**Lab 6**

**How to Use Xilinx ISE 12 Project Navigator to create an up counter that will drive the seven segment display on the NEXYS2**

**Acknowledgements:**

Developed by Craig Kief, Engineering Faculty at University of New Mexico, Albuquerque, New Mexico. *Funded by the National Science Foundation (NSF) under the ATE Program. This effort was assisted by Quinlan Cao and Andrew Dodd.*

**Lab Summary:**

Counters allow hardware to do division. It is possible to be able 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, it would flash so often that it would appear to be on constantly.

**Lab Goal:**

This project has several goals. The first is to show how to develop and download a simple up counter. The second is how to drive the counted output to the four each seven segment digits on the NEXYS2 board.

**Learning Objectives**

1. Understand the hierarchical “tree” structure of projects and their test modules.

2. Using the wizard, create a test bench for the modules.

3. Create a test bench for the system.

**Method of Evaluation:** Your grade will be determined by your instructor.

**Time Required**: 2 hours

**Lab Preparation**

Read this document completely first before actually accomplishing the lab. This document was written by someone with many years of experience in this field and a great deal of effort was put into hoping to impart this knowledge to those looking to improve their skills.

**Equipment and Materials**

Access to Xilinx ISE software is all that is needed for this activity.

|  |  |
| --- | --- |
| **Software needed** | **Quantity** |
| The following items from the Xilinx: [www.xilinx.com](http://www.xilinx.com/)  ● free software **ISE** *WebPACK* | 1 |

**Additional References:**

Xilinx ISE 12 Software manuals found on Xilinx web site: www.xilinx.com and hardware manuals found at the Digilent Corporation’s website at [www.digilentinc.com](http://www.digilentinc.com/).

**Lab Procedure**

**Introduction**

On the wiki where you found this tutorial (http://vhdl\_fpgas.ece.unm.edu/index.php/Main\_Page) it is also possible to find the file counter.zip file. Download and unzip this file (of source files) onto your hard drive. Older versions of the Xilinx software had a serious problem with directory names or file names that had spaces in them. To this day, older designers will religiously avoid this practice. I would recommend unzipping the file into a directory called “Temp” or “Xilinxlabs” or something similar. There is nothing special about this project. The code may not even be that good. It was chosen due to its size and level of complexity. It is a good place to start.

**Procedure**

1. Download the .zip file. Make sure to unzip the zip file so that it can be used in the project (you can do this by going to the zip folder and clicking “extract all files” near the top of the window). The zip file contains three files. Two of the files are design files (.vhd) and one file is the UCF file. Create a new project using the ISE software (File: New Project). Name it “counter”. Press “Next” and a “Project Settings” window will open. Change all of the settings to those shown in Figure 1.



*Figure 1*

Now click “Next” and “Finish”. You can now begin adding source files.

2. Now, right click in the Design Section in the upper left corner of the ISE Design Suite (underneath where it says “Hierarchy” ). Choose the “Add Source” option and select the three source files (counter.vhd, counter.ucf, and display\_controller.vhd) from the unzipped folder. Press “Open” and then “Ok” on the “Adding Source Files” window.

4. Now that you have successfully added the source files, you can create the .bit file. Click on the top-level .vhd module labeled “counter - Behavioral (counter.vhd)”. Now look down below the “Hierarchy” to where it says “Processes”. Double-click on “Generate Programming File”. This will automatically make it perform three steps. It will first “Synthesize”, then progress to “Implement Design” and finally “Generate Programming File”. While you can click individually on one step at a time, it is often easier to just click on “Generate Programming File”. So now that you have your .bit file, it has been converted to be loaded onto your FPGA.

5. Next, you will open up “Digilent Adept”, the software that transfers the programming file (.bit file) to your FPGA board. Connect your FPGA to your computer, making sure the power is turned “On” on the board. Now click on “Initialize Chain” at the bottom of the Adept window. It should look like this:



*Figure 2*

6. Click on “Browse” and find your “counter.bit” file. Press open. A warning may occur that looks like this:



*Figure 3*

Click on “Yes”. Next click “Program”. The same warning will occur, but just press “Yes” again. Once you do that, your FPGA should begin flashing in fantastic ways! It will automatically count from 0-C5C1. Since this program is using the hexadecimal system, “C5C1” represents 50,625, and will therefore count up to “C5C1” and then revert back to “0” and start over. It 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.

If you hold the button on the far right on the FPGA board (B18), it will reset and go back to 0. If you let go, it will automatically begin counting again. The far left switch (R17) is a pause switch. When in the up position, it will pause the counting. When down, counting continues.

Now that you have successfully downloaded the bit file to your Nexys2 board lets dissect the code.

Lets first look at the Digilent Nexys2 Board Reference Manual (Nexys2\_rm.pdf). Turn to page 5/17 and you will see Figure 8. On your operating Nexys2 board notice that LEDs LD0 through LD3 are counting. Looking back at Figure 8, ( upper right hand corner) LD0 through LD3 are outputs J14, J15, K15 and K14 from the FPGA.

LED LD7 is the blinking LED on your board connected to pin R4 of the FPGA

Still referring to Page 5/17 of the reference manual. Look at the four hex displays on the lower right hand side of figure 8. The four PNP transistors (AN0 through AN3) above the HEX displays enable each of the HEX displays individually. Only one HEX display is enabled at a time. This is why the correct refresh frequency must be used so that we don’t see flickering or unrecognizable characters. Notice that each of the PNP bases is connected to a specific FPGA output F17, H17, C18, and F15.

Looking at page 5/17 notice the signals CA, CB, CC, CD, CE, CF, CG and this signals come from the FPGA L18, F18, D17, D16, G14, J17, and H14. DP is the decimal point on the HEX display, it is not used in this lab.

Looking at Page 6/17 Figure 9 you will see each segment of the HEX display. For example FPGA output L18 (CA) connects to ***each*** A segment of the four HEX displays and FPGA output H14 connects to ***each*** G segment of each of the four HEX displays. These HEX displays are COMMON ANODE meaning that the segment will illuminate when a LOW is applied to the segment needed to be illuminated.

If the letter A needed to be displayed, segments E, F, A, B, C, and G would be turned on with a LOW.

Remember that transistor AN0, AN1, AN2, or AN3 must be turned on to illuminate the specific HEX display

Let’s take a look at the UCF file. As you know the UCF file connect the specific FPGA outputs to the specific signals

***THE SYSTEM CLOCK 50MHZ IS CONNECTED TO FPGA PIN B8***

NET "clk" LOC = "B8"; # Check Reference manual for description page 4;

NET "reset" LOC = "B18"; #Push Button BTN0 Page 5 of reference manual;

NET "pause" LOC = "R17"; #Slide Switch SW7 Page 5 of reference manual;

***THESE ARE THE OUTPUTS TO THE LEDS LD0 TO LD3***

NET "count\_ out(0)" LOC = "J14"; #LED0 See Page 5 of reference manual;

NET "count\_out(1)" LOC = "J15"; #LED1 See Page 5 of reference manual;

NET "count\_out(2)" LOC = "K15"; #LED2 See Page 5 of reference manual;

NET "count\_out(3)" LOC = "K14"; #LED3 See Page 5 of reference manual;

***THIS IS LED LD7***

NET "count\_blink" LOC = "R4"; #LED7 See Page 5 of reference manual;

***THESE ARE THE OUTPUTS THAT ENABLE THE SPECIFIC HEX DISPLAY THROUGH THE PNP***

NET "an(0)" LOC = "F17"; #Transistor AN0 See page 5 of reference manual;

NET "an(1)" LOC = "H17"; #Transistor AN1 See page 5 of reference manual;

NET "an(2)" LOC = "C18"; #Transistor AN2 See page 5 of reference manual;

NET "an(3)" LOC = "F15"; #Transistor AN3 See page 5 of reference manual;

***THESE ARE THE INDIVIDUAL HEX DISPLAY SEGMENT CA,CB,CC,CD,CE,CF,CG***

NET "sseg(0)" LOC = "L18"; #Segment A of HEX display see page 6 of reference manual;

NET "sseg(1)" LOC = "F18"; #Segment B of HEX display see Page 6 of reference manual;

NET "sseg(2)" LOC = "D17"; #Segment C of HEX display see Page 6 of reference manual;

NET "sseg(3)" LOC = "D16"; #Segment D of HEX display see Page 6 of reference manual;

NET "sseg(4)" LOC = "G14"; #Segment E of HEX display see Page 6 of reference manual;

NET "sseg(5)" LOC = "J17"; #Segment F of HEX display see Page 6 of reference manual;

NET "sseg(6)" LOC = "H14"; #Segment G of Hex display see Page 6 of reference manual;

Before we start dissecting the code for the lab, let’s go over a simple N-bit counter

***This table is used for a 50MHZ input clock***

Clock divide frequencies

q(1) Frequency(Hz) Period(ms)

i 50000000.00 0.00002

0 25000000.00 0.00004

1 12500000.00 0.00008

2 6260000.00 0.00016

3 3125000.00 0.00032

4 1562500.00 0.00064

5 781250.00 0.00128

6 390625.00 0.00256

7 195312.50 0.00512

8 97656.25 0.01024

9 48828.13 0.02048

10 24414.06 0.04096

11 12207.03 0.08192

12 6103.52 0.16384

13 3051.76 0.32768

14 1525.88 0.65536

15 762.94 1.31072

16 381.47 2.62144

17 190.73 5.24288

18 95.37 10.48576

19 47.68 20.97152

20 23.84 41.94304

21 11.92 83.88608

22 5.96 167.77216

23 2.98 335.54432

***Example1 Clock divider***

Library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_unsigned.all

entity clkdiv is

port(

clr : in STD\_LOGIC; -- Switch of your choice

clk : in STD\_LOGIC; -- System clock of 50Mhz (B8)

clk25 : out STD\_LOGIC -- Freq of 25Mhz ( FPGA output)

clk190 : out STD\_LOGIC -- Freq of 190hz ( FPGA output)

clk3 : out STD\_LOGIC -- Freq of 2.98hz (FPGA outpu

);

End clkdiv;

architecture clkdiv of clkdiv is

signal count : STD\_LOGIC\_VECTOR(23 downto 0); -- This happens internally, its an array used -- store incrementing counts

begin

process(clk, clr)

begin

if clr = ‘1’ then -- If button pressed (your choice)

count <= (others => ‘0’) -- A process of assigning a 0 to every cell of the

--array count

elsif (clk’event and clk = ‘1’) then -- If clock is rising increment array count

count <= count + 1;

end process;

clk25 <= count(0); -- 25MHz -- Think of the count array as a counter that is

clk190 <= count(17); -- 190Hz -- incrementing

clk3 <= count(23); -- 3 Hz -- and Bit 0, Bit 17 and Bit 23 are being accessed.

--Any bit between 0 and 23 can be accessed for

--any one of the frequencies listed on the frequency table

end clkdiv

Note: In the UCF file you will set an FPGA output to clk25, clk190, and clk3

***Display\_controller.vhd***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

* -- clk is the 50Mhz system clock FPGA pin B8
* -- reset is is FPGA pin B18
* --hex3, hex2,hex1 and hex0 are four bit arrays used to store hex display character
* --an (an0,an1, an2, an3) four bit array that connect to the display transistors
* --sseg 7bit array that stores the individual segments of the hex display
* -- refer to ucf file and Nexys2 manual

entity display\_controller is

port(

clk, reset: in std\_logic;

hex3, hex2, hex1, hex0: in std\_logic\_vector(3 downto 0);

an: out std\_logic\_vector(3 downto 0);

sseg: out std\_logic\_vector(6 downto 0)

);

end display\_controller;

-- This architecture below describes the behavior of the multiplexer using its individual processes as well as

-- its seven-segment decoder to display numbers on the anodes using cathodes.

architecture arch of display\_controller is

-- each 7-seg led enabled (2^18/4)\*25 ns (40 ms)

constant N: integer:=18;

-- These signals are operations within the FPGA that help the processes overall but that we cannot see.

--q\_reg and q\_next are 18 bit arrays, in this application N=18

--sel is a 2 bit array that selects the hex display

--hex is a 4 bit array the contains the hex value to be displayed

signal q\_reg, q\_next: unsigned(N-1 downto 0);

signal sel: std\_logic\_vector(1 downto 0);

signal hex: std\_logic\_vector(3 downto 0);

begin

-- This process controls the reset button of the clock.

process(clk, reset)

begin

if reset='1' then --If button pressed all bits in q\_reg set to 0

q\_reg <= (others=>'0');

elsif (clk'event and clk='1') then --If clock is rising q\_next assigned to q\_reg

q\_reg <= q\_next;

end if;

end process;

-- State logic for the counter --Increment q\_reg and assign to q\_next

q\_next <= q\_reg + 1;

-- 2 MSBs of counter to control 4-to-1 multiplexing

--assign bits17 and 16 to sel array

sel <= std\_logic\_vector(q\_reg(N-1 downto N-2));

-- This is the 2:4 decoder, which converts a two-bit input into a four-bit input for the seven-segment decoder.

--The value in the sel array will determine which of the Case statements is selected. Since

-- the sel array is two bits then there are only four possible cases 0,1,2,3.

--The value in an will determine which transistor is turned on that selects one of four hex

--displays.

--The hex0, hex1, hex2, and hex3 value comes from the Counter.vhd file

process(sel, hex0, hex1, hex2, hex3)

begin

case sel is

when "00" =>

an <= "1110";

hex <= hex0;

when "01" =>

an <= "1101";

hex <= hex1;

when "10" =>

an <= "1011";

hex <= hex2;

when others =>

an <= "0111";

hex <= hex3;

end case;

end process;

-- The value that was assigned to the hex array is used to assign the sseg array. Example: If

-- the hex array contained the value 1001 (9) then the sseg array would be assigned the

-- value 0010000 which would display a 9 on the selected hex display. See reference

--if needed

with hex select

sseg(6 downto 0) <=

"1000000" when "0000", -- 0

"1111001" when "0001", -- 1

"0100100" when "0010", -- 2

"0110000" when "0011", -- 3

"0011001" when "0100", -- 4

"0010010" when "0101", -- 5

"0000010" when "0110", -- 6

"1111000" when "0111", -- 7

"0000000" when "1000", -- 8

"0010000" when "1001", -- 9

"0001000" when "1010", -- A, which signifies "10"

"0000011" when "1011", -- B, which signifies "11"

"1000110" when "1100", -- C, which signifies "12"

"0100001" when "1101", -- D, which signifies "13"

"0000110" when "1110", -- E, which signifies "14"

"0001110" when others; -- F, which signifies "15"

end arch;

This file Counter.vhd is the Mother file or main file

File display\_controller.vhd is the child file

***Counter.vhd***

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 10:26:20 06/14/2011

-- Design Name:

-- Module Name: counter - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_arith.ALL;

use IEEE.STD\_LOGIC\_unsigned.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Below are our entities which consist of a clock of 50 MHz, Reset button, Pause Button,

-- A count out which is nothing more than a counter on the LED's which mirrors the counting on the seven-segment display

-- but in binary code, a count blink which simply displays the rate at which the counters are counting, anodes (an) which

-- trigger the seven segment, and cathodes (sseg) which are on the anodes themselves.

--clk is the 50Mhz system clock FPGA pin B8

--reset is push button FPGA pin B18 switch BTN0

--pause is slide switch FPGA pin R17 Switch 7

--count out is 16 bit array that contains value to be displayed on LEDs LD0,LD1,LD2,LD3

--count blink is output FPGA pin R4 LD7

--an is four bit array that enables transistor

--sseg 7 bit array that contains character to be displayed on hex display

entity counter is

Port ( clk : in STD\_LOGIC;

reset : in STD\_LOGIC;

pause : in STD\_LOGIC;

count\_out : out STD\_LOGIC\_VECTOR (15 downto 0);

count\_blink : out STD\_LOGIC ;

an: out STD\_LOGIC\_VECTOR(3 downto 0);

sseg: out STD\_LOGIC\_VECTOR(6 downto 0));

end counter;

-- The architecture below consists of a component describing the multiplexer as well as a component describing the

-- analog-to-digital converter. It also consists of the proccesses of the counter itself.

architecture Behavioral of counter is

-- This component is the multiplexer, which connects the anodes to the counting LED's of the counter proccesses.

--Check the child file (display\_controller.vhd) and you will see that this the entity

component display\_controller is

port(

clk, reset: in std\_logic;

hex3, hex2, hex1, hex0: in std\_logic\_vector(3 downto 0);

an: out std\_logic\_vector(3 downto 0);

sseg: out std\_logic\_vector(6 downto 0));

end component;

--temp\_count is a 16bit array that is initialized to zero

--clk\_divider is a 24 bit array that is initialized to zero

--temp\_count is used to count up to 64k and the values are displayed on the hex display

--clk\_divider is used as a clock to update and refresh the hex displays. If the correct frequency

--is not used the display will flicker or characters will not be recognizable

signal temp\_count : std\_logic\_vector(15 downto 0) := x"0000";

signal slow\_clk : std\_logic ;

signal clk\_divider : std\_logic\_vector(23 downto 0) := "000000000000000000000000";

-- These port maps rename the entities of the components so that they are recognizable and can be used in architecture.

-- The hex counters are sent to the temp\_count which will be diplayed on the LED's.

--The names on the left of the => are from the child file (display\_controller.vhd) and the files

--on the right of the => are from this file. This is how information is shared by mother and child --files

begin

display\_out : display\_controller

port map(

clk => clk,

reset => reset,

hex3 => temp\_count (15 downto 12),

hex2 => temp\_count (11 downto 8),

hex1 => temp\_count (7 downto 4),

hex0 => temp\_count (3 downto 0),

an => an,

sseg => sseg

);

-- These below will have to be changed appropriately as all are set to open right now.

-- These are counter codes that count in binary form.

-- slow\_clk and clk\_divider adjust the speed at which counting takes place.

-- If it seems as if the anodes are constantly displaying one nummber, the counting may be taking place at

-- high speed; simply slow it down to watch the counter's complete function to make sure all is working properly.

clk\_division : process (clk, clk\_divider)

begin

if clk'event and clk ='1' then

clk\_divider <= clk\_divider + 1;

end if;

slow\_clk <= clk\_divider(21); --slow clock running at 11.92Hz

count\_blink <= clk\_divider(23); --LED7 blinking at 2.98Hz

end process;

--This is the counter for values that are displayed on HEX display

counting : process(reset, pause, slow\_clk, temp\_count)

begin

if reset = '1' then

temp\_count <= x"0000";

elsif pause = '1' then

temp\_count <= temp\_count;

else

if slow\_clk'event and slow\_clk= '1' then

if temp\_count < 50625 then

temp\_count <= temp\_count + 1;

else

temp\_count <= x"0000";

end if;

end if;

end if;

count\_out <= temp\_count;

end process;

Congratulations, you now have completed the counter tutorial!

This is another explanation of the “counter.vhd” file

Open the “counter.vhd” file. Once again, there are three basic parts to a synthesizable VHDL file.

i. library IEEE

This is where libraries are declared. Libraries allow the use of certain commands and operators.

ii. entity “entity name” is

This is where the inputs and outputs are defined.

iii. architecture “architecture name” of “entity name” is

This is where we define the entity’s behavior using VHDL.

The tricky part to this project is not the actual counter. That is fairly straight forward as you will see when we get to that point. The tricky part to this project is actually the displaying of data onto the seven segment. The entire four digits share a similar set of seven segment wires. The trick is to drive the anode of each of the four of them at different times.

Shown above is the entity declaration for the counter itself. As you can see from the picture below taken from the manual, there are seven cathode lines that go to the display. These seven cathode lines service the entire four digits. To do this correctly, it is necessary to jump from one anode to the other very quickly. The anode/digit is light by driving the input low.

But, back to the counter first. Here is the code for the counter.

***counting : process(reset, pause, slow\_clk, temp\_count)***

***begin***

***if reset = '1' then***

***temp\_count <= x"0000";***

***elsif pause = '1' then***

***temp\_count <= temp\_count;***

***else***

***if slow\_clk'event and slow\_clk= '1' then***

***if temp\_count < 50625 then***

***temp\_count <= temp\_count + 1;***

***else***

***temp\_count <= x"0000";***

***end if;***

***end if;***

***end if;***

***count\_out <= temp\_count;***

***end process;***

The code is an “up” counter since it has a “+”. This is actually just numerical addition. If the student glances back to the library declaration of the counter file they will observe this.

*use IEEE.STD\_LOGIC\_arith.ALL;*

*use IEEE.STD\_LOGIC\_unsigned.ALL;*

These two lines allow the VHDL to be able 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.

In the case of the code above, this counter counts up to around 50,000 (65,535 to be exact). This number “50625” was chosen because it is 15 to the fourth power and is therefore close to the maximum number possible to be displayed. The key here to remember is that the designer can use to count up to any number they choose. Once it hits the magic number the designer can also determine where the count will again begin. The designer could create a counter that counts from 50 to 150. Whatever is needed. There is also a clock divider that slows the 50MHz onboard oscillator down to a level where it can be viewed. If the entire signal was being driven at 50MHz, the digits would all seem to glow as they would be changing to fast for the human eye to differentiate.

The clock divider determines the frequency of the slow clock. Basically the slow clock is the output of the clock divider’s MSB. When the clock divider’s MSB is low then slow clock is low. When the clock divider’s MSB is high then slow clock is high.

Every rising edge of the clock signal (50 MHz clock) will increment the clock divider. That means that the frequency of the slow clock is the frequency of the clock signal divided by , where *n* is the number of bits in the clock divider signal. Each period of the slow clock increments the counter so the time it takes for the counter to increment is divided by the frequency of the clock signal.

The clock input is tied to the 50 MHz clock and the clock divider signal has two bits. The frequency of the slow clock is calculated to be 12.5 MHz and the period is 80 ns.

12.5 MHz is much too fast. The counter is incrementing every 80 ns. If the clock divider is changed to 24 bits then the slow clock will operate at less than 3 Hz and the counter will increment every 0.3 seconds.

Making the clock divider signal 24 bits will slow the counter enough so the changes can be seen in the LED’s.