Creating Your First Design with the Spartan-3A Evaluation Kit

Version 10.1.00

Revision History

Version	Description	Date
10.1.00	Initial release	May 27, 2008

Overview

If you've never designed with an FPGA before, or just need a refresher in using the latest Xilinx implementation tools, then this tutorial should serve as a starting point for your exciting journey. Creating a design in an FPGA can be as simple as a 10-minute exercise, or as complicated as creating a complete system-on-a-chip taking several designers multiple months to complete. It would be miss-leading to say that you can learn everything you need to know in this short 30 minute tutorial. Instead, this step-by-step guide is simply a way to become familiar with the basics of the FPGA design process using the Xilinx ISE Foundation tools and the Spartan-3A Evaluation board. Think of it as the hardware equivalent to that famous "Hello World" design example in software realm.

This tutorial will take you through the entire design process of creating a very simple "blink-the-LED" example. The steps covered are as follows:

- Creating a New Project
- Adding a New Source
- Behavioral Simulation
- Adding Timing Constraints
- Adding Pin Constraints
- Implementing the Design
- Verifying the Results
- Configuring the FPGA
- Modifying the Design
- Programming the SPI Flash

In the end you will see the results of your effort through a functioning design running on the Spartan-3A Eval board. Everything you need to accomplish this is included in the Avnet Spartan-3A Evaluation Kit.

To get the most from the tutorial, it is assumed that you have a basic understanding of FPGA architectures and some knowledge of hardware design languages (HDLs). The tutorial is written with both a VHDL and Verilog design example, although detailed knowledge of either is not required.

Beyond this tutorial, there are numerous books and training classes available that can help you hone your design skills. You are just beginning to scratch the surface of the very exciting field of FPGA design. The possibilities are endless, so read on and enjoy this exciting new world.

Objectives

After completing this tutorial you should be familiar with the following:

1) An introductory design flow used in creating simple FPGA designs
2) The basics in using the Xilinx ISE Foundation tools to implement your FPGA design
3) How to use the Avnet programming utility to configure the FPGA and SPI memory

Requirements

This tutorial is written for the Avnet Spartan-3A Evaluation Kit, which contains the Xilinx ISE Design Suite DVD, a Spartan-3A Evaluation board, a USB cable, and a downloadable FPGA

programming utility called AvProg. The following sections explain the software and hardware setup that's required to complete the Blink LED design.

Software

You will need to have the ISE WebPACK or Foundation software installed on a Windows compatible PC. The Installation and registration instructions are provided on the inside of the DVD jacket. The WebPACK version of the ISE software is free and supports the Spartan-3A FPGA family, which is what is used on the Spartan-3A Evaluation board. It's recommended that you install the WebPACK version for now, since it does not have the 60-day time-out limitation of the ISE Foundation Evaluation.

You will also want to visit the Xilinx web site (www.xilinx.com/support) and download the latest service pack. Although not required, it's always recommended to have the latest service pack installed to fix any potential bugs or errata that may exist.

Programming the Spartan-3A FPGA via the included USB cable, requires an Avnet developed utility called AvProg. This Windows based application takes the implemented FPGA design file from the ISE tool and downloads it to the FPGA through the USB cable. More details on this will be covered in the configuration section, so for now you just need to download and install the AvProg utility. The AvProg utility and installation instructions can be downloaded by clicking on the Technical Support link at www.em.avnet.com/spartan3a-evl.

Hardware

Besides the Spartan-3A Evaluation board and USB cable, the only other hardware required is a Windows compatible PC. Although the Xilinx ISE tools will also run under Linux, the AvProg programming utility is only supported under Windows.

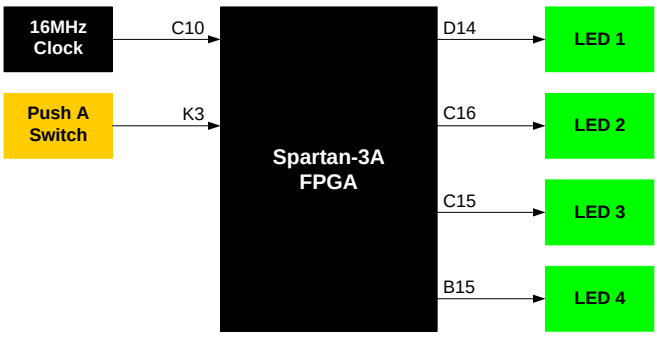
Setup

The Spartan-3A Evaluation board contains several jumpers that can be changed to alter the board's functionality. For the purpose of this tutorial, the factory default jumper settings should be used. Before starting, verify the following jumper settings are in place.

Jumper	Function	Pin Setting	Mode	Description
JP1	Flash Write Protect	Pins 1-2	Open	No write protect
JP2	Power Source #1	Pins 1-2	Closed	USB Power
JP3	Power-On Reset	Pins 1-2	Open	No Reset
JP4	Mode	Pins 1-2	Open	|
		Pins 3-4	Closed	| → Master SPI
		Pins 5-6	Closed	| Mode
		Pins 7-8	Open	|
JP5	FPGA Suspend	Pins 2-3	Closed	Disable suspend mode
JP6	External SPI	Pins 1-2	Closed	Disable external SPI
JP7	Power Source #2	Pins 1-2	Closed	USB Power

Getting Started

In this tutorial, you will create a simple binary counter that drives the LEDs on the Spartan-3A Evaluation board. A block diagram of the hardware features used on the board is shown below.



A 16 MHz clock source will drive the counter. The Push A button will act as a counter disable, such that when you place your finger on the Push A pad, the counter will stop counting up and the LEDs will display the current value held. Removing your finger will start the counter. The pin numbers shown next the FPGA will be important as you develop your design. The FPGA must use these specific pins since they are physically connected to the input and output devices used on the Spartan-3A Evaluation board.

So let's get started in creating your first FPGA design. To start ISE, double-click on the ISE icon,

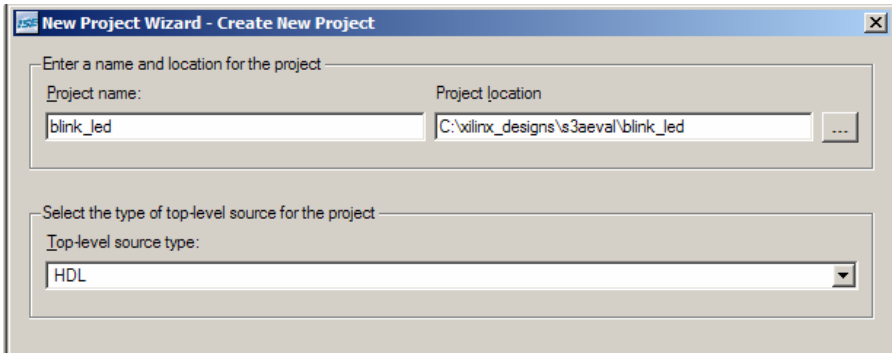
Or start ISE from the Start menu by selecting:

Start → All Program → Xilinx ISE 10.1 → Project Navigator

Creating a New Project

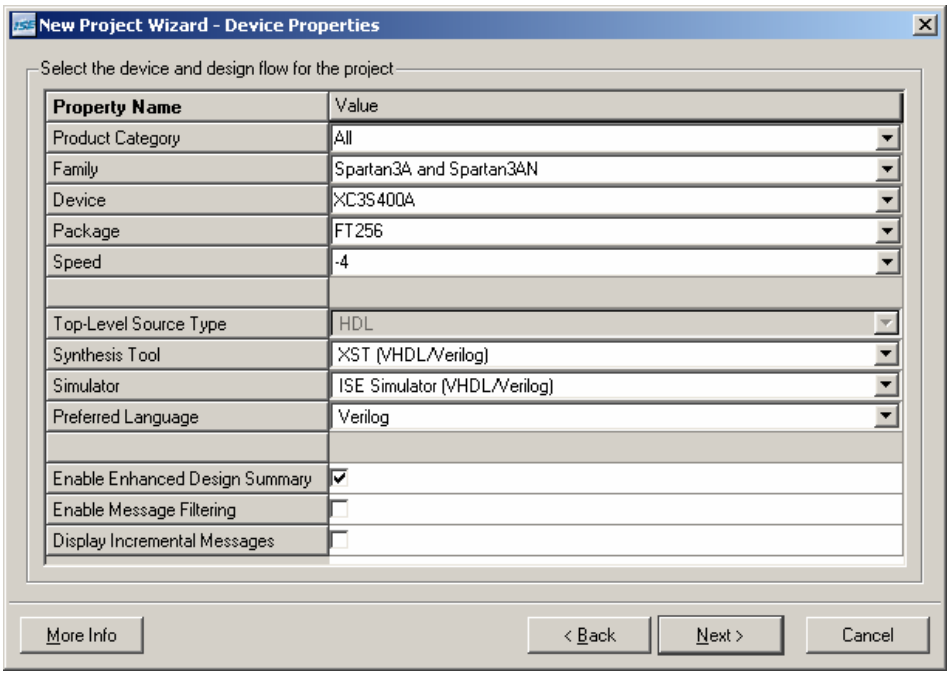
First, you need to setup a new project which specifies the parameters of your design and creates a directory containing all the project files. To create a new project:

1) Select **File > New Project…** The New Project Wizard window appears.
2) Type **blink_led** in the Project Name field.
3) Verify the Project location directory is where you want your design to be created. ISE automatically creates a **blink_led** directory for your project.

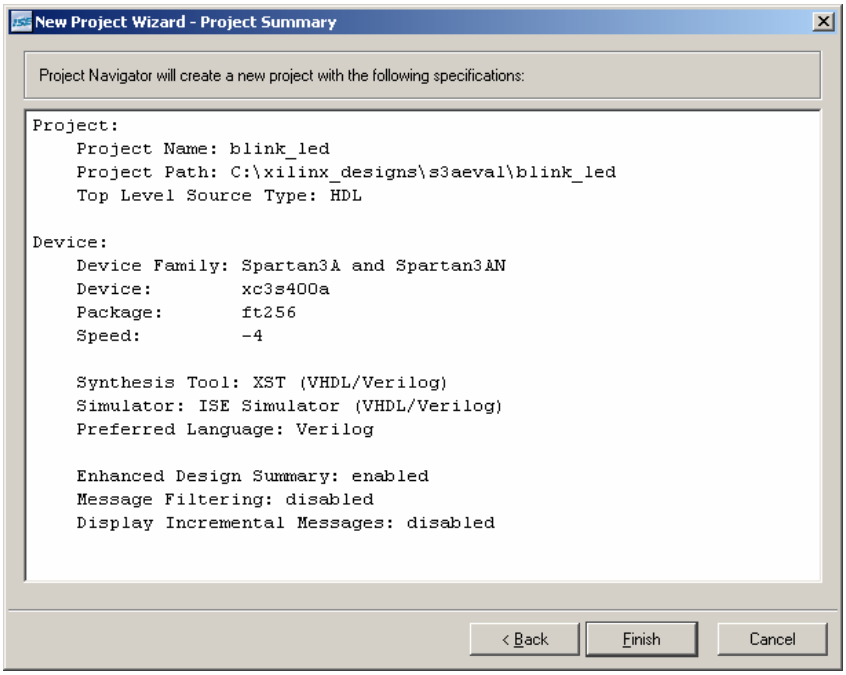


4) Verify that HDL is selected from the Top-level source type list.
5) Click **Next**.

6) Fill in the properties table as shown below. For the preferred language, you can select Verilog or VHDL.



7) Click **Next**.
8) The Create a new source window appears. Bypass this for now by clicking **Next**.
9) The Add Existing Sources window appears next. Bypass this by clicking **Next**.
10) A Project Summary window will appear next. Verify that your summary matches that shown below.



