Xilinx Tutorial

Spartan III FPGA board
Programming example

By Ali Kaichouhi

Introduction

The goal of this tutorial is to learn you how to program the xilinx spartan III FPGA board. For the programming we use the ISE 6.3.03i (Integrated Synthesis Environment) software(tutorial assumes that the software is already installed and configured on a windows 2000/XP machine). First an introduction of the spartan III FPGA board and a short overview of the Xilinx ISE software are given. Then a complete tutorial of a 0-9 seven segment display counter is given

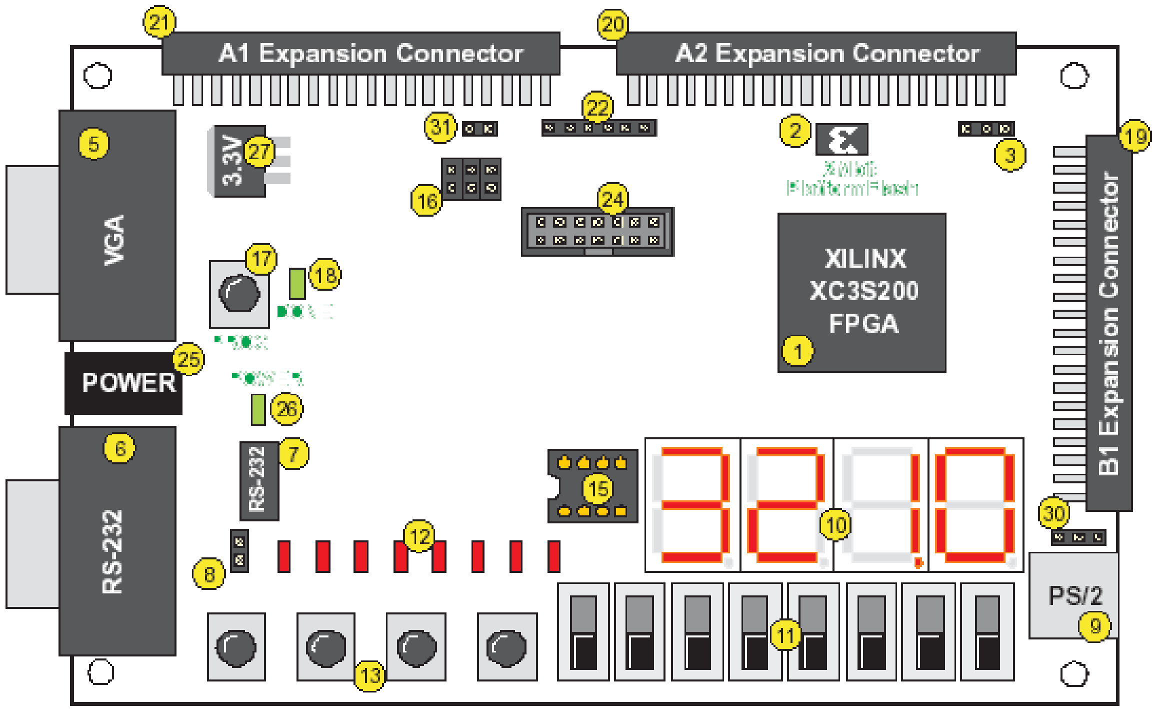


Figure 1 The topside view of Xilinx Spartan III Board

The core part of the spartan III board (see fig. 1) is the Xilinx Spartan XC3S200 chip ❶ . It consists of 4320 logic cell equivalents, twelve 18K-bit block RAMs(216K bits), twelve 18x18 hardware multipliers, four Digital Clock Managers(DCMs), and up to 173 user-defined I/O signals. These I/O signals are directly connected to the expansion connectors A1, A2, B1. Connector A1 has 32 user I/O pins, connector A2 and B1 have 34 user I/O pins each.

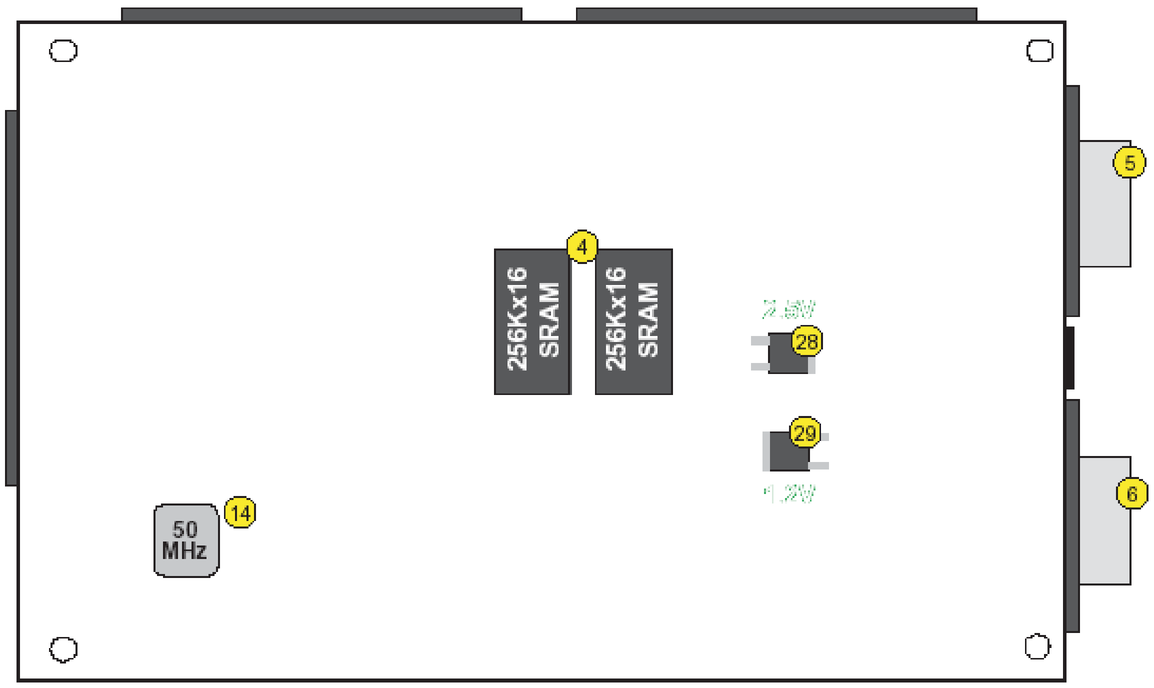


Figure 2 The bottom side view of Xilinx Spartan FPGA Board

You can connect any digital interface to these connectors that provide signals for the XC3S200 chip. For example, sensor interface, servo motor signals etc. Pin 1 on each connector is GND. Similarly, pin2 is always +5VDC output from the switch power supply. Pin 3 is always the output from the +3.3V DC. The board has a 2Mbit XCF02S programmable configuration flash PROM ❷. 1Mbit is reserved to store non-volatile program data. It can be programmed using a parallel port to JTAG cable ❷❸. The JTAG connector end fits directly to the header stake pins ❷❷(See fig. 3). Make sure that the signals at the end of the JTAG cable align with the labels listed on the board.

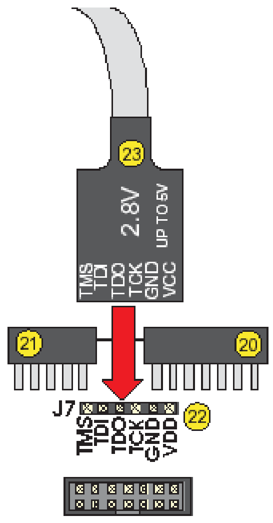
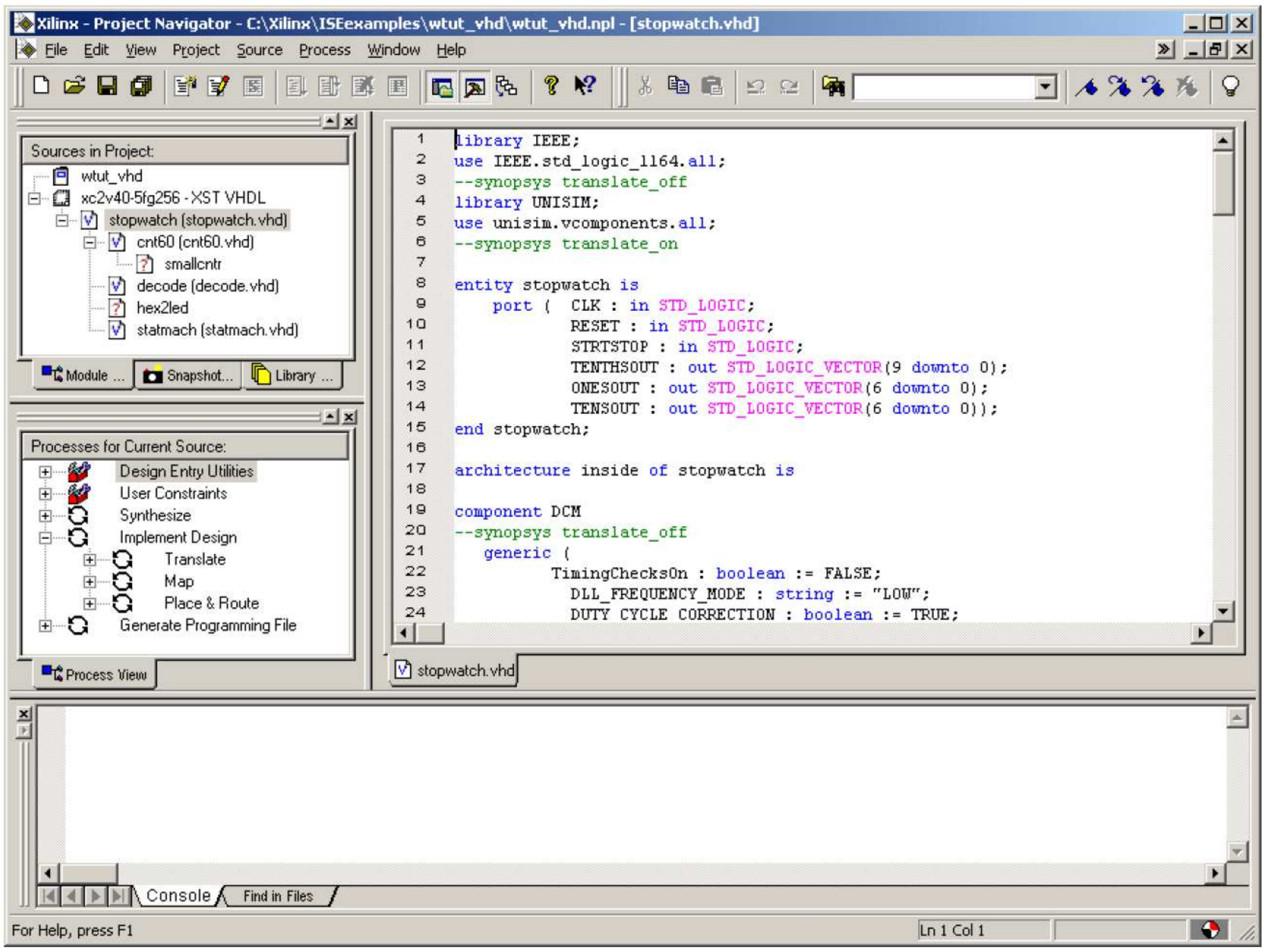


Figure 3 JTAG connector

The other end is connected to the parallel port of your PC. The platform flash has three settings which are controlled by the jumper JP1 ❸. These are Default, Flash read, Disable. Default means that the FPGA boots from Platform Flash. And no additional data storage is available. The Flash read mode means the FPGA boots from Platform Flash, which is permanently enabled. The FPGA can read additional data from Platform Flash. When in disabled setting the Platform Flash is disabled. Other configuration data source provides FPGA boot data. To make these three settings available all three jumpers on the J8 header ❶❻ must be installed. The board has also eight slide switches ❶❶. These switches are connected to an associated FPGA pin. They are not available through the expansion connectors. When in the UP or ON position, a switch connects the FPGA pin to +3.3V, a logic High. In DOWN or OFF position, the switch connects the FPGA pin to ground, a logic Low. The spartan III board has a dedicated 50 Mhz oscillator source ❶❹, which is located at the bottom side(see fig. 2). You can also derive other frequencies from this source using the FPGAs Digital Clock Managers(DCMs). An optional socket for another clock source is provided ❶❺. The spartan III has four momentary-contact push button switches ❶❸. Pressing a push button generates a logic High on the associated FPGA pin. The board has also eight surface-mount LEDs ❶❷. To light an individual LED, drive the associated FPGA control signal High. The Spartan III board has a four-character seven segment LED display controlled by FPGA user I/O pins. Each digit shares eight common control signals to light individual LED segments. Each digit has a separate anode control input to select the appropriate character segment. And it is also comprised with a VGA port ❺, RS-232 port ❻, PS/2 mouse/keyboard port ❾.



```
library IEEE;
use IEEE.std_logic_1164.all;
--synopsys translate_off
library UNISIM;
use unisim.vcomponents.all;
--synopsys translate_on

entity stopwatch is
    port (  CLK : in STD_LOGIC;
            RESET : in STD_LOGIC;
            STRTSTOP : in STD_LOGIC;
            TENTHSOUT : out STD_LOGIC_VECTOR(9 downto 0);
            ONESOUT : out STD_LOGIC_VECTOR(6 downto 0);
            TENSOUT : out STD_LOGIC_VECTOR(6 downto 0));
end stopwatch;

architecture inside of stopwatch is

component DCM
--synopsys translate_off
   generic (
          TimingChecksOn : boolean := FALSE;
            DLL_FREQUENCY_MODE : string := "LOW";
            DUTY CYCLE CORRECTION : boolean := TRUE;
```

Figure 4 The ISE interface

Figure 4 shows the ISE interface. The interface is divided in four frames. The top left frame is *Sources in Project*. This window shows the name of your project, the specified part type and the design flow/synthesis tool, and design source files. With the Snapshot tab you can view the snapshots taken. A snapshot is a copy of all the files of your project including synthesis and simulation subdirectories(it is stored as read only). To continue from the taken snapshot you have to restore it. This capability provides storing different versions of a design. The Library tab shows all the libraries involved in the design project.
The *Processes for Current Source* frame contains the Process View tab. It changes when selecting a different source in the Sources in Project frame. The process Window provides the following functions:

- *Design Entry Utilities*
  Provides access symbol generation, launching the Modelsim simulator, view command line log file, view vhdl instantiation template.
- *User Constraints*
  Provides access to editing location and timing constraints on the chip.
- Synthesis
  Provides access to Check Syntax, View RTL Schematic, and synthesis reports.
- *Implement design*
  Provides access to implementation tools, design flow reports, and point tools.
- *Generate Programming File*
  Provides access to the configuration tools and bitstream generation.

