ISE 6 In-Depth Tutorial

XILINX®

About This Tutorial

About the In-Depth Tutorial

This tutorial gives a description of the features, tools and design flows in ISE 6. The primary focus of this tutorial is to show the relationship among the Xilinx® and third-party design entry, implementation and simulation tools.

This guide is a learning tool for designers who are unfamiliar with the features of the ISE software or those wanting to refresh their skills and knowledge.

You may choose to follow one of three tutorial flows available in this document. For information about the tutorial flows, see "Tutorial Flows."

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this page. You can also directly access some of these resources using the provided URLs.

Resource	Description/URL
Software Manuals	The collection of software manuals is available from the software Help menu (**Help → Online Documentation**) and at http://support.xilinx.com/support/sw_manuals/xilinx6/
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Forums	Discussion groups and chat rooms for Xilinx software users http://toolbox.xilinx.com/cgi-bin/forum

Resource	Description/URL
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx® device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/xlnx/xil_tt_home.jsp

Tutorial Contents

This guide covers the following topics.

- **Chapter 1**, "Overview of ISE and Synthesis Tools," introduces you to the ISE primary user interface, Project Navigator, and the synthesis tools available for your design.
- **Chapter 2**, "HDL-Based Design," guides you through a typical HDL-based design procedure using a design of a runner's stopwatch.
- **Chapter 3**, "Schematic-Based Design," explains many different facets of a schematic-based ISE design flow using a design of a runner's stopwatch. This chapter also shows how to use ISE accessories such as StateCAD™, Project Navigator, CORE Generator™, and ISE Text Editor.
- **Chapter 4**, "Behavioral Simulation," explains how to use the ModelSim Simulator to simulate a design before design implementation and to verify that the logic that you have created is correct.
- **Chapter 5**, "Design Implementation," describes how to Translate, Map, Place, Route (Fit for CPLDs), and generate a Bit file for designs.
- **Chapter 6**, "Timing Simulation," explains how to perform a timing simulation using the post-place & route simulation netlist and the block and routing delay information to give an accurate assessment of the behavior of the circuit under worst-case conditions.

Tutorial Flows

This document contains three tutorial flows. In this section, the three tutorial flows are outlined and briefly described in order to help you determine which sequence of chapters applies to your needs. The tutorial flows include:

- HDL Design Flow
- Schematic Design Flow
- Implementation-only Flow

HDL Design Flow

The HDL Design flow is as follows:

- **Chapter 2**, "HDL-Based Design"
- **Chapter 4**, "Behavioral Simulation"
  Note that behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 5**, "Design Implementation"
- **Chapter 6**, "Timing Simulation"
  Note that timing simulation is optional; however, it is strongly recommended.

Schematic Design Flow

The Schematic Design flow is as follows:

- **Chapter 3**, "Schematic-Based Design"
- **Chapter 4**, "Behavioral Simulation"
  Note that behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 5**, "Design Implementation"
- **Chapter 6**,"Timing Simulation"
  Note that timing simulation is optional; however, it is strongly recommended.

Implementation-only Flow

The Implementation-only flow is as follows:

- **Chapter 5**, "Design Implementation"
- **Chapter 6**, "Timing Simulation"
  Note that timing simulation is optional; however, it is strongly recommended.

Table of Contents

Chapter 1

Overview of ISE and Synthesis Tools

This chapter includes the following sections:

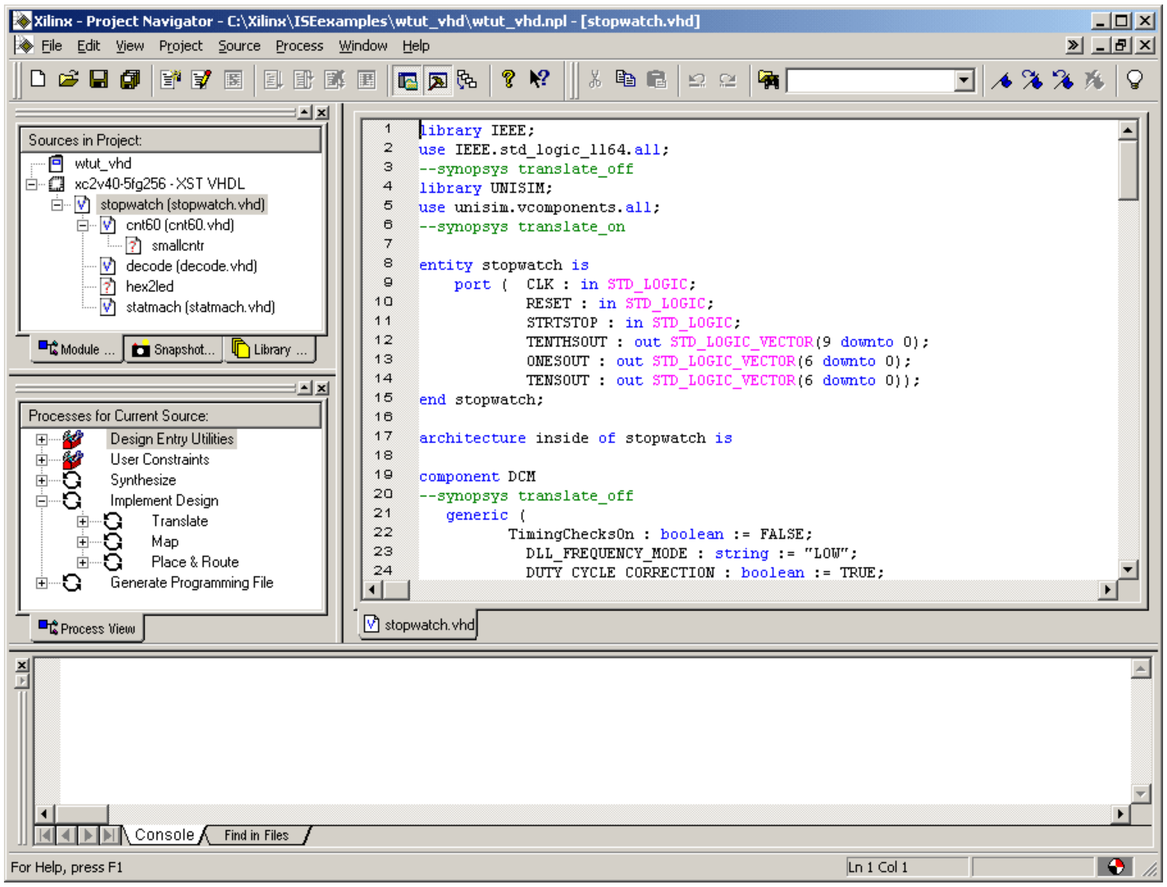
- "Overview of ISE"
- "Overview of Synthesis Tools"

Overview of ISE

ISE controls all aspects of the design flow. Through the Project Navigator interface, you can access all of the various design entry and design implementation tools. You can also access the files and documents associated with your project. Project Navigator maintains a flat directory structure; therefore, you can maintain revision control through the use of snapshots.