11) Click **Finish**.

ISE creates a **blink_led** project directory for your FPGA design and shows the targeted FPGA device (Spartan-3A XC3S400A-4FT256).

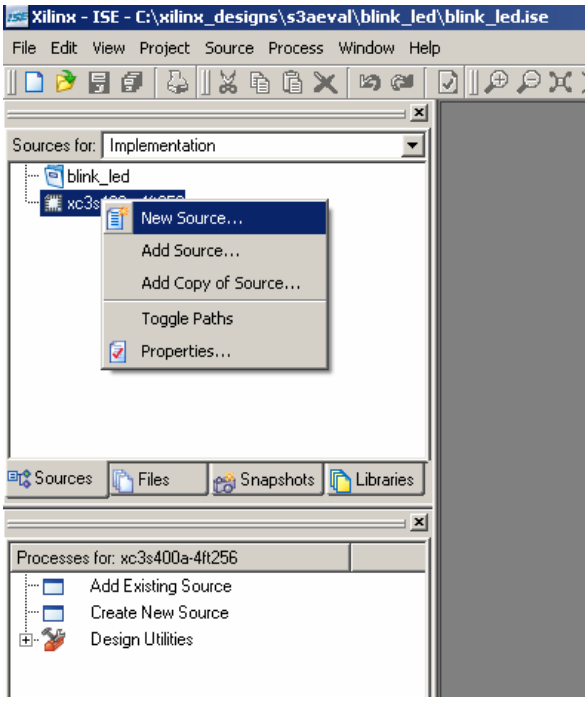
Adding a New Source

Next you need to create a top-level "source" file that defines your design. You have the option of creating either a VHDL or Verilog source file. Pick either the Verilog or VHDL flow and follow the appropriate section below. Skip the other language option and continue with the Behavioral Simulation section.

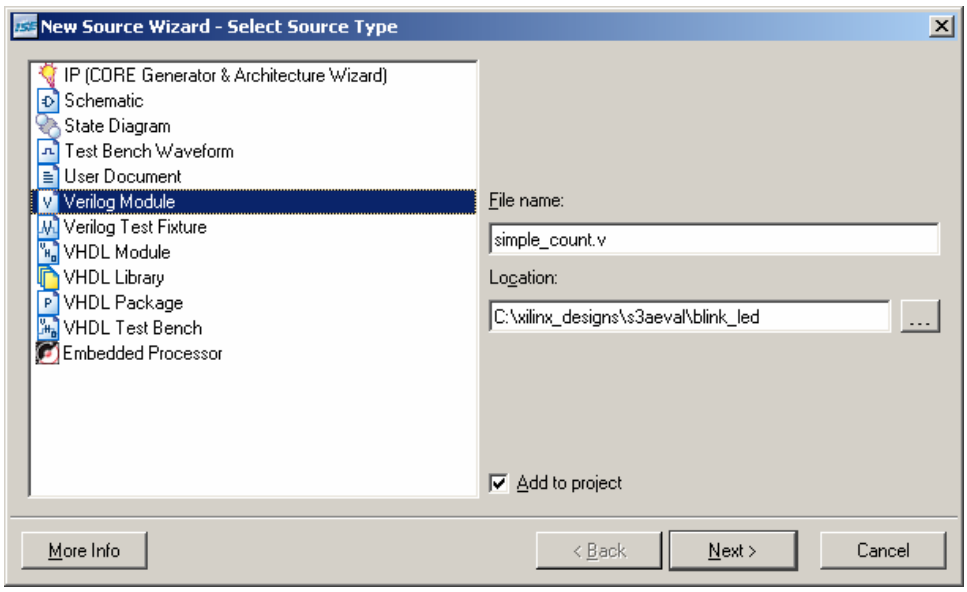
Creating a Verilog Source

Create the top-level Verilog source file for the project as follows:

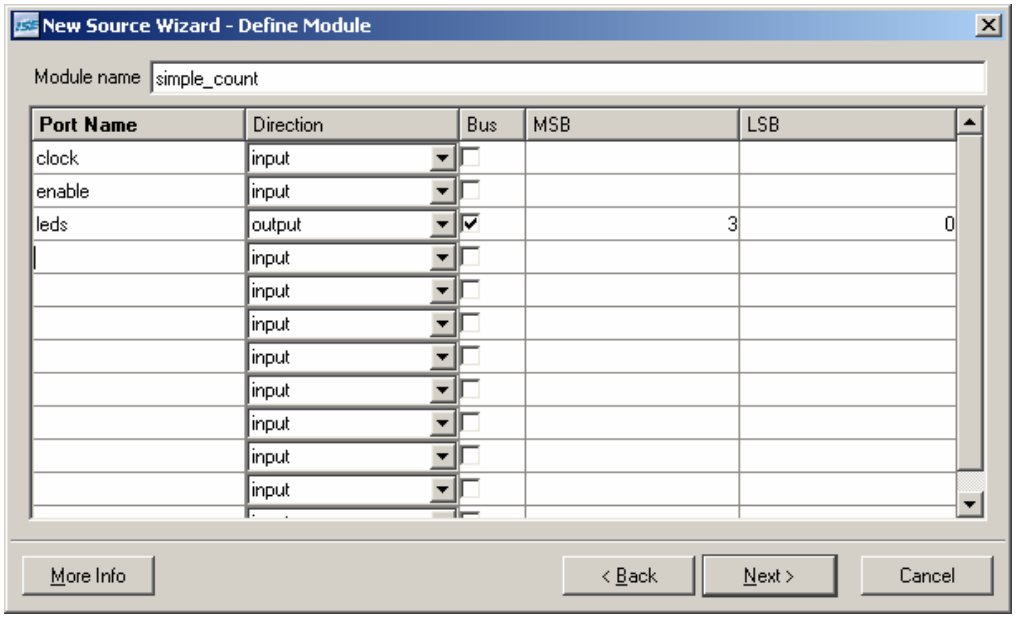
1) In the Sources window, right-click on the **xc3s400a-4ft256** device to pull up the menu and select **New Source…**



2) Select **Verilog Module** as the source type in the New Source dialog box.
3) Type in the top level module name **simple_count.v** as shown.



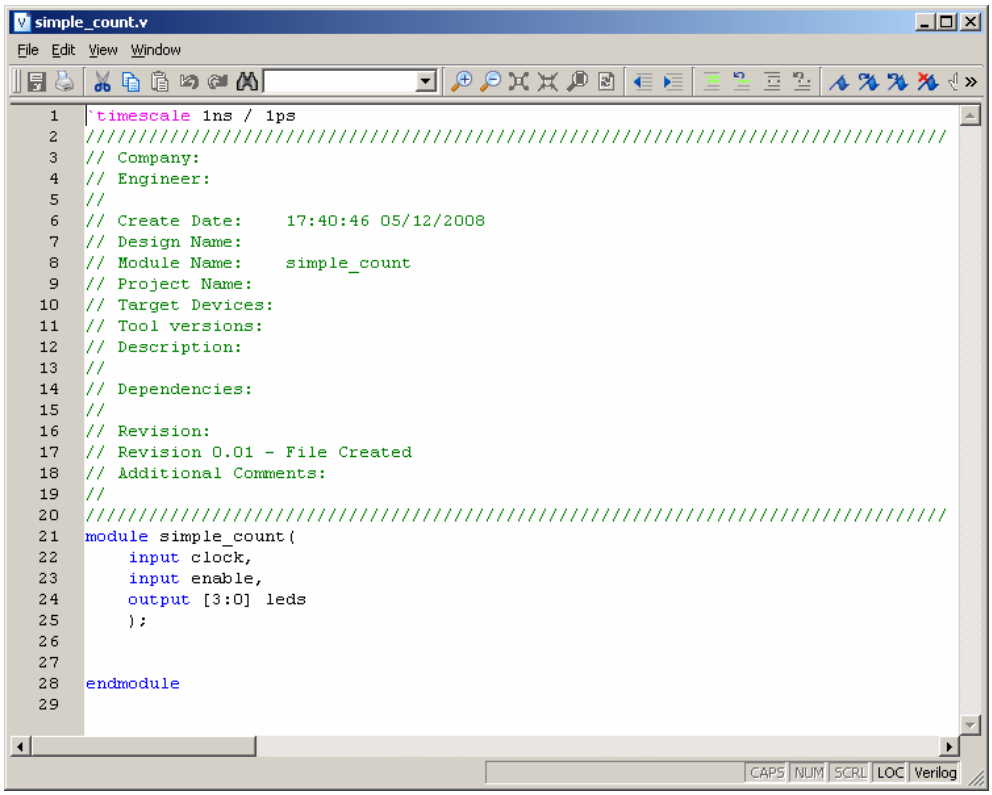
4) Verify the **Add to project** is checked.
5) Click **Next**
6) Enter the input and output signals for your simple_count design by completing the port names and information as shown below. Note that the **leds** port is a 4-bit output bus.



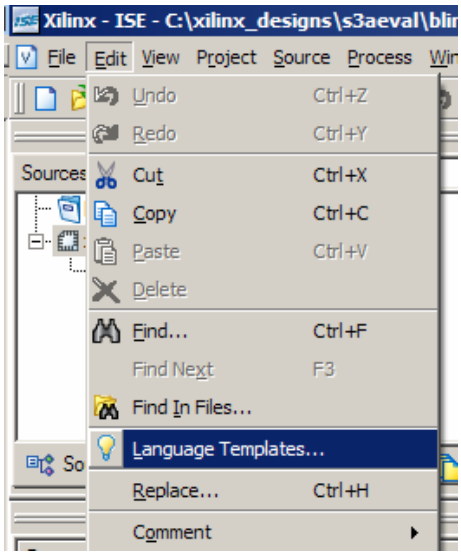
7) Click **Next**, then **Finish**.

ISE automatically creates your **simple_count.v** Verilog source file and displays it as a tab in the Workspace area on the right of Project Navigator. This source file defines the module inputs and outputs only. Your next step is to add the behavioral description of the simple_count module. If you don't have a lot of experience with HDLs (Verilog or VHDL), there is no need to worry. ISE

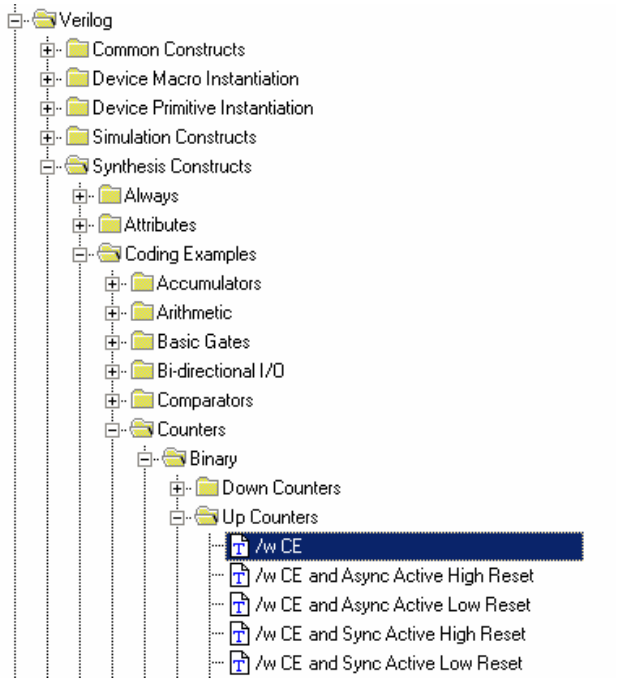
contains Language Templates with many practical coding examples that you can easily use as a starting point for creating your design.