Each Process is dependent on other processes. If for example you run the *Implementation* process then the Project Navigator will also run the *Synthesis* process because it needs to be up to date before the *Implement* process can run.
The frame at the bottom is the *console* window. Here you can see error messages, warnings and information messages. To jump to the error in the VHDL source file just double click on the error red icon. Warnings are displayed with yellow icons.
The last and biggest frame is the text editor. Here you can edit your VHDL/Verilog source files. You can also use the VHDL Bencher(Project --> New Source --> Test Bench Wave Form) to create a graphical test wave form and then generate a VHDL/Verilog test bench file.

In this tutorial an example of a design with VHDL is presented. The purpose of this tutorial is to learn how to program and get familiar with the Spartan III FPGA board. This tutorial assumes that you have some basic knowledge of hardware programming languages.

Design example: 0-9 seven segment display counter

Design information:

This design is a VHDL(hardware description language) based design. It's hierarchical, which means that the top level is a VHDL file that references several other lower-level macros. The lower-level macros are either VHDL modules or CORE Generator modules. The CORE Generator is a design tool that delivers parameterized cores optimized for Xilinx FPGAs. It provides functions such as adders, accumulators, multipliers, counters, and system level building blocks such as filters, transforms, FIFOs, and memories.
The design starts with a template of the design defining the inputs and outputs. From here additional code has to be written both manually and by using the ISE Project Navigator. When the code has been entered and verified for syntax errors, then the design is going to be simulated using the Modelsim simulator. When the results are as expected then the next step is to Assign Package Pins under the User Constraints process. This means drag and drop the used signals in the design on the Package Pins of the FPGA. A constraint file (UCF) will be written that contains these pin assignments.
The next step is to start the XST (Xilinx Synthesis Tool). A netlist file(NGC) is generated. This is a translation of the VHDL design file into gates optimized for the target architecture. The netlist file together with the pin assignment constraint file is then passed to the Implementation Design process. This consists of several synthesis steps like Translation, Mapping and Place & route. At the end, an NGD file is created that is needed for the generation of a configuration data file. This is a bitstream file for downloading to a target device, in this case a PROM device (XCF02S).   After this a PROM file with IMPACT software is created to program the PROM device with iMPACT. The Spartan III board is programmed via the JTAG cable connected to the parallel port of your PC.

The design presented is a simple 0-9 seven segment LED display counter. It's self running, which means that it doesn't need an input from outside. The only input is a clock signal from a crystal oscillator on the FPGA board. It has 13 output datalines.

A list of the I/O and functional blocks of the 7-segment LED display counter is given below.

INPUT	
CLOCK	Clock source is a 50 Mhz crystal oscillator

Tabel 1

OUTPUT	
B[6:0]	FPGA connections to seven segment display(Active Low)
AN[3:0]	Digit Enable(Anode Control) Signals (Active Low)
DP	Connection to display point segment of the selected display(Active Low)

Tabel 2

Functional Block	***Description***
DCM1	Clocking Wizard macro with internal feedback and duty cycle correction
counter1	VHDL-based module running on 1.5625MHz. It has 21 datalines and counts from 0 to 1.5625E6. It sets a Q_THRESH0 threshold signal on high when this maximum value is reached
hex2led	VHDL-based macro. This macro decodes a 4 digit hexadecimal value to a 7-segment LED display format.

Tabel 3

➊ Creating a new project

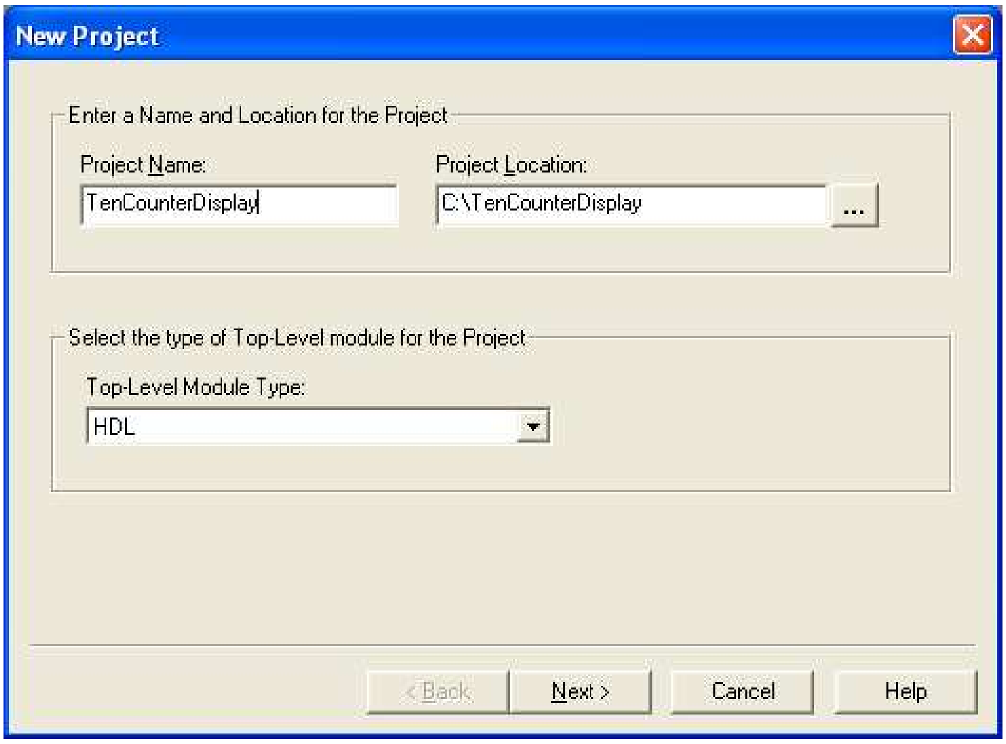
1. Start the Project Navigator by double clicking the shortcut on the desktop or select **Start** ⇨ **Programs** ⇨ **Xilinx ISE 6** ⇨ **Project Navigator**.

Figure 5 Project Navigator shortcut icon

The Xilinx Project Navigator window appears.

Note: You can stop the tutorial at any time and save your work by selecting **File** ⇨ **Save All**.

2. Select **File** ⇨ **New Project**. New Project window appears. Fill in the window as shown in figure 6 . You may choose another Project Name- and location if you want. Click **Next**.


New Project

Enter a Name and Location for the Project

Project Name: TenCounterDisplay

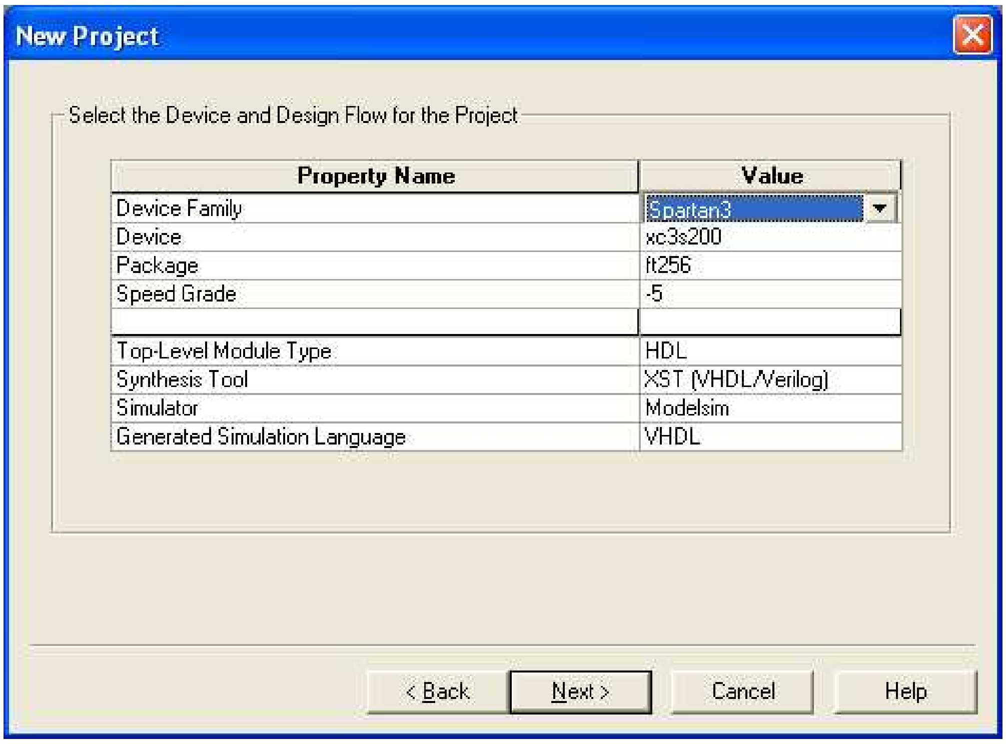
Project Location: C:\TenCounterDisplay

Select the type of Top-Level module for the Project

Top-Level Module Type: HDL

Figure 6 Specifying a new- project and location and Top-Level module type

3. The next **New Project** window appears. Fill in as shown below in figure 7. Click **Next**.

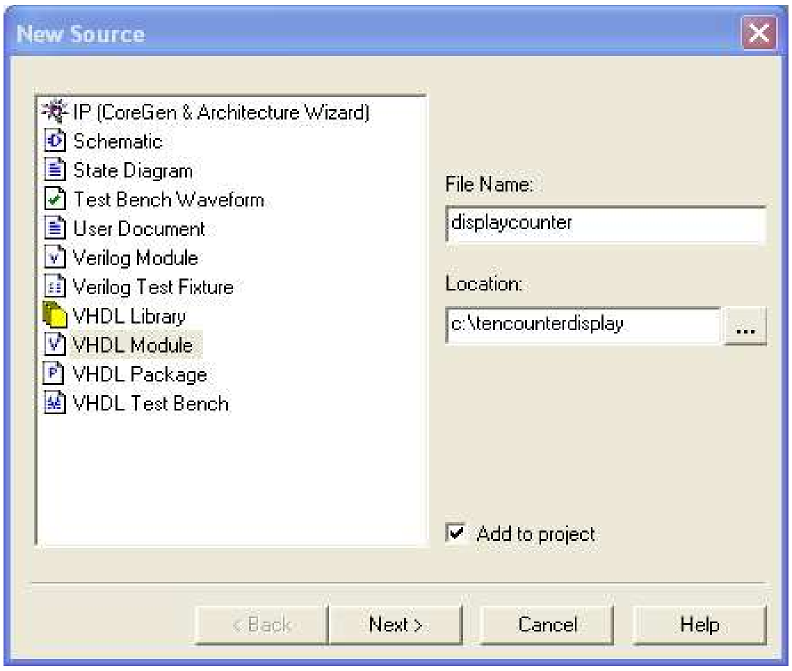


Property Name	Value
Device Family	Spartan3
Device	xc3s200
Package	ft256
Speed Grade	-5
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim
Generated Simulation Language	VHDL

Figure 7 Specifying Synthesis Tool

❷ Creating a VHDL based module

1. On the next window that appears click **New Source**. The **New Source** window appears. Select **VHDL Module** and type displaycounter in the **File Name** field. You may choose another name here. Make sure that **Add to project** is checked. Click **Next**.



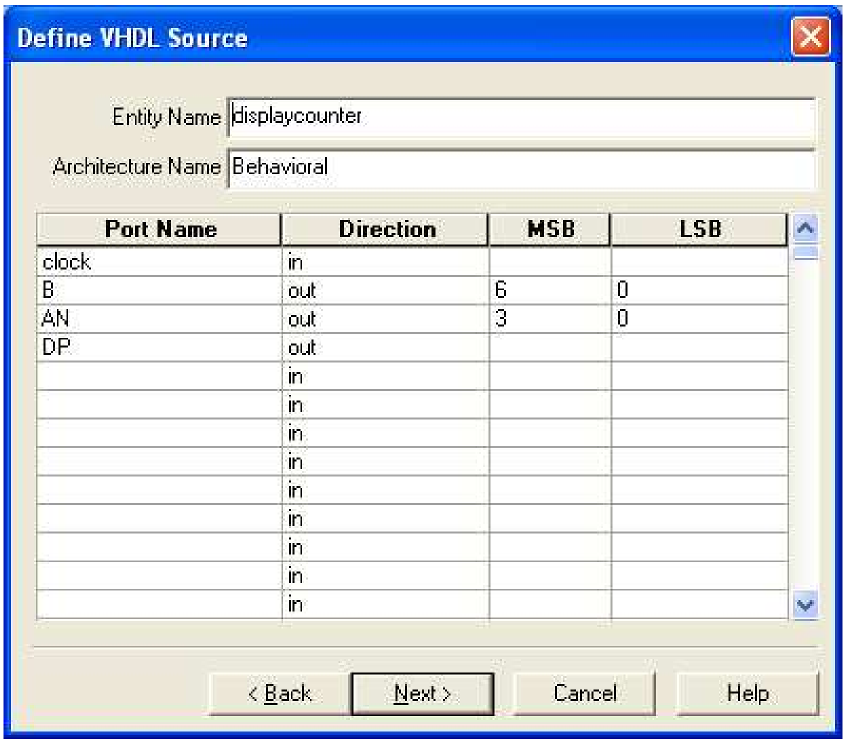
File Name: displaycounter

Location: c:\tencounterdisplay

☑ Add to project

Figure 8 Specifying a new source file

2. The **Define VHDL Source** window appears. Here you define the inputs and outputs. Fill in as shown in figure 9. Click **Next**.



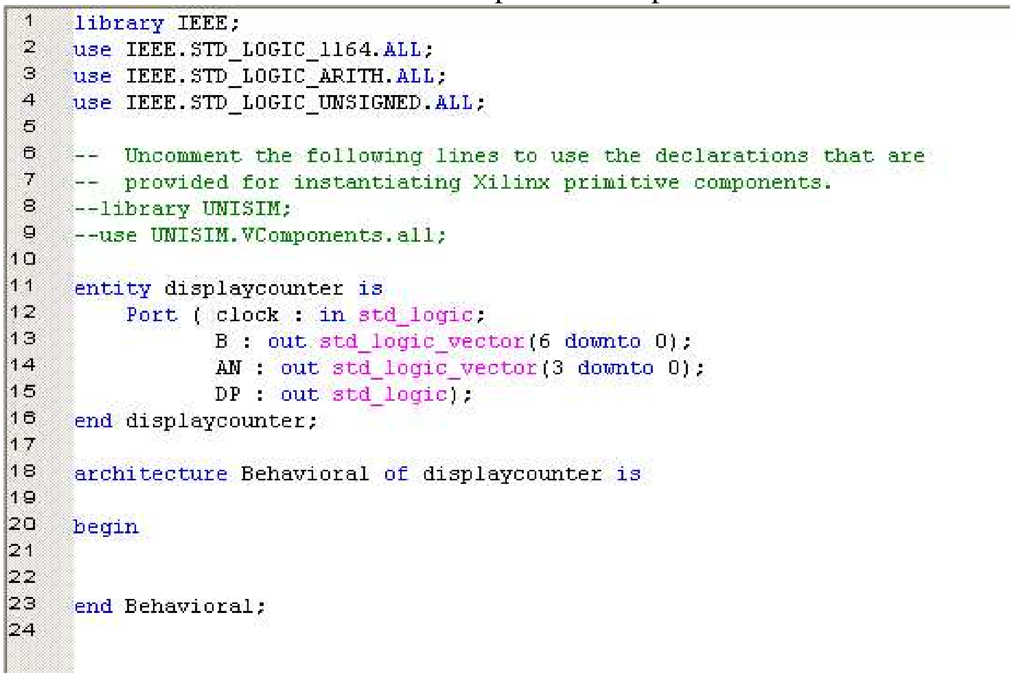
Entity Name: displaycounter

Architecture Name: Behavioral

Port Name	Direction	MSB	LSB
clock	in		
B	out	6	0
AN	out	3	0
DP	out		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Figure 9 Defining inputs and outputs of the VHDL design

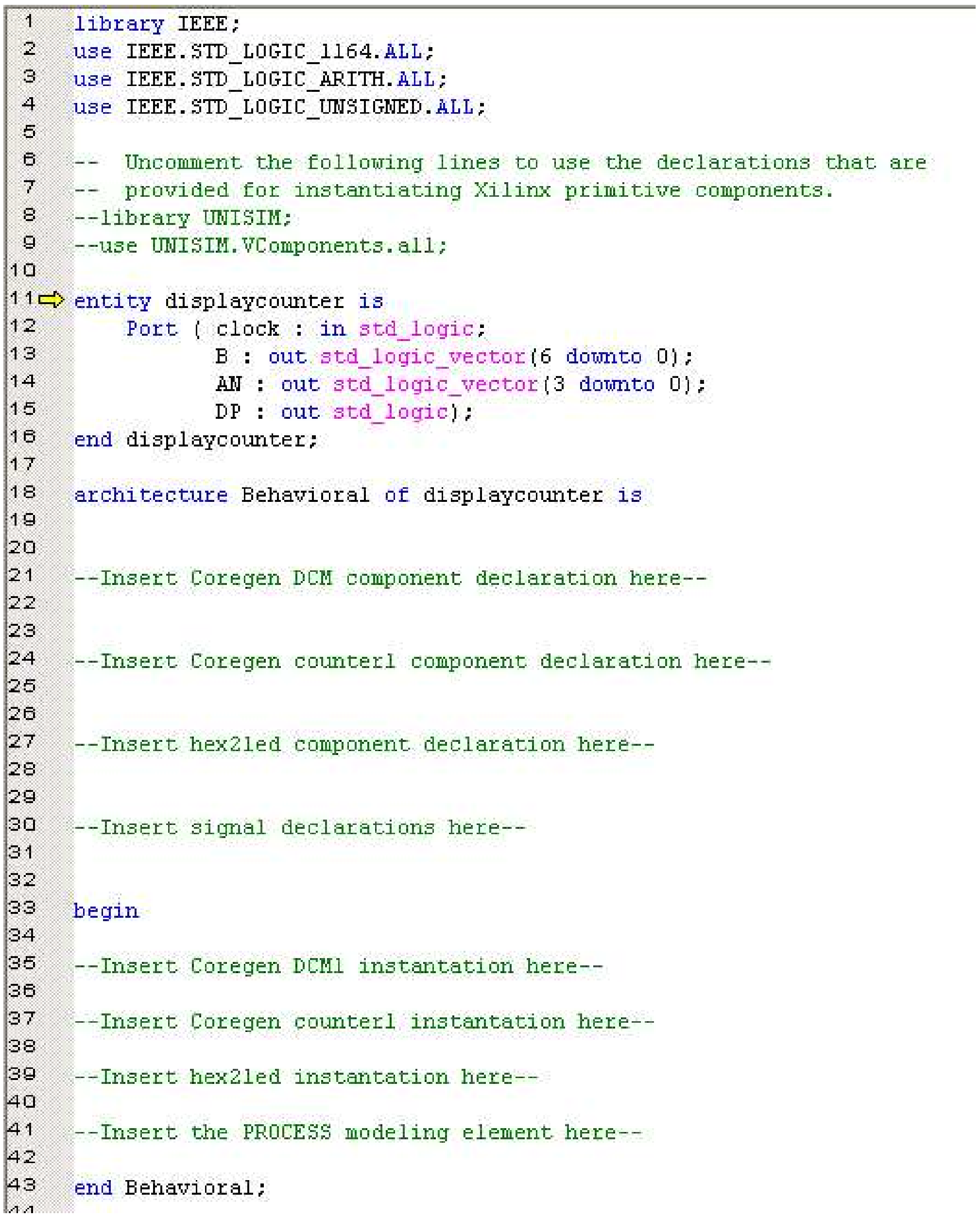
3. The **New Source Information** window appears. Click **Finish**. Click **Next** and then **Finish**. This will create a new skeleton source with the defined inputs and outputs.



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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity displaycounter is
    Port ( clock : in std_logic;
           B : out std_logic_vector(6 downto 0);
           AN : out std_logic_vector(3 downto 0);
           DP : out std_logic);
end displaycounter;

architecture Behavioral of displaycounter is

begin


end Behavioral;

```

Figure 10 New skeleton source

4. Modify the skeleton file as shown below in figure 11. Only the lines of the form **-- <*text*> --** have been added. These are comment lines that point to positions where additional code should be inserted.



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

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity displaycounter is
    Port ( clock : in std_logic;
           B : out std_logic_vector(6 downto 0);
           AN : out std_logic_vector(3 downto 0);
           DP : out std_logic);
end displaycounter;

architecture Behavioral of displaycounter is


--Insert Coregen DCM component declaration here--


--Insert Coregen counter1 component declaration here--


--Insert hex2led component declaration here--


--Insert signal declarations here--


begin

--Insert Coregen DCM1 instantation here--

--Insert Coregen counter1 instantation here--

--Insert hex2led instantation here--

--Insert the PROCESS modeling element here--

end Behavioral;
```

Figure 11 Added text lines to the Top-Level module

5. Click **File** ⇨ **Save**.

After you have created the top-level VHDL module displaycounter, next a low-level VHDL module called hex2led has to be added. The skeleton of this module has already been created for this project, but it must be added to it. The VHDL module is then connected to your top-level VHDL design through instantiation and compiled with the rest of the design. This macro serves to convert 4-bit HEX value into a 7-segment LED display format. To add the source file to the project:

6. Select *TenCounterDisplay* in the Sources in Project window.
   Next, add the remaining VHDL file to the project.
7. Select **Project** ⇨ **Add Source**.
8. Download the *hex2led.zip* file and unzip it in your project directory.
9. Select *hex2led.vhd* from the project directory. Click **Open**.
10. In the Choose Source Type dialog box, select **VHDL Design File**. Click **OK**.
11. The VHDL file *hex2led.vhd* has been added to the project.

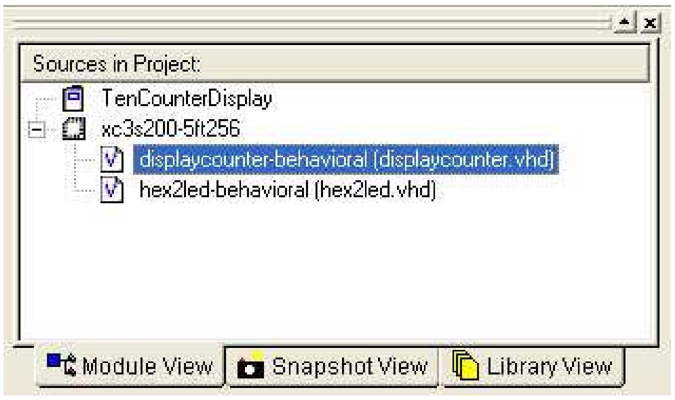
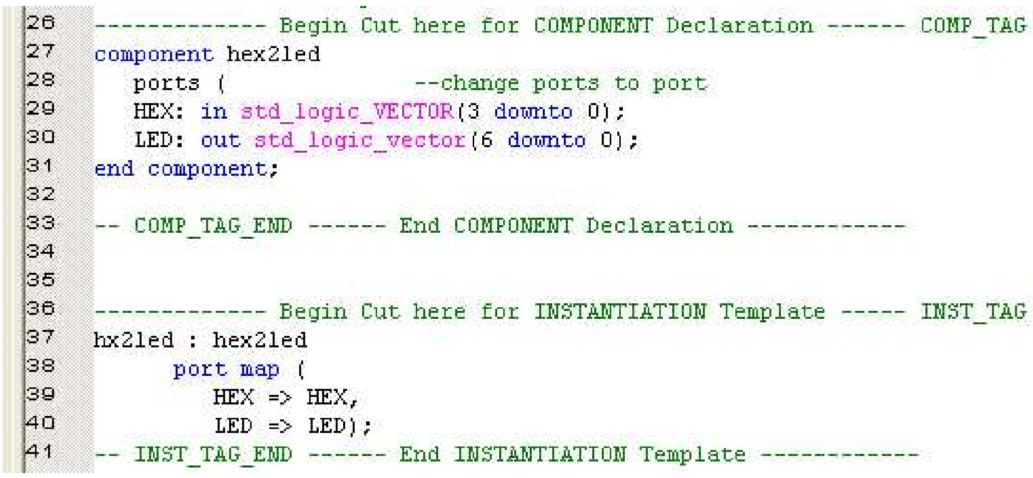


Figure 12 Module *hex2led.vhd* has been added to the project

Next you are going to instantiate the hex2led module in the top-level module displaycounter. To instantiate:

12. Place the cursor after the line in *displaycounter.vhd* that states:
   `"--Insert hex2led component declaration here--"`
13. Select **Edit** ⇨ **Insert File** and choose *hex2led.vho*.
14. Click **Open**.
   The VHDL template file for the instantiation is inserted.

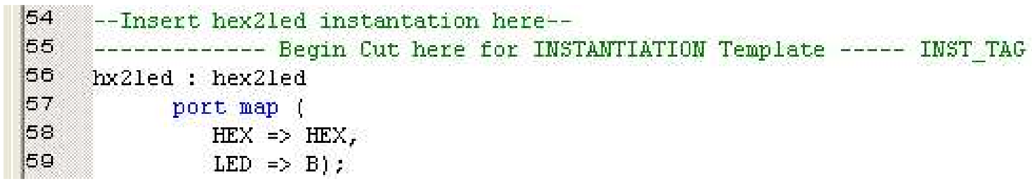


```
------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component hex2led
   ports (             --change ports to port
   HEX: in std_logic_VECTOR(3 downto 0);
   LED: out std_logic_vector(6 downto 0);
end component;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------


------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
hx2led : hex2led
      port map (
         HEX => HEX,
         LED => LED);
-- INST_TAG_END ------ End INSTANTIATION Template ------------
```

Note: The component declaration does not need to be modified.

15. Highlight the inserted code from
   `"------------- Begin Cut here for INSTANTIATION Template"`
   to
   `"LED => LED);"`

16.Select **Edit** ⇨ **Cut**.
17.Place the cursor after the line that states:
`"--Insert hex2led instantiation here--"`
18.Select **Edit** ⇨ **Paste** to place the instantiation here.
19.Edit this instantiated code to connect the B output port of the counterdisplay design to the LED port.



```
--Insert hex2led instantation here--
------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
hx2led : hex2led
      port map (
         HEX => HEX,
         LED => B);
```

Figure 13 Port B connected to Port LED

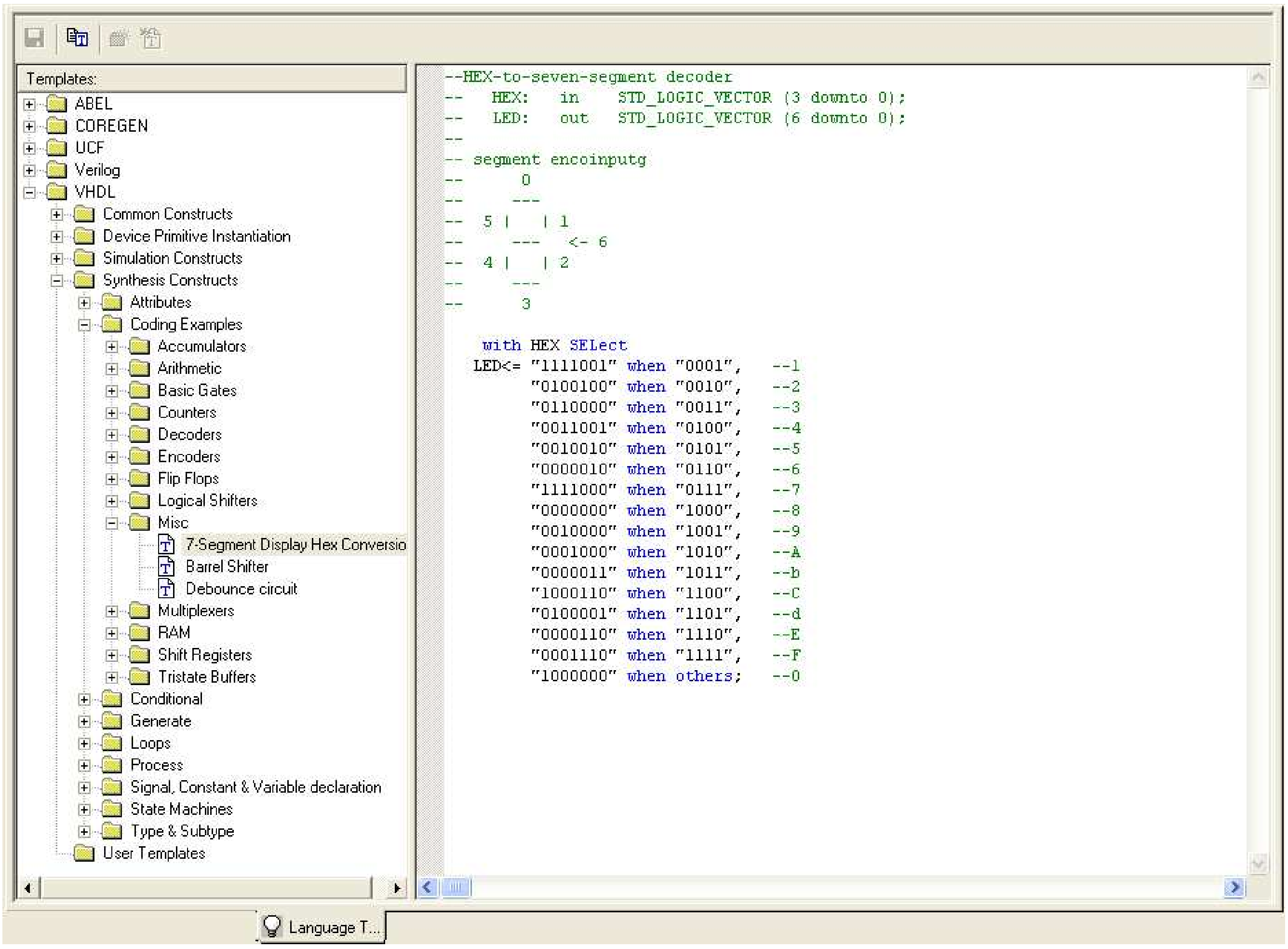
To send data to the HEX input port like on a real physical data line in a circuit a signal declaration of this port has to be made. To do this:
20.Highlight the following line:
`"signal HEX: std_logic_vector(3 downto 0):="0000";"`
21.Press and hold down the **CTRL** key while pressing **C**.
22.Place the cursor after the line that states:
"--Insert signal declarations here--"
23.Press and hold down the **CTRL** key while pressing **V**.
The HEX signal port is now declared.
24.Click **File** ⇨ **Save**.

To check the syntax of source files:
25.Select *displaycounter.vhd* in the Sources in Project window.
Upon selecting the VHDL file, the Processes for Source window displays all processes available for this file.
26.Double click **Check Syntax** in the Synthesize-XST hierarchy.

The hex2led design contains a syntax error that must be corrected. The red "x" beside the Check Syntax process indicates an error was found during analysis. Project Navigator reports errors in red and warnings in yellow in the Console window.

To display the error in the source file:
27.Double click on the error message in the console window.
28.Correct any errors in VHDL source file. The comments next to the error explain this simple fix.
29.Select **File** ⇨ **Save** to save the file.
30.Re-analyze the file by selecting the VHDL file and double-clicking **Check Syntax** in the Synthesize-XST hierarchy.
The ISE Language Templates include VHDL constructs and synthesis templates which represent commonly used logic components, such as counters, D-flip flops, multiplexers and primitives. You will use the HEX2LED Converter template for this exercise. This template provides source code to convert a 4-bit to 7-segment LED display format. To invoke the Language Templates and select the template for this tutorial:

31.From Project Navigator, select **Edit** ⇨ **Language Templates**.
32.To expand the view click the + next to the topic VHDL.
33.Under the VHDL hierarchy, expand the Synthesis Constructs hierarchy.
34.Under the Synthesis Constructs hierarchy, expand the Coding Examples hierarchy.
35.Under the Coding Examples hierarchy, expand Misc and click the **7-Segment Display Hex Conversion** template. The contents display in the right hand pane.



```
--HEX-to-seven-segment decoder
--   HEX:   in    STD_LOGIC_VECTOR (3 downto 0);
--   LED:   out   STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoinputg
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---
--      3

    with HEX SELect
   LED<= "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
         "0000011" when "1011",   --b
         "1000110" when "1100",   --C
         "0100001" when "1101",   --d
         "0000110" when "1110",   --E
         "0001110" when "1111",   --F
         "1000000" when others;   --0
```

Figure 14 The language template of the 4 bit to 7-segment display converter

36.Highlight the code from the line that states:
`"with HEX SELect"`
to
`" "0010000" when "1001", --9  "`

37.Select **Edit** ⇨ **Copy**.
38.Double click on the source file *hex2led.vhd* in the Sources in Project window. The *hex2led.vhd* source file opens in the ISE text editor window.

39. Place the cursor after the line in *hex2led.vhd* that states:
`"begin"`
40. Select **Edit** ⇨ **Paste**.

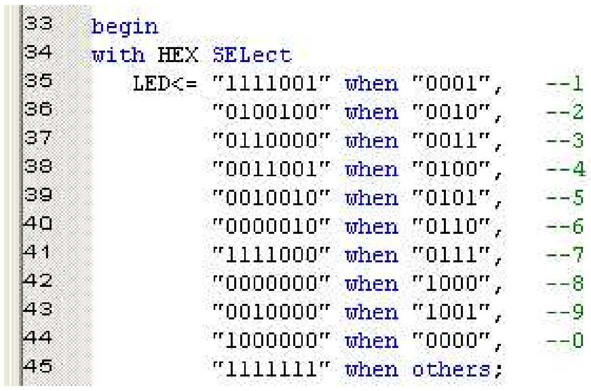
41.Highlight these following lines:

```
‶ "1000000" when "0000",   --0
  "1111111" when others; ″
```

42.Press and hold down the **CTRL** key while pressing **C**.
43.Place the cursor after the line that states:

```
‶  ″0010000″ when ‶1001″,  --9  ″
```

44. Press and hold down the **CTRL** key while pressing **V**.
The new lines have been added.



```
begin
with HEX SELect
    LED<= "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
          "1000000" when "0000",   --0
          "1111111" when others;
```

Figure 15 New line has been added to the template

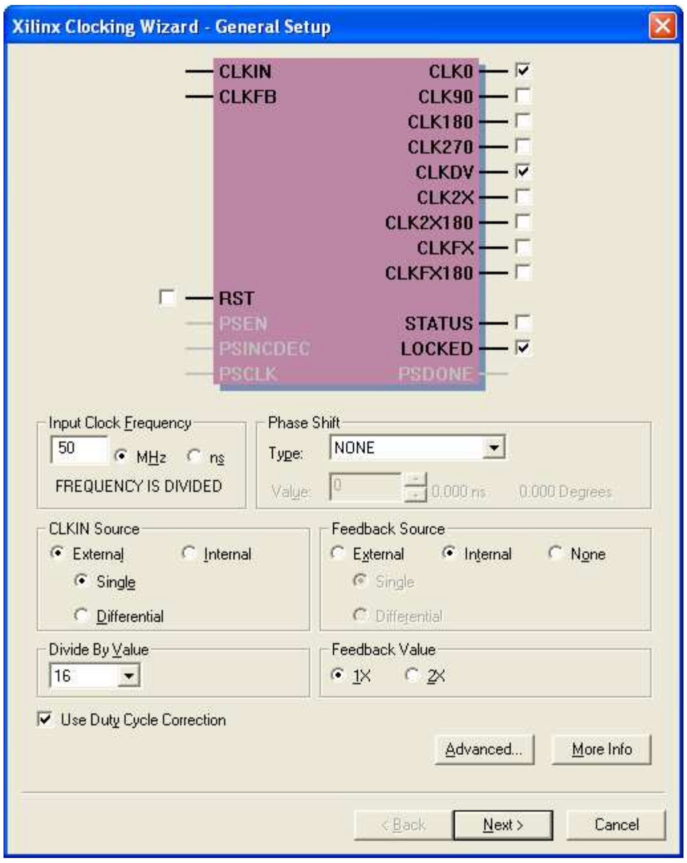
45. Save the design using **File** ⇨ **Save** and close *hex2led.vhd* and the language template in the ISE text editor.

❸ Creating CORE Generator modules

CORE generator is a graphical interactive design tool used to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and preoptimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic for arithmetic functions and on-chip RAM for dual-port and synchronous RAM.
In this section you will first create a DCM(digital clock manager) module called DCM1. This is a clock frequency divider with internal feedback and 50 % duty cycle correction. It lowers the 50 Mhz crystal oscillator frequency to 1.5625 Mhz. The second module is a 0 to 1.5625E6 counter called counter1. It's operating on the generated frequency of 1.5625MHz of the DCM1 module. When the count value of 1.5625E6 is reached then it sets a Q_THRESH threshold signal on high. The duration of this count is exactly one second. This is the amount of time that each digit number is displayed. The counter resets itself and start to count again for one second. Then on the check of Q_THRESH the next digit value is displayed and so on.

1. In **Project Navigator** window select **Project** ⇨ **New Source**.
2. Select **IP (CoreGen & Architecture Wizard)** as the source type.
3. Enter 'DCM1' in the **File Name** field.

4. Make sure that Add to Project is checked. Click **Next**.
5. Click **Clocking** ⇨ **Single DCM**.
6. Click **Next** and then **Finish**. The Clocking Wizard is launched.
7. Make sure you set the settings as shown below in figure 16.


Xilinx Clocking Wizard - General Setup

CLKIN
CLKFB
CLK0 [✓]
CLK90 [ ]
CLK180 [ ]
CLK270 [ ]
CLKDV [✓]
CLK2X [ ]
CLK2X180 [ ]
CLKFX [ ]
CLKFX180 [ ]
[ ] RST
PSEN
PSINCDEC
PSCLK
STATUS [ ]
LOCKED [✓]
PSDONE

Input Clock Frequency: 50 [✓ MHz] ns
FREQUENCY IS DIVIDED

Phase Shift
Type: NONE
Value: 0 0.000 ns 0.000 Degrees

CLKIN Source: [✓ External] Internal
[✓ Single] Differential

Feedback Source: External [✓ Internal] None
[✓ Single] Differential

Divide By Value: 16

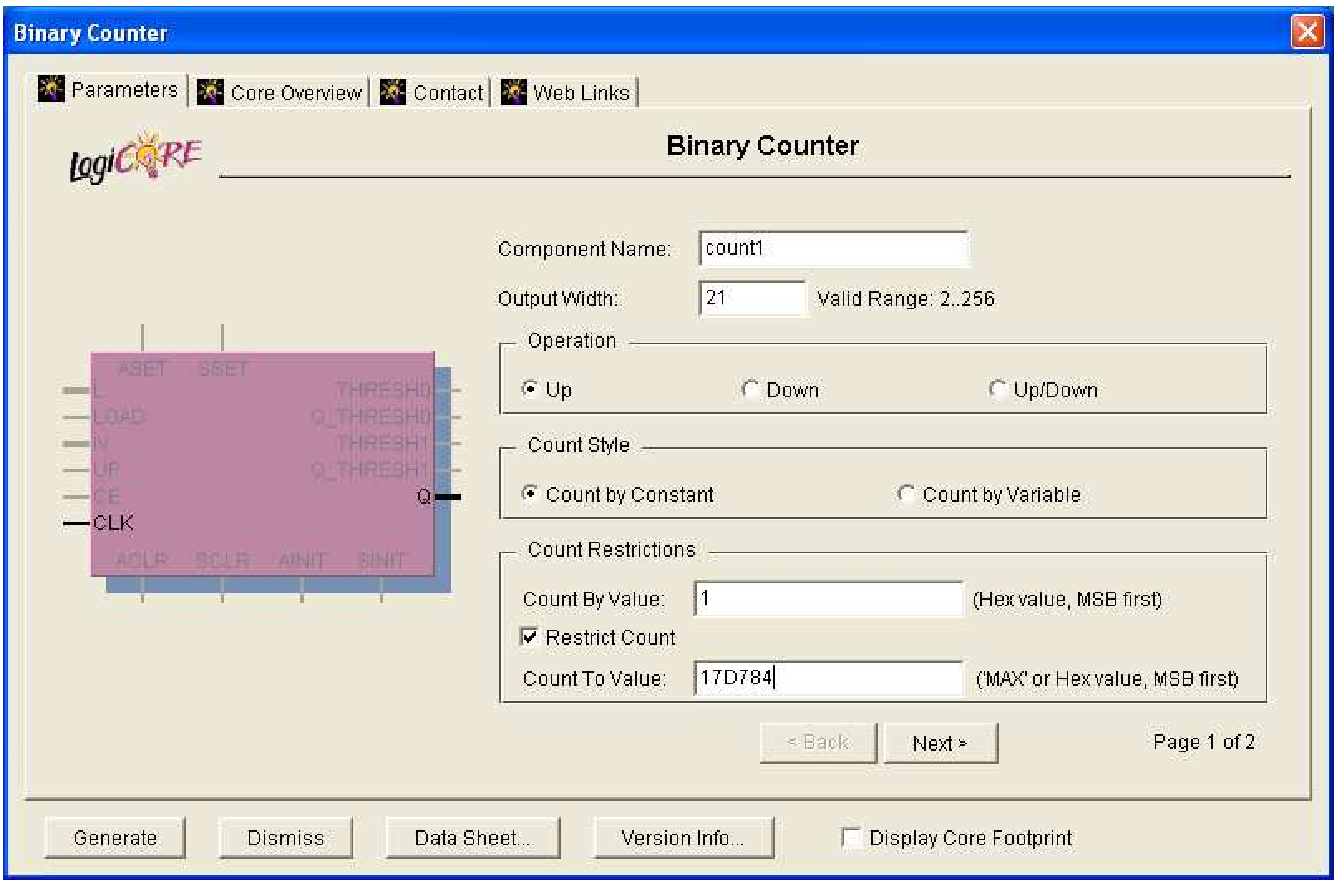
Feedback Value: [✓ 1X] 2X

[✓] Use Duty Cycle Correction

Figure 16 Setting the clocking wizard

8. Click the **Advanced** button.
9. Select Wait for DCM lock before DONE signal goes high.
10. Select Divide Input Clock By 2.
11. Click **OK** and **Next**.
    An information message displays the locked signal and the STARTUP_WAIT option.
12. Click **Finish**.
    DCM1.xaw is added to the list of project source files in the Sources in Project window.

13. In **Project Navigator** window select **Project** ⇨ **New Source**.
14. Select **IP (CoreGen & Architecture Wizard)** as the source type.
15. Enter 'count1' in the **File Name** field.
16. Make sure that Add to Project is checked. Click **Next**.
17. Click **Basic Elements** ⇨ **Counters** ⇨ **Binary Counter**.
18. Click **Next** and then **Finish**. The Binary Counter wizard is launched.
19. Make sure you set the settings as shown below in figure 17. The hexadecimal value of 17D784 is equal to a decimal value of 1.5625E6.



Binary Counter

Component Name: count1

Output Width: 21 Valid Range: 2..256

Operation: [✓ Up] Down Up/Down

Count Style: [✓ Count by Constant] Count by Variable

Count Restrictions

Count By Value: 1 (Hex value, MSB first)

[✓ Restrict Count]

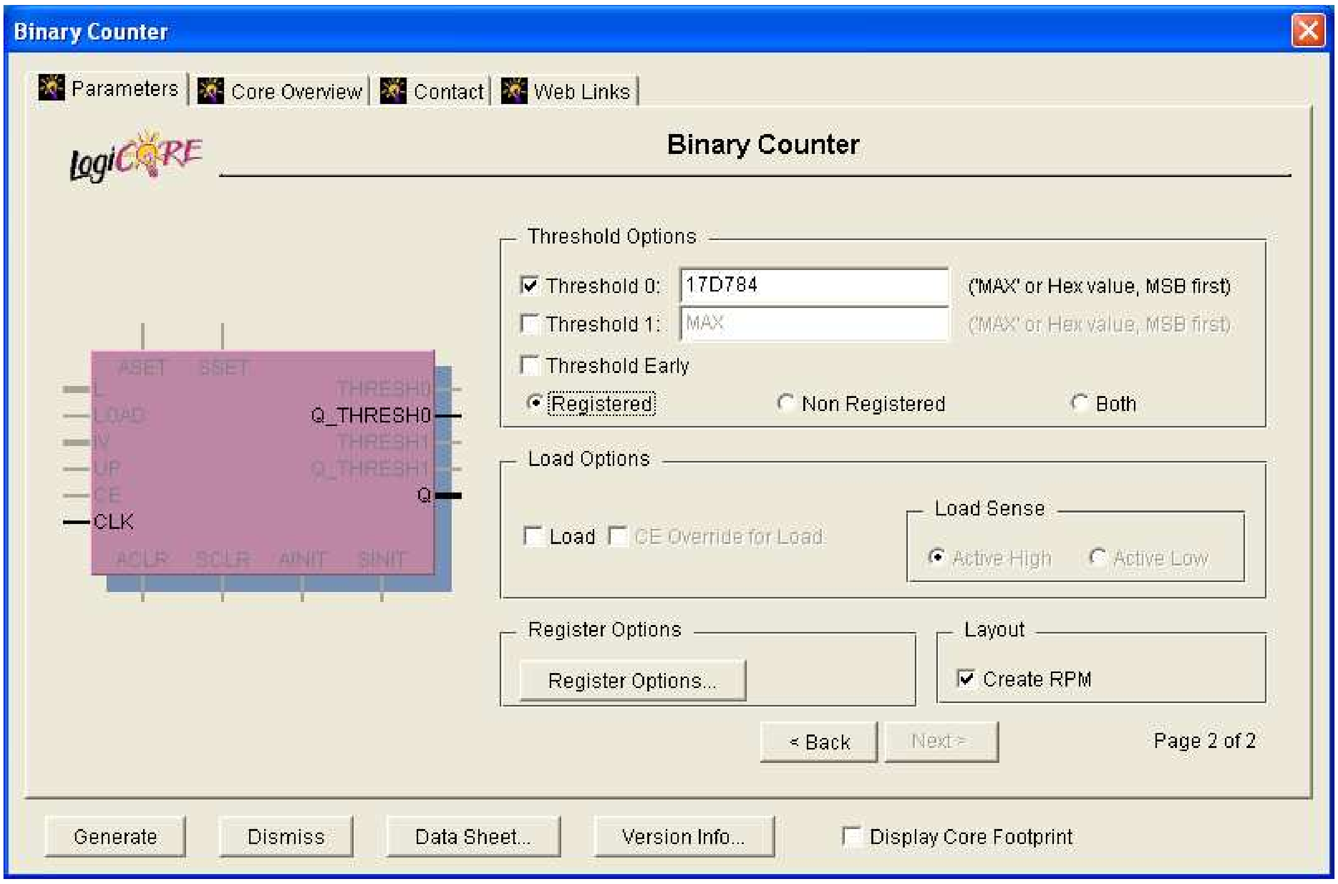
Count To Value: 17D784 ('MAX' or Hex value, MSB first)

☐ Display Core Footprint

Figure 17 Setting the binary counter

20.Click **Next**.

21.Continue to fill in the Binary Counter dialog box with the settings as shown below in figure 18.



Binary Counter

Parameters | Core Overview | Contact | Web Links

Binary Counter

Threshold Options

[✓] Threshold 0: 17D784 ('MAX' or Hex value, MSB first)

[ ] Threshold 1: MAX ('MAX' or Hex value, MSB first)

[ ] Threshold Early

[✓ Registered] Non Registered Both

Load Options

[ ] Load [ ] CE Override for Load

Load Sense: [✓ Active High] Active Low

Register Options: Register Options...

Layout: [✓] Create RPM

< Back Next > Page 2 of 2

Generate Dismiss Data Sheet... Version Info... [ ] Display Core Footprint

Figure 18 Continue setting the binary counter

22.Click the **Register Options** button.
23.Select **Clock Enable**. Click **OK**.
24.Click **Generate**.

The module has been created and automatically added to the project library. A number of other files are added to the project directory. These files are:

- *count1.sym*
  This is a schematic symbol file.
- *count1.edn*
  This file is the netlist that is used during the Translate phase of implementation.
- *count.vho* or *count1.veo(for verilog)*
  This is the instantiation templates that is used to incorporate the CORE Generator module in your source HDL.
- *count.vhd* or *count.v(for verilog)*
  These are simulation-only files.
- *count1.xco*
  This file stores the configuration information for the count1 module and is used as a project source.
- *coregen.prj*
  This file stores the CORE Generator configuration for the project.

25.Click **OK**.

❹ Instantiating the CORE Generator modules

Next, instantiate the DCM1 and the count1 macro for your VHDL design. First instantiate the DCM1 macro for your VHDL design:

1. In Project Navigator, in the Sources in Project window, select *DCM1.xaw*.
2. Double click **View VHDL Instantiation Template** in the Processes for Source window.
3. Form the newly opened VHDL Instantiation Template copy the component declaration template:

```
COMPONENT DCM1
PORT(
        CLKIN_IN : IN std_logic;
        CLKDV_OUT : OUT std_logic;
        CLKIN_IBUFG_OUT : OUT std_logic;
        CLK0_OUT : OUT std_logic;
        LOCKED_OUT : OUT std_logic
          );
END COMPONENT;
```

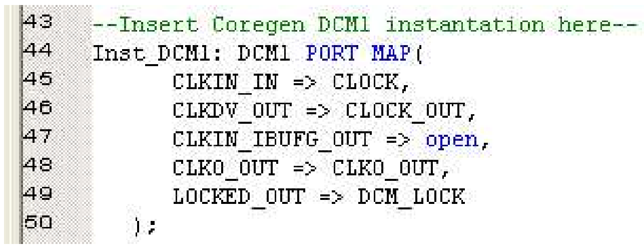
4. Paste the component declaration into the section in *displaycounter.vhd* labeled
   --Insert Coregen DCM component declaration here--

5. Copy the instantiation template from the newly opened VHDL Instantiation Template:

```
Inst_DCM1: DCM1 PORT MAP(
        CLKIN_IN => ,
        CLKDV_OUT => ,
        CLKIN_IBUFG_OUT => ,
        CLK0_OUT => ,
        LOCKED_OUT =>
);
```

6. Paste the instantiation template into the section in *displaycounter.vhd* labeled
   --Insert Coregen DCM1 instantiation here--

7. Make the necessary changes as shown below in figure 19:



```
--Insert Coregen DCM1 instantation here--
Inst_DCM1: DCM1 PORT MAP(
      CLKIN_IN => CLOCK,
      CLKDV_OUT => CLOCK_OUT,
      CLKIN_IBUFG_OUT => open,
      CLK0_OUT => CLK0_OUT,
      LOCKED_OUT => DCM_LOCK
   );
```

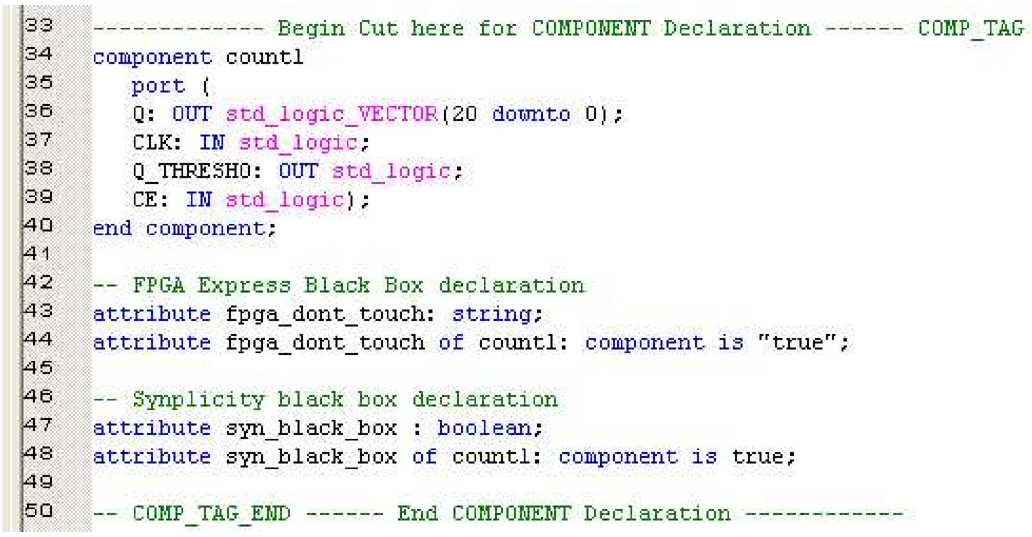
Figure 19 Port mapping for the DCM1 instantiation

8. Close the *DCM1.vhi* module in the ISE text editor.
9. To connect a clock signal to the *CLKIN_IN* port of the DCM1 the clock input of the display counter has to be connected to it and the output ports of the DCM1 should become available as signals. To do this all these ports must be declared as signals. To declare signals of the DCM1 ports:
10. Highlight these following lines:
```
"signal CLOCK_OUT: std_logic;
 signal DCM_LOCK: std_logic;
 signal CLK0_OUT: std_logic;"
```
11. Press and hold down the **CTRL** key while pressing **C**.
12. Place your cursor after the line that states:
```
"signal HEX: std_logic_vector(3 downto 0):="0000";"
```
13. Press and hold down the **CTRL** key while pressing **V**.

Next, instantiate the CORE Generator count1 module in the VHDL code.

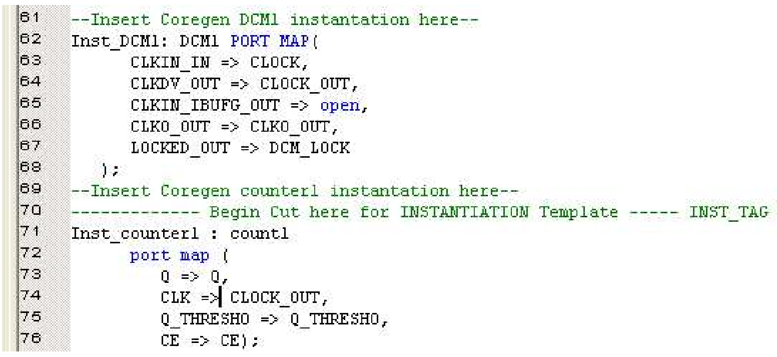
1. Place your cursor after the line that states:
```
"--Insert Coregen counter1 component declaration here--"
```

2. Select **Edit** ⇨ **Insert File** and choose *count1.vho*. Click **Open**.
   The VHDL template file for the CORE Generator instantiation is inserted.
   Note: The component declaration does not need to be modified.


```
------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component count1
   port (
   Q: OUT std_logic_VECTOR(20 downto 0);
   CLK: IN std_logic;
   Q_THRESH0: OUT std_logic;
   CE: IN std_logic);
end component;

-- FPGA Express Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of count1: component is "true";

-- Synplicity black box declaration
attribute syn_black_box : boolean;
attribute syn_black_box of count1: component is true;

-- COMP_TAG_END ------ End COMPONENT Declaration ------------
```
Figure 20 The component declaration of the *count1* counter module

3. Highlight the inserted code from
```
"------------- Begin Cut here for INSTANTIATION Template -----"
```
   to
```
"CE => CE);"
```

1. Select **Edit** ⇨ **Cut**.
2. Place the cursor after the line that states:
   `"--Insert Coregen counter1 instantiation here--"`
3. Select **Edit** ⇨ **Paste** to place the instantiation here.
4. Change `"your_instance_name" to Inst_counter1.`
5. Edit this instantiated code to connect the *CLK* port to the *CLOCK_OUT* port of the DCM1 CORE Generator module as shown below in figure 21.



```
--Insert Coregen DCM1 instantation here--
Inst_DCM1: DCM1 PORT MAP(
      CLKIN_IN => CLOCK,
      CLKDV_OUT => CLOCK_OUT,
      CLKIN_IBUFG_OUT => open,
      CLK0_OUT => CLK0_OUT,
      LOCKED_OUT => DCM_LOCK
   );
--Insert Coregen counter1 instantation here--
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
Inst_counter1 : count1
      port map (
         Q => Q,
         CLK => CLOCK_OUT,
         Q_THRESH0 => Q_THRESH0,
         CE => CE);
```

Figure 21 CLK input connected to the CLOCK_OUT of the DCM1 module

6. This step is not necessary. But it's better for the readableness of your source VHDL file to delete the comment lines (-- <*text*>--) that have been created by the Core Generator for the count1 module.

To make the input and output ports of the count1 counter available as signals, these ports have to be declared as signals:

7. Highlight the following lines:
```
" signal Q: std_logic_vector(20 downto 0);
  signal Q_THRESH0: std_logic;
  signal CE: std_logic; "
```
8. Press and hold down **CTRL** key while pressing **C**.
9. Place the cursor after the line that states:
   `"signal CLK0_OUT: std_logic;"`
10. Press and hold down the **CTRL** key while pressing **V**.

You are almost at the end of source code entry. You have added all the components that are needed and made all the required connections between these components. The last thing to do is to add the Process modeling element with code that describe the behavior of the 7-segment LED counter.

11.Highlight these following lines:

```
  " CE <= '1';
    DP <= '0';
    AN <= ″1110″;
    PROCESS
      BEGIN
         wait until Q_THRESH0'event and Q_THRESH0 ='1';
           if   HEX =  ″1001″
           then HEX <= ″0000″;
           else HEX <= HEX + ″0001″;
           end if;
    END PROCESS; ″
```

12.Press and hold down the **CTRL** while pressing **C**.
13.Place the cursor after the line that states:

```
  "--Insert the PROCESS modeling element here-- ″
```

14.Press and hold down the **CTRL** key while pressing **V**.
15.Save the design using **File** ⇨ **Save**.

❺ Simulation

The design code entry is completed. But you don't know if the design is working properly or not. It would be nice if you can simulate the behavior of the design in a simulator before translating the design and implement it into the spartan III FPGA board. For this purpose we use the ModelSim simulator. In order to use this simulator, a test bench is required to provide stimulus to the design. To create the test bench:

1. Select **Project** ⇨ **New Source**.
   A dialog box opens in which you specify the type of source you want to create.
2. Select **VHDL Test Bench**.
3. In the File Name field type '*displaycounter_tb*'.
4. Make sure that Add to Project is checked. Click **Next**.
   A dialog box opens in which you select the source file.
5. Select **displaycounter**.
6. Click **Next** ⇨ **Finish.**
   A VHDL test bench has been created from source file '*displaycounter.vhd*'. It is opened in the ISE text editor window.

The Top-Level module *displaycounter* has been instantiated as a component in the test bench file. You can think of it as a black box that contains the behavior of the Top-Level design. To access the component interface, all the ports have been declared as signals. The only thing that is missing is a clock generator that connects to the clock input of the Top-Level design. To add the clock generator:

7. Highlight these following lines:

```
"generatorclock: process
  begin
     clock <= '1';
     loop
        wait for (ClockPeriod/2);
         CLOCK <= not CLOCK;
      end loop;
  end process;"
```

8. Press and hold down the **CTRL** key while pressing **C**.
9. Place the cursor after the line in *'displaycounter_tb.vhd'* that states:
   `"BEGIN"`
10. Press and hold down the **CTRL** key while pressing **V**.
11. Highlight this following line:
   `"constant ClockPeriod : Time := 20 ns;"`
12. Place the cursor after the line that states:
   `"SIGNAL DP :  std_logic;"`
13. Press and hold down the **CTRL** while pressing **V**.
14. Save the test bench file using **File** ⇨ **Save**.

The test bench file is now complete. A complete count of 0-9 needs a simulation time of 10 seconds. However at an operation frequency of 50 Mhz the simulator will create a lot of simulation data. The overall duration of the simulation will also take a long time. In order to ease the simulator you are going to decrease the display time interval between each digit value from one second to one millisecond. So that only 10 ms seconds is needed to see a complete run. To do this the maximum count value of *counter1* has to be set to a hexadecimal value of 61A.

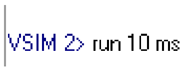
15. Double click on the source file *count1.xco* in the Sources in Project window.
    The Binary Counter wizard is launched.
16. Enter '61A' in the **Count To Value:** field.
17. Click **Next**.
18. Enter '61A' in the **Threshold 0:** field.
19. Click **Generate**.
20. Click **OK**.

Next, launch the ModelSim simulator.

21.Select *displaycounter_tb.vhd* in the Sources in Project window.
Upon selecting the VHDL file, the Processes for Source window displays all processes available for this file.

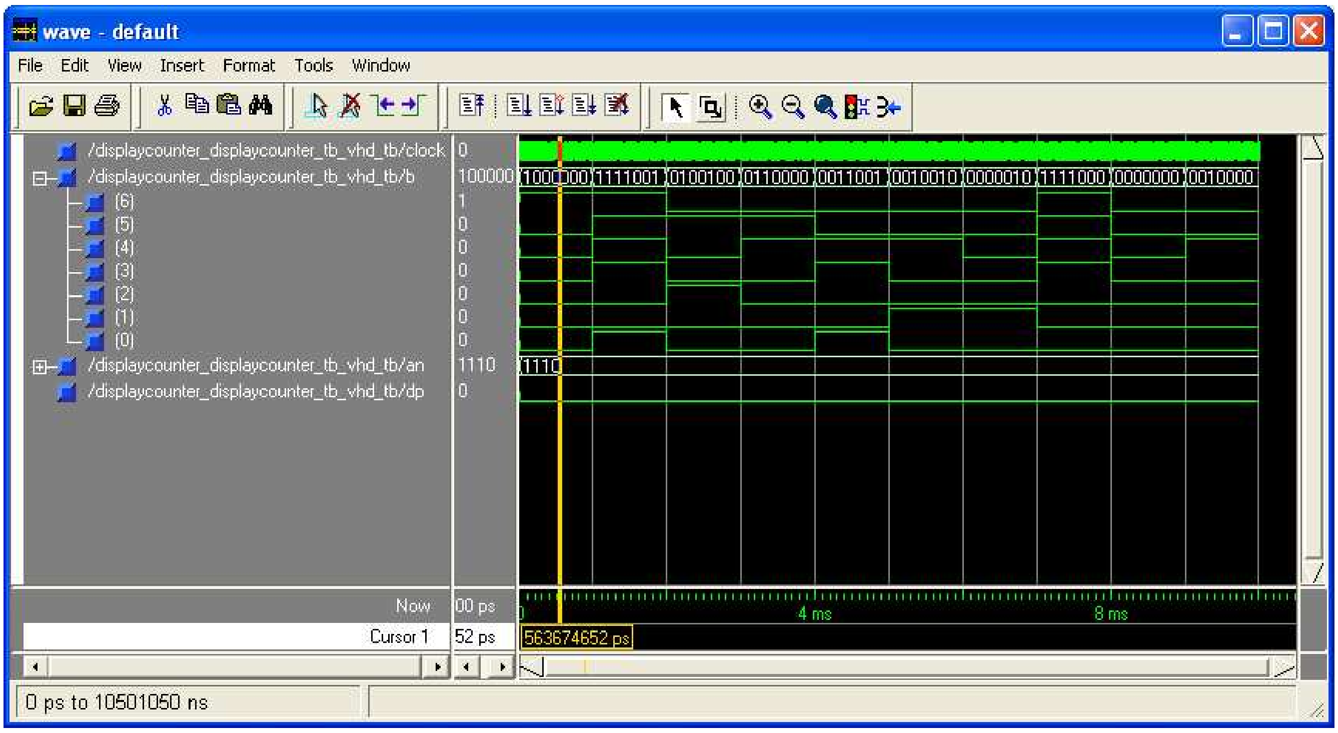
22.Double click **Simulate Behavioral Model** in the ModelSim Simulator hierarchy.
The ModelSim simulator is launched.

26.At the ModelSim command prompt, enter 'run 10 ms' and hit enter.



```
VSIM 2> run 10 ms
```

27.The simulation will run for 10 ms. The waveforms for LED output B should now be visible in the Waveform Window.



wave - default
File Edit View Insert Format Tools Window
/displaycounter_displaycounter_tb_vhd_tb/clock 0
/displaycounter_displaycounter_tb_vhd_tb/b 100000
(6) 1
(5) 0
(4) 0
(3) 0
(2) 0
(1) 0
(0) 0
/displaycounter_displaycounter_tb_vhd_tb/an 1110
/displaycounter_displaycounter_tb_vhd_tb/dp 0
1000000 1111001 0100100 0110000 0011001 0010010 0000010 1111000 0000000 0010000
1110
Now 00 ps
Cursor 1 52 ps
563674652 ps
4 ms
8 ms
0 ps to 10501050 ns

Figure 22 Waveforms are shown in the waveform window after simulation

Next, set the counter value of the counter1 module back to a hexadecimal value of 17D784.

28. Double click on the source file *count1.xco* in the Sources in Project window.
The Binary Counter wizard is launched.

29.Enter ' 17D784 ' in the **Count To Value:** field.

30.Click **Next**.

31.Enter ' 17D784 ' in the **Threshold 0:** field.

32.Click **Generate**.

33.Click **OK**.

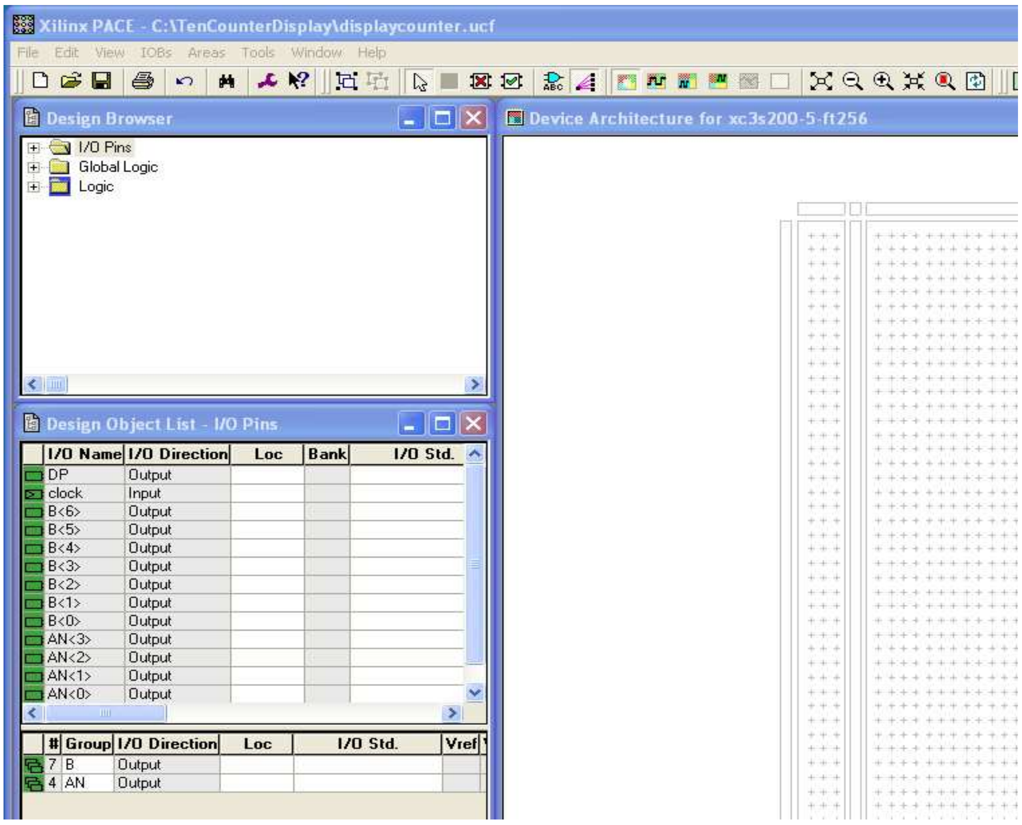
34.Close *displaycounter_tb.vhd* in the ISE editor.

❻ Assign Package Pins

The design code entry is finished and the design has been simulated. Now that we know that the design works, the next step is to assign package pins to the signals of your design. You are going to do this by using the Pinout Area Constraints Editor (PACE) tool. PACE generates a valid User Constraints File(UCF). This file will contain information about the I/O pin assignments. The Translate step shown under the Implement Design Process uses this file, along with the design source netlist(NGD), to produce a new NGD file. This section describes the creation of I/O assignments for several signals.

Launch PACE and place I/O pins in the Package Pin Window using the drag and drop method:

1. Select *displaycounter.vhd* in the Sources in Project window.
   Upon selecting the VHDL file, the Processes for Source window displays all processes available for this file.
2. Double click **Assign Package Pins** under the User Constraints hierarchy.
   A message displays asking to create a constraints file for the Top-Level module *displaycounter.*
3. Click **Yes**.
   The PACE editor is launched.


Xilinx PACE - C:\TenCounterDisplay\displaycounter.ucf
File Edit View IOBs Areas Tools Window Help
Design Browser
I/O Pins
Global Logic
Logic
Design Object List - I/O Pins
Device Architecture for xc3s200-5-ft256

I/O Name	I/O Direction	Loc	Bank	I/O Std.
DP	Output			
clock	Input			
B<6>	Output			
B<5>	Output			
B<4>	Output			
B<3>	Output			
B<2>	Output			
B<1>	Output			
B<0>	Output			
AN<3>	Output			
AN<2>	Output			
AN<1>	Output			
AN<0>	Output			

#	Group	I/O Direction	Loc	I/O Std.	Vref
7	B	Output			
4	AN	Output			

Figure 23 The PACE tool

4. Select the Package View window.
   This window shows the graphical representation of the device package.

5. In the Design Object List window, click, drag and drop the following signals to the specific location in the Package Pins window:
   - DP ⇨ P16
   - clock ⇨ T9
   - B6 ⇨ N16
   - B5 ⇨ F13
   - B4 ⇨ R16
   - B3 ⇨ P15
   - B2 ⇨ N15
   - B1 ⇨ G13
   - B0 ⇨ E14
   - AN3 ⇨ E13
   - AN2 ⇨ F14
   - AN1 ⇨ G14
   - AN0 ⇨ D14

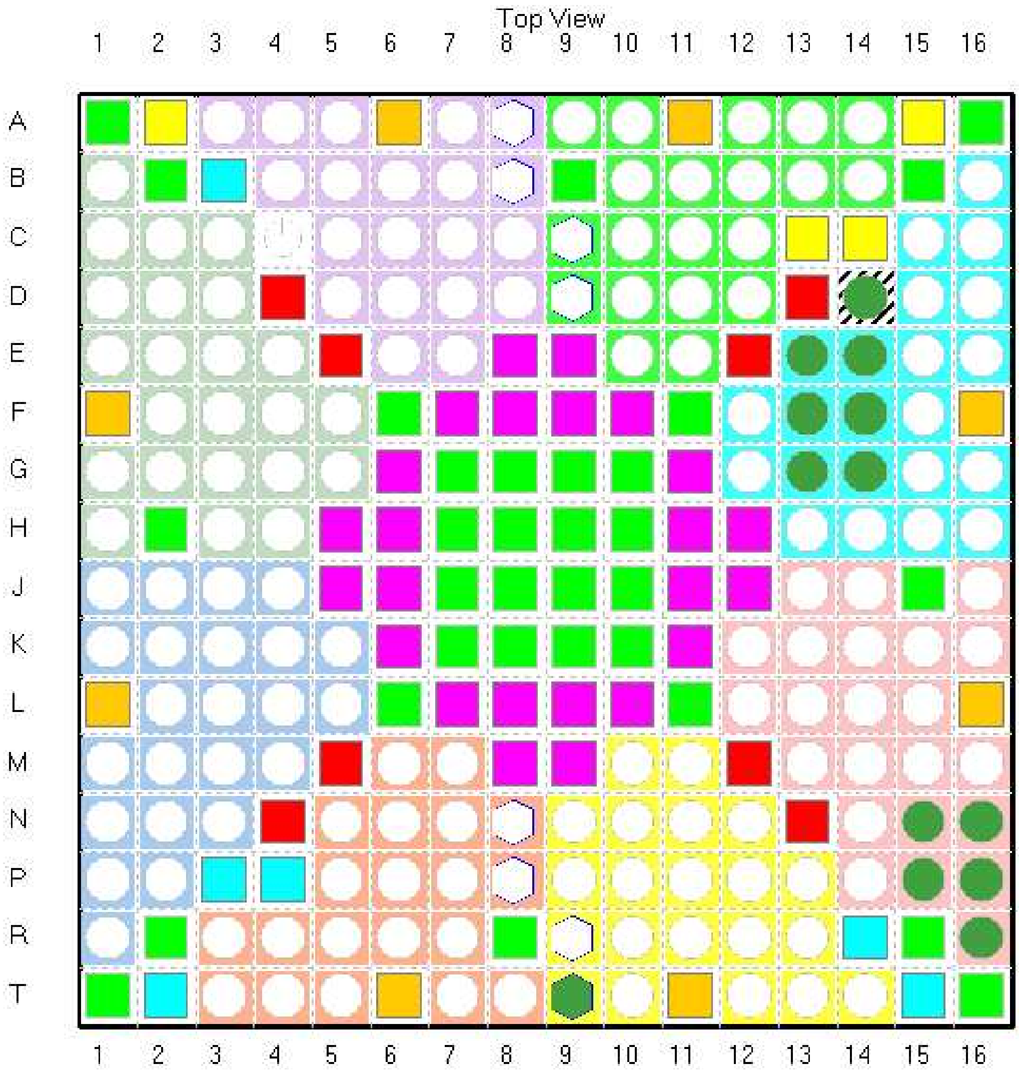


Figure 24 Drag and Drop I/Os in the Package Pins window

6. Once the pins are locked down, select **File** ⇨ **Save**. The changes made in PACE are now saved in the *displaycounter.ucf* file in your current working directory.
7. Exit PACE.

❼ Synthesizing and Implementing the Design

So far you have used XST for verifying syntax. But it also translates and optimizes the VHDL code into a set of components that the synthesis tool can recognize and translates it to the target technology's primitive components. At the back-end of this tool a netlist file NGC is created that is passed to the Implement Design Process. The Implement Design is the process of translating, mapping, placing and routing of your design. During translation the program NGDBuild converts input design netlists and writes results to a single merged NGD netlist and adds a user constraint file (UCF). Mapping allocates CLB(Configurable Logic Blocks) and IOB(IO Blocks) resources for all basic logic elements in the design. The design is then placed and routed after the mapped design is evaluated.

In the previous section you assigned the package pins. Since the PACE editor needs a NGD file from the Implement Design Process which in turn needs a NGC netlist file from the Synthesize-XST Process all these processes are started automatically. Completed steps in the processes are marked with a ✔. See figure 25.

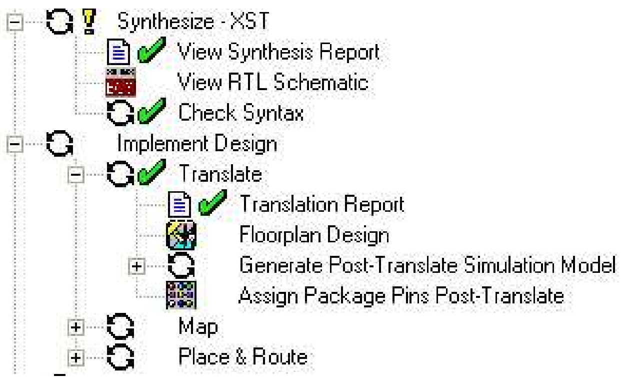


Figure 25 Completed steps are marked with a ✔

Next, finish the Implement Design process.
1. Double click **Implement Design** in the Process for Source window.
   Implement Design process runs the Map, Place & Route processes and finishes.

XST can generate a schematic representation of the VHDL code that you have entered. A schematic view of the code is helpful for analyzing your design to see a graphical connection between the various components that XST has inferred. To view a schematic representation of your RTL code:

2. Double click **View RTL Schematic** under the Synthesize-XST process hierarchy.
   The RTL Viewer displays the schematic.

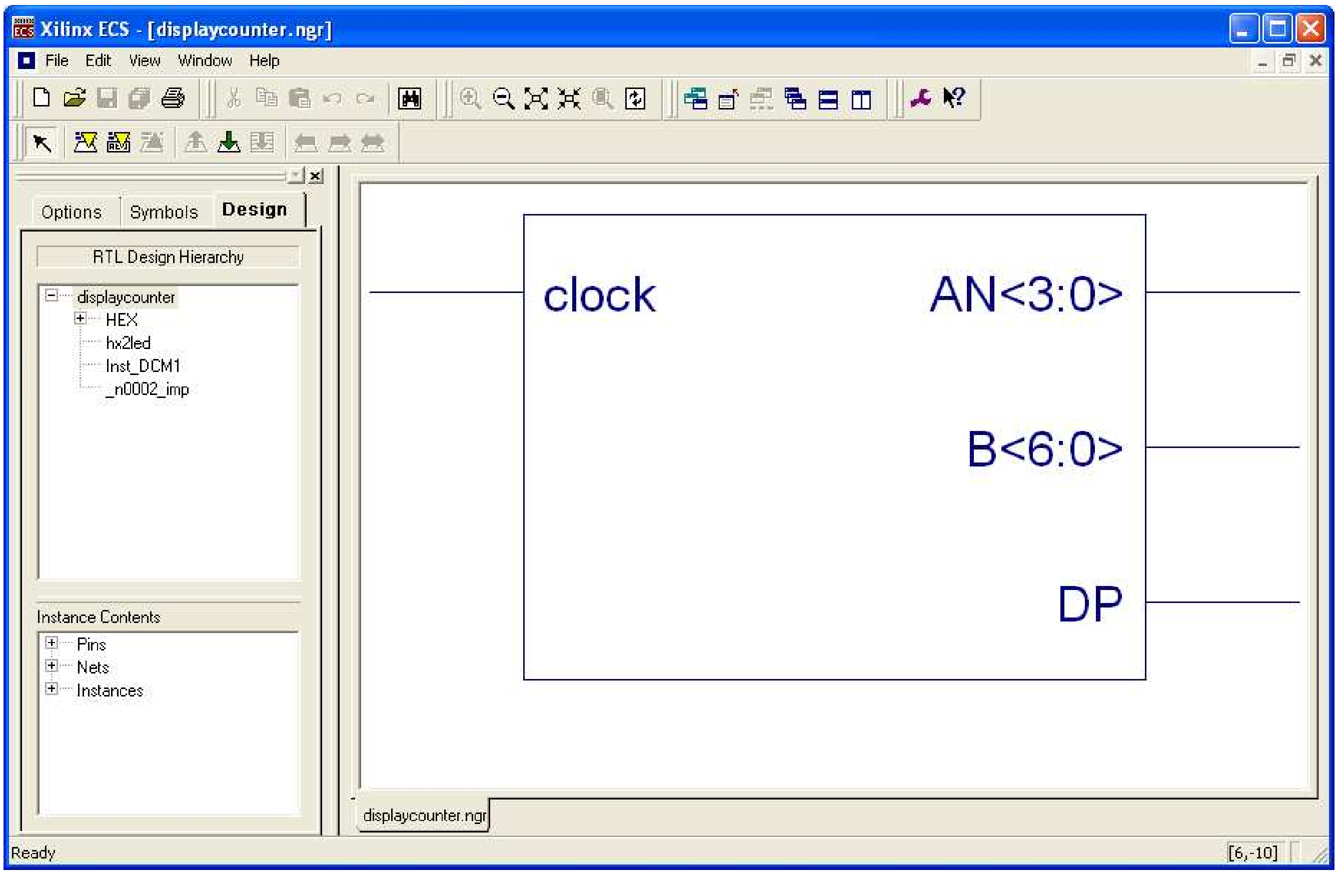


Figure 26 ECS RTL Viewer

Right click on the schematic to view various options for the schematic viewer.

❽ Generating BIT file and programming the device

This is the final step in the design flow. In this section you are going to create a configuration bitstream file for downloading into a PROM device. This device is a xilinx Flash Prom XCF02S denoted as number ❷ in figure 1. To program the device you are going to create a PROM file with the iMPACT programming tool. To do the programming you have to connect the Xilinx Spartan III FPGA board to the parallel port of your PC.

Create a bitstream for the target device as follows:

1. Double click **Generate Programming File** in the Process for Source window to create a bitstream of this design.
   The bitstream comes from the BitGen program and creates the *displaycounter.bit* file.
2. Verify that the file is in the project directory.

Next, launch iMPACT and create a PROM file as follows:

3. In the Processes for Source window, expand the **Generate Programming File** hierarchy

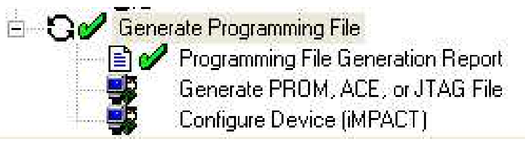
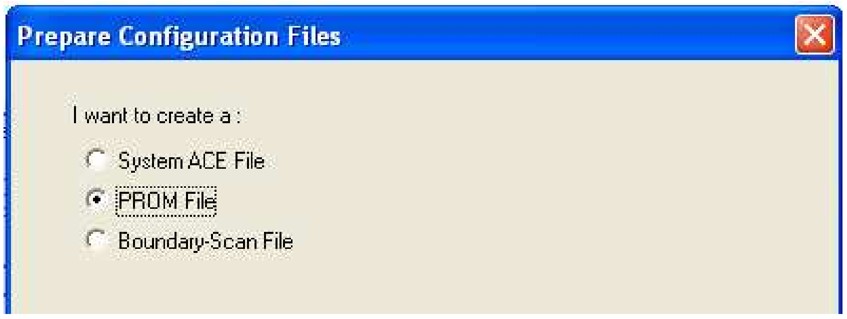

Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)

Figure 27 Generate Programming File hierarchy

4. Double click **Generate PROM, ACE, or JTAG File**.
   A wizard opens.
5. In the Prepare Configuration Files dialog box, under "`I want to create a:`", select **PROM File**.
6. Click **Next**.

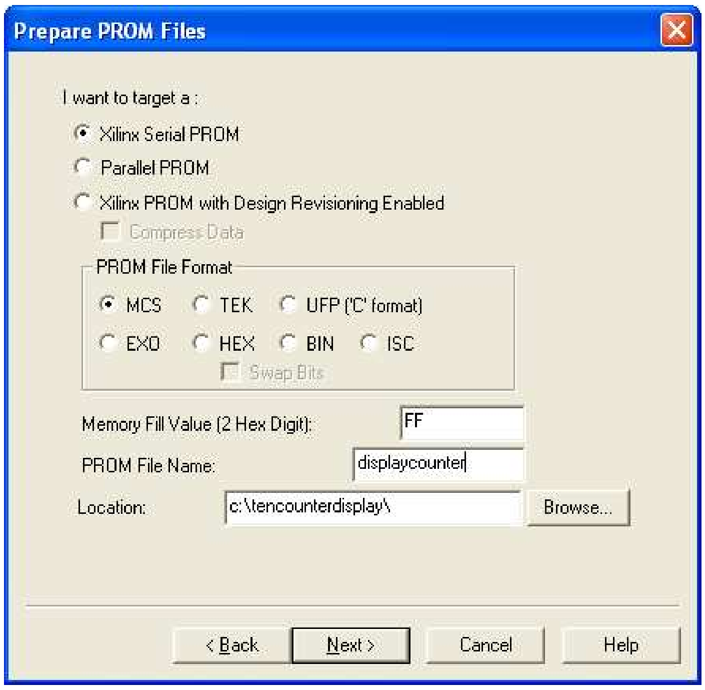

Prepare Configuration Files

I want to create a :

( ) System ACE File
(•) PROM File
( ) Boundary-Scan File

Figure 28 Prepare Configuration Files Dialog

7. Fill in the Prepare PROM Files dialog box is as shown below in figure 29.


Prepare PROM Files

I want to target a :

(•) Xilinx Serial PROM
( ) Parallel PROM
( ) Xilinx PROM with Design Revisioning Enabled
[ ] Compress Data

PROM File Format
(•) MCS ( ) TEK ( ) UFP ('C' format)
( ) EXO ( ) HEX ( ) BIN ( ) ISC
[ ] Swap Bits

Memory Fill Value (2 Hex Digit): FF
PROM File Name: displaycounter
Location: c:\tencounterdisplay\

Figure 29 Prepare PROM Files Dialog

8. Click **Next**.
9. In the Specify Xilinx Serial PROM device dialog box, select the PROM device as shown in figure 30.

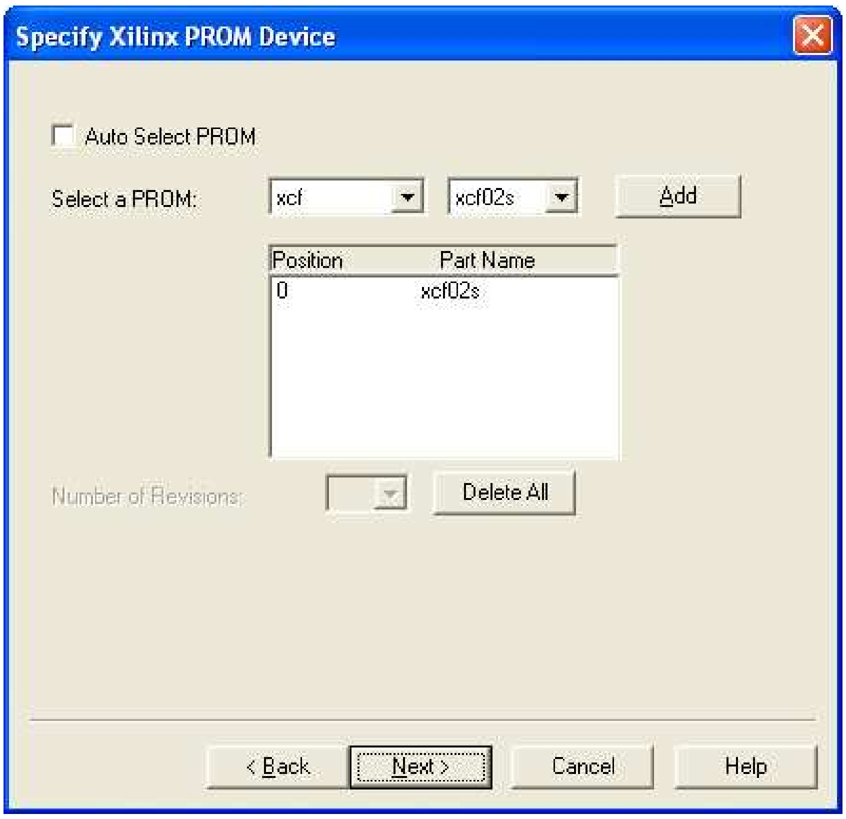


Figure 30 Specify Xilinx Serial PROM Dialog Box

10. Click **Next**.
11. In the File Generation Summary dialog box, click **Next**.
12. In the Add Device File dialog box, click **Add File** and select the *displaycounter.bit* file.

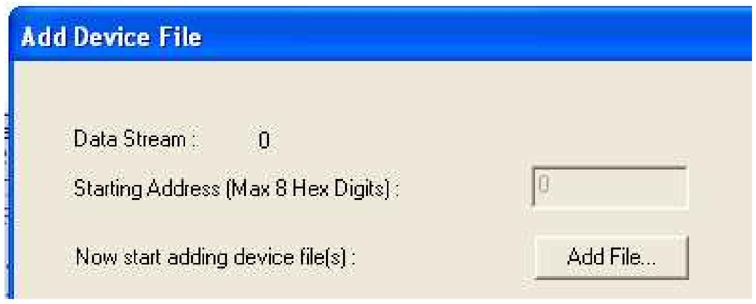

Add Device File
Data Stream : 0
Starting Address (Max 8 Hex Digits) :
0
Now start adding device file(s) :
Add File...

Figure 31 Add Device File Dialog Box

13. Click **No** when you are asked if you would like to add another design file to the data stream.
14. Click **Finish**.
    iMPACT displays the PROM associated with your bit file.
15. When asked to generate a file now, click **Yes**. This creates the PROM file.
16. Exit iMPACT.

17.Next, check that the jumpers on the J8 header ❶❻ are all installed and the jumper on the JP1 header ❸ is set on default on the Spartan III FPGA board.

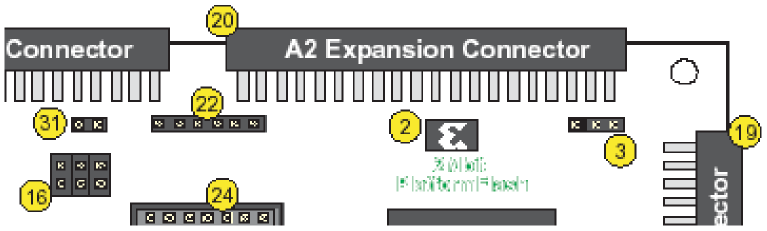


Figure 32 J8 and JP1 header are indicated as numbers 16 and number 3

18.Connect the parallel port end of the JTAG cable ❷❸to the PC's parallel port.

19.Connect the JTAG connector end of the JTAG cable to the J7 header stake pins ❷❷as shown in figure 33.

Make sure that the signals at the end of the JTAG cable align with the labels listed on the board.

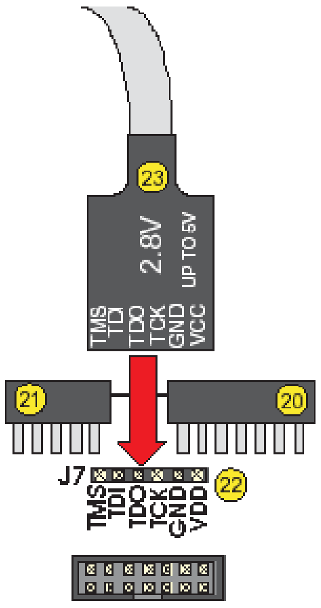


Figure 33 JTAG cable connection on the J8 connector

20.Connect the AC wall adapter +5V DC output to the AC power adapter input ❷❺.

There is no power switch to the board. To disconnect power, remove the AC adapter from the wall or disconnect the dc connector plug. The POWER indicator LED ❷❻ lights up when power is properly applied. And the DONE indicator LED ❶❽ also lights up.

21.Double click **Configure Device (iMPACT**).

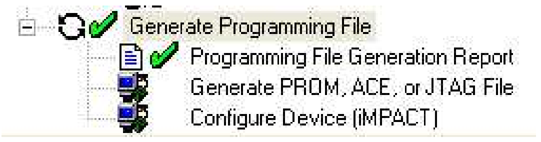
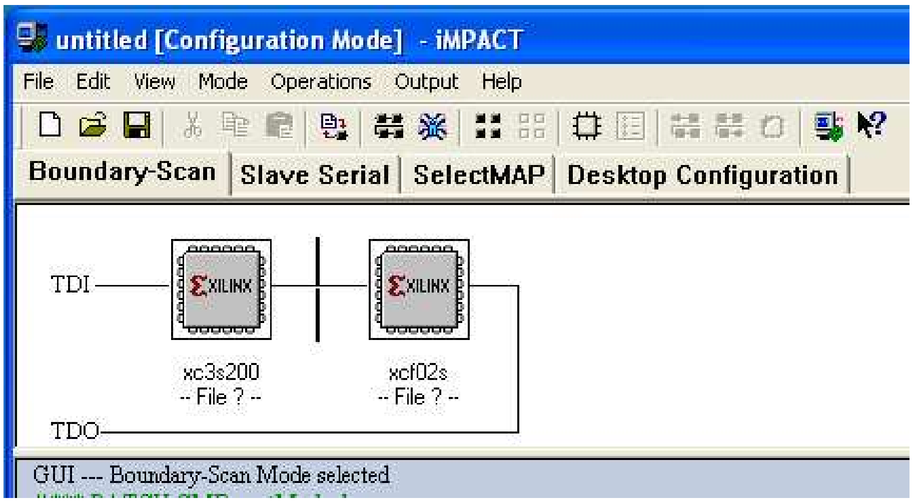

Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)

Figure 34 Generate Programming File hierarchy

iMPACT is launched.

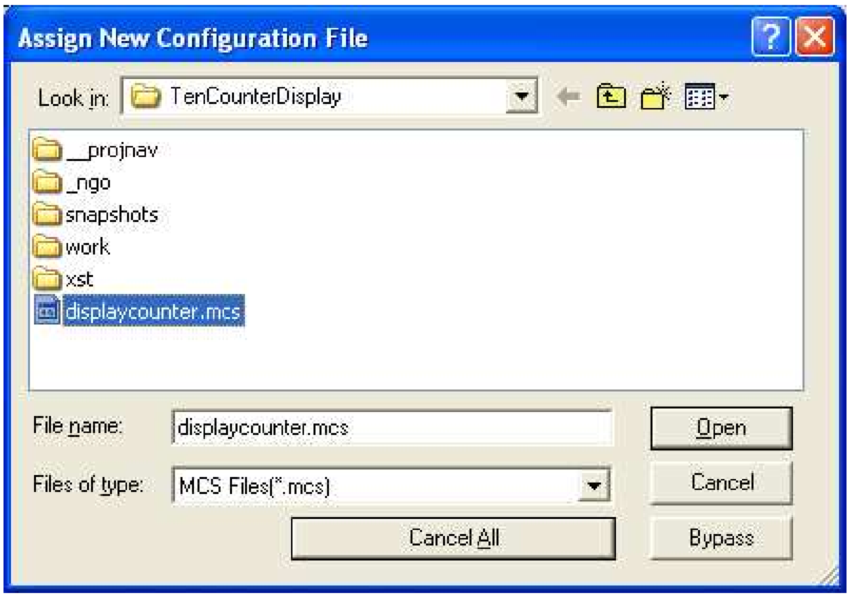
22.In the Configure Devices dialog box, under "`I want to configure device via:`", select **Boundary-Scan Mode**.
23.Click **Next**.
24.In the Boundary-Scan Mode Selection dialog box, select **Automatically connect to cable and identify Boundary-Scan chain**.
25.Click **Finish**.
iMPACT has establishes a connection with the FPGA xc3s200 and xcf02s devices.



GUI --- Boundary-Scan Mode selected

Figure 35 iMPACT established a connection with the devices

The Boundary-Scan Chain Contents Summary dialog box appears saying that there were two devices detected in the boundary-scan chain and that iMPACT will now direct you to associate a programming file with each device, starting with the first.
26.Click **OK**.
27.Right click on the xcf02s device and select **Assign New Configuration File**.
28.In the Assign New Configuration File dialog box, select *displaycounter.mcs* and click **Open**.



File name: displaycounter.mcs
Files of type: MCS Files(*.mcs)

Figure 36 Assign New Configuration dialog box

A new configuration bitstream file has been assigned to the xcf02S device.

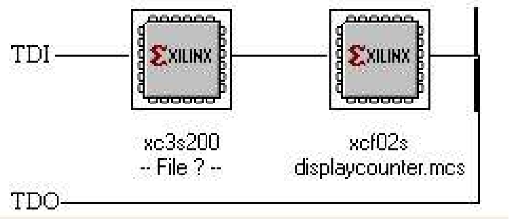


Figure 37 New configuration file is assigned to the xcf02s device

29.Right click on the xcf02s device and select **Program**.
   The Program Options dialog box appears.

30.Make sure that **Erase Before Programming** and **Verify** is checked.
31.Click **OK**.
   The xcf02s device will be erased and programmed.