Project Navigator Interface

The Project Navigator Interface is divided into four main subwindows, as seen in Figure 1-1. On the top left is the Sources in Project window which hierarchically displays the elements included in the project. Beneath the Sources in Project window is the Processes for Current Source window which displays available processes. The third window at the bottom of the Project Navigator is the Console window which displays status messages, errors, and warnings, and which is updated during all project actions. The fourth window to the right is a multi-document Interface (MDI) window for viewing ascii text files and HDL Bencher™ Waveforms. Each window may be resized, undocked from Project Navigator or moved to a new location within the main Project Navigator window. The default layout can always be restored by selecting **View** → **Restore Default Layout**. These windows are discussed in more detail in the following sections.



```
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 1-1: **Project Navigator**

Sources in Project Window

This window consists of three tabs which provide information for the user. Each tab is discussed in further detail below.

Module View

The Module View tab displays the project name, any user documents, the specified part type and design flow / synthesis tool, and design source files. Each file in the Module View has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). For a complete list of possible source types and their associated icons, see the Project Navigator online help. Select **Help** → **ISE Help Contents**, select the Index tab and click Source / file types.

If a file contains lower levels of hierarchy, the icon has a + to the left of the name. HDL files have this + to show the entities (VHDL) or modules (Verilog) within the file. You can expand the hierarchy by clicking the +. You can open a file for editing by double-clicking on the filename.

Snapshot View

The Snapshot View tab displays all snapshots associated with the project currently open in Project Navigator. A snapshot is a copy of the project including all files in the working directory, and synthesis and simulation subdirectories. A snapshot is stored with the project for which is was taken, and can be viewed in the Snapshot View. You can view the reports, user documents, and source files for all snapshots. All information displayed in the Snapshot View is read-only. Using snapshots provides an excellent version control system, enabling subteams to do simultaneous development on the same design.

Note: Remote sources are not copied with the snapshot. A reference is maintained in the snapshot.

Library View

The Library View tab displays all libraries associated with the project open in Project Navigator.

Processes for Current Source Window

This window contains the Process View tab.

Process View

The Process View tab is context sensitive and changes based upon the source type selected in the Sources for Project window. From the Process View tab, you can run the functions necessary to define, run and view your design. The Process Window provides access to the following functions:

- **Design Entry Utilities**

  Provides access to symbol generation, instantiation templates, HDL Converter, View Command Line Log File, Launch MTI, and simulation library compilation.

- **User Constraints**

  Provides access to editing location and timing constraints.

- **Synthesis**

  Provides access to Check Syntax, synthesis, View RTL Schematic, and synthesis reports. This varies depending on the synthesis tools you use.

- **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.

The Processes for Current Source window incorporates automake technology. This enables the user to select any process in the flow and the software automatically runs the processes necessary to get to the desired step. For example, when you run the Implementation process, Project Navigator also runs the synthesis process because implementation is dependent on up-to-date synthesis results.

Note: To view a running log of command line arguments in the Console window, expand Design Entry Utilities and select View Command Line Log File. See the Using Command Line section of Chapter 5, "Design Implementation" for further details.

Console Window

The Console window displays errors, warnings, and informational messages. Errors are signified by a red box next to the message, while warnings have a yellow box. Warning and Error messages may also be viewed separately from other console text messages by selecting either the Warnings or Errors tab at the bottom of the console window.

Error Navigation to Source

You can navigate from a synthesis error or warning message in the Console window to the location of the error in a source HDL file. To do so, select the error or warning message, right-click the mouse, and from the menu select **Goto Source**. The HDL source file opens and the cursor moves to the line with the error.

Error Navigation to Solution Record

You can navigate from an error or warning message in the Console window to the relevant solution records on the support.xilinx.com website. These type of errors or warnings can be identified by the web icon to the left of the error. To navigate to the solution record, select the error or warning message, right-click the mouse, and from the menu select **Goto Solution Record**. The default web browser opens and displays all solution records applicable to this message.

MDI Window

In the multi-document interface (MDI) window, you can access the ISE Text Editor, the ISE Language Templates, and HDL Bencher™.

Text Editor

Source files and other text documents can be opened in a user designated editor. The editor is determined by the setting found by selecting **Edit → Preferences, Editor tab.** The default editor is the ISE Text Editor. ISE Text Editor enables you to edit source files and to access the ISE Language Templates, which is a catalog of ABEL, Verilog and VHDL language and User Constraint File templates. You can use and modify these templates for your own design

HDL Bencher

HDL Bencher is a PC-based test bench and test fixture creation tool integrated in the Project Navigator framework. HDL Bencher can be used to graphically enter stimuli and the expected response, then generate a VHDL test bench or Verilog test fixture. For future details see the "Creating a Test Bench Waveform Using HDL Bencher" in Chapter 4.

Using Snapshots

Snapshots enable you to maintain revision control over the design. A snapshot contains a copy all of the files in the project directory. See also "Snapshot View."

Creating a Snapshot

To create a snapshot:

1. Select **Project → Take Snapshot**.
2. In the Take a Snapshot of the Project dialog box, enter the snapshot name and any comments associated with the snapshot.

In the Snapshot View, the snapshot containing all of the files in the project directory along with project settings is displayed.

Restoring a Snapshot

Since snapshots are read-only, a snapshot must be restored in order to continue work. When you restore a snapshot, it replaces the project in your current session. To restore a snapshot:

1. In the Snapshot View, select the snapshot.
2. Select **Project → Make Snapshot Current**.

Before the snapshot replaces the current project, you must place the current project in a snapshot so that your work is not lost

Viewing a Snapshot

The Snapshot View contains a list of all the snapshots available in the current project. To review a process report or verify process status within a snapshot:

1. Expand the snapshot source tree and select the desired source file.
2. Right-click the mouse over the desired process report.
3. From the menu, select **Open Without Updating**.

Using Project Archives

You can also archive the entire project into a single compressed file. This allows for easier transfer over email and storage of numerous projects in a limited space.

Creating an Archive

To create an archive:

1. Select **Project → Archive**.
2. In the Create Zip Archive dialog box, enter the archive name and location.

The archive contains all of the files in the project directory along with project settings. Remote sources are not zipped up into the archive.

Restoring an Archive

You cannot restore an archived file directly into Project Navigator. The compressed file can be extracted with any ZIP utility and you can then open the extracted file in Project Navigator.

Overview of Synthesis Tools

You can synthesize your design using three synthesis tools. The following section lists the devices supported by each synthesis tool and includes some process properties information.

Xilinx Synthesis Technology (XST)

This synthesis tool is part of the ISE package and is available for both an HDL- or Schematic-based design flow.

Supported Devices

- Virtex™/-E /-II, Virtex -II Pro™
- Spartan™-II /-IIE/-3
- XC9500™ /XL/XV
- Coolrunner™ /XPLA3/-II

Process Properties

Process properties enable you to control the synthesis results of XST. Two commonly used properties are Optimization Goal and Optimization Effort. Using these properties you can control the synthesis results for area or speed, and the amount of time the synthesizer runs.

For more information, see the *XST User Guide*. This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

Synplify/Synplify Pro

This synthesis tool is not part of the ISE package and is not available unless purchased separately. This synthesis tool is not available for a schematic-based design.

Supported Devices

- Virtex™/-E /-II, Virtex -II Pro™
- Spartan™-II /-IIE/-3
- XC9500™ /XL/XV
- Coolrunner™ /XPLA3/-II

Process Properties

Process properties enable you to control the synthesis results of Synplify/Synplify Pro. Most of the commonly used synthesis options available in the Synplify/Synplify Pro stand-alone version are available for Synplify/Synplify Pro synthesis through ISE.

For more information about the specific synthesis options, see the Synplify / Synplify Pro online help.

LeonardoSpectrum

This synthesis tool is not part of the ISE package and is not available unless purchased separately. Two commonly used properties are Optimization Goal and Optimization Effort. Using these properties you can control the synthesis results for area or speed and the amount of time the synthesizer runs. This synthesis tool is available for both an HDL-based and Schematic-based design flow.

Supported Devices

- Virtex™ / -E / -II, Virtex -II Pro™
- Spartan™-II / -IIE / -3
- XC9500™ / XL / XV
- Coolrunner™ / XPLA3 / -II

Process Properties

Process properties enable you to control the synthesis results of LeonardoSpectrum. Most of the commonly used synthesis options available for the LeonardoSpectrum stand-alone version are available for LeonardoSpectrum synthesis through ISE.

For more information, see the LeonardoSpectrum online help.

Chapter 2

HDL-Based Design

This chapter includes the following sections:

Overview of HDL-Based Design

This chapter guides you through a typical HDL-based design procedure using a design of a runner's stopwatch. The design example used in this tutorial demonstrates many device features, software features, and design flow practices you can apply to your own design. This design targets a Virtex™-II device; however, all of the principles and flows taught are applicable to any Xilinx® device family, unless otherwise noted.

The design is composed of HDL elements and a CORE Generator™ macro. You can synthesize the design using Xilinx Synthesis Technology (XST), LeonardoSpectrum, or Synplify.

This chapter is the first in the "HDL Design Flow." It is followed by Chapter 4, "Behavioral Simulation", in which you simulate the HDL code using the ModelSim Simulator. In Chapter 5, "Design Implementation", you will implement the design using the Xilinx Implementation Tools, and generate a bitstream.

For an example of how to design with CPLDs, see the *ISE Software Interactive Tutorial for Xilinx CPLDs* http://support.xilinx.com/support/techsup/tutorials/index.htm.

Getting Started

The following sections describe the basic requirements for running the tutorial.

Required Software

To perform this tutorial, you must have the following software and software components installed:

- Xilinx® Series ISE 6.x
- ModelSim (necessary for Behavioral and Timing Simulation)
- Virtex™-II libraries and device files

Note: For detailed software installation instructions, refer to the *ISE Installation Guide and Release Notes.* This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

This tutorial assumes that the software is installed in the default location *c:\xilinx*. If you have installed the software in a different location, substitute *c:\xilinx* with your installation path.

Optional Software Requirements

The following third-party synthesis tools are incorporated into this tutorial, and may be used in place of the XST synthesis tool:

- Synplify/Synplify PRO 7.3 (or above)
- LeonardoSpectrum 2003.a (or above)

VHDL or Verilog?

This tutorial supports both VHDL and Verilog designs, and applies to both designs simultaneously, noting differences where applicable. You will need to decide which HDL language you would like to work through for the tutorial, and download the appropriate files accordingly. Starting with the 6.1i release, XST can now synthesize a mixed-language design. However, this tutorial does not go over the mixed language feature.

Installing the Tutorial Project Files

The Stopwatch tutorial projects can be downloaded from http://support.xilinx.com/support/techsup/tutorials/tutorials6.htm. Download either the VHDL or the Verilog design flow project files.

After you have downloaded the tutorial project files from the web, unzip the tutorial projects in the *c:\xilinx* directory, and replace any existing files.

After you unzip the tutorial project files in *c:\xilinx*, the directory *wtut_vhd* (for a VHDL design flow) or *wtut_ver* (for a Verilog design flow) is created within *c:\xilinx\ISExamples*, and the tutorial files are copied into the directories.

The following table lists the locations of both the complete and incomplete projects.

Note: The runner's stopwatch design is referred to a Watch for brevity.

Table 2-1: **Tutorial Project Directories**

Directory	Description
wtut_vhd	Incomplete Watch Tutorial - VHDL
wtut_ver	Incomplete Watch Tutorial - Verilog
watchvhd	Solution for Watch - VHDL
watchver	Solution for Watch - Verilog

Note: Do not overwrite any files in the solutions directories.

The *watchvhd* and *watchver* solution projects contain the design files for the completed tutorials, including HDL files and the bitstream file. To conserve disk space, some intermediate files are not provided.

Starting the ISE Software

Once the tutorial files are downloaded and unzipped, start the ISE software and open the design project file.

To launch the ISE software package:

1. Double-click the ISE Project Navigator icon on your desktop or select **Start** → **Programs** → **Xilinx ISE** → **Project Navigator**.

