level VHDL designs that instance components created throughout the course.

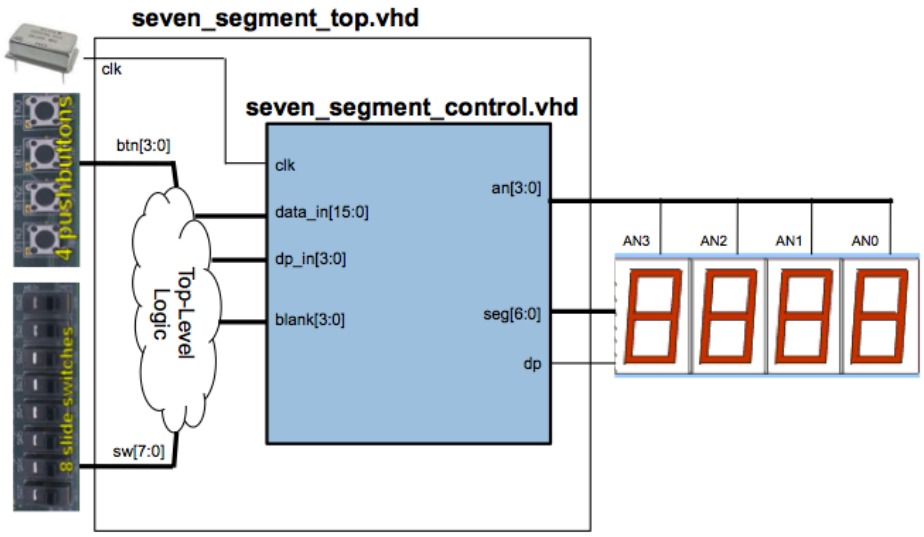

Figure 4.4: Seven Segment Controller Hierarchy.

Create a *new* project with a new top-level VHDL file with the entity named "`seven_segment_top`" and add ports for the clock, switches, buttons, and seven segment display signals as suggested in Table 4.2. Instance your seven segment controller that you created in the previous exercise. As described in the textbook, there are two ways of instancing your seven segment display controller. You can instance it as a *component instantiation* or as a *entity instantiation*.

Port Name	Direction	Width
clk	Input	1
btn	Input	4
sw	Input	8
seg	Output	7
dp	Output	1
an	Output	4

Table 4.2: Interface for the Top Entity of the Sevent Segment Controller.

Once you have instanced your controller, you will need to create additional logic in the top-level design that provides a stimulus to the inputs of your seven segment controller. This logic will determine what is displayed on the four digits of the seven segment controller. Create additional logic that conforms to the following instructions. Note that each statement is presented in priority order (i.e., the first statment has priority over the following statements).

- When BTN3 is pressed, blank all segments and digit points on the display.
- When BTN2 is pressed, display the following characters on the segments "BEEF". No digit points should be asserted.
- When BTN1 is pressed:
  - Display the hexadecimal value of the 8 switches on the right two digits.
  - Blank the left two segments.
  - Assert the digit point associated with AN1 digit (the digit point should be displayed between the two lit digits).
- Create a separate free running 32-bit counter within the top-level module. A "free-running" counter means that the counter should increment every cycle and that there is no enable or logic to halt counting. The value of this counter will be to display under the following conditions (assuming none of the conditions above are met):

- When BTN0 is pressed, display the bottom 16 bits of a free running 32-bit counter. Assert ALL digit points.
- When none of the buttons are pressed, display the top 16 bits of the free running counter. Assert the digit point associated with the left most digit.

You will need to simulate this simple top-level design to make sure it does what you expect. Since you have already simulated the seven-segment display controller in detail, it is only necessary to simulate this new top-level functionality. Make sure you simulate this functionality before proceeding to synthesis.

Upload: Copy and paste your working top-level VHDL file after you are confident it is operating correctly.

Exercise 4 – Synthesis and Implementation

Once you are happy with the simulation of your top-level design, you are ready to synthesize the design. Like all labs, you need to create a .ucf file that defines all of the pins that you use. Like all I/O pins used in your design, you must specify the location of the clock pin. In addition to the location constraint, you must provide a timing constraint to this clock signal in your .ucf file. This timing constraint tells the design tools that the clock signal will operate at 50 MHz. The synthesis and implementation tools will use this information to map your design and attempt to synthesize logic that will not contain any timing violations. We will review the timing of this circuit later in the lab.

```
# Clock
NET "clk" LOC="B8";

# Define a new timing constraint indicating a 50 MHz clock period
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 20 ns HIGH 50 %
```

After synthesizing your circuit, watch the following screencast to learn more about the reporting of sequential circuits. After reviewing the screen cast, review your logs until you are satisfied that there are no errors or warnings that will prevent your circuit from working, download your circuit to the board and make sure it works on the board as you expect it to.

Screencast: Sequential Circuit Synthesis

Question: What is the minimum clock period of your circuit (review the "Post-PAR Static Timing Report" and search for "Minimum period")?

Question: Review the "Map Report" and determine the number of "slices" used by your design.

Personal Exploration

Spend some additional time in this lab by exploring new concepts and ideas for your seven-segment display. Ideas for personal exploration include:

- Review the FPGA editor layout of your design and comment on its properties
- Create a different top-level design that does something different with the display (a stopwatch, button counter, calculator, etc.)
- Experiment with the size of the counter and report on what happens as you increase/decrease the anode on time

Note that you do not need to submit any of the code you developed as part of your personal exploration. Provide a simple summary of the ideas you explored.

Pass Off

- Show your segment display controller simulation running within the testbench.
- Demonstrate to the TA a working circuit on your board.