Tutorial 5
4- Bit Counter with Xilinx ISE 9.2 and Spartan 3E

Introduction

This tutorial will introduce a 4-bit counter. With four bits the counter will count from 0 to 9, ignore 10 to 15, and start over again. The timing of the counter will be controlled by a clock signal that is chosen by the programmer. There will also be a reset button and a pause switch.

The VHDL code was already written in the last tutorial, but this tutorial will break the code down into understandable segments. VHDL is an acronym inside of an acronym. The 'V' stands for Very High Speed Integrated Circuit (VHSIC) and 'HDL' stands for Hardware Descriptive Language. VHDL is a powerful language with numerous language constructs that are capable of describing very complex behavior needed for today's programmable devices.

Objective

The objective of this lab is to understand the segments of VHDL code and the concepts of a counter. A button will be used to clock the counter. Whenever the button is pushed the counter value will be incremented by 1. LED's will be used to project the numerical value (in binary) of the counter output. After the proper function of the counter is verified, a 50 MHz clock from the Spartan board will replace the button and the programmer will adjust the speed so the counter operation can be visualized from the LED's.

Process

1. Create VHDL code in Xilinx ISE 9.2.
2. Synthesize VHDL code and use ModelSim to check behavior.
3. Upload counter to Spartan board and increment using button.
4. Reprogram board using the 50 MHz clock and adjust speed.

Implementation

1. Start Xilinx ISE 9.2 and create a new project called "tutorial_5." Choose a project location (folder on the C drive) that will be easy to find. On the **Device Properties** window choose the settings shown in figure 1.

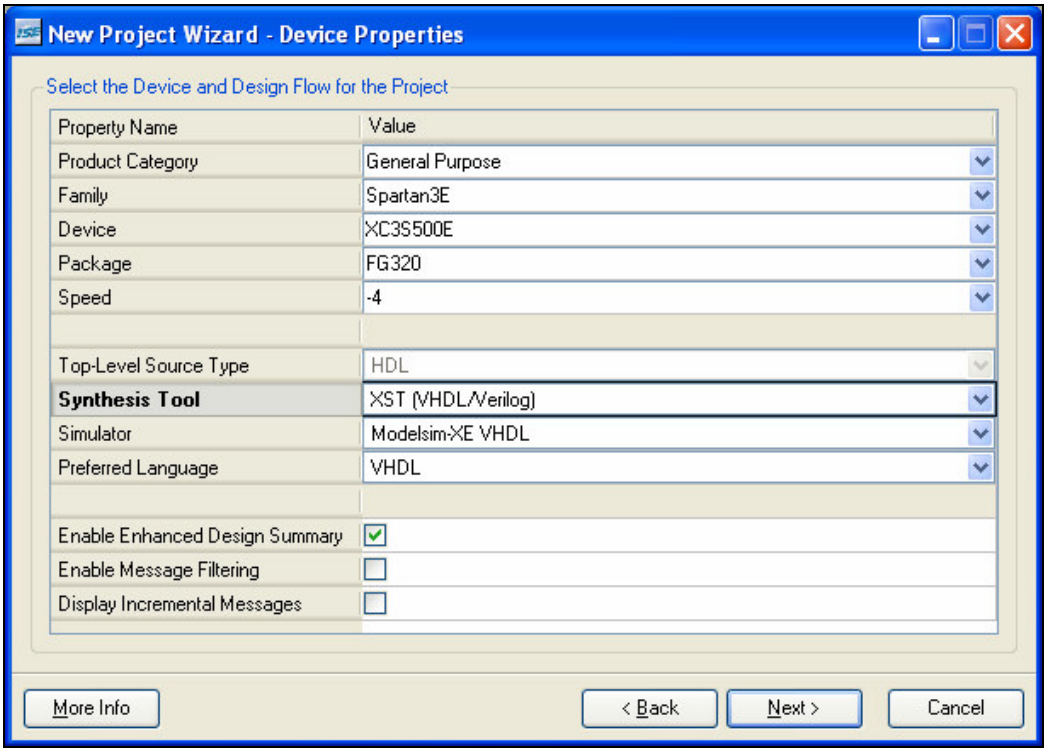


Figure 1 New Project Wizard - Device Properties settings.

Click "Next>" until the **Summary** window appears, then click "Finish."

2. Under the "Project" menu select "New Source…" In the **Select Source Type** window, highlight "VHDL Module" and enter the file name "counter". Click "Next>".