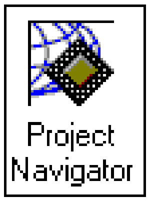


Figure 2-1: **Project Navigator Desktop Icon**

2. From Project Navigator, select **File → Open Project**. The Open Project dialog box appears.

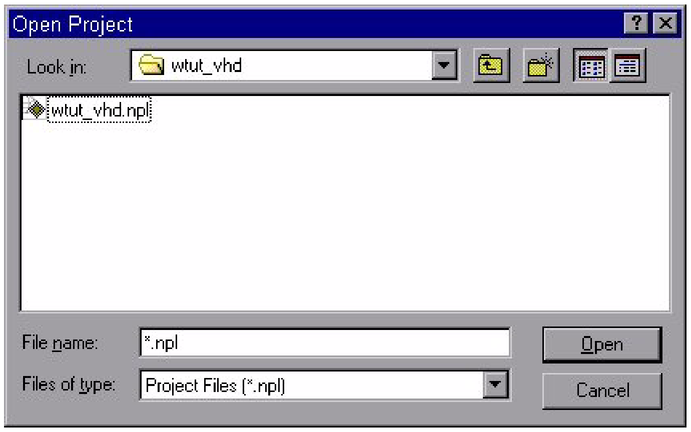


Figure 2-2: **Getting Started Dialog Box**

3. In the Directories list, browse to *c:\xilinx\ISEexamples\wtut_vhd* or *c:\xilinx\ISEexamples\wtut_ver.*
4. Double-click *wtut_vhd.npl* (VHDL design entry) or *wtut_ver.npl* (Verilog design entry).

Stopping the Tutorial

You may stop the tutorial at any time and save your work by selecting **File → Save All**.

Design Description

The design used in this tutorial is a hierarchical, HDL-based design, which means that the top-level design file is an HDL file that references several other lower-level macros. The lower-level macros are either HDL modules or CORE Generator™ modules.

The design begins as an unfinished design. Throughout the tutorial, you will complete the design by generating some of the modules from scratch and by completing others from existing files. When the design is complete, simulate it to verify the design's functionality.

The tutorial design is a simple runner's stopwatch. There are three external inputs and three external output buses in the completed design. The system clock is an externally generated signal. The following list summarizes the input lines and output buses.

Note: Throughout this tutorial, the runner's stopwatch design that you are working on is referred to as Watch.

Inputs

The following are input signals for the tutorial Watch design.

- **STRTSTOP**

  Starts and stops the stopwatch. This is an active low signal which acts like the start/stop button on a runner's stopwatch.

- **RESET**

  Resets the stopwatch to 00.0 after it has been stopped.
- **CLK**

  Externally generated system clock.

Outputs

The following are outputs signals for the design.

- **TENSOUT[6:0]**

  7-bit bus which represents the tens digit of the stopwatch value. This bus is in 7-segment display format viewable on the 7-segment LED display.
- **ONESOUT[6:0]**

  Similar to TENSOUT bus above, but represents the ones digit of the stopwatch value.
- **TENTHSOUT[9:0]**

  10-bit bus which represents the tenths digit of the stopwatch value. This bus is one-hot encoded.

Functional Blocks

The completed design consists of the following functional blocks.

- **STATMACH**

  State Machine module defined and implemented in StateCAD™.
- **CNT60**

  HDL-based module which counts from 0 to 59, decimal. This macro has two 4-bit outputs, which represent the ones and tens digits of the decimal values, respectively.
- **TENTHS**

  CORE Generator™ 4-bit binary encoded counter. This macro outputs a 4-bit code which is decoded to represent the tenths digit of the watch value as a 10-bit one-hot encoded value.
- **HEX2LED**

  HDL-based macro. This macro decodes the ones and tens digit values from hexadecimal to 7-segment display format.
- **SMALLCNTR**

  A simple Counter.
- **DECODE**

  Decodes the CORE Generator output from 4-bit binary to a 10-bit one-hot output.
- **DCM1**

  Clocking Wizard macro with internal feedback and duty-cycle correction.

Design Entry

For this hierarchical design, you will examine HDL files, correct syntax errors, create an HDL macro, and add a CORE Generator™ module, and you will create and use each type of design macro. All procedures used in the tutorial can be used later for your own designs.

With the *wtut_vhd.npl* or *wtut_ver.npl* project open in Project Navigator, the Sources in Project window displays all of the source files currently added to the project, with the associated entity or module names (see Figure 2-3). In the current project, *smallcntr* and *hex2led* are instantiated, but the associated entity or module is not defined in the project. Instantiated components with no entity or module declaration are displayed with a red question-mark.

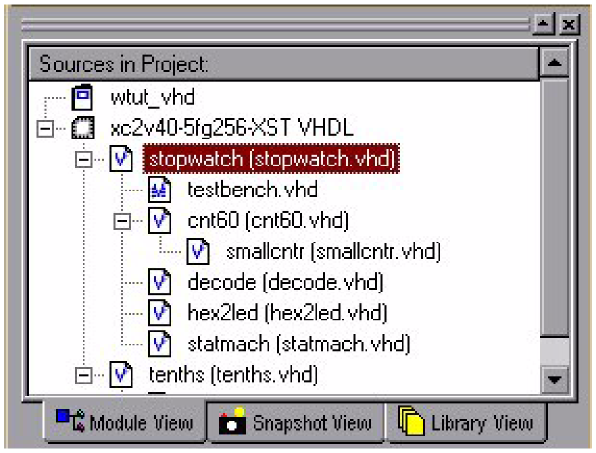


Figure 2-3: **Sources in Project Window**

Adding Source Files

HDL files must be added to the project before they can be synthesized. Four HDL files have already been added to this project. One file must still be added.

1. Select *stopwatch.vhd* or *stopwatch.v* in the Sources in Project window.

   Upon selecting the HDL file, the Processes for Current Source window displays all processes available for this file.

   Next, add the remaining HDL file to the project.

2. Select **Project** → **Add Source**.
3. Select *smallcntr.vhd* or *smallcntr.v* from the project directory.
4. In the Choose Source Type dialog box, select **Verilog/VHDL Module**.
5. Click **OK**.

The red question-mark (?) for smallcntr should change to a V.

Figure 2-4: **smallcntr.vhd file in Sources in Project window**

Checking the Syntax

To check the syntax of source files:

1. Select *stopwatch.vhd* or *stopwatch.v* in the Sources in Project window.

   Upon selecting the HDL file, the Processes for Current Sources in Project window displays all processes available for this file.
2. Double-click **Check Syntax** in the Synthesize hierarchy.

Note: Check Syntax is not available when Synplify is selected as the synthesis tool.

Correcting HDL errors

The SMALLCNTR 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:

1. Double-click on the error message in the console window.
2. Correct any errors in the HDL source file. The comments next to the error explain this simple fix.
3. Select **File** → **Save** to save the file.
4. Re-analyze the file by selecting the HDL file and double-clicking **Check Syntax** in the Synthesize hierarchy.

Creating an HDL-Based Module

Next, create a module from HDL code. With ISE, you can easily create modules from HDL code using the ISE Text Editor tool. The HDL code is then connected to your top-level HDL design through instantiation and is compiled with the rest of the design.

Now, you will author a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 7-segment LED display format.

Using the New Source Wizard and ISE Text Editor

In order to create the module, in the New Source Wizard, create a file and specify the name and ports of the component. The resulting “skeleton” HDL file is then modified further in the ISE Text Editor.

To create the source file:

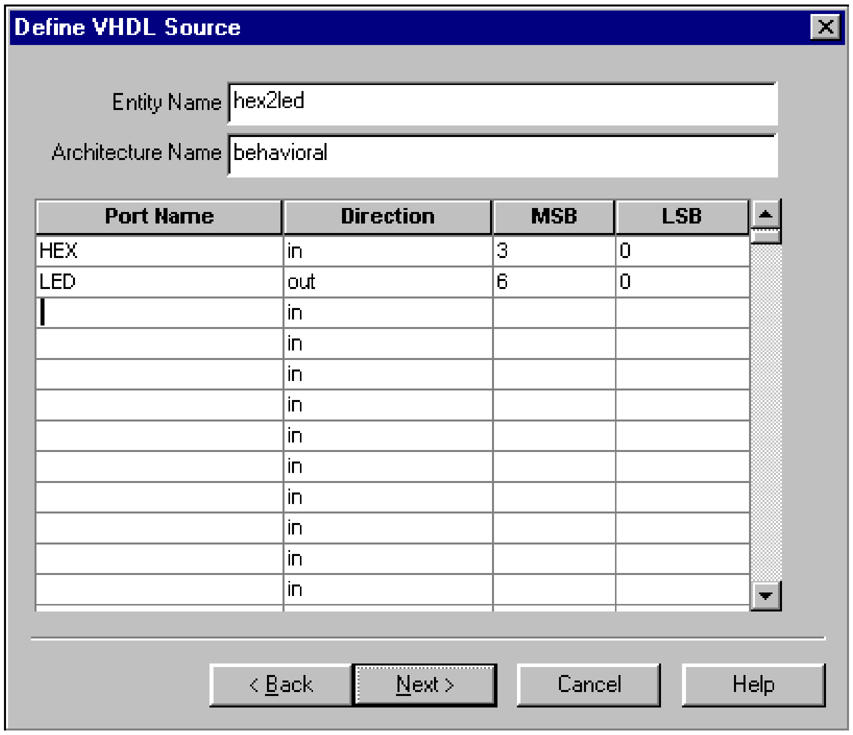
1. Select **Project** → **New Source**.

   A dialog box opens in which you specify the type of source you want to create.
2. Select **VHDL Module** or **Verilog Module**.
3. In the File Name field, type ‘hex2led’.
4. Click **Next**.

The *hex2led* component has a 4-bit input port named hex and a 7-bit output port named led. To enter these ports:

1. Click in the Port Name field and type **HEX**.
2. Click in the Direction field and set the direction to **in**.

3. In the MSB field enter **3**, and in the LSB field enter **0**. Refer to Figure 2-5.



Entity Name: hex2led

Architecture Name: behavioral

Port Name	Direction	MSB	LSB
HEX	in	3	0
LED	out	6	0
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Figure 2-5: **New Source Wizard for VHDL**

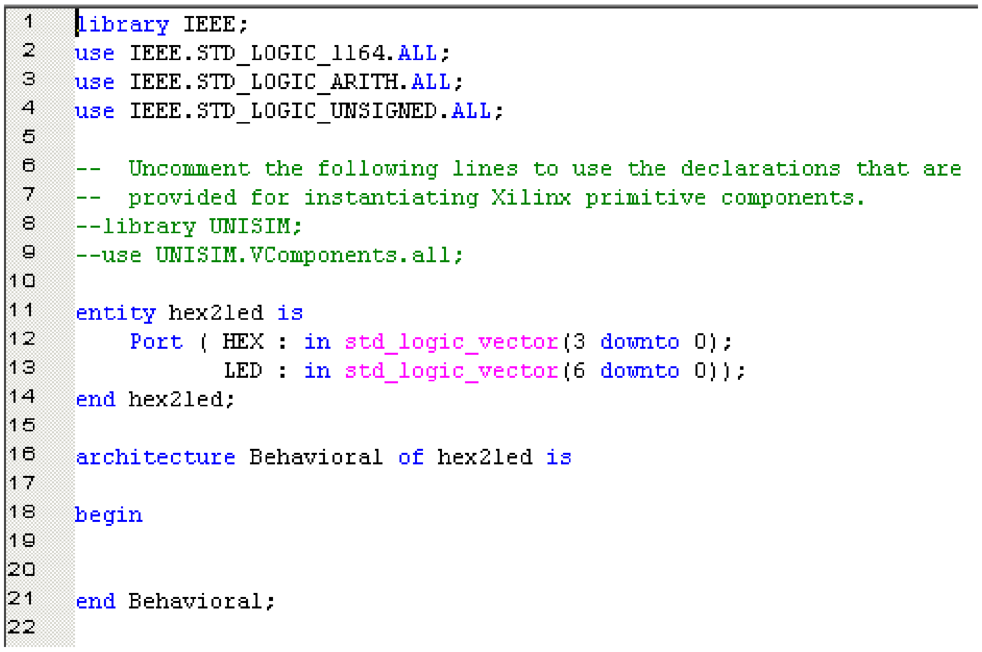
Repeat the previous steps for the **LED[6:0]** output bus. Be sure that the direction is set to **out**.

4. Click **Next** to complete the Wizard session.

   A description of the module displays.

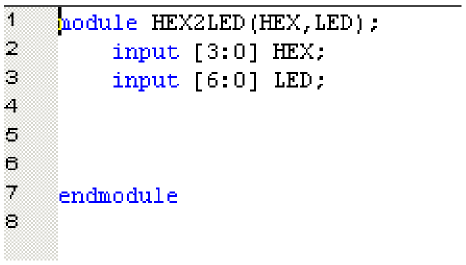
5. Click **Finish** to open the empty HDL file in the ISE Text Editor.

The VHDL file is found in Figure 2-6. The Verilog HDL file is found in Figure 2-7.