8) Place the cursor on the line below the " **);** " and left-click to place a holding spot for adding new text.
9) Open the Language Templates by selecting **Edit → Language Templates…**



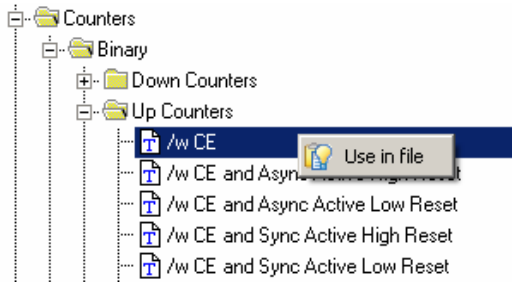
10) Using the "**+**" symbol, expand the Verilog template list so that you can view the **Binary → Up Counters** examples as shown.



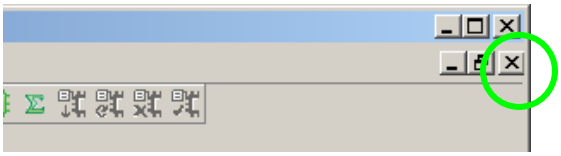
11) Left-click on the first up counter that specifies a Count Enable (CE) function, and review the example source code for this function in the window to the right.

```
reg [<upper>:0] <reg_name>;

always @(posedge <clock>)
   if (<clock_enable>)
      <reg_name> <= <reg_name> + 1;
```

12) Insert this source into your simple_count module by right-clicking on the /W CE counter and selecting **Use In File**. The example code will be added to your source.



13) Click on the Language Template tab to again view the template window. Close the template window by clicking the “X” in the upper right corner of the window.



The template provides a starting point for your counter design. You will need to make some minor edits to the simple_count.v file to make it specific to your design requirements.

14) Edit your simple_count.v file to match the example shown below. In general the changes required are:

- Add module specific signal and port names
- Define an internal 4-bit register called count and initialize it to 0
- Add the "~" symbol in "`if (~enable)`" to denote an active low enable signal
- Assign the leds to the same value as the internal register count

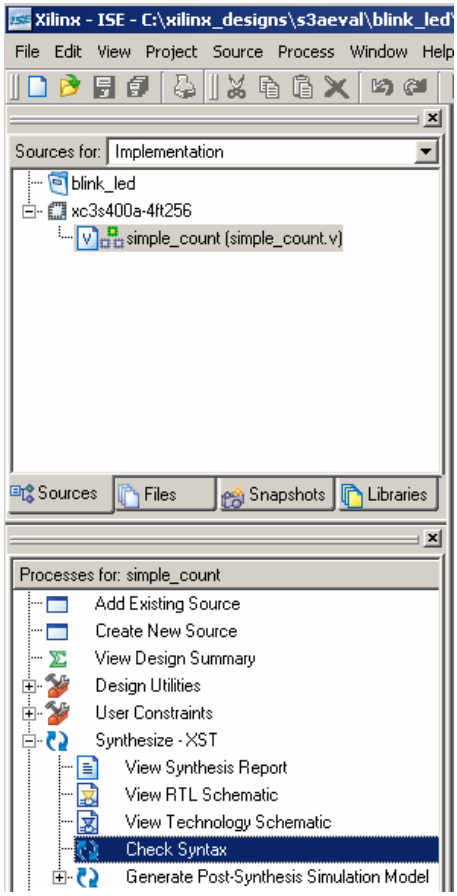
```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    11:02:16 05/18/2008
7  // Design Name:
8  // Module Name:    simple_count
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module simple_count(
22     input clock,
23     input enable,
24     output [3:0] leds
25     );
26
27    reg [3:0] count = 0;
28
29    always @(posedge clock)
30       if (~enable)
31          count <= count + 1;
32
33    assign leds[3:0] = count[3:0];
34
35 endmodule
36
```

15) You have now completed the top level source file for your simple count design. Save the file by selecting **File → Save**.

To be sure there are no errors or typos you can verify the source file by checking the syntax.

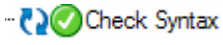
16) Verify that **Implementation** is selected from the drop-down list in the Sources window.

17) Select the **simple_count.v** design in the Sources window to display the related processes in the Process window.
18) Click the "**+**" next to the Synthesize-XST process to expand the process group.
19) Double-click on the **Check Syntax** process.



You can check for errors in the Console tab of the Transcript window along the bottom of the Project Navigator window. Correct any errors in your source file and repeat the **Check Syntax** process.

20) A green check mark signifies that there are no errors.

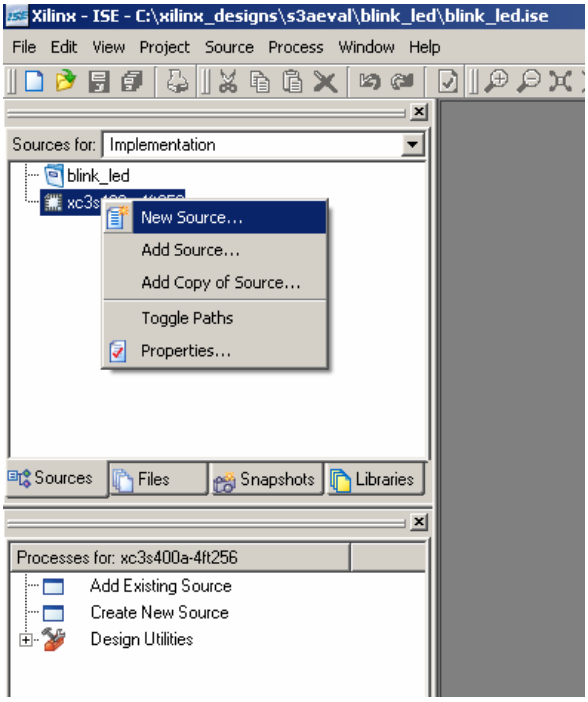


21) Close the **simple_count.v** file.

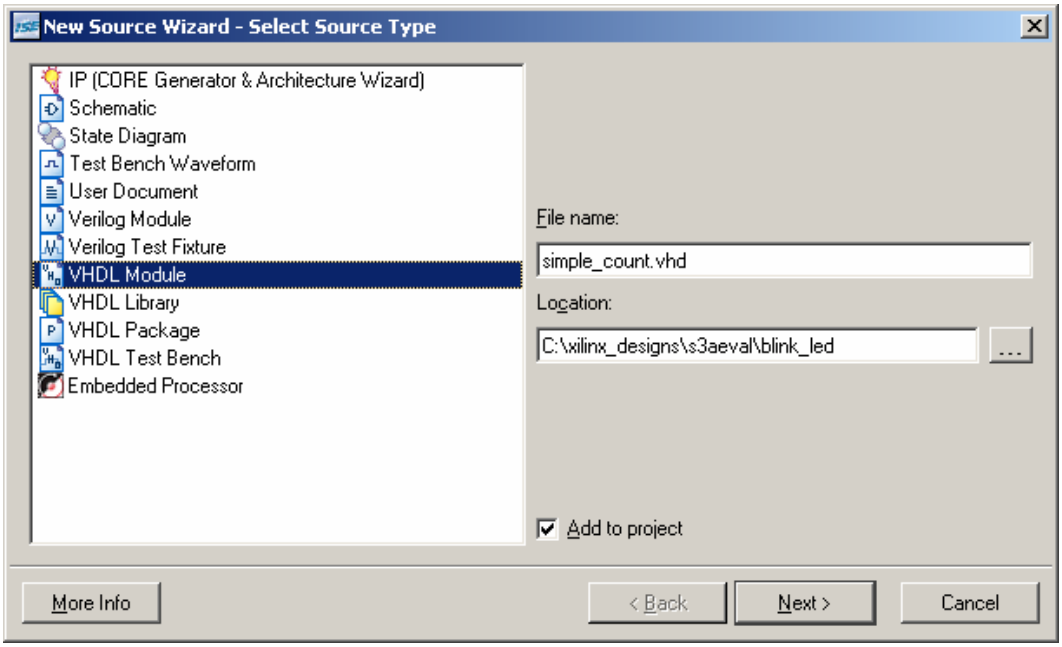
Creating a VHDL Source

Create the top level VHDL source file for the project as follows:

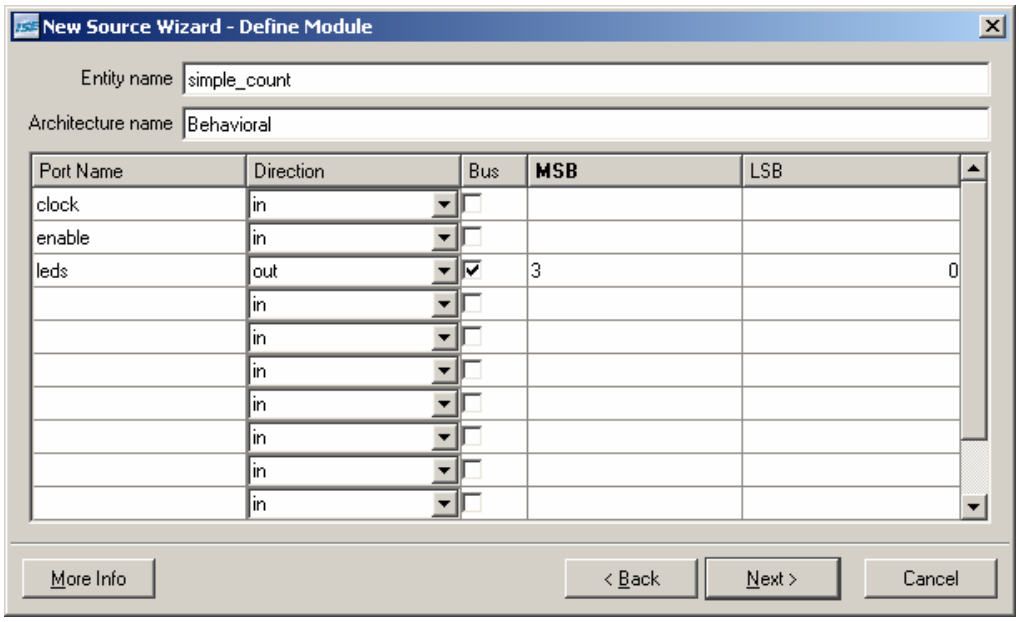
1) In the Sources window, right-click on the **xc3s400a-4ft256** device to pull up the menu and select **New Source…**



2) Select **VHDL Module** as the source type in the New Source dialog box.
3) Type in the module name **simple_count.vhd** as shown.



4) Verify the **Add to project** is checked.
5) Click **Next**
6) Enter the input and output signals for your Blink design by completing the port names and information as shown below. Note that the **leds** port is a 4-bit output bus.

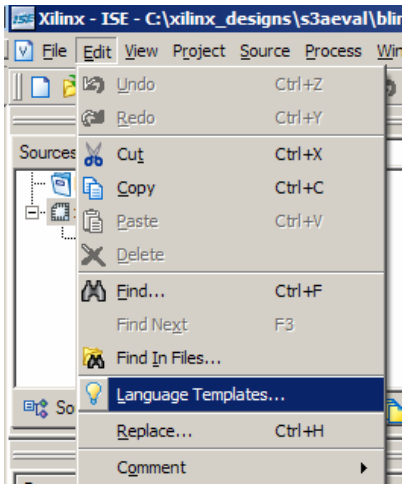


7) Click **Next**, then **Finish**.

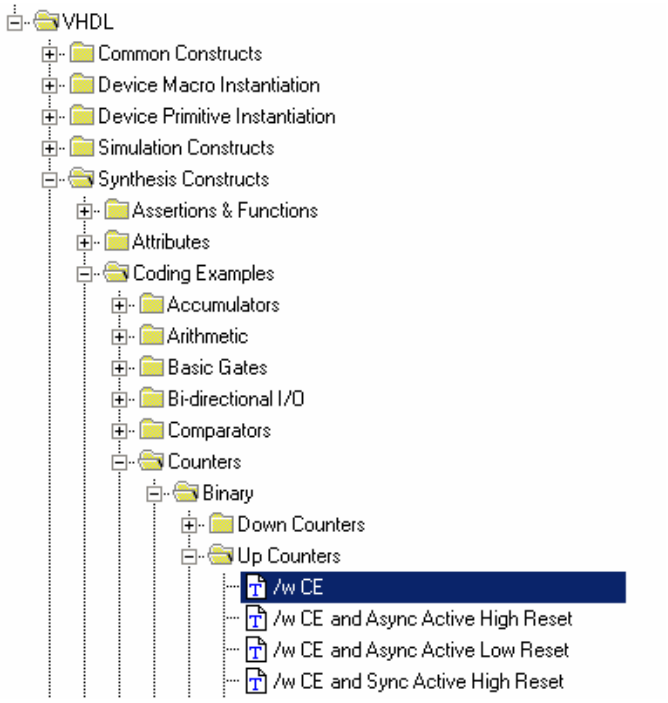
ISE automatically creates your **simple_count.vhd** VHDL source file and displays it as a tab in the Workspace area on the right of Project Navigator. This source file defines the module inputs and outputs only. Your next step is to add the behavioral description of the simple_count module. If you don't have a lot of experience with HDLs (Verilog or VHDL), there is no need to worry. ISE contains Language Templates with many practical coding examples that you can easily use as a starting point for creating your design.

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    21:16:19 05/25/2008
6  -- Design Name:
7  -- Module Name:    simple_count - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity simple_count is
31     Port ( clock : in  STD_LOGIC;
32            enable : in  STD_LOGIC;
33            leds : out  STD_LOGIC_VECTOR (3 downto 0));
34 end simple_count;
35
36 architecture Behavioral of simple_count is
37
38 begin
39
40 end Behavioral;
41
42
```

8) Place the cursor on the line below the "`begin`" statement and left-click to place a holding spot for adding new text.
9) Open the Language Templates by selecting **Edit → Language Templates…**



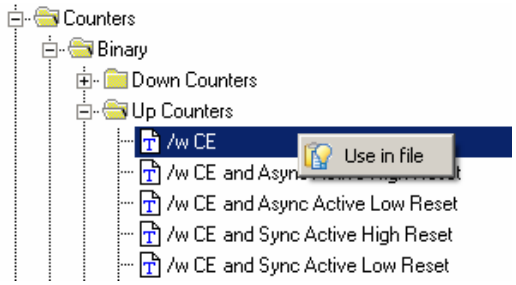
10) Using the "**+**" symbol, expand the VHDL template list so that you can view the **Binary → Up Counters** examples as shown.



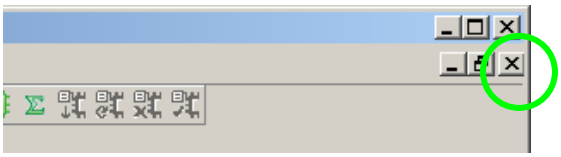
11) Left-click on the top counter that specifies a Count Enable (CE) function, and review the example source code for this function in the window to the right.

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <clock_enable>='1' then
         <count> <= <count> + 1;
      end if;
   end if;
end process;
```

12) Insert this source into your simple_count module by right-clicking on the /W CE counter and selecting **Use In File**. The example code will be added to your source.



13) Click on the Language Template tab to again view the template window. Close the template window by clicking the “X” in the upper right corner of the window.



The template provides a starting point for your counter design. You will need to make some minor edits to the **simple_count.vhd** file to make it specific to your design requirements.

14) Edit your simple_count.vhd file to match the example shown below. In general the changes required are:

- Add module specific signal and port names
- Define an internal 4-bit register called count and initialize it to 0
- Assign the leds to the same value as the internal register count
- Change the polarity of the enable input. When the enable switch is pressed it outputs a “1” to the FPGA, which should disable the counter. Normally, the enable switch is outputting a “0” which should allow the counter to count up.

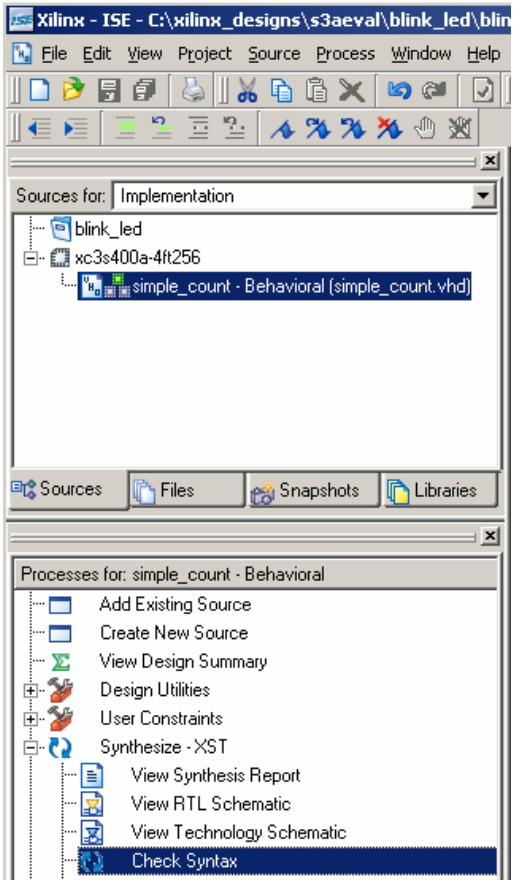
```
 5  -- Create Date:    21:16:19 05/25/2008
 6  -- Design Name:
 7  -- Module Name:    simple_count - Behavioral
 8  -- Project Name:
 9  -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity simple_count is
31      Port ( clock : in  STD_LOGIC;
32             enable : in  STD_LOGIC;
33             leds : out  STD_LOGIC_VECTOR (3 downto 0));
34  end simple_count;
35
36  architecture Behavioral of simple_count is
37
38  signal count: std_logic_vector (3 downto 0) := "0000";
39
40  begin
41  process (clock)
42  begin
43     if clock='1' and clock'event then
44        if enable='0' then
45           count <= count + 1;
46        end if;
47     end if;
48  end process;
49
50  leds <= count(3 downto 0);
51
52  end Behavioral;
```

15) You have now completed the top level source file for your blink led design. Save the file by selecting **File → Save**.

To be sure there are no errors or typos, you can verify the source file by checking the syntax.

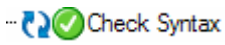
16) Verify that **Implementation** is selected from the drop-down list in the Sources window.

17) Select the **simple_count.vhd** design in the Sources window to display the related processes in the Process window.
18) Click the "**+**" next to the Synthesize-XST process to expand the process group.
19) Double-click on the **Check Syntax** process.



You can check for errors in the Console tab of the Transcript window along the bottom of the Project Navigator window. Correct any errors in your source file.

20) A green check mark signifies that there are no errors.



21) Close the **simple_count.vhd** file.

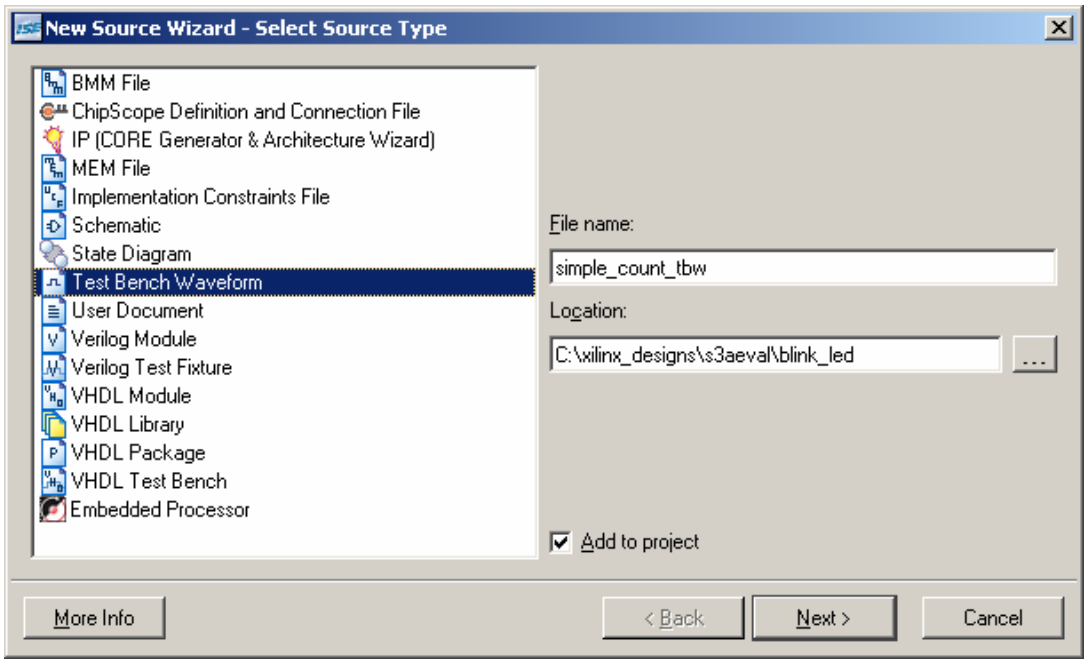
Behavioral Simulation

Simulating your design's behavior is an important step that can save time in the long run. The ISE WebPACK tools include a built-in simulator called ISIM. The WebPACK version of ISIM can be used to simulate smaller designs, or you can purchase the full ISIM package as an upgrade.

To perform your simulation, you need to create a test bench. A test bench is a description of the input waveforms or stimulus to your design, simulating what your design would see on its inputs so you can see the resulting outputs.

The following steps show you how to create a test bench waveform and run a behavioral simulation on your simple_count top level source file.

1) Right-click on the **simple_count** HDL file in the sources window.
2) Create a new test bench source by selecting **New Source…**.
3) In the New Source Wizard, select the **Test Bench Waveform** as the source type, and enter **simple_count_tbw** in the file name field.



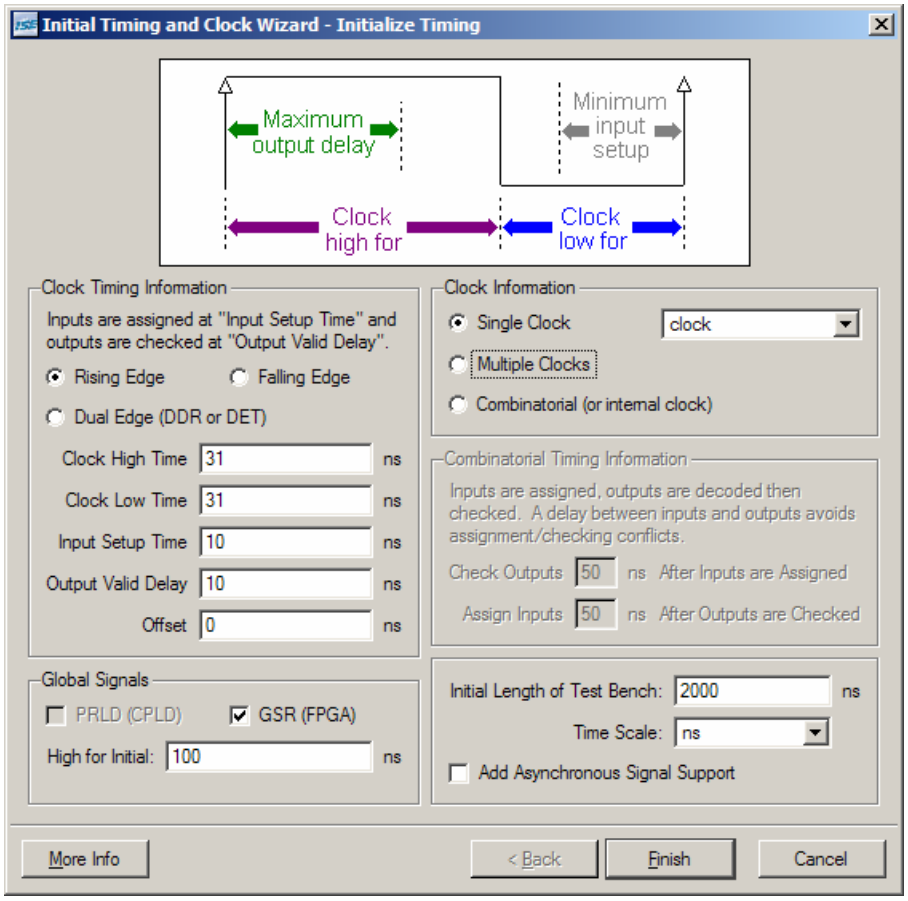
4) Click **Next**.
5) The Associated Source window shows that the source will be added to the project, associated to the **simple_count** top level source file. Click **Next**.
6) Click **Finish**.

The Timing and Clock Wizard window appears next. This wizard provides an easy way to define the 16 MHz clock that will drive the binary counter in your design. A 16 MHz clock has a period equal to 62.5 ns. For the purpose of the behavioral simulation, you can define the clock high period to be approximately half this time, or 31 ns. The clock low time can also be specified at 31 ns.

Inputs to the simple counter should be valid at least 10 ns prior to the clock rising edge, and outputs from the counter should be valid within 10 ns after the rising edge of clock. These are somewhat arbitrary values and do not have any significant effect on the behavioral simulation results. Set the **Offset** to 0 since you will be using the GSR (Global Set/Reset) function for an initial 100 ns of the simulation. For additional information on these parameters, click on the **More Info** button in the lower left corner of the Clock Wizard window.

You will want to run the simulation for 2000 ns, so set the Initial Length of Test Bench to **2000 ns**.

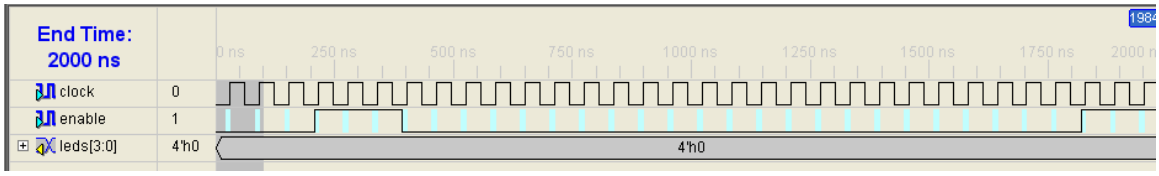
Verify that your wizard settings are the same as that shown in the example below.



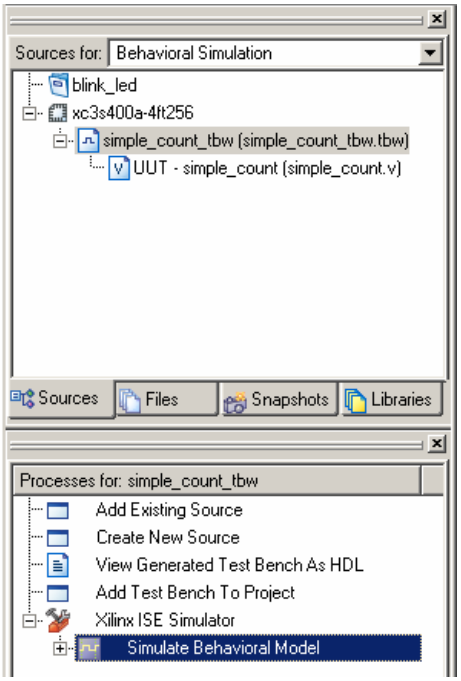
7) Click **Finish**.

A waveform display will appear showing the **clock**, **enable**, and **led** signals. The blue shaded areas that precede the rising clock edge correspond to the Input setup Time (10 ns) that you set in the Timing wizard.

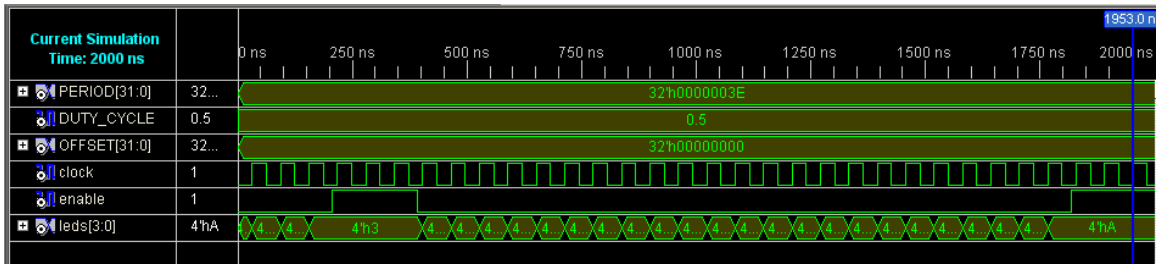
8) Define the timing of the **enable** signal by first clicking on the enable waveform at approximately **210 ns** to toggle the enable signal high. This will disable the counter.
9) Click at **400 ns** to toggle the enable low and continue the count
10) Click at **1800 ns** to toggle it high.
11) Verify your waveform matches the figure below.

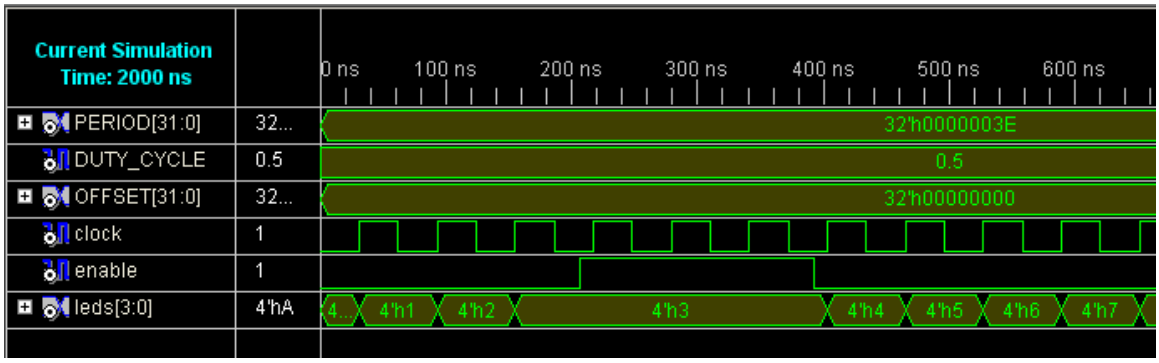


12) Save the waveform.
13) Verify that **Behavioral Simulation** and **simple_count_tbw** are selected in the Sources window.



14) In the processes tab, click the "**+**" to expand the Xilinx ISE Simulator processes and double-click on the **Simulate Behavioral Model** process. This runs the simulator using your **simple_count_tbw** testbench.
15) To view the simulation results, select the **Simulation** tab and zoom in to see the leds count as expected.





Note: You can ignore the PERIOD, DUTY_CYCLE, and OFFSET signals.

16) Close the simulation view. If you are prompted with the following message, “You have an active simulation open. Are you sure you want to close it?”, click **Yes**.

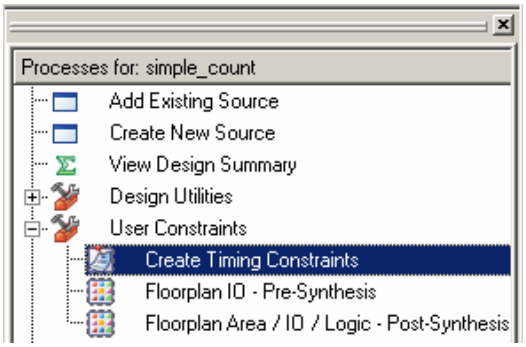
Design simulation is an important step in the design process that often gets over-looked. Spending some simulation time up front can often pay for itself in saving time during hardware debug. For more information on the many capabilities and simulation options that ISE contains, use the built-in ISE help information at **Help > Help Topics > FPGA Design > Simulation.**

Adding Timing Constraints

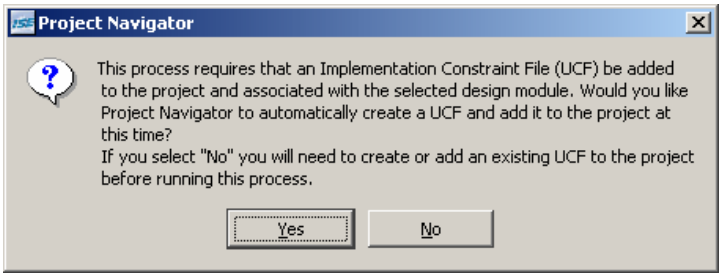
The ISE tools implement your design based on specific timing parameters, or constraints that you provide it. If you do not provide ISE with any timing specifications for your design, the tool will implement the design to the best of its ability. Often this is adequate, however, it’s better to provide some level of timing constraints for the design so you can be sure the actual implementation will meeting your timing needs.

For the simple_count design, there is a 16 MHz input clock, which specifies that the design implementation must work at that clock speed or higher. To guide the ISE implementation tools into meeting this specification, you need to provide a clock period timing constraint for the 16 MHz clock. The ISE tools provide a timing constraints editor to assist in creating these specifications.

1) Verify that the Sources option is showing **Implementation**.
2) Expand the User Constraints processes by click on the “**+**”.
3) Double-click on the **Create Timing Constraints** process.

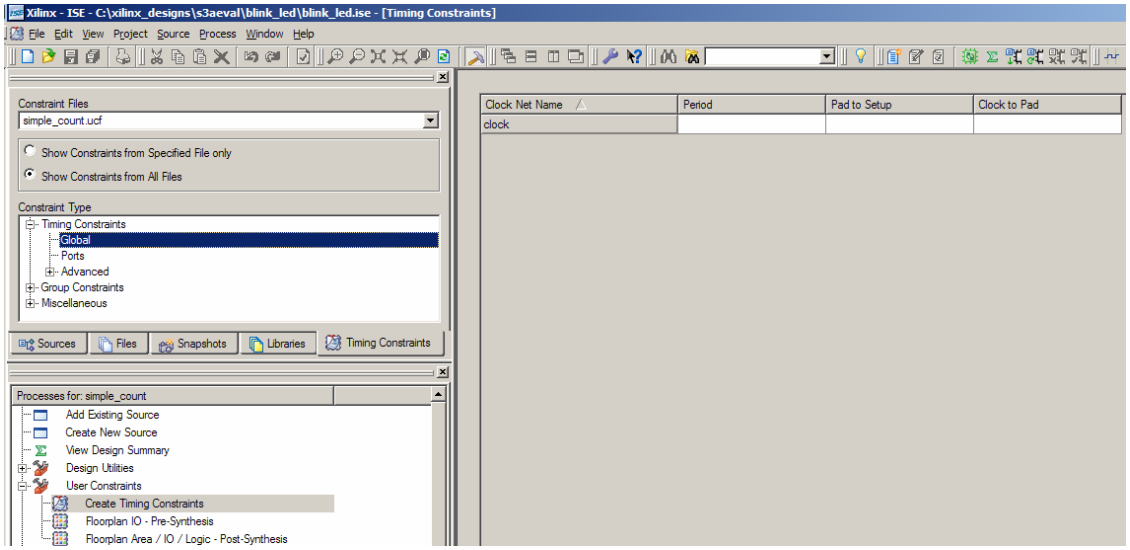


ISE runs the Synthesis and Translate steps and automatically creates a User Constraints File (UCF). You will be prompted with the following message:

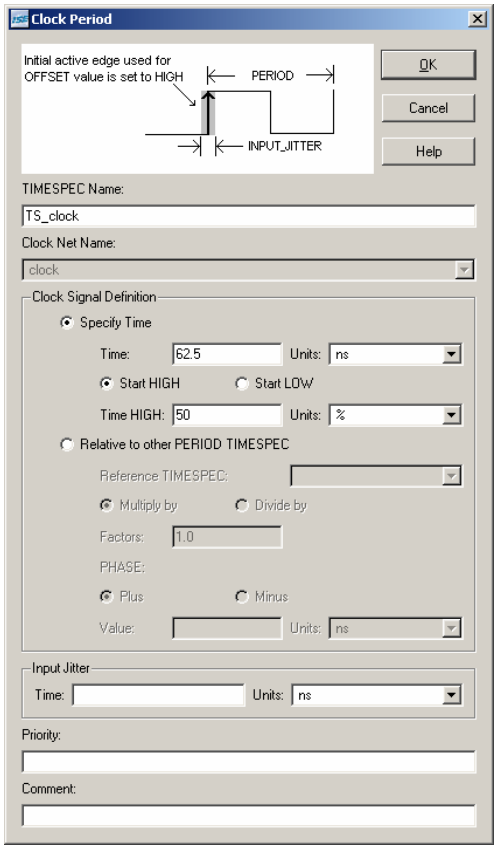


4) Click **Yes** to add the UCF file to your project.

The **simple_count.ucf** file is added to your project and is visible in the Sources tab under the top-level simple_count HDL file. The Timing Constraints Editor automatically opens as shown in the following figure.



5) Double-click in the clock **Period** cell and enter **62.5 ns** to define the board input clock frequency of 16 MHz. The 50% duty cycle and other default setting are fine as shown in the following figure.



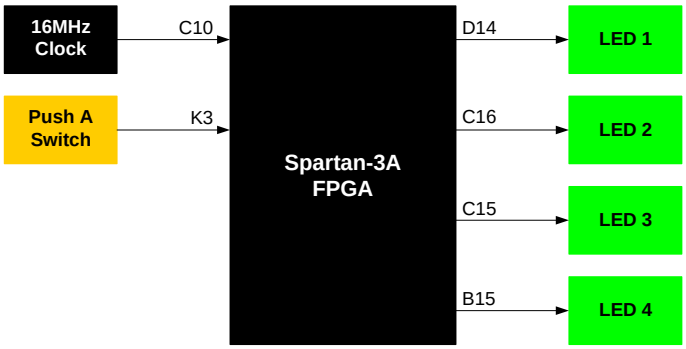
6) Note the TIMESPEC Name of **TS_clock**. ISE is creating a timing specification for the clock signal such that the implementation tools will try to meet the 16 MHz speed requirement.
7) Click **OK**.
8) Save your timing constraints by clicking the **Save** icon .
9) Close the timing constrains editor.

The clock period timing constraint is just one of many different timing constraints that you can place on your FPGA design. For additional information on timing constraints, select **Help > Help Topics > FPGA Design > Constraints** and review the topics covered in this section.

Adding Pin Constraints

Besides timing constraints, pin constrains are also very important in achieving a successful FPGA design. In the case of the Spartan-3A Evaluation board, the I/O pins of the FPGA are already connected to various circuits on the board. As you create your FPGA design, you need to tell the ISE tools what pins these signals are connected to. There are several ways you can enter I/O or pin constraints into your design UCF file. You will use the **Floorplan IO – Pre-Synthesis** process in this tutorial.

For the simple_count example, there are two input signals and four output signals that require pin constraints. They are:



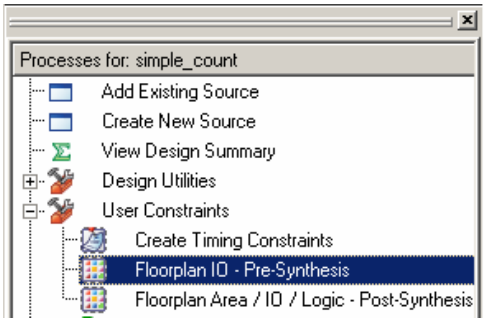
Inputs:

clock	Pin C10	
enable	Pin K3	Normally Low, goes High when pushed

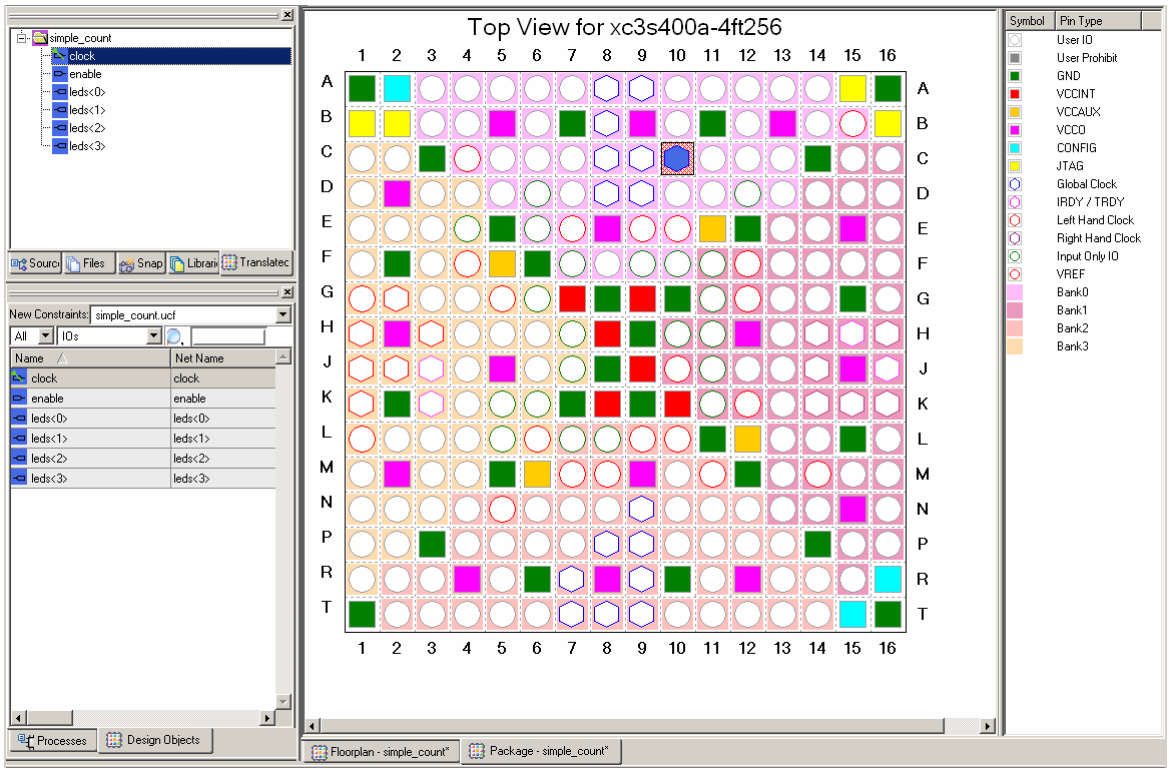
Outputs:

led 1	Pin D14	|
led 2	Pin C16	| "1" turns LED on
led 3	Pin C15	|
led 4	Pin B15	|

1) In the Process window, double click on the **Floorplan IO – Pre-Synthesis** process.



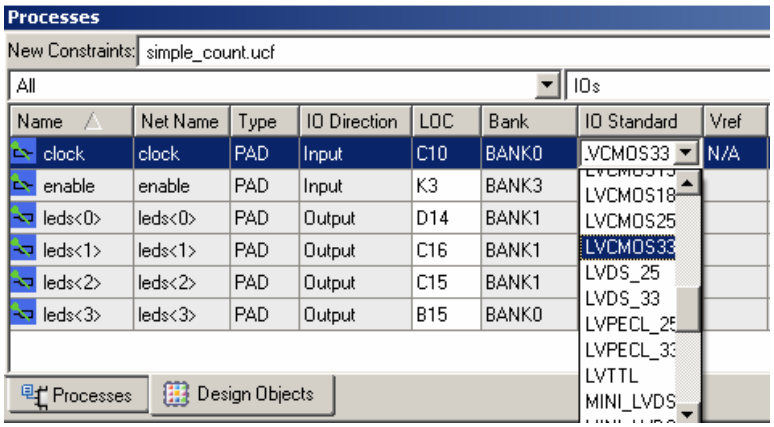
2) If a Pin Ahead message window appears, click **OK**.
3) The center **Package** view tab shows the FPGA pins with the various color and symbol codes depending on the pin features. In the upper left window, the **Translated** netlist tab shows the I/O pins contained in the simple_count design.
4) To place the input and output signals at the specified pin locations, simply select a signal from the list in the upper left Translated window and drag it over to the correct pin location in the Package view. Do this for all the input and output pins listed.



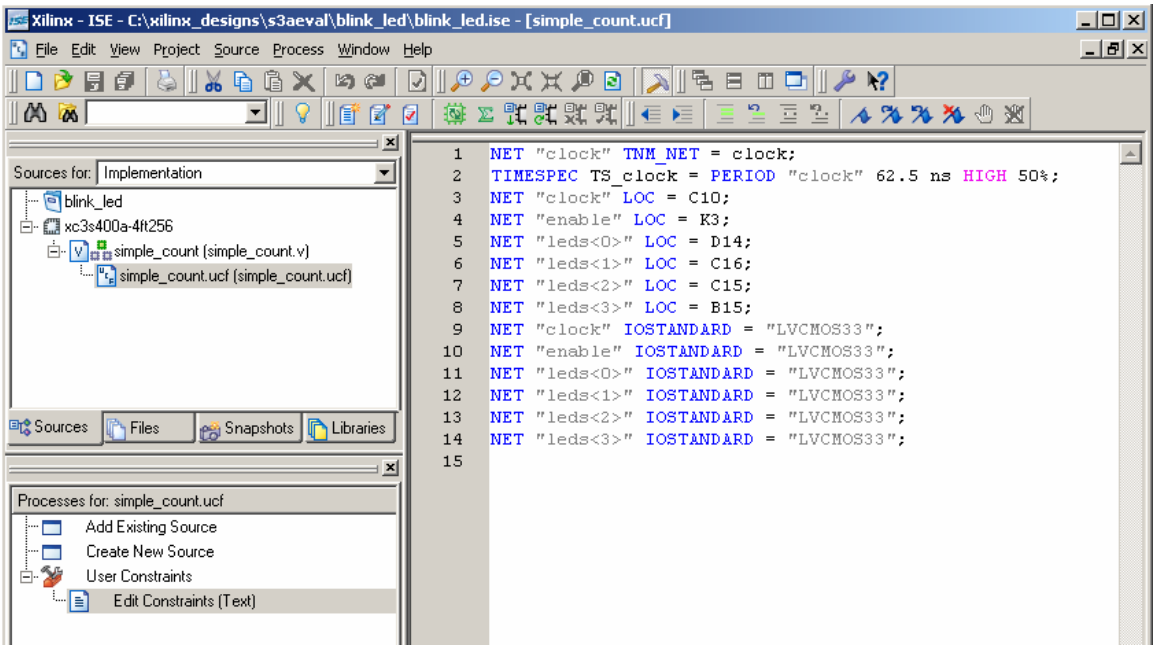
Note that your 16 MHz clock signal goes into one of the 16 Global Clock pins on the FPGA. Clock input signals should connect to global clock pins to utilize the internal global clock buffers inside the FPGA.

5) In the **Design Objects** tab, you should notice the addition of the pin numbers to the **LOC** cells of the corresponding signals. To view more of the Design Objects window, place your cursor along the line dividing the Design Objects tab and the Package window. Drag the window to the right to extend the Design Objects window and shrink the Package window.

6) In addition to the pin location constraints, you will also want to define the I/O standard that each pin should use. In the case of the Spartan-3A Evaluation board, you should set these to LVCMOS33 as shown in the figure below. Click your cursor in the **IO Standard** cell of each pin and select the **LVCMOS33** option from the drop-down menu.



7) Click the **Save** icon  to save your I/O constraints to the UCF file.
8) Close the **Package** and **FloorPlan** tabs.
9) To verify that the timing and pin constraints are accurately added to the user constraints file, select the **simple_count.ucf** file in the Sources window.
10) Double-click on the **Edit Constraints (Text)** option in the Process window and verify the constraints match those shown below.



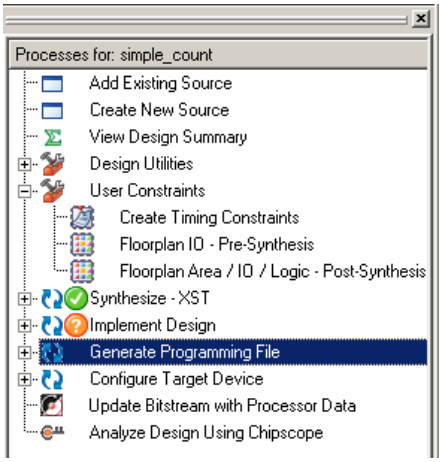
11) Close the constraints file.

Implementing the Design

You are now ready to implement your design. Instead of just running the Implement Design process, you can run the Generate Program File process instead. This will automatically run the Implement Design processed first, and then create the programming file you will need to load into the FPGA.

Because the Spartan-3A Evaluation kit uses a separate low cost USB cable and the AvProg programming utility to program the FPGA, you should not run the Configure Target Device process. If you happen to have a special Xilinx USB JTAG that connects to the J5 header on the Spartan-3A board, then you could run this process.

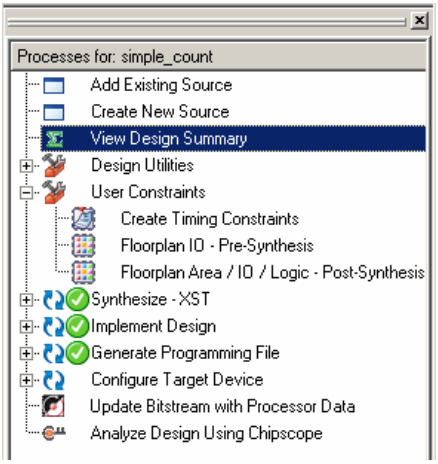
1) Double-click on the **Generate Programming File** process to implement your design.



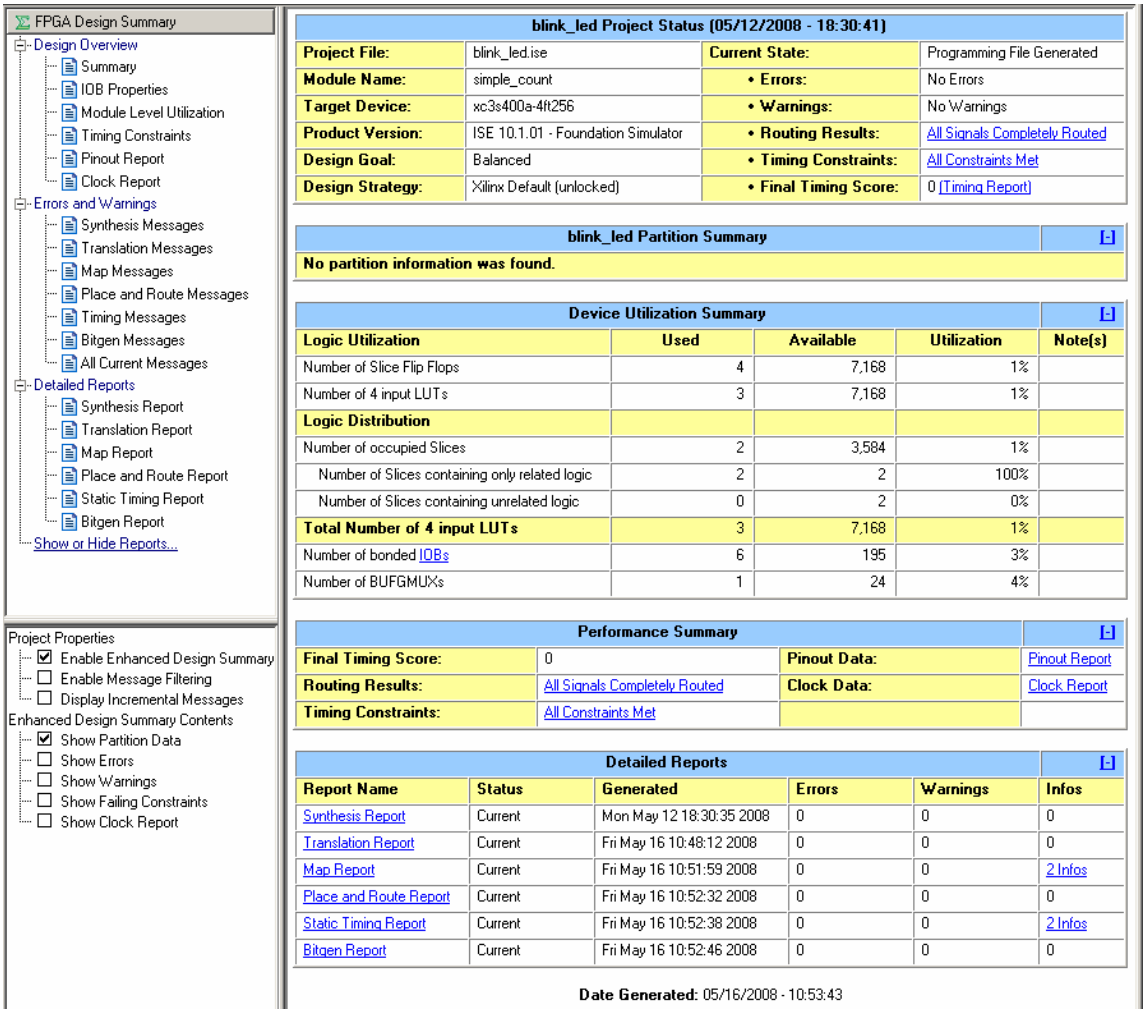
Verifying the Results

Once your design has been implemented, you will want to verify the completed results to make sure the timing and I/O constraints where met and to get a feel for the FPGA design resources required.

1) Double-click on the **View Design Summary** process to view the implementation results.



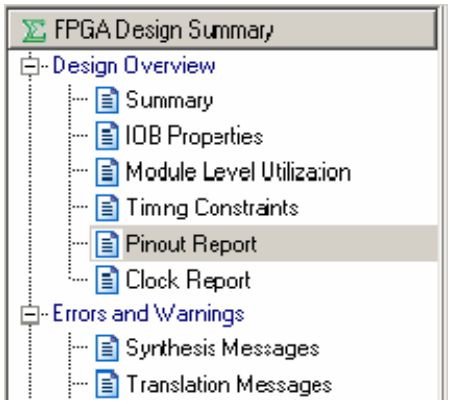
2) This will bring up the **Design Summary** window as shown below.



3) In the upper right section of the Project Status, you can quickly get a feel for the implementation results by seeing there were no Errors or Warnings, all signals routed, and the timing constraints were met.
4) In the Device Utilization Summary section you can see that the design required 4 flip-flops, 6 I/Os, and 1 BUFGMUX. It is always a good sanity check to compare this with what you think the design should be.
5) In the FPGA Design Summary window along the left of the window, click on the **Static Timing Report** under the Detailed Reports section. If you scroll to the bottom of this report, you should see something similar to what is shown below. All timing constraints should be met and the maximum clock frequency for the design should be something much greater than the specified 16 MHz.

```
--------------------------------------------------------------------------------


All constraints were met.


Data Sheet report:
------------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clock
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clock          |    2.121|         |         |         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0

Constraints cover 10 paths, 0 nets, and 13 connections

Design statistics:
   Minimum period:   2.121ns(1)    (Maximum frequency: 471.476MHz)


------------------------------------Footnotes-----------------------------------
1)  The minimum period statistic assumes all single cycle delays.

Analysis completed Fri May 16 10:52:38 2008
--------------------------------------------------------------------------------
```

6) Next, click on the **Pinout Report** and verify that the I/O assignments are what you specified.



C9		DIFFMTB	IO_L10P_0/GCLK6	UNUSED		0				
C10	clock	IBUF	IO_L09P_0/GCLK4	INPUT	LVCMOS33	0				IBUF
C11		DIFFMTB	IO_L07P_0	UNUSED		0				
C12		DIFFMTB	IO_L03P_0	UNUSED		0				
C13		DIFFSTB	IO_L01N_0	UNUSED		0				
C14			GND							
C15	leds<2>	IOB	IO_L24N_1/A25	OUTPUT	LVCMOS33	1	12	SLOW	NONE**	
C16	leds<1>	IOB	IO_L24P_1/A24	OUTPUT	LVCMOS33	1	12	SLOW	NONE**	
D1		DIFFMLB	IO_L03P_2	UNUSED		2				

As you can see from the Design Summary window, there is an abundance of information available for the implemented design. For this simple design example, there is not much need to explore them all in detail. However, if you have a design that is not working, they can prove invaluable in helping to debug a problem.

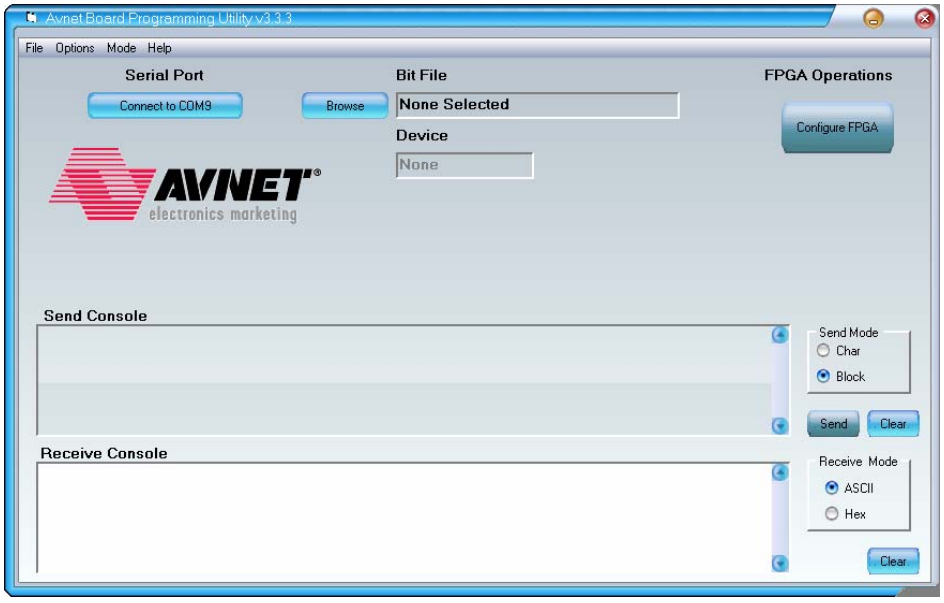
7) Close the Design Summary window when you are done reviewing the different design reports.

Configuring the FPGA

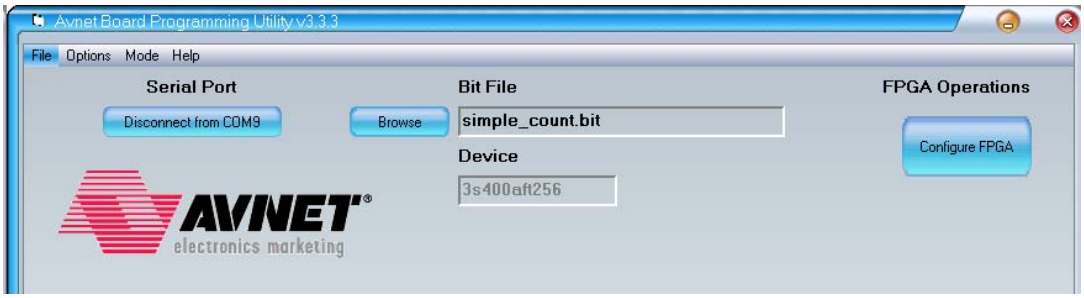
The output of the Generate Programming File process is a configuration file, or a bit file. For the simple_count design, the ISE tools created a **simple_count.bit** file that you will use to configure the FPGA.

Since the Spartan-3A Evaluation board has its own FPGA programming capability built-in, you will use the Avnet utility called **AvProg** to program the FPGA with the simple_count.bit file created in ISE.

1) Be sure the board jumper settings are as specified in the Hardware setup section of this tutorial.
2) Connect the USB cable to the Spartan-3A Evaluation board (Connector P1) and then connect to the USB port of your PC.
3) As a minimum, the 5V power LED (D1) should illuminate.
4) Start the AvProg programming utility with **Start > All Programs > Avnet > AvProg**.



5) Click the **Connect to COMx** button to establish a communications link between AvProg and the Spartan-3A board. The actual COM port number can vary depending on how Windows assigns the USB port to a specific serial port. AvProg automatically detects the correct COM port number assigned to the Spartan-3A Evaluation board.
6) Click on the **Browse** button to select the **simple_count.bit** file from your design. Browse to the directory of our ISE project and select the bit file.



7) AvProg reads the bit file and displays the FPGA information contained in the bit file under the Device window. Verify that this is the **3s400aft256**.
8) Click the **Configure Device** button to program the FPGA.
9) Click **Yes** when asked if the bit file is for the 3s400aft256 device.
10) AvProg will download the **simple_count.bit** file to the FPGA and once configured the blue LED should turn on, signifying that the FPGA is configured. The Receive Console window should display "**FPGA programmed successfully!**" message when complete.
11) The four LEDs should be counting. However, since the counter is running at 16 MHz, the LEDs are actually toggling so fast, that you cannot see the counting and they all appear as if they are on.
12) Place you finger on the **Push_A** CapSense switch. The LEDs should stop counting and hold a fixed value. Release your finger and then press again. The value should change. Given the high speed of the counter, your random stops of the counter are all you can expect.

Obviously, having the LEDs display the counter values at a 16 MHz rate is not very practical. It would be better if we could slow this down to something that the human eye could perceive. An easy way to do this is to increase the counter size to something like a 26-bit counter, and connect the LEDs to the upper 4-bits of this counter. A quick modification to the design should fix the problem.

Modifying the Design

1) From the Sources window in ISE, double-click on the top level **simple_count** HDL file.
2) Modify the source code to increase the counter registers from 4-bits to 26-bits. The examples below show the code changes for both the VHDL and Verilog versions.

```
entity simple_count is
    Port ( clock : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           led : out  STD_LOGIC_VECTOR (3 downto 0));
end simple_count;

architecture Behavioral of simple_count is

signal count: std_logic_vector (25 downto 0) := "00000000000000000000000000";

begin
process (clock)
begin
   if clock='1' and clock'event then
      if enable='0' then
         count <= count + 1;
      end if;
   end if;
end process;

led <= count(25 downto 22);

end Behavioral;
```

VHDL Modifications

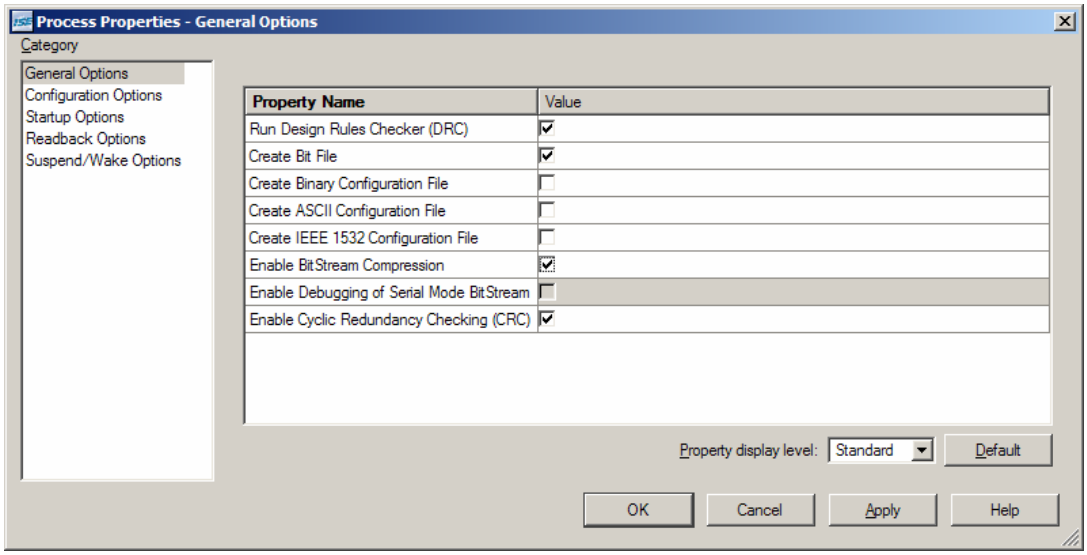
```
module simple_count(
    input clock,
    input enable,
    output [3:0] leds
    );

   reg [25:0] count = 0;

   always @(posedge clock)
      if (~enable)
         count <= count + 1;

   assign leds[3:0] = count[25:22];

endmodule
```

Verilog Modifications

3) Click the **Save** button.

In Spartan-3A FPGAs, it is possible to compress the bit file to reduce the size. This is useful when you are storing the bit file in some external configuration memory since you will not need as much storage space for the FPGA configuration data. To see the size of the previous simple_count.bit file, browse the ISE design directory using Windows Explorer and note the file size. It should be about 200K bytes.

4) To compress the bit stream, right-click on the **Generate Programming File** and select **Properties**.
5) Click on the **Enable Bit Stream Compression** value to select it.



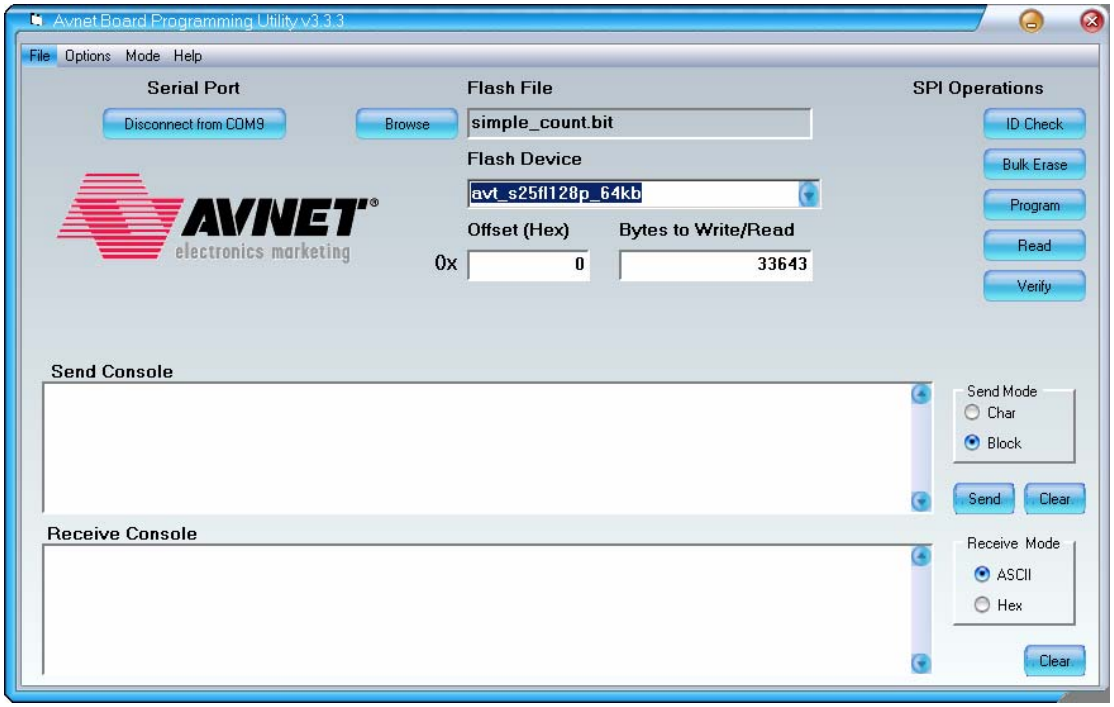
6) Click **OK**.
7) Double-click on the **Generate Programming File** to re-compile the design.
8) When completed, go back to AvProg, re-select the new **simple_count.bit** file and press the **Configure FPGA** button.
9) The new design should now show the counting LEDs at a much slower rate.
10) Notice that the configuration download from AvProg was significantly faster. Browse to the design directory using Windows Explorer and note the files size of the compressed simple_count.bit file. It's about 7x smaller (~30KB).

Programming the SPI Flash

The Spartan-3A Evaluation board contains a 128 Mbit Spansion SPI Flash memory that can be used to store a FPGA bit file. With the SPI memory programmed, that FPGA will configure itself on power-up, assuming the MODE jumpers are set for Master SPI mode.

Your final step in the tutorial will be to program the Spansion SPI memory and verify that the FPGA can configure from it.

1) From AvProg, select **Mode > Program SPI Flash** from the menu items along the top bar.
2) Browse to the same **simple_count.bit** file you used in the last step. Note the file size is displayed in the Bytes to Write/Read cell.
3) Select the Spansion **avt_s25fl128p_64kb** SPI from the Flash Device drop down menu.



4) If you are not sure if the SPI memory is erased or not, be safe and click on the **Bulk Erase** button to erase the SPI flash.
5) Once the SPI is erased, click the **Program** button.
6) When the SPI Flash is done being programmed, close AvProg, unplug the USB cable from the PC, then plug the cable back in.
7) On power-up, the FPGA should configure itself from the Spansion SPI memory and your simple_count design should be functioning.

You have now successfully completed your first design for the Spartan-3A Evaluation kit. There is much more to learn and explore, and the Help menu in ISE is a great place to start. You will also see there are many more features and ways of doing things than what was shown in this tutorial. As you become more familiar with the tools, you will likely find your preferences. In addition, there are other reference designs and tutorials available for this board through the Avnet Reference Design Center at www.em.avnet.com/spartan3a-evl.