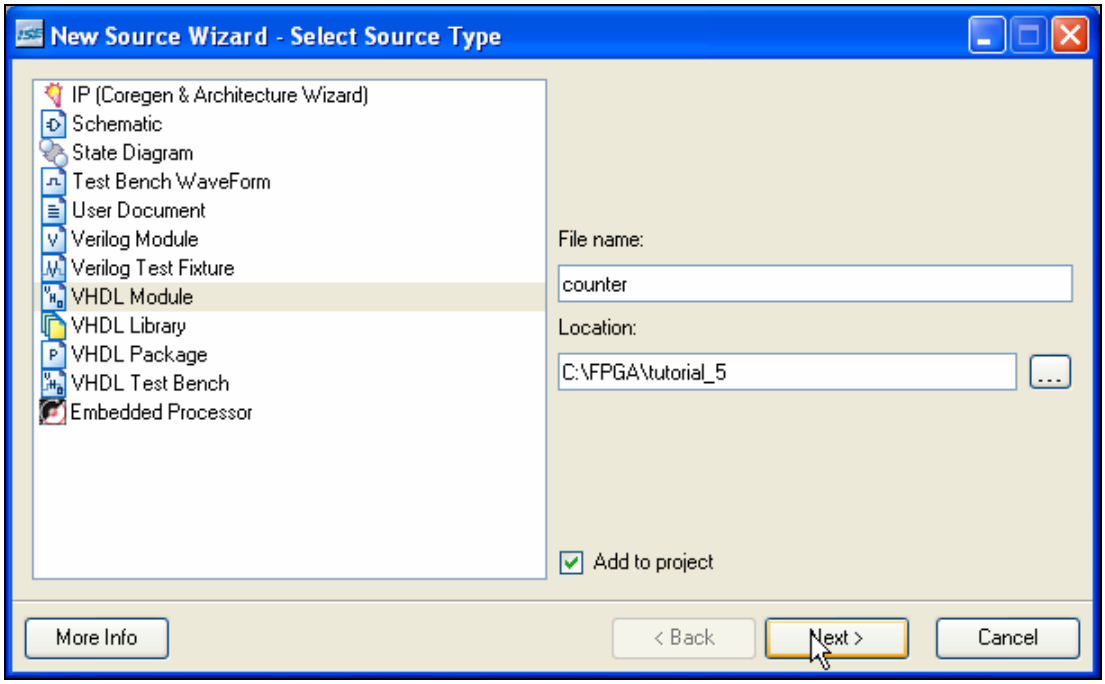


Figure 2 Creating a new source with the New Source Wizard.

3. In the **Define Module** window, list the port names in the "Port Name" Column. The ports are the lines going into and out of the device. Specify whether the port is an input (in) or an output (out) in the "Direction" column. The 'inout' selection will not be used in this tutorial. The box in the "Bus" column gets checked if the line will carry more than one bit. If "Bus" is selected, specify the most significant bit (MSB) and the least significant bit (LSB) in the appropriate column. Figure 3 shows the setup for the counter. Notice all ports carry one bit except for 'count_out', which will carry four bits (3 downto 0). When finished entering data, click "Next>".

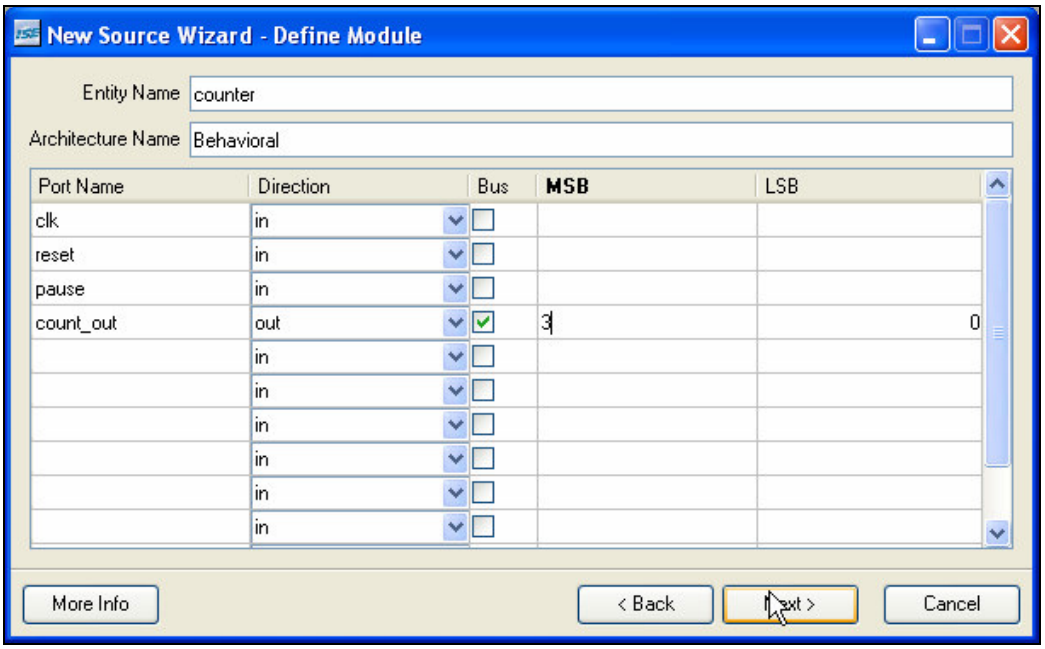


Figure 3 Counter setup in the Define Module window.

On the **Summary** window, click "Finish". The *counter.vhd* file will appear in the ISE workspace along with a design summary. Go ahead and close the design summary by right clicking on the "Design Summary" tab and selecting "Close".

4. There are three basic parts to a 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.

ISE's New Source Wizard took care of the library declarations and the entity. The programmer will write the architecture that defines the devices behavior.

5. The first part of writing the architecture is to declare some signals. Think of a signal as a wire that carries data between logic circuits that make up the counter. Signals exist within the device and are not inputs or outputs. That is why signals are not declared in the entity.

   Three signals will be declared. A signal to carry the value of the counter called 'temp_count', a signal to carry the adjusted clock signal called 'slow_clk', and a signal that determines how much the clock will be slowed called 'clk_divide'.

   The signals are declared after the architecture statement and before the begin command:

```
architecture Behavioral of counter is

  signal declarations go here

begin
```

   The syntax of a signal declaration looks like this:

```
signal signal name : type_name := "expression" ;
```

   Here are the signal declarations to type into the architecture:

```
signal temp_count : std_logic_vector(3 downto 0) := "0000" ;
signal slow_clk   : std_logic ;
signal clk_divider : std_logic_vector(1 downto 0) := "00";
```

   Signal 'temp_count' is of type 'std_logic_vector' which carries 4 bits. The bits are given an initial value of "0000". Notice the expression in enclosed in double quotes (""). This is the case for expressions of multiple bits. When writing an expression for a single bit the syntax calls for single quotes (''). This is a common syntax error when writing VHDL code.

   Signal 'slow_clk' is of type 'std_logic' and no initial condition is specified. Do not forget the semicolon at the end of each declaration.

   Signal 'clk_divide' is of type 'std_logic_vector' which carries 2 bits. The bits are given an initial value of "00".

   Checking syntax after each bit of programming is a good habit to get into. It is much easier to trouble shoot small sections of code rather than writing the whole program and then going back to find an error. To check syntax in ISE, expand the "Synthesize - XST" process in the **Processes** window and double click on the "Check Syntax" process. A green checkmark will appear next to the process icon if no errors are found. Errors that are found will be displayed in the ISE **Transcript** window along the bottom of the screen.

Save the changes in the VHDL file and check the syntax before going on to step 6.

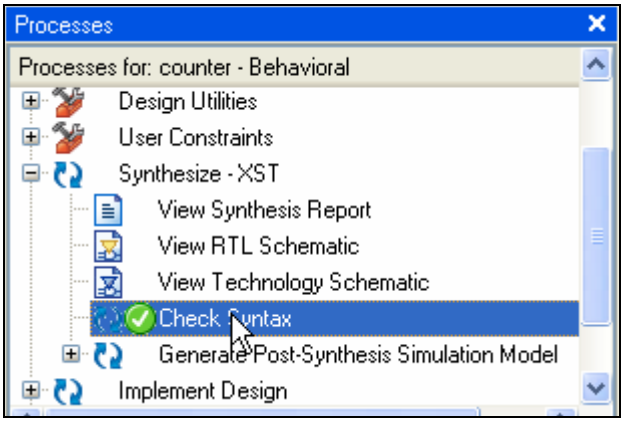


Figure 4 Checking VHDL syntax in ISE.

6. In VHDL, the most common way to implement sequential circuits is by using a process. There can be multiple processes in a design. All processes will execute concurrently. A process communicates with the rest of the design using signals or ports declared outside of the process (this was done in step 5). A process is activated when there is a change in a signal that has been predefined in a sensitivity list. Two processes will define the behavior of the counter.

   Below the signal declarations, the command word 'begin' declares the beginning of 'Behavioral', which is the name that was given to the architecture. Below "begin", enter the first process for the clock divider as shown below.

```
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(1);

end process;
```

   Here is the explanation of the 'clk_division' process:

   a. `clk_division : process (clk, clk_divider)`
      The name of the process is 'clk_division' and the sensitivity list includes the 'clk' input. The process will run every time the clk or clk_divider signal changes.

b. **begin**
This begins the code that describes the process.

c. **if** clk'event **and** clk = '1' **then**
clk_divider <= clk_divider + 1;
end if;
If the 'clk' signal is a rising edge then 'clk_divider' gets incremented.

d. slow_clk <= clk_divider(1);
'slow_clk' gets the MSB (most significant bit) of the 'clk_divider' signal. The more bits that 'clk_divider' has, the longer the 'slow_clk' period will be.

e. **end process;**
This is the end command that goes at the end of each process. It is the termination of the begin command.

7. The next process defines the counting. It follows much of the same syntax but uses several more 'if' statements. Type the following process.

```
counting : process(reset, pause, slow_clk, temp_count)

begin

 if reset = '1' then
   temp_count <= "0000";
 elsif pause = '1' then
   temp_count <= temp_count;
 else
   if slow_clk'event and slow_clk= '1' then
      if temp_count < 9 then
         temp_count <= temp_count + 1;
      else
         temp_count <= "0000";
      end if;
   end if;
 end if;

count_out <= temp_count;

end process;
```

Here is the explanation of the counting process. The first, second, and last lines have been omitted. Except for the names chosen by the programmer, the lines are in all processes.
:

a. **if** reset = '1' **then**
temp_count <= "0000";
If 'reset' is high then 'temp_count' gets zero (counter is reset). This is not dependent on a clock signal so the 'reset' is asynchronous.

b.
```
elsif pause = '1' then
   temp_count <= temp_count;
```
If 'pause' is high then 'temp_count' gets 'temp_count' (counter doesn't change). This 'pause' signal is also asynchronous. (the **elsif** spelling is intentional, do not use **elseif**)

c.
```
else
   if slow_clk'event and slow_clk = '1' then
```
If 'reset' and 'pause' are low then 'slow_clk' is checked for a rising edge. The process drops into the next if statement if a rising edge is detected.

d.
```
if temp_count < 9 then
   temp_count <= temp_count + 1;
else
   temp_count <= "0000";
end if;
```
'temp_count' will be incremented if it has a value of less than nine. If equal to or greater than nine, temp count gets zero.

e.
```
   end if;
end if;
```
Each 'if' statement must be closed with an 'end if'. This can get tricky when you have several if statements inside of each other. Indenting (tab) the text helps keep track of the hierarchy of the 'if' statements.

f.
```
count_out <= temp_count;
```
'count_out' gets the value of 'temp_count'. This is the output of the counter.

8. Below the processes is the final line of the VHDL module, which terminates the 'begin' command that appears just below the architecture declaration.

```
end Behavioral;
```

Save the changes then double click the "Synthesize - XST" process in the **Processes** window to synthesize *counter.vhd*. A syntax check will be done as part of this process and any errors will appear in the ISE **Transcript** window. A green check mark will appear next to the process icon if no errors are found.

9. A testbench is a VHDL file that tests the behavior of a particular device. Like all VHDL files, it has three basic parts: the library declarations, the entity, and the architecture. The New Source Wizard will take care of the library declarations and the entity. The process will be designed to test the counter.

   Under the "Project" menu select "New Source" or click on the New Source button on the Project Navigator toolbar.

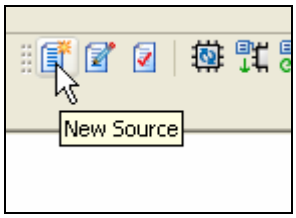


Figure 5 New Source Button.

In the **New Source Wizard - Select Source Type** window, select "VHDL Test Bench", enter "counter_tb" as the file name and then click "Next>". It is common practice to give a testbench file the same name as the device it is testing. Adding "tb" to the file name identifies it as a testbench.

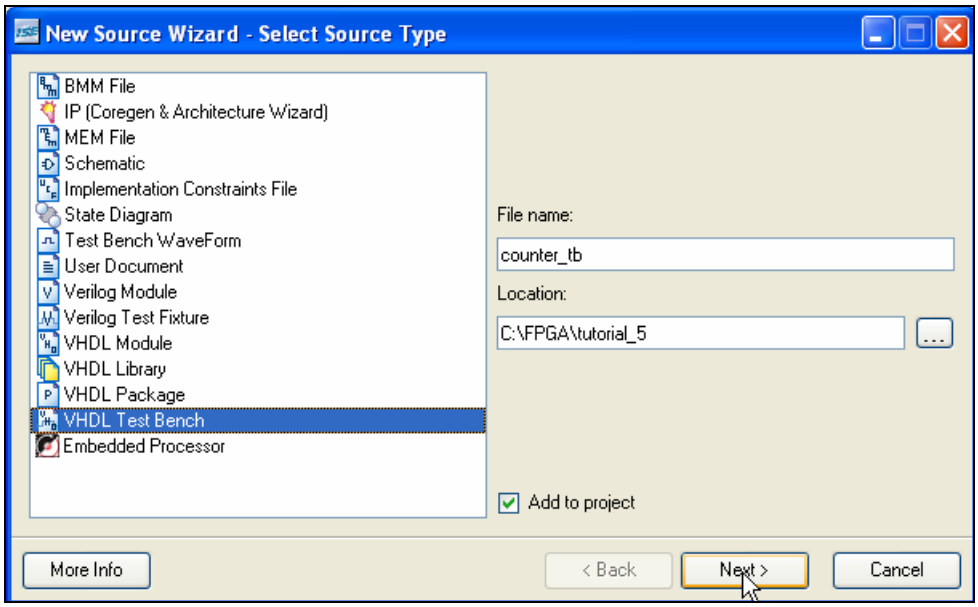


Figure 6 Creating a testbench with the New Sources Wizard.

In the **Associate Source** window 'counter' already highlighted because it is the only source available. Click "Next>", and then click "Finish" on the **Summary** window.

The testbench will open in the ISE workspace for editing. The New Sources Wizard has made the library declarations, and taken care of the entity. Writing the architecture is all that remains.

10. The architecture starts with a component declaration. This makes the counter available for use inside the testbench entity. Signals are declared after the component declaration. The signals will connect the counter to the testbench entity.

    Below the signal declarations, declare a time constant that will define the period of the clock signal.

```
constant clk_period : time := 10 ns;
```

    The New Source Wizard has already begun the architecture by instantiating the counter as the unit under test (uut). The instantiation wires the counter to the testbench.

    Below the instantiation, a process called "tb" has been started. Change the name from "tb" to "clock" and delete the existing commands inside the process. Create a process that will complement the 'clk' signal every 10 ns (clk_period). This will give the clock signal a period of 20 ns.

```
clock : PROCESS
      BEGIN

      clk <= NOT clk;
            wait for clk_period;

      END PROCESS;
```

    The next process, "pause_test", will control the pause signal. Pause will stay low for 74 clk_periods (37 clock signal periods) and then go high for 6 clk_periods (3 clock signal periods).

```
pause_test : PROCESS
      BEGIN

      pause <= '0';
            wait for clk_period*74;
      pause <= '1';
            wait for clk_period*6;

      END PROCESS;
```

The last process, “reset_test”, will control the reset signal. Reset will be low for 53 clock signal periods and then go high for 1 clock signal period.

```
reset_test : PROCESS
      BEGIN

      reset <= '0';
            wait for clk_period*106;
      reset <= '1';
            wait for clk_period*2;

      END PROCESS;
```

The last line of the testbench is the end command which is the termination of the begin command referring to architecture behavior. Save the changes made to the testbench file.

11. Simulate the behavior of the counter using ModelSim. Select “Behavioral Simulation” from the pulldown menu in the **Sources** window and highlight *counter_tb.vhd*. This will bring up the “ModelSim Simulator” toolbox in the **Processes** window. Click on the small box with the “+” symbol next to the toolbox and then double click on “Simulate Behavioral Model”.

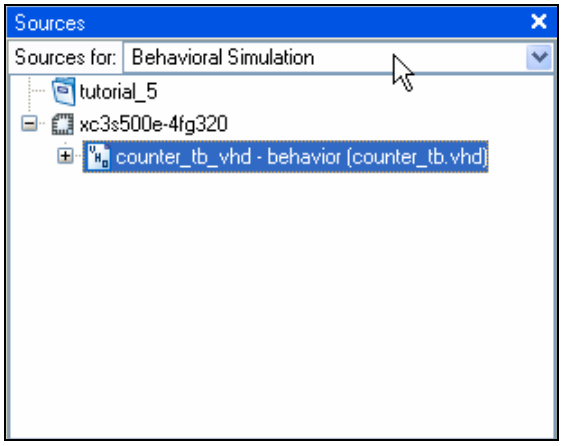


Figure 7 Pulldown menu in 'Sources' window.

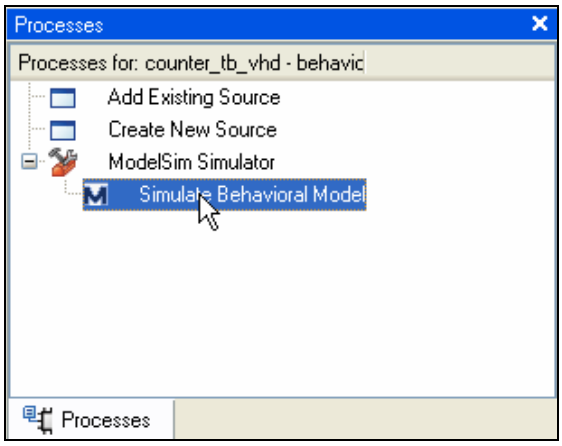


Figure 8 Starting ModelSim.

ModelSim will generate a behavioral waveform if there are no errors in the testbench. The waveform can be used to test the counter and aid in its design.

If there are errors, Modelsim’s **Transcript** window will display information to help troubleshoot the problem. Figure 10 shows the transcript error message generated when a semicolon was removed from line 54 in counter_tb.vhd.

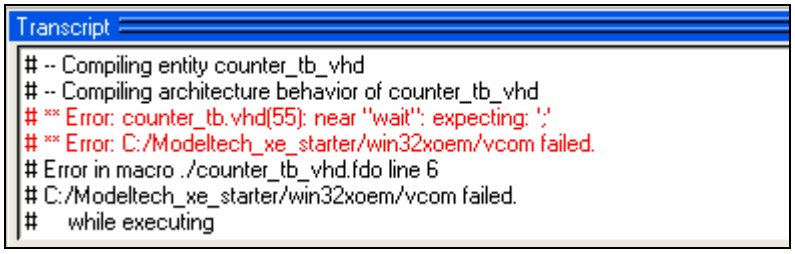


Figure 9 Error message in Transcript window.

Once the waveform has been generated, undock the **Wave** window and zoom out to see the features of the waveform.

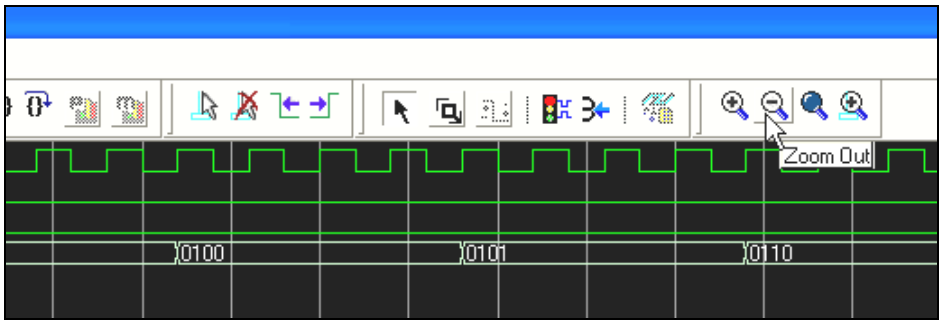


Figure 10 ModelSim zoom tools.

The wave representing ‘count_out’ is displayed in binary. The data can be displayed in an easier to read format. To change to a base ten display, right click on the wave name in the left column, select “radix”, and select “unsigned”.

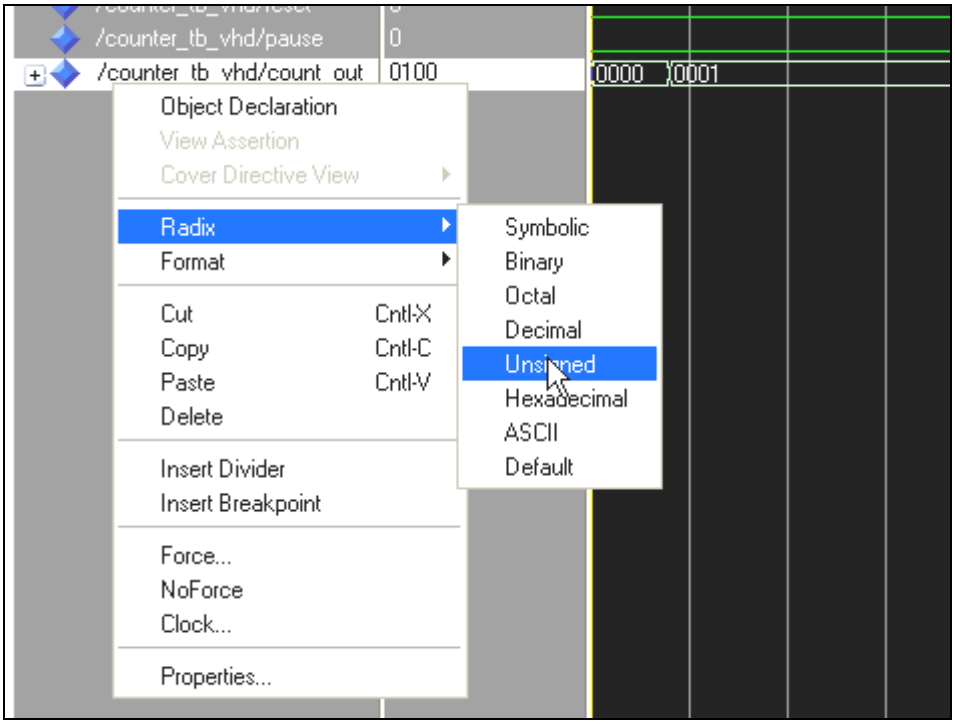


Figure 11 Changing radix of count_out waveform.

12. Once the counter is working properly it is time to create a UCF (user constraint file) and assign the FPGA pins to the counter's inputs and outputs. The clock and reset will be assigned to the north and south buttons. The pause will be assigned to switch 0, and bits 0-3 of count_out will be assigned to LED's 0-3. Pin assignments can be found in the Spartan 3E Starter User Guide

    Select "Synthesis/Implementation" from the pulldown menu on the **Sources** window. In the **Processes** window expand the "User Constraints" toolbox and double click on "Assign Package Pins". Choose "Yes" when Project Navigator asks to add a UCF file to the project.

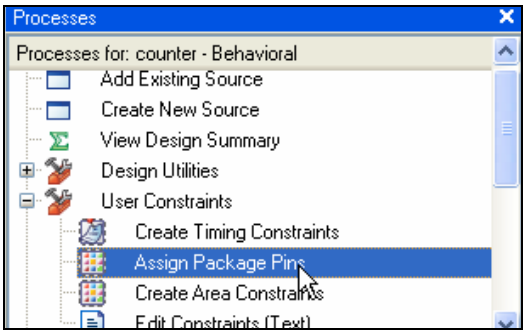
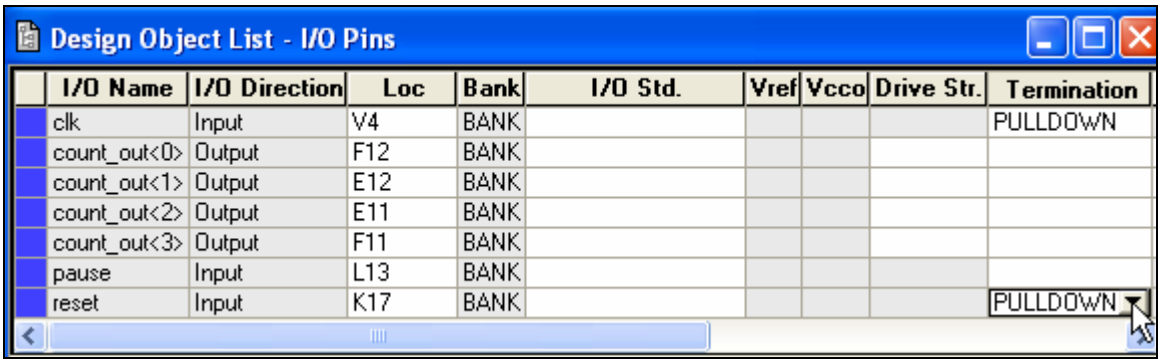


Figure 12 Starting Xilinx Pace to create UCF.

After the Xilinx PACE program starts, resize the **Design Object List - I/O Pins** window until the **Termination** column is visible. Enter the pin assignments in the **Loc** column. The Spartan board's buttons need the "PULLDOWN" constraint to function properly. Enter "PULLDOWN" in the **Termination** column for 'clk' and 'reset'. Refer to figure 14.



I/O Name	I/O Direction	Loc	Bank	I/O Std.	Vref	Vcco	Drive Str.	Termination
clk	Input	V4	BANK					PULLDOWN
count_out<0>	Output	F12	BANK					
count_out<1>	Output	E12	BANK					
count_out<2>	Output	E11	BANK					
count_out<3>	Output	F11	BANK					
pause	Input	L13	BANK					
reset	Input	K17	BANK					PULLDOWN

Figure 13 Pin assignments in Xilinx PACE.

Click the save button when the pins assignments have been entered. When the **Bus Delimiter** window comes up, make sure "XST Default: <>" is selected and press "OK". Close the Xilinx PACE program.

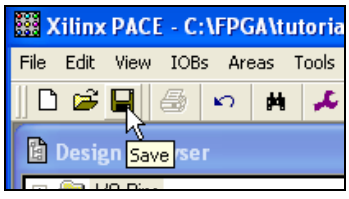


Figure 14 Saving in Xilinx PACE.

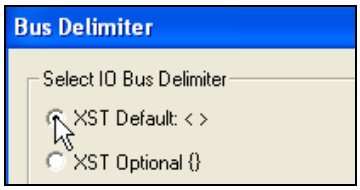


Figure 15 Select XST Default <>.

Double click “Edit Constraints (text)” in the **Processes** window to view the UCF in the ISE workspace. This will make it easy to edit the UCF later on in step 15.

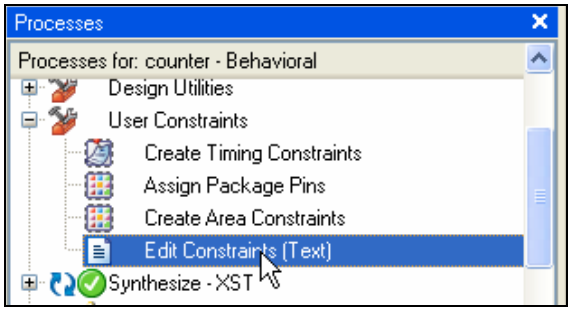


Figure 16 Opening the UCF in ISE workspace.

13. Plug the Spartan 3E board into your computer and turn the board’s power on. Expand the “Generate Programming File” process in the **Processes** window and double click “Configure Device (iMPACT).

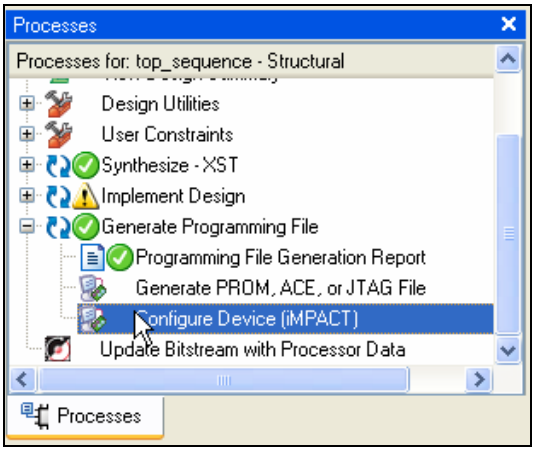


Figure 17 Selecting iMPACT will run all processes.

All the processes will run (this may take a minute or two). Ignore the warning on the “Implement Design” process. When all the processes have finished, iMPACT will start. Select the top radio button and click “Finish”.

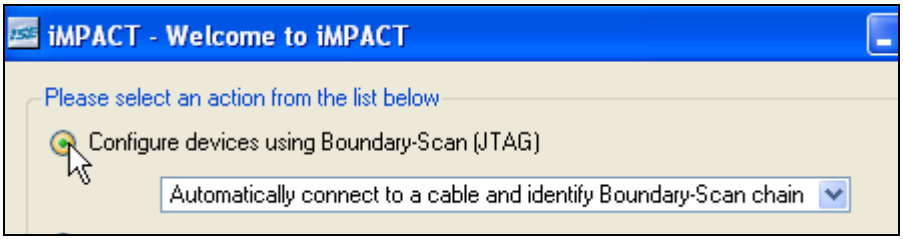


Figure 18 Welcome to iMPACT.

iMPACT will run a boundary scan that will appear in the ISE workspace. Assign *count.bit* to the FPGA (xc3s500e) and bypass the other two devices.

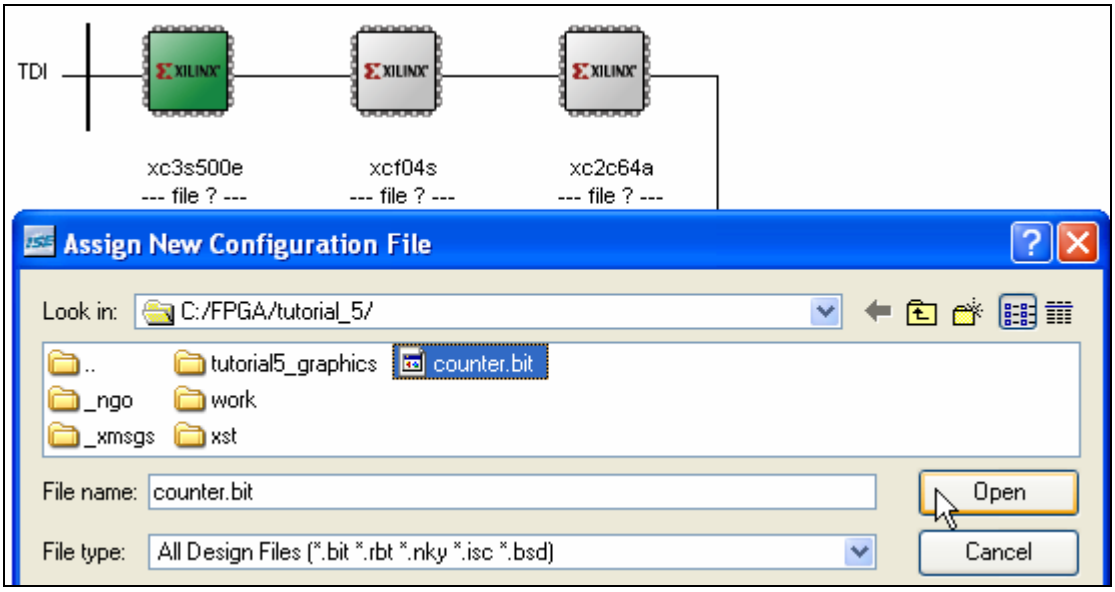


Figure 19 Assigning bit file to the FPGA.

14. Highlight the FPGA icon, right click the white space inside the ISE workspace, and select "Program…" Click "OK" on the **Programming Properties** window.

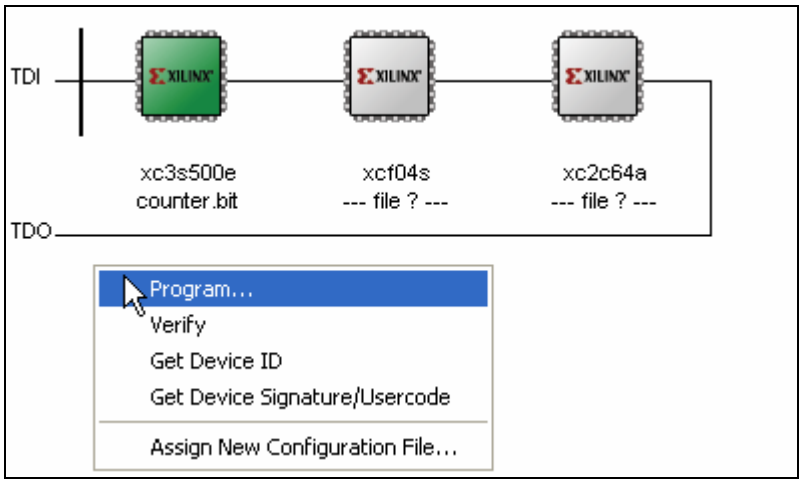


Figure 20 Programming the FPGA.

The Spartan board should now be programmed. Pressing the north button will increment the counter. You may have to press the button a few times because of the clock divider. The four LED's represent a four bit binary number. Switch 0 is the pause and the south button is the reset. Verify the counter is working properly.

15. Close iMPACT by closing the “Boundary Scan” in the ISE workspace (do not save changes when prompted).

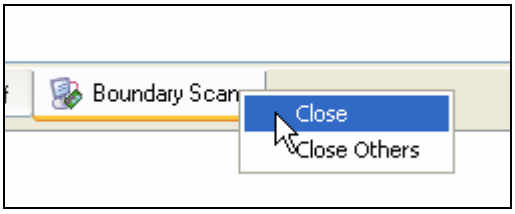


Figure 21 Closing iMPACT.

If not already done, open the UCF in the ISE workspace by double clicking “Edit Constraints (text)” in the Processes window.

Connect the Spartan board’s 50 MHz clock to the counter. Edit the line in the UCF that connects “clk” by removing the “PULLDOWN” constraint and changing “V4” to “C9”. Save the changes and reprogram the Spartan board (steps 12 and 13).

```
NET "clk" LOC = "C9";
```

The counter is now being clocked by the Spartan board’s 50 MHz clock and not the push button. All other pin assignments are the same.

Notice that all four LED’s appear to be on. They are changing, but because the counter is operating at a speed of 50 MHz, the changes cannot be seen. Toggle the pause switch on and off to see that the counter is working.

16. 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 2^n, 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 2^n 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.

$$f_{slow\_clk} = 50\,MHz\ /(2^2) = 12.5\,MHz \quad \Rightarrow \quad 1/12.5\,MHz = 80\,ns$$

12.5 MHz is much too fast. The counter is incrementing every 80ns. 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.

$$f_{slow\_clk} = 50\,MHz\ /\ (2^{24}) = 2.98\,Hz \quad \Rightarrow \quad 1/2.98\,Hz = 0.34\,s$$

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

To change the clock divider, start by closing iMPACT (as in step 14) and opening *counter.vhd* in the ISE workspace.

Edit the clock divider signal to make it 24 bits. Instead of typing 24 zeros in the initial expression, hex notation is used (specified by the "x") so only six zeros are needed.

```
signal clk_divider : std_logic_vector(23 downto 0) := x"000000";
```

The other change to make is in the clock division process. The slow clock gets the most significant bit (MSB) of the clock divider so change the (1) to a (23).

```
slow_clk <= clk_divider(23);
```

Save the changes made to *counter.vhd* and reprogram the Spartan board (steps 12 and 13).

Commented copies of *counter.vhd*, *counter_tb.vhd*, and *counter.ucf* are attached to the back of this tutorial for reference.

This tutorial was authored by Stephen Tomany. Stephen is a Junior in the Electrical Engineering Department at The University of New Mexico in Albuquerque. Questions or comments can be sent to stomany@unm.edu.

Rev. 01/24/08

counter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity counter is

port (      clk : in std_logic;
                  reset : in std_logic;
                  pause : in std_logic;
                  count_out : out std_logic_vector(3 downto 0));
end counter;

architecture Behavioral of counter is

signal temp_count : std_logic_vector(3 downto 0) := x"0";
signal slow_clk   : std_logic;

-- Clock divider can be changed to suit application.
-- Clock (clk) is normally 50 MHz, so each clock cycle
-- is 20 ns. A clock divider of 'n' bits will make 1
-- slow_clk cycle equal 2^n clk cycles.
signal clk_divider : std_logic_vector(23 downto 0) := x"000000";

begin

-- Process that makes slow clock go high only when MSB of
-- clk_divider goes high.

clk_division : process (clk, clk_divider)

begin
      if (clk = '1' and clk'event) then
            clk_divider <= clk_divider + 1;
      end if;

      slow_clk <= clk_divider(23);

end process;


counting : process(reset, pause, slow_clk, temp_count)

begin
      if reset = '1' then
            temp_count <= "0000";         -- Asynchronous reset.
      elsif pause = '1' then
            temp_count <= temp_count;     -- Asynchronous count pause.
      else

            if slow_clk'event and slow_clk ='1' then  -- Counting state
                  if temp_count < 9 then
                        temp_count <= temp_count + 1; -- Counter increase
                  else
                        temp_count <= "0000";         -- Rollover to zero
```

```
                end if;
            end if;
        end if;

count_out <= temp_count;            -- Output

end process;

end Behavioral;                     -- End module.
```

counter_tb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY counter_tb_vhd IS
END counter_tb_vhd;

ARCHITECTURE behavior OF counter_tb_vhd IS

      -- Component Declaration for the Unit Under Test (UUT)
      COMPONENT counter
      PORT(
            clk : IN std_logic;
            reset : IN std_logic;
            pause : IN std_logic;
            count_out : OUT std_logic_vector(3 downto 0)
            );
      END COMPONENT;

      --Inputs
      SIGNAL clk :  std_logic := '0';
      SIGNAL reset :  std_logic := '0';
      SIGNAL pause :  std_logic := '0';

      --Outputs
      SIGNAL count_out :  std_logic_vector(3 downto 0);

      constant clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
      uut: counter PORT MAP(
            clk => clk,
            reset => reset,
            pause => pause,
            count_out => count_out
      );

      clock : PROCESS
      BEGIN

      clk <= NOT clk;
            wait for clk_period;

      END PROCESS;

      pause_test : PROCESS
      BEGIN

      pause <= '0';
            wait for clk_period*74;
      pause <= '1';
            wait for clk_period*6;
```

```
	END PROCESS;

	reset_test : PROCESS
	BEGIN

	reset <= '0';
		wait for clk_period*106;
	reset <= '1';
		wait for clk_period*2;

	END PROCESS;

END;
```

counter.ucf

```
# UCF for the 4 bit counter


# Change "clk" pin assignment to "V4" (BTN_NORTH) to
# manually control the clock signal (don't forget "PULLDOWN")

# Change "clk" to "C9" (CLK_50MHz) to run counter off
# 50MHz clock

NET "clk"  LOC = "C9" ;
NET "count_out<0>"  LOC = "F12"  ; # LED<0>
NET "count_out<1>"  LOC = "E12"  ; # LED<1>
NET "count_out<2>"  LOC = "E11"  ; # LED<2>
NET "count_out<3>"  LOC = "F11"  ; # LED<3>

NET "pause"  LOC = "L13"  ; # SW<0>
NET "reset"  LOC = "K17" | PULLDOWN ;
```