```
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 hex2led is
    Port ( HEX : in std_logic_vector(3 downto 0);
           LED : in std_logic_vector(6 downto 0));
end hex2led;

architecture Behavioral of hex2led is

begin


end Behavioral;

```

Figure 2-6: **Skeleton VHDL File in the ISE Text Editor**



```
module HEX2LED(HEX,LED);
    input [3:0] HEX;
    input [6:0] LED;



endmodule

```

Figure 2-7: **Skeleton Verilog File in the ISE Text Editor**

In the ISE Text Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are displayed in blue, data types in red, comments in green, and values in black. This color-coding enhances readability and recognition of typographical errors.

Using the Language Templates

The ISE Language Templates include HDL 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 value to 7-segment LED display format.

Note: You can add your own templates to the Language Templates for components or constructs you use often.

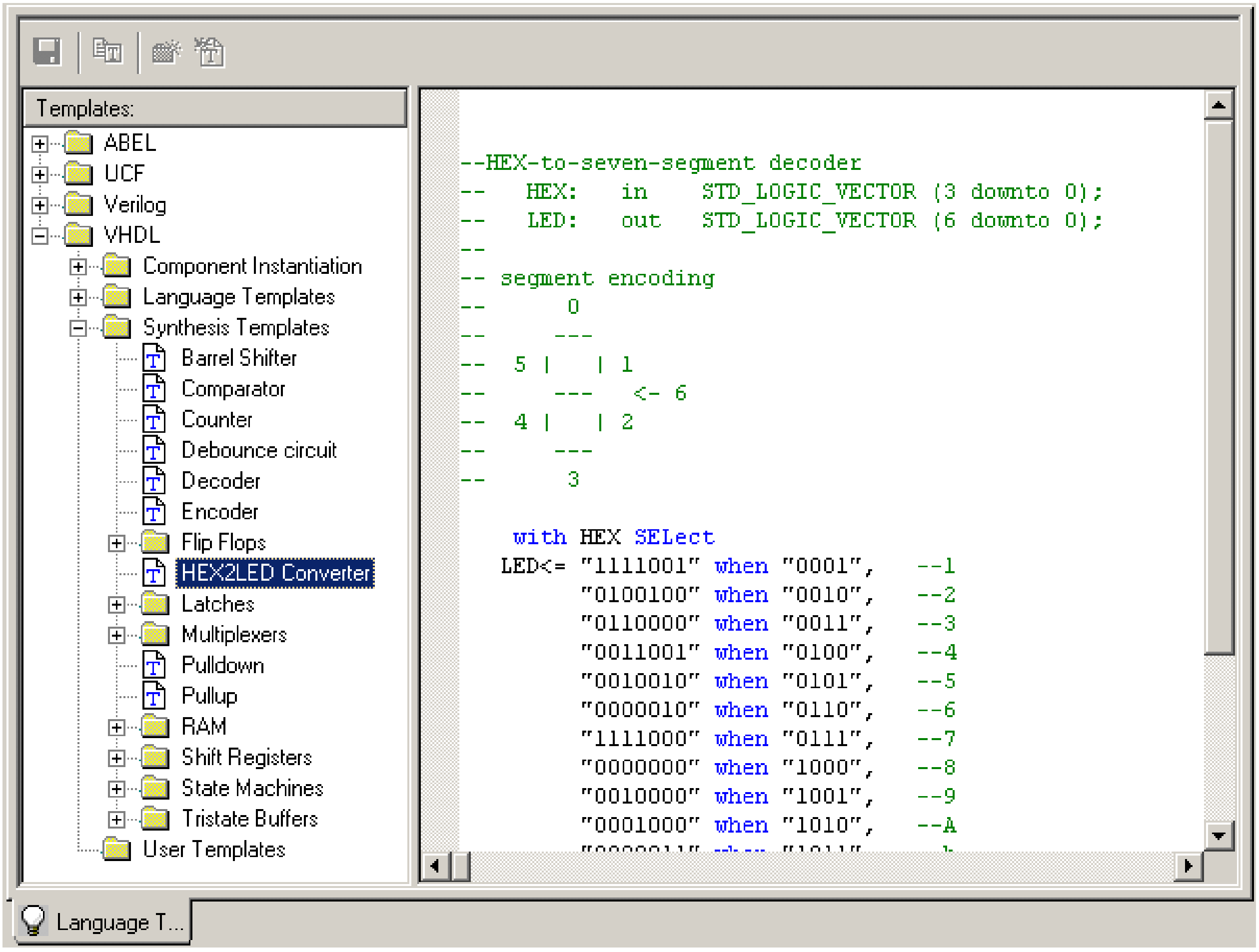
To invoke the Language Templates and select the template for this tutorial:

1. From Project Navigator, select **Edit → Language Templates**.

   Each HDL language in the Language Templates is divided into four sections: Component Instantiations, Language Templates, Synthesis Templates, and User Templates. To expand the view of any of these sections, click the + next to the topic. Click any of the listed templates to view the template contents in the right-hand pane.

2. Under either the VHDL or Verilog hierarchy, expand the Synthesis Templates hierarchy and select the template called **HEX2LED Converter**. Use the appropriate template for the language you are using.

3. To preview the HEX2LED Converter template, click the template in the hierarchy. 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 encoding
--       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
```

Figure 2-8: **Language Templates**

Adding the Language Template to Your File

You will now use the drag and drop method for adding templates to your HDL file. A copy and paste function is also available from the Language Templates Edit Menu and the right-click menu.

To add the template to your HDL file using the drag and drop method:

1. In the Language Templates, click and drag the HEX2LED Converter name into the *hex2led.vhd* file under the architecture statement, or the *hex2led.v* file under the module declaration.
2. Close the Language Templates window.
3. (Verilog only) After the input and output statements and before the HEX2LED converter that you just added, add the following line of code to the HDL file to allow an assignment:

   ```
   reg [6:0] LED;
   ```

   You now have complete and functional HDL code.
4. Save the file by selecting **File** → **Save**.
5. Select *hex2led* in the Sources in Project window and double-click **Check Syntax** under Synthesize in the Processes for Current Source window.
6. Exit the ISE Text Editor.

Creating a CORE Generator Module

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 pre-optimize 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 create a CORE Generator module called Tenths. Tenths is a 4-bit binary encoded counter. The 4-bit number is decoded to count the tenths digit of the stopwatch's time value.

Creating the CORE Generator Module

Create the CORE Generator module using the New Source Wizard in Project Navigator. This invokes CORE Generator in which you can select and define the type of module you want.

To create the module:

1. In Project Navigator, select **Project** → **New Source**.
2. Select **IP (CoreGen & Architecture Wizard)** as the source type.
3. Enter 'tenths' in the File Name field.
4. Click **Basic Elements** → **Counters** → **Binary Counters**.
5. Click **Next** and then **Finish**.
6. Fill in the Binary Counter dialog with the following settings:
   - Component Name: **tenths**

     Defines the name of the module.
   - Output Width: **4**

     Defines the width of the output bus.
   - Operation: **Up**

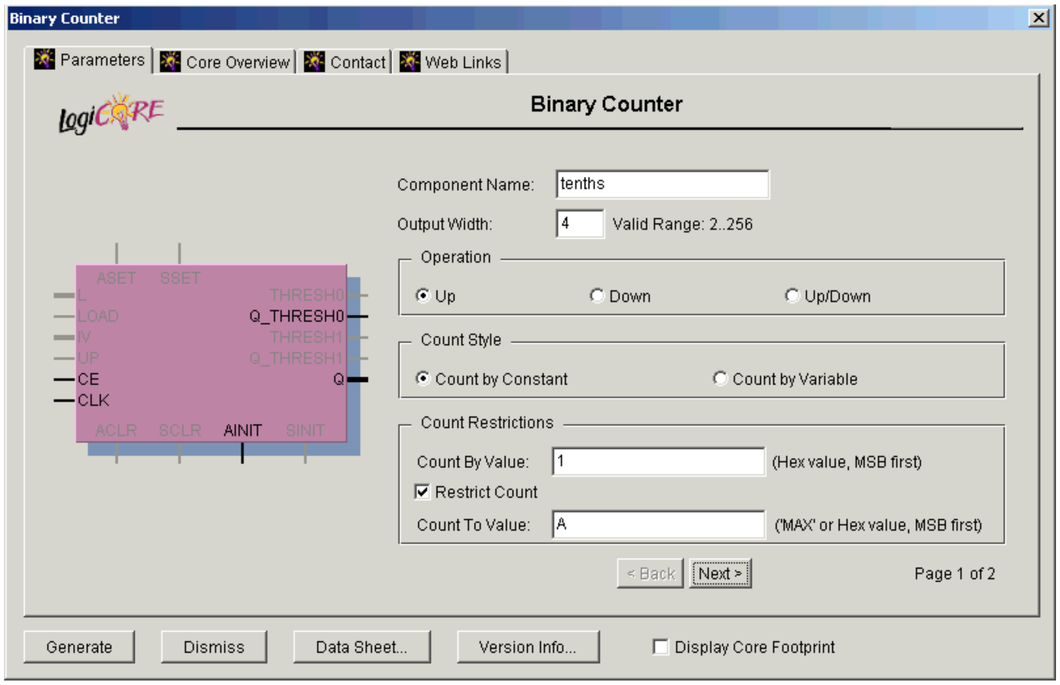
     Defines how the counter will operate. This field is dependent on the type of module you select.

- Count Style: **Count by Constant**

  Allows counting by a constant or a user supplied variable.

- Count Restrictions: **Enable and Count To Value A (HEX)**

  This dictates the maximum count value.



Component Name: tenths
Output Width: 4 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: A ('MAX' or Hex value, MSB first)
< Back | Next > | Page 1 of 2
Generate | Dismiss | Data Sheet... | Version Info... | Display Core Footprint

Figure 2-9: **CORE Generator Module Selector**

7. Select the **Next** button.
8. Continue to fill in the Binary Counter dialog with the following settings:
   - Threshold Options: **Threshold 0 set to A**

     Signal goes high when the value specified has been reached.

   - Threshold Options: **Registered**
9. Click the **Register Options** button to open the Register Options dialog box.
10. In the Register Options dialog box, enter the following settings:
    - Clock Enable: **Selected**
    - Asynchronous Settings: **Init with a value of 1**
    - Synchronous Settings: **None**
11. Click **OK**.
12. Check that *only* the following pins are used (used pins will be highlighted on the model symbol to the left side of the CORE Generator™ window):
    - **AINIT**
    - **CE**
    - **Q**
    - **Q_THRESH0**
    - **CLK**

13. Click **Generate**.

The module is created and automatically added to the project library.

A number of other files are added to the project directory. These files are:

- *tenths.sym*

  This is a schematic symbol file.
- *tenths.edn*

  This file is the netlist that is used during the Translate phase of implementation.
- *tenths.vho* or *tenths.veo*

  This is the instantiation template that is used to incorporate the CORE Generator™ module in your source HDL.
- *tenths.vhd* or *tenths.v*

  These are simulation-only files.
- *tenths.xco*

  This file stores the configuration information for the Tenths module and is used as a project source.
- *coregen.prj*

  This file stores the CORE Generator configuration for the project.

14. Click **Dismiss**.

Instantiating the CORE Generator Module in the HDL Code

Next, instantiate the CORE Generator module in the HDL code using either a VHDL flow or a Verilog flow.

VHDL Flow

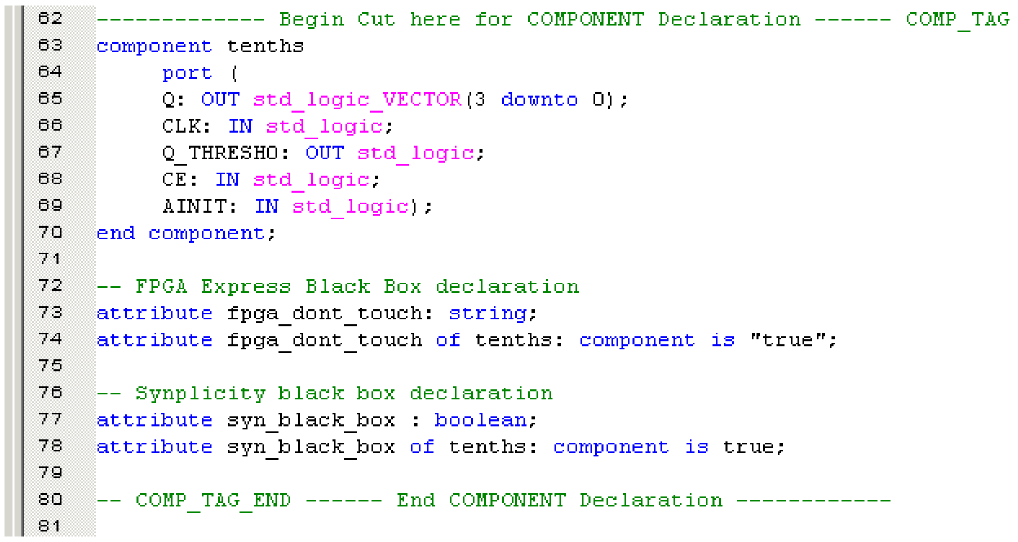
To instantiate the CORE Generator module using a VHDL flow:

1. In Project Navigator, double-click *stopwatch.vhd* to open the file in ISE Text Editor.
2. Place your cursor after the line that states:

   ```
   “-- Insert Coregen Counter Component Declaration”
   ```
3. Select **Edit** → **Insert File** and choose *Tenths.vho*.

   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 tenths
     port (
     Q: OUT std_logic_VECTOR(3 downto 0);
     CLK: IN std_logic;
     Q_THRESH0: OUT std_logic;
     CE: IN std_logic;
     AINIT: IN std_logic);
end component;

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

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

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

Figure 2-10: **VHDL Component Declaration of CORE Generator Module**

4. Highlight the inserted code from

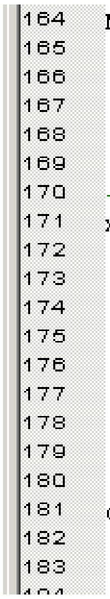
   `“-- Begin Cut here for INSTANTIATION Template”`

   to

   `“AINIT=>AINIT);”`
5. Select **Edit** → **Cut**.
6. Place the cursor after the line that states:

   `“--Insert Coregen Counter Instantiation”`
7. Select **Edit** → **Paste** to place the instantiation here.
8. Change “your_instance_name” to XCOUNTER.

9. Edit this instantiated code to connect the signals in the Stopwatch design to the ports of the CORE Generator™ module as shown in Figure 2-11.



```
MACHINE:statmach port map(CLK=>clk_dcm,
                          RESET=>RESET,
                          STRTSTOP=>strtstopinv,
                          CLKEN=>clkenable,
                          RST=>rstint);

-- Insert Coregen Counter Instantiation.
xcounter : tenths
      port map (
         Q => Q,
         CLK => CLK_dcm,
         Q_THRESH0 => xtermcnt,
         CE => clkenable,
         AINIT => rstint);



decoder: decode port map (
         binary => Q,
         one_hot => xcountout);
```

Figure 2-11: **VHDL Component Instantiation of the CORE Generator Module**

10. Save the design using **File** → **Save**, and close the ISE Text Editor.

Verilog Flow

To instantiate the CORE Generator module using a Verilog flow:

1. In Project Navigator, double-click *stopwatch.v* to open the file in the ISE Text Editor.
2. Select **File** → **Open** and open the tenths.veo file.
3. Changes the **Files of Type** from **Source Files** to **All Files**.
4. Select *tenths.veo*.
5. Highlight the inserted code in *tenths.veo* from

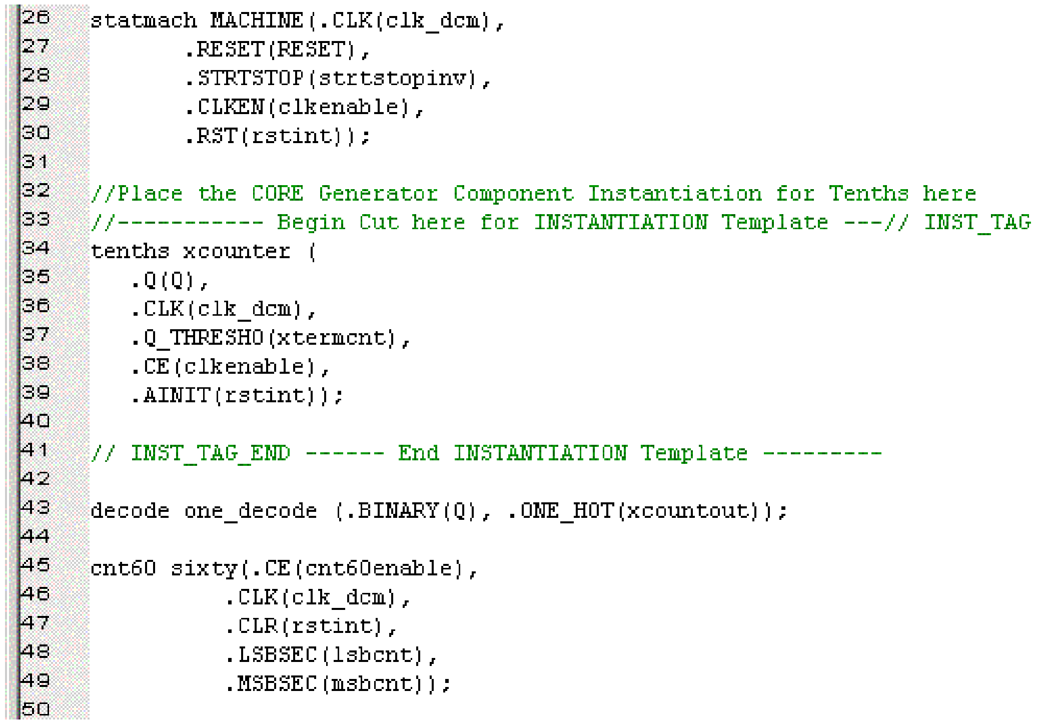
   `“//----------- Begin cut here for INSTANTIATION TEMPLATE ---//”`

   to

   `“// INST_TAG_END ------ End INSTANTIATION Template”`
6. Select **Edit** → **Copy**.
7. Place the cursor after the line in *stopwatch.v* that states:

   `“// Place the Coregen Component Instantiation for Tenths here.”`
8. Select **Edit** → **Paste** to place the instantiation here.
9. Change “YourInstanceName” to XCOUNTER.

10. Edit this code to connect the signals in the Stopwatch design to the ports of the CORE Generator™ module as shown in Figure 2-12.



```
statmach MACHINE(.CLK(clk_dcm),
       .RESET(RESET),
       .STRTSTOP(strtstopinv),
       .CLKEN(clkenable),
       .RST(rstint));

//Place the CORE Generator Component Instantiation for Tenths here
//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG
tenths xcounter (
   .Q(Q),
   .CLK(clk_dcm),
   .Q_THRESH0(xtermcnt),
   .CE(clkenable),
   .AINIT(rstint));

// INST_TAG_END ------ End INSTANTIATION Template ---------

decode one_decode (.BINARY(Q), .ONE_HOT(xcountout));

cnt60 sixty(.CE(cnt60enable),
          .CLK(clk_dcm),
          .CLR(rstint),
          .LSBSEC(lsbcnt),
          .MSBSEC(msbcnt));

```

Figure 2-12: **Verilog Component Instantiation of the CORE Generator Module**

11. Save the design using **File** → **Save** and close *stopwatch.v* in the ISE Text Editor.

Creating a DCM Module

The Clocking Wizard, a Xilinx® Architecture Wizard, enables you to graphically select Digital Clock Manager (DCM) features that you wish to use. In this section, create a basic DCM module with CLK0 feedback and duty-cycle correction.

Using DCM Wizard

To create the DCM1 module:

1. In Project Navigator, select **Project** → **New Source**.
2. In the New Source dialog box, select **IP (CoreGen & Architecture Wizard)** and type ‘dcm1’ for the File Name.
3. Click **Next**.
4. In the Select Core Type window, select **Clocking** → **Single DCM**.
5. Click **Next,** then **Finish.** The Clocking Wizard is launched
6. Verify that RST, CLK0 and LOCKED are selected.
7. Type 50 for the Input Clock Frequency.

8. Verify the following settings:
   - CLKIN Source: **External**
   - Feedback Source: **Internal**
   - Feedback Value: **1X**
   - Phase Shift: **None**
   - Duty Cycle Correction: **Yes**
9. Select the **Advanced** button.
10. Select Wait for DCM lock before DONE signal goes high.
11. Select **OK** and **Next**.

    An informational message displays the locked signal and the STARTUP_WAIT option.
12. Select **Finish**.

dcm1.xaw is added to the list of project source files in the Sources in Project window.

Instantiating the DCM1 Macro - VHDL Design

Next, instantiate the DCM1 macro for your VHDL or Verilog design. To instantiate the DCM1 macro for the VHDL design:

1. In Project Navigator, in the Sources in the Project window, select *dcm1.xaw*.
2. Double-click **View HDL Instantiation Template** under the Design Entry Utilities in the Processes for Source window.
3. From the newly opened HDL Instantiation Template copy the component declaration template:

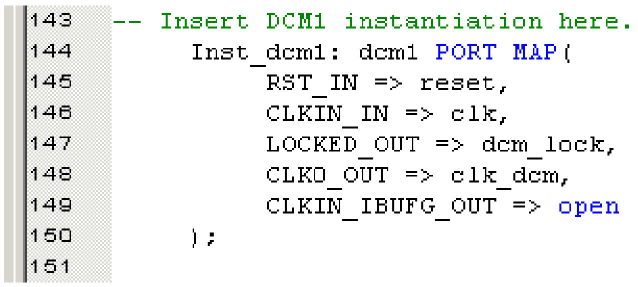
```
COMPONENT dcm1
PORT (
      RST_IN : IN std_logic;
      CLKIN_IN : IN std_logic;
      LOCKED_OUT : OUT std_logic;
      CLK0_OUT : OUT std_logic;
      CLKIN_IBUFG_OUT : OUT std_logic
      );
END COMPONENT;
```

4. Paste the component declaration into the section in *stopwatch.vhd* labeled
   `-- Insert DCM1 component declaration here.`
5. Copy the instantiation template from the newly opened HDL Instantiation Template:

```
Inst_dcm1: dcm1 PORT MAP(
     RST_IN => ,
     CLKIN_IN => ,
     LOCKED_OUT => ,
     CLK0_OUT => ,
     CLKIN_IBUFG_OUT =>
);
```

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

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



```
-- Insert DCM1 instantiation here.
    Inst_dcm1: dcm1 PORT MAP(
         RST_IN => reset,
         CLKIN_IN => clk,
         LOCKED_OUT => dcm_lock,
         CLK0_OUT => clk_dcm,
         CLKIN_IBUFG_OUT => open
    );
```

Figure 2-13: **VHDL Instantiation for dcm1**

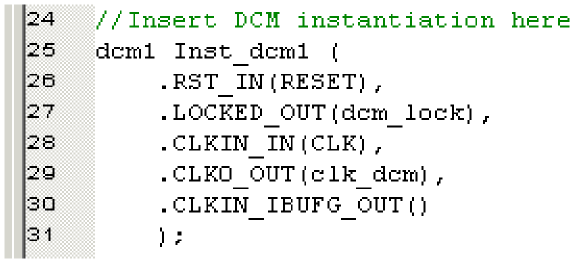
Instantiating the DCM1 Macro - Verilog

To instantiate the DCM1 macro for your Verilog design:

1. In Project Navigator, in the Sources in the Project window, select *dcm1.xaw*.
2. Double-click **View HDL Instantiation Template** under Design Entry Utilities in the Processes for Source window.
3. From the newly opened HDL Instantiation Template, copy the instantiation template:

```
dcm1 instance_name (
     .RST_IN(RST_IN),
     .LOCKED_OUT(LOCKED_OUT),
     .CLKIN_IN(CLKIN_IN),
     .CLK0_OUT(CLK0_OUT),
     .CLKIN_IBUFG_OUT(CLKIN_IBUFG_OUT)
     );
```

4. Paste the instantiation template into the section in stopwatch.v labeled `//Insert DCM1 instantiation here.`
5. Make the necessary changes as shown in the figure below.



```
//Insert DCM instantiation here
dcm1 Inst_dcm1 (
     .RST_IN(RESET),
     .LOCKED_OUT(dcm_lock),
     .CLKIN_IN(CLK),
     .CLK0_OUT(clk_dcm),
     .CLKIN_IBUFG_OUT()
     );
```

Figure 2-14: **Verilog Instantiation for dcm1**

Synthesizing the Design

So far you have used XST for verifying syntax. Next, you will synthesize the design. The synthesis tool uses the design's HDL code and generates a supported netlist type (EDIF or NGC for the Xilinx® implementation tools).

The synthesis tools perform three general steps (although all synthesis tools further break down these general steps) to create the netlist:

- **Analyze / Check Syntax**

  Checks the syntax of the source code.
- **Compile**

  Translates and optimizes the HDL code into a set of components that the synthesis tool can recognize.
- **Map**

  Translates the components from the compile stage into the target technology's primitive components.

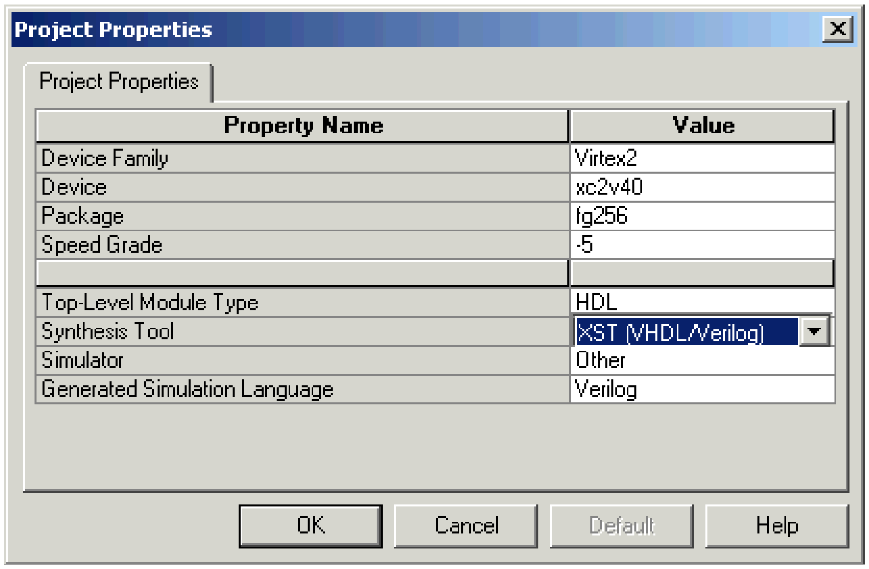
The synthesis tool of the Watch design project is XST; however, you can change the tool at any time during the design flow. Changing the design flow results in the deletion of implementation data. You have not yet created any implementation data.

For projects that contain implementation data, Xilinx recommends that you take a snapshot of the project before changing the synthesis tool to preserve this data. For more information about taking a snapshot, see "Creating a Snