

Simulation Lab 6

Create an Up-Counter Driving the Seven Segment Display



National
Science
Foundation

Funded in part, by a grant from the
National Science Foundation
DUE 1003736 and 1068182

Acknowledgements

Developed by Craig Kief, Alonzo Vera, Alexandria Haddad, and Quinlan Cao, at the Configurable Space Microsystems Innovations & Applications Center (COSMIAC). Based on original tutorial developed by Bassam Matar, Engineering Faculty at Chandler-Gilbert Community College, Chandler, Arizona. *Funded by the National Science Foundation (NSF).*

Lab Summary

Counters allow hardware to do division. It is possible to slow a clock down to a level that the user can see on a led by using a counter. For example, in this project, one of the higher bits in the counter is driven to the LED. If the 50MHz clock was driven to the LED, without slowing the clock, the LED would flash so quickly that it would appear always on.

Lab Goal

The student will learn how to download and/or develop a simple up-counter and how to send the counted output to the four digit seven-segment LED display on the FPGA board.

Learning Objectives

1. Create a new project and add project files
2. Understand the hierarchical “tree” structure of projects and their test modules.
3. Learn how to read the FPGA’s Reference Manual and the code for this Lab.
4. Understand how the seven-segment display works.

Grading Criteria

Your grade is determined by your instructor.

Time Required

2 hours

Lab Preparation

- Read this document completely first before actually accomplishing the lab.
- Navigate to from the COSMIAC FPGA web page (http://cosmiac.org/Projects_FPGA.html). Download and unzip the **counter.zip** file and extract the Lab 6 source files onto your hard drive to a location you can easily find.

NOTE: Older versions of Xilinx ISE do not handle directory or file names with spaces. It is recommended that you follow the practice of seasoned developers/designers and avoiding using spaces in your directory/file names. Extract the files into a directory named **Labs**, **Xilinxlabs**, or any name of your choosing. Ensure that any parent directories **do not** contain spaces, i.e. “My Documents.”

Equipment and Materials

Students should work in teams of two or three. Each team of students will need the following supplies:

Supplies	Quantity
<ol style="list-style-type: none">1. ISE[®] Design Suite (or WebPACK[™]) software from the Xilinx website, www.xilinx.com, if you don't already have it installed. Your classroom should have a full working version of Xilinx ISE[®] Design Suite.2. FPGA kit including download and power cable(s).3. Free Digilent Adept software (instructions for download and installation are included at the beginning of this lab): http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2	1

Additional References

The FPGA reference manual, data sheet, and any other supporting documents that may be of use.

ISE Design Suite User Guide :

<http://www.xilinx.com/support/index.htm#nav=sd-nav-link-106173&tab=tab-dt>

Digilent Adept User Guide:

<http://www.digilentinc.com/Data/Software/Adept/Adept%20Users%20Manual.pdf>

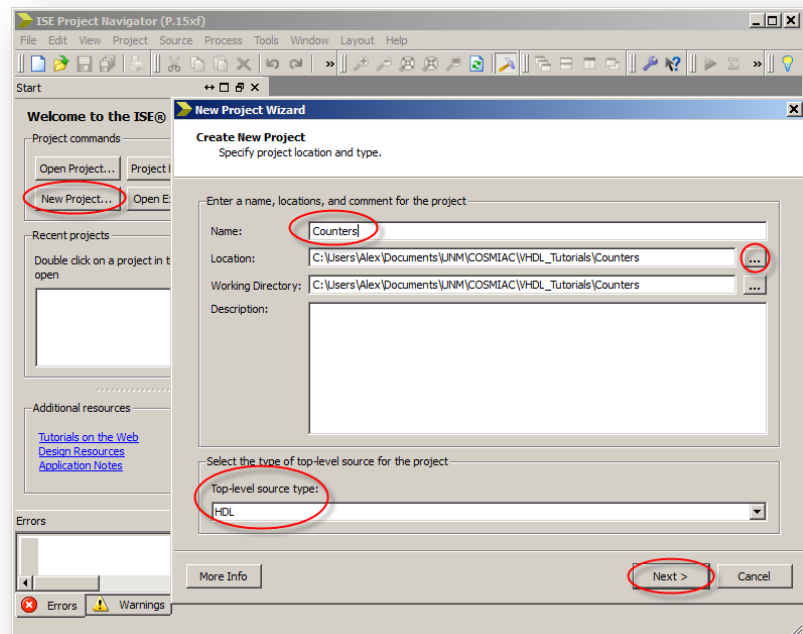


Lab Procedure 1: Counters

NOTE: Make sure you have completed the Lab Preparation before starting this Lab Procedure.

NOTE: If you've recently completed Lab 6, try creating the project and adding the source files on your own, without using the instructions. The project name is **Counter**, and the source files are **display_controller.vhd**, **counter.vhd**, and **counter4.vhd**.

1. Open **Xilinx ISE Design Tool Project Navigator**.
2. Click **OK** to close the Tip of the Day.
3. Start a new project by selecting **File → New Project** from the menu or click the **New Project** button. The New Project Wizard starts.
 - a. Type **Counter** in the Name text box.
 - b. Select a location on your computer to save your project files by clicking the ellipsis (...) button to the right of the **Location** text box.
 - c. Under **Top-level source type**, select **HDL**.
 - d. Click **Next**.



NOTE: File names must start with a letter. Use underscores (_) for readability. Do not use hyphens (-); although the file name will work, the entity name will not.

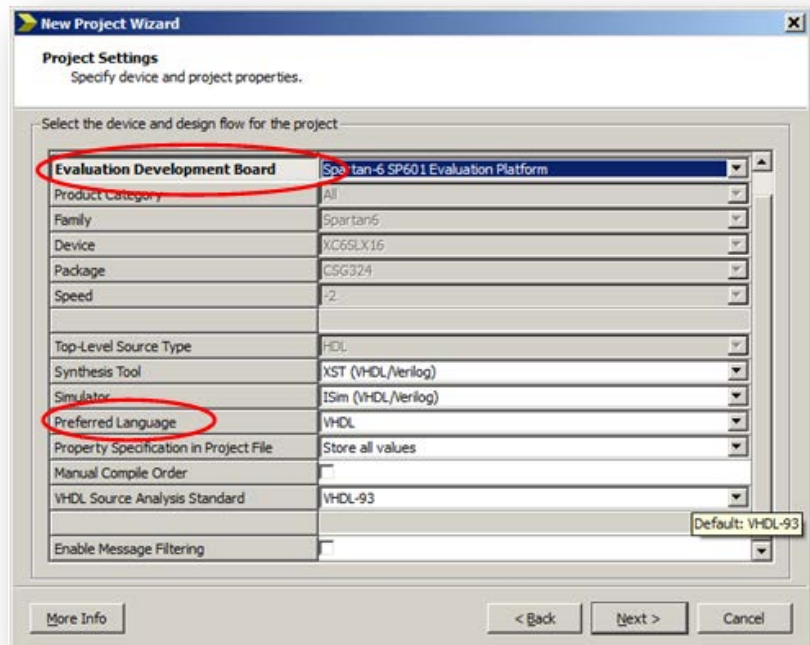


4. The Project Settings dialog box opens.
5. Select **Spartan-6 SP601 Evaluation Platform** from the **Evaluation Development Board** drop down menu.

The **Product Category**, **Family**, **Device**, **Package**, and **Speed** should all automatically populate (top half of the screen). You will need to set some options in the lower half of the dialog box.

Select **VHDL** from the **Preferred Language** drop down menu.

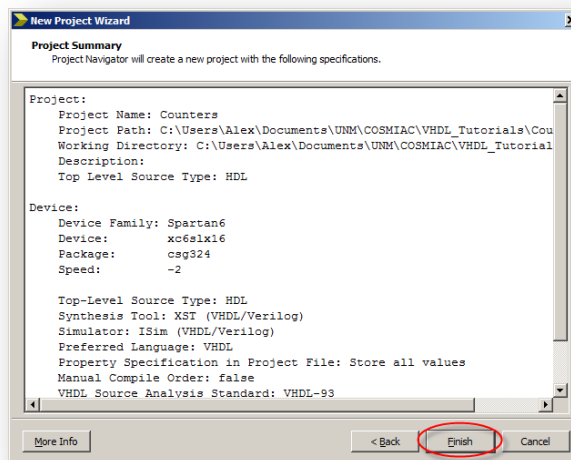
The Project Settings should resemble the figure to the right.



6. Click **Next**.

NOTE: The options specified are for the **Spartan 6 LX FPGA**. Your board might be different than the board used for these instructions. Board specifications are printed on the FPGA chip in the middle of the board. The board information is also listed on the box it came in.

7. The Project Summary displays. Verify the file type and name are correct then click **Finish** to complete the New Project creation process.

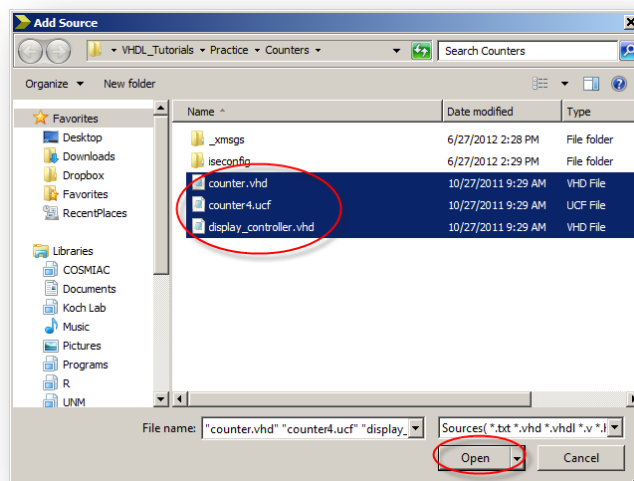
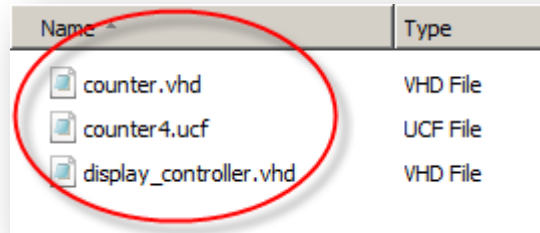




NOTE: There are three main files associated with this tutorial. The project files are in the **counter.zip** file.

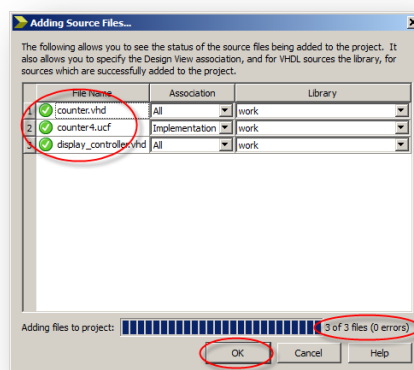
We will be adding the files from the counter.zip file to our new project.

8. Using Windows Explorer **move** these three files to your project folder.
9. In Xilinx ISE® select **Project** → **Add Source** from the menu or **Right-click** in the Hierarchy pane and select **Add Source** from the shortcut menu. The **Add Source** dialog box opens.
10. **Select** the three files that were moved to the project directory earlier. **Click** the **Open** button. The **Adding Source Files...** dialog box opens.



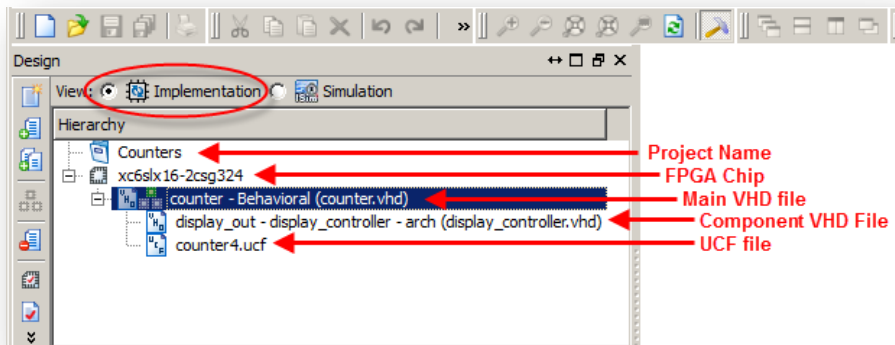
11. Click the **OK** button to complete the add files process.

NOTE: All three project files display with a green checkmark to the left of the file name. This lets us know that ISE understands the design association of each of the files.





12. The files are added to the **Counter** project.



NOTE: It is important to understand the file Hierarchy pane. The files display in a tree structure similar to the file and directory structure on a computer.

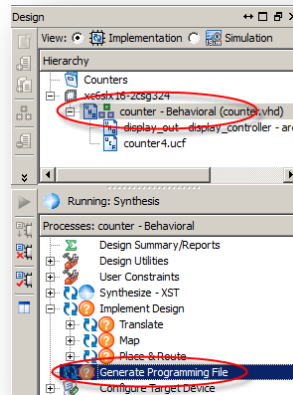
When Implementation is selected the UCF file should be below the design file it is associated with.

This image shows that at the top level is the project, then the chip type. Under the chip are the VHDL file, component VHDL file, and the associated UCF file.



Lab Procedure 2: Generate the Programming File (.bit) and Implement design to FPGA board

1. Select the top-level .vhd module, **counter - Behavioral (counter.vhd)**.
2. In the Processes pane double-click **Generate Programming File**. ISE will synthesize, implement and create the .bit file in one step.



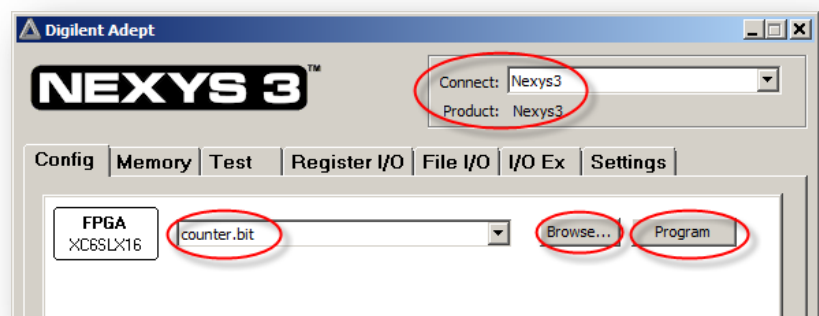
3. Open **Digilent Adept** by selecting Start → All Programs → Digilent → Adept → Adept. Your system might have slightly different Start Menu options.



4. Choose **Nexys3** from the **Connect Product** drop down menu.

NOTE: If you do not see Nexys3 as an option, check the USB connection to ensure the board is connected to the computer.

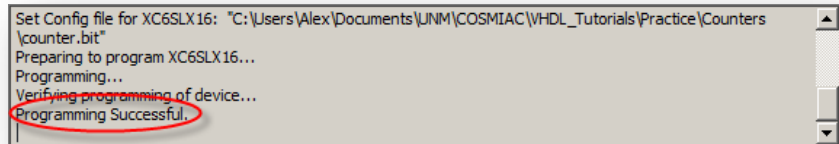
5. Click the **Browse** button. The Open dialog box displays.
6. Navigate to the folder of the **Counters** project and select the **counter.bit** file.
7. Click the **Open** button. The file is displayed in the dropdown list next to the FPGA icon.
8. Click the **Program** button. The bit file is programmed to the FPGA





board.

9. Once the program is downloaded to the FPGA board the status window will display **Programming Successful.**



10. Close Adept.

Now that the FPGA is programmed, the small green LEDs should be flashing quite rapidly, as they count in binary. The red cathode LEDs should be quickly counting in hexadecimal, from 0 – C5C1 (or 50,625 in decimal) after which they will start at 0 again.

NOTE: We used a binary and a hexadecimal counter for this Lab. Binary is a base 2 system. Hexadecimal is a base 16 system, so instead of going from 0-9, it goes from 0-15. The numbers 0-9 are still the same, but the letters A, B, C, D, E, and F represent 10-15 respectively. Our number system is decimal, or base 10. The base of a number is represented by the subscript base number, i.e. fifteen is represented as 15_{10} in decimal (base 10), 1111_2 in binary (base 2), or F_{16} in hexadecimal (base 16). Another example using twenty-seven: 27_{10} , 11011_2 , or $1B_{16}$.

The button on the far right (BTN1 labeled D9) is programmed to reset the count to start at 0. Holding the button will hold the count at 0. As soon as you release the button the counter will start counting again. You can configure the reset command to any button that you want.

NOTE: The Nexys 3 has a red reset button. This reset will delete the current program and reset the board. It will **not** reset the count of our design.

The far left switch (SW7 labeled T5) is a pause switch. When in the up (high) position, it will pause the counting. Moving the switch down (low) continues counting.



Lab Procedure 3: Review Reference Manual and VHDL code

Now that you have successfully downloaded the bit file to your FPGA board you may be wondering how the code for this program was written. Let's take a look at the board's reference manual and the VHDL code.

NOTE: This lab is written using the Nexys 3 board. If you are using a different board your instructor will let you know where to find the correct reference manual. You can also perform a Google search for “*Board Name* Reference Manual” (type the name of your board, such as Nexys 2 instead of *Board Name*).

Step 1: Reference Manual

Locate the I/O section in the reference manual. In the Nexys 3 reference manual this section is titled **Basic I/O**, and starts on page 18, as shown on the right. The figure on page 18 (and shown to the right) describe the input switches and buttons, and the output anode LEDs and four seven-segment LED displays.

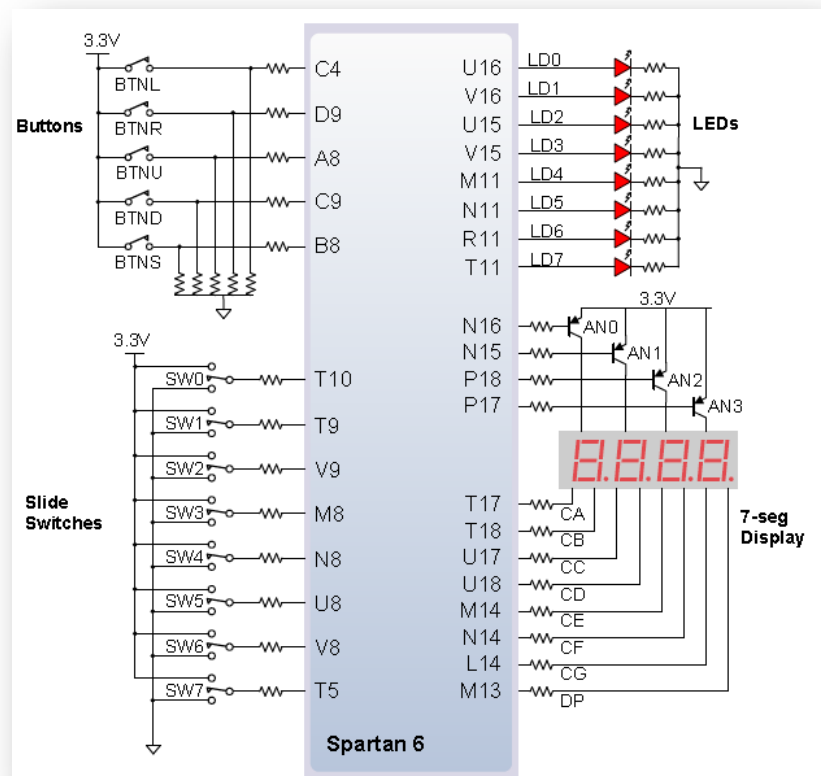
Look at your FPGA board and notice that LD0 through LD3 are blinking (they are actually displaying the four least significant bits of COUNT_OUT, which is counting in binary).

Looking back at the reference manual diagram, find the **LEDs** section at the top right. Note that LD0 through LD3 are connected to pints U16, V16, U15, and V15 on the FPGA.

The blinking LED is LD7 and is connected to pin T11 of the FPGA.

Directly below the **LEDs** section of the diagram is the **7-seg Display** section. There are four anodes labeled AN0, AN1, AN2, and AN3, connected to output pins N16, N15, P18, and P17, respectively. In the diagram, directly below the anodes are the seven cathodes pin assignments. These displays are *common anode*, each of the four separate anodes share the cathode signal.

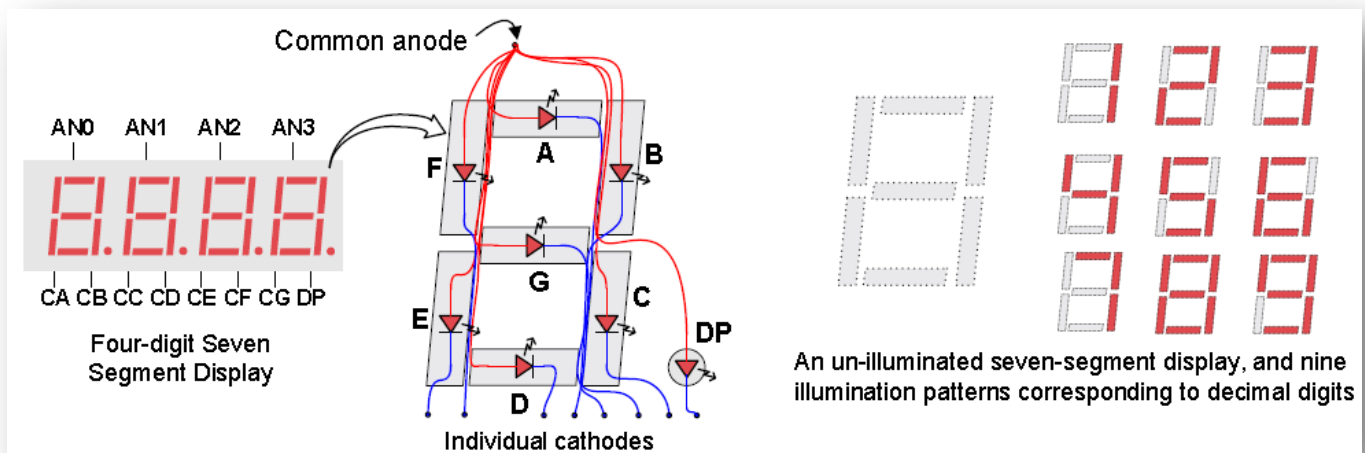
The anodes can only display the cathode signals one at a time. As described in the reference manual, “a scanning display controller circuit can be used to show a four-digit number on this display. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than the human eye can detect.”





Cathode signals labeled CA, CB, CC, CD, CE, CF, CG are connected to the FPGA pins T17, T18, U17, U18, M14, N14, and L14. The decimal point on the cathode display is labeled DP and is connected to pin M13; it is not used in this lab.

The next page of the reference manual explains the operation of the seven-segment display. For example FPGA output T17 (CA) connects to each A segment of the four anodes and FPGA output L14 connects to each G segment of the four anode displays.



The cathodes display with a LOW signal. To display the letter A, segments E, F, A, B, C, and G would be turned on with a LOW. Remember that anode AN0, AN1, AN2, or AN3 must also be turned on to illuminate the cathode display.



Step 2: VHDL Code

– COUNTER.VHD – Main Module

The tricky part to this project is not the actual counter. That is fairly straightforward as you will see when we get to that point. The tricky part to this project is actually how to display the results of the count onto the four anode, seven-segment display. As we've just discussed, each of the four anodes share a single set of seven cathode displays. The trick is to drive the count results to each of the four anodes at different times while making the display look like they are displaying at the same time. To do this correctly, it is necessary to jump from one anode to the other very quickly.

Open the “counter.vhd” file. As discussed in previous labs, there are three basic parts to a synthesizable VHDL file:

library IEEE:	This is where libraries are declared. Libraries allow the use of certain commands and operators.
entity “<i>entity_name</i>” is:	This is where the inputs and outputs are defined.
architecture “<i>architecture_name</i>” of “<i>entit_name</i>” is:	This is where we define the entity's behavior using VHDL.

First we will discuss the counter. The first 56 lines of code are the set-up of the hardware. We tell the hardware what libraries and packages to use. Next we define the I/O ports that we want the counter to use. We let the hardware know that we are using an additional component. We then map the ports of the component to the counter ports and declare additional signals that are needed. Finally, at line 67 we start to describe the hardware using two processes that will run concurrently. Let's take a look.

The code is an “up” counter since it has a “+”. This is actually just numerical addition. Looking at the first 4 lines of code you will notice lines 3 and 4 declare the *STD_LOGIC_arith* and *STD_LOGIC_unsigned* packages. These two packages allow the VHDL to use and understand arithmetic operations such as addition, subtraction, and less than. If these two lines are removed, the project will have errors since the rest of the program will not know how to deal with mathematical operations.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_arith.ALL;
4 use IEEE.STD_LOGIC_unsigned.ALL;
5
```

Lines 7 – 20 are the entity declaration for the counter. There are three inputs and four outputs. The counter has a clock, reset, and a pause input. The clock is what will drive the count. The reset and pause inputs do just what they say, reset the count or pause the count. The COUNT_OUT output is a 16-bit port. We will only display that last 4 bits of COUNT_OUT on LD0 – LD3. The COUNT_BLINK output will display on LD7 and will blink as the count is incremented. The last two outputs AN and SSEG are used to control the display on the four anodes. The anode that is currently used is selected using AN and SSEG controls which of the seven-cathodes to turn on.



```

7 -----
8 entity counter is
9     Port ( CLK : in  STD_LOGIC;                -- 100MHz clock
10           RESET : in  STD_LOGIC;               -- reset button
11           PAUSE : in  STD_LOGIC;               -- pause button
12           COUNT_OUT : out STD_LOGIC_VECTOR (15 downto 0); -- 16 bit counter on the LEDs which mirrors
13                                                         -- the counting on the seven-segment display
14                                                         -- in binary instead of hex
15           COUNT_BLINK : out STD_LOGIC ;         -- displays the rate at which the counters are counting
16           AN: out STD_LOGIC_VECTOR(3 downto 0); -- four digit anode display
17           SSEG: out STD_LOGIC_VECTOR(6 downto 0)); -- seven cathodes for each of the four anodes
18
19 end counter;
20 -----

```

Lines 22 – 116 contain the Architecture statement. This is where the hardware behavior is described. Lines 24 – 32 describe a component of the counter.vhd. In order to control the display of the anodes we created a sub-module called display_controller. When we generated the programming file earlier, recall that we only had to call the counter.vhd file. That is because we included the display_controller in the counter as a component. If you are familiar with C++ this is similar to including a class file.

When you include additional modules in a main vhd module you declare them as a component of the main module. This is a very simple task. Just use the component statement, as shown, and copy the entity port (); from the sub-module.

```

21
22 architecture Behavioral of counter is
23
24 -- Include the display_controller.vhd as a component in counter.vhd
25 -- This will connect the anodes to the counting LED's of the counter processes
26 component display_controller is                -- see display_controller.vhd for
27     port(                                       -- descriptions of these ports
28         CLK, RESET: in std_logic;
29         HEX3, HEX2, HEX1, HEX0: in std_logic_vector(3 downto 0);
30         AN: out std_logic_vector(3 downto 0);
31         SSEG: out std_logic_vector(6 downto 0));
32 end component;

```

Lines 34 – 41 declare the signals that are used for the counter and to slow the clock speed.

```

34 -- create signals used for the counter.vhd processes
35 -- TEMP_COUNT is used for the counter
36 -- SLOW_CLK and CLK_DIVIDER are used to slow the speed of the CLK so that the anode display
37 -- flashes at a speed that is discernable to the human eye
38 -- ... see CLK_DIVISION and COUNTER process for further explanation
39 signal TEMP_COUNT : std_logic_vector(15 downto 0) := x"0000"; -- counter
40 signal SLOW_CLK : std_logic ; -- new 'slow' clock that is used instead of the actual CLK
41 signal CLK_DIVIDER : std_logic_vector(23 downto 0) := x"0000000"; -- 24-bit divider, x = hex (16 bits) + 6 zeros = 24

```



The next 12 lines of code map the display_controller component's ports to the counter's ports. When you use components this is an important step as it tells the synthesizer where to send the data from one module to the other.

```

44 -- map the ports of the display_controller.vhd to the counter.vhd
45 begin
46 DISPLAY_OUT : display_controller
47     port map (
48         CLK => CLK,                -- display_controller CLK gets counter CLK
49         RESET => RESET,            -- display_controller RESET gets counter RESET
50         HEX3 => TEMP_COUNT (15 downto 12), -- display_controller HEX3 gets counter signal TEMP_COUNT, bits 15-12
51         HEX2 => TEMP_COUNT (11 downto 8), -- display_controller HEX2 gets counter signal TEMP_COUNT, bits 11-8
52         HEX1 => TEMP_COUNT (7 downto 4),  -- display_controller HEX1 gets counter signal TEMP_COUNT, bits 7-4
53         HEX0 => TEMP_COUNT (3 downto 0),  -- display_controller HEX0 gets counter signal TEMP_COUNT, bits 3-0
54         AN => AN,                  -- display_controller AN gets counter AN
55         SSEG => SSEG               -- display_controller SSEG gets counter SSEG
56     );

```

The next two processes are the 'driving' instructions of counter.vhd. The processes run in parallel (concurrently).

The CLK_DIVISION process creates an illusion of “slowing the clock” so that the display is discernible by the human eye. If the entire signal was being driven at 100MHz, all the output LEDs would appear all on, all the time, as they would be changing too fast for the human eye to perceive. This “slow clock” is created by incrementing the CLK_DIVIDER signal by one on every rising edge of the board's 100MHz clock. Then the 21st bit of the CLK_DIVIDER is sent to the SLOW_CLK signal.

The bit size of CLK_DIVIDER determines the frequency of SLOW_CLK. Making the CLK_DIVIDER signal 24 bits will slow the counter enough so the changes can be seen in the LEDs. Basically the SLOW_CLK signal is the clock divider's 21st bit. When the clock divider's 21st bit is low then SLOW_CLK is low. When the clock divider's 21st bit is high then slow clock is high.

```

67 CLK_DIVISION : process (CLK, CLK_DIVIDER) -- sensitive to CLK and CLK_DIVIDER
68                                         -- everytime an event occurs on one of these signals the process runs
69 begin
70     if CLK'event and CLK = '1' then      -- if there is a rising edge of the board's CLK
71         CLK_DIVIDER <= CLK_DIVIDER + 1;  -- increment CLK_DIVIDER +1
72     end if;
73     SLOW_CLK <= CLK_DIVIDER(21);          -- SLOW_CLK gets bit 21 of CLK_DIVIDER
74     -- *****
75     -- for the test bench change SLOW_CLK <= CLK_DIVIDER to 2, back to 21 for compile
76     -- *****
77     COUNT_BLINK <= CLK_DIVIDER(23);      -- COUNT_BLINK gets bit 23 of CLK_DIVIDER
78                                         -- COUNT_BLINK is merely a single LED that flashes as CLK_DIVIDER increments
79                                         -- It is for show purposes only
80 end process;                             -- end of the CLK_DIVISION process

```

The counting process is where we actually perform the count. First we describe what should happen when the reset and the pause are activated. When the reset button is pressed, the count starts at 0. When the pause switch is thrown, the display will pause at the current count, until the pause switch is turned off.

Finally we describe how the board should perform the count on lines 99 – 103. This counter counts up to 65,535₁₀, or FFFF₁₆. First the counter checks to see if TEMP_COUNT is less than 65,535. If it is, then



TEMP_COUNT is incremented by one. If TEMP_COUNT is greater than 65,535 then TEMP_COUNT is set back to 0 and the count starts over.

NOTE: As the designer you could set the counter to count from 0 – 150, or 1 – 200, whatever you need for your particular project.

```

83 -- COUNTING process controls the RESET, PAUSE, and counting functionality of the board.
84 --
85 COUNTING : process(RESET, PAUSE, SLOW_CLK, TEMP_COUNT) -- sensitive to RESET, PAUSE, SLOW_CLK, TEMP_COUNT
86 -- everytime an event occurs on one of these signals the process
87 -- runs
88 begin
89     -- RESET control
90     if RESET = '1' then -- when the reset button is pressed
91         TEMP_COUNT <= x"0000"; -- set TEMP_COUNT = 0
92
93     -- PAUSE control
94     elsif PAUSE = '1' then -- or if the pause switch is on
95         TEMP_COUNT <= TEMP_COUNT; -- hold the display at the current value of TEMP_COUNT
96
97     else -- Otherwise (reset and pause are NOT activated)
98         -- counter
99         if SLOW_CLK'event and SLOW_CLK = '1' then -- when there is a rising edge of SLOW_CLK (SLOW_CLK equals 1)
100             if TEMP_COUNT < 65535 then -- and TEMP_COUNT is less than 65535 (FFFF in hexadecimal)
101                 TEMP_COUNT <= TEMP_COUNT + 1; -- increment TEMP_COUNT by 1
102             else -- otherwise (TEMP_COUNT is greater than 65535)
103                 TEMP_COUNT <= x"0000"; -- set TEMP_COUNT back to 0
104             end if;
105         end if;
106     end if;
107
108     COUNT_OUT <= TEMP_COUNT; -- COUNT_OUT gets TEMP_COUNT
109     -- COUNT_OUT only displays the 4 LSB (see UCF)
110     -- recall that TEMP_COUNT is also ported to
111     -- HEX3, 2, 1, and 0 in the port map for the
112     -- display controller
113
114 end process;

```

– DISPLAY_CONTROLLER.VHD – Component Module

The display_controller module is what controls the four digit seven-segment display. The first 16 lines of code are the library and package declarations and the entity section. There are six inputs and two outputs for the display_controller.

NOTE: Notice that the HEX3, HEX2, HEX1, and HEX0 are all declared on line 12. Whenever you have more than one port that has the same specifications, you can declare them all in the same line of code.

As in the counter module, the display_controller has a CLK and a RESET port. Additionally there are three 4-bit ports, HEX3 – HEX0. The two output ports AN and SSEG are the same configuration and the same name as the counter module. However, they are not the same ports, they just “look” the same.



```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6
7
8
9  entity display_controller is
10     port(
11         CLK, RESET: in std_logic;           -- 100MHz clock and RESET button (single bit)
12         HEX3, HEX2, HEX1, HEX0: in std_logic_vector(3 downto 0); -- 4-bit inputs for the LED lights
13         AN: out std_logic_vector(3 downto 0); -- 4-bit anode controller
14         SSEG: out std_logic_vector(6 downto 0) -- 7-bit cathode controller
15     );
16 end display_controller;
17

```

The next section, on lines 27 – 133 is the Architecture of display_controller. In the declarative section of the architecture, on Line 32 we declare a constant integer, N. We use N to define the size of the signal vectors Q_REG and Q_NEXT. The signals are used to slow the display speed. Finally we have the SEL and HEX signals which determine which anode to display and what value to send to the cathodes for display.

```

27 architecture arch of display_controller is
28
29     -- *****
30     -- CHANGE N value to 2 for test bench, back to 20 for compile
31     -- *****
32     constant N: integer := 20;
33         -- constant 'N' set to an integer, used to slow the
34         -- display speed so that regular humans can see it
35         -- changing this number to anything < 25 and the display
36         -- is too slow, > 18 and the display is too fast
37
38     signal Q_REG, Q_NEXT: unsigned(N-1 downto 0); -- Q_REG used to set the value of SEL, see process
39     signal SEL: std_logic_vector(1 downto 0);    -- SEL value determines which anode to display
40     signal HEX: std_logic_vector(3 downto 0);    -- HEX value determines the cathode display, see end of code

```

After the Begin statement we have a process that describes what happens on a reset or rising edge of the clock. When reset is pressed then Q_REG is set to zero. If the reset isn't activated then if there is a clock event and it is equal to one, then Q_REG gets the current value of Q_NEXT.

NOTE: The (others=>'0') command is used because we do not know the exact size of Q_REG (it is a vector of size N). This command allows us to set all the bits of Q_REG, whether there are 8 bits or 40 bits, to zero.

NOTE: Remember processes run concurrently in vhdl. So the entire time that this code is running on your hardware, every process from counter and display_control will all be running at the same time. Recall that CLK_DIVISION in counter is also performing some actions on every rising edge of the board's clock at the same exact instant that this display_controller process is sending Q_NEXT's value to Q_REG.



```

42 begin
43
44   -- This process controls the RESET button of the clock
45   process(CLK, RESET)
46   begin
47       if RESET='1' then
48           Q_REG <= (others=>'0');
49       elsif (CLK'event and CLK='1') then
50           Q_REG <= Q_NEXT;
51       end if;
52   end process;
53

```

The next two statements increment Q_REG by one, and send the 1st and 2nd MSB to SEL. E.g. If N = 8, then Q_REG and Q_NEXT are 8-bit vectors. When line 60 executes, bit 7 and bit 6 (remember that with 8 bits, bits are numbered 76543210) are sent to SEL. SEL will be used next to select the anode to display.

```

54
55   -- State logic for the counter
56   Q_NEXT <= Q_REG + 1;
57
58
59   -- 2 MSBs of counter to control 4-to-1 multiplexing
60   SEL <= std_logic_vector(Q_REG(N-1 downto N-2));
61
62

```

This process selects the anode to display, based on the value of SEL and sets the value for the cathode display at the selected anode. Only one anode can be on at any one instant. The anode that is on is the one with the low signal (or a zero). SEL is a 2-bit value and we used a case statement to send a value to AN with a 0 value in the bit position of the anode that we want on as shown in the following table.

SEL value	AN value	Anode On	Bit sent to UCF
00	1110	AN0	AN(0)
01	1101	AN1	AN(1)
10	1011	AN2	AN(2)
11	0111	AN3	AN(3)

What we will do is send one bit of the 4-bit vector AN, as indicated in the table to the UCF file. Notice that the only time that the first bit of AN (or AN(0)) is equal to zero and therefore, the only time anode 0 gets a zero value is when SEL = 00. This logic is the same for each of the other anodes.

After value of AN is set, on lines 76, 82, 86, and 90, we set the value of HEX. The signal HEX will determine which of the cathodes will light on the selected anode. HEX is a 4-bit signal that we use for the numbers 0 – 15 (decimal) or 0 – F (hexadecimal). The values of HEX0 – 3 are set from the 16-bit TEMP_COUNT signal in the counter module.



```

70  process(SEL, HEX0, HEX1, HEX2, HEX3)      -- sensitivity list includes SEL, HEX0, HEX1, HEX2, HEX3
71  begin                                     -- begin the process
72      case SEL is                          -- beginning of case statement, when SEL is...
73
74          when "00" =>                    -- when SEL is 00
75              AN <= "1110";                -- This sets AN0 on
76              HEX <= HEX0;                -- send HEX0 to the first digit (HEX)
77                                          -- for cathode display
78                                          -- HEX0 is the 4 least significant bits
79
80          when "01" =>                    -- when SEL is 01
81              AN <= "1101";                -- This sets AN1 on
82              HEX <= HEX1;                --
83
84          when "10" =>                    -- when SEL is 10
85              AN <= "1011";                -- This sets AN2 on
86              HEX <= HEX2;                --
87
88          when others =>                  -- when SEL is anything else (11)
89              AN <= "0111";                -- This sets AN3 on
90              HEX <= HEX3;                -- send HEX3 to the first digit (HEX)
91                                          -- for cathode display
92                                          -- HEX3 is the 4 most significant bits
93
94      end case;                            -- end case
95  end process;                            -- end process
96

```

Lines 111 – 133 is the code that describes to the four anodes how to display the hexadecimal numbers 0 – F. Recall that when a cathode is low (or zero) it will light. This code uses a 4-bit signal called HEX to define what number to display. When HEX is 0001_2 (or 1_{10}) then we send 1000000 to SSEG the 7-bit cathode output. We will configure each bit of SSEG separately in the UCF in just a moment.

There are 7 cathodes, A - G, and in the UCF we will link them as follows:

Cathodes-->	G	F	E	D	C	B	A
SSEG bit -->	6	5	4	3	2	1	0

```

111  with HEX select
112      SSEG(6 downto 0) <=
113          "1000000" when "0000", -- 0, cathode G is OFF
114          "1111001" when "0001", -- 1, cathodes B and C are ON
115          "0100100" when "0010", -- 2, cathodes C and F are OFF
116          "0110000" when "0011", -- 3, cathodes E and F are OFF
117          "0011001" when "0100", -- 4, cathodes A, D, and E are OFF
118          "0010010" when "0101", -- 5, cathodes B and E are OFF
119          "0000010" when "0110", -- 6, cathode B is OFF
120          "1111000" when "0111", -- 7, cathodes A, B, and C are ON
121          "0000000" when "1000", -- 8, ALL cathodes are ON
122          "0010000" when "1001", -- 9, cathode E is OFF
123          "0001000" when "1010", -- 10, displays A (10 in Hex), cathode D is OFF
124          "0000011" when "1011", -- 11, displays b (11 in Hex), cathodes A and B are OFF
125          -- use a lower case b because capitol B looks like 8
126          "1000110" when "1100", -- 12, displays C (12 in Hex), cathodes B, C, and, g are OFF
127          "0100001" when "1101", -- 13, displays d (13 in Hex), cathodes A and F are OFF
128          -- use a lower case d because capitol D looks like 0
129          "0000110" when "1110", -- 14, displays E (14 in Hex), cathodes B and C are OFF
130          "0001110" when others; -- 15, displays F (15 in Hex), cathodes B, C, and D are OFF
131
132  end arch;
133

```



– COUNTER4.UCF – User Constraints File

The UCF file is what we use to configure the output of the previous code. Basically all ports are assigned to some pin on the board. The CLK is the only pin assignment in this file that isn't readily obvious by looking at the board labels. Check your board's reference manual for the correct pin assignment for the internal clock.

```

1  ## These pin assignments for for the Nexys 3, Spartan 6 board.
2  ## If you are using a different board, change the LOC = " " assignments
3  ## as needed. Check your board's reference manual clock and I/O sections
4
5
6  NET "CLK" LOC = "V10";           # Check Reference manual for correct assignment for clock
7  NET "RESET" LOC = "D9";          # reset uses BTNR, the reset button on the board resets the entire board
8  NET "PAUSE" LOC = "T5";          # setting SW7 to high will pause the counter
9  NET "COUNT_OUT(0)" LOC = "U16"; # LED0
10 NET "COUNT_OUT(1)" LOC = "V16"; # LED1
11 NET "COUNT_OUT(2)" LOC = "U15"; # LED2
12 NET "COUNT_OUT(3)" LOC = "V15"; # LED3
13 NET "COUNT_BLINK" LOC = "T11";  # LED7
14 NET "AN(0)" LOC = "N16";          # Transistor AN0
15 NET "AN(1)" LOC = "N15";          # Transistor AN1
16 NET "AN(2)" LOC = "P18";          # Transistor AN2
17 NET "AN(3)" LOC = "P17";          # Transistor AN3
18 NET "SSEG(0)" LOC = "T17";        # Segment A of HEX
19 NET "SSEG(1)" LOC = "T18";        # Segment B of HEX
20 NET "SSEG(2)" LOC = "U17";        # Segment C of HEX
21 NET "SSEG(3)" LOC = "U18";        # Segment D of HEX
22 NET "SSEG(4)" LOC = "M14";        # Segment E of HEX
23 NET "SSEG(5)" LOC = "N14";        # Segment F of HEX
24 NET "SSEG(6)" LOC = "L14";        # Segment G of Hex

```



Lab Procedure 5: Test your Knowledge

1. Create a new ISE project called **New_Count**.
2. Find and include the three files from the **Counter** project:

counter 4.ucf
counter.vhd
display_counter.vhd

Answer the following questions:

3. Why is the Hierarchy important? Where in the Hierarchy should the UCF fall?

4. What does the .bit file do?

5. What is the difference between a decimal, binary, and hexadecimal number?

6. Where can you find information about the FPGA board you are using?



7. The Nexys 3 board has a seven segment display. What does the term *common anode* mean?

8. What is the clock speed of the board you are using?

9. What purpose does the *STD_LOGIC_arith* package serve for your code?

10. How did we “slow the clock” in this lab?

11. What is a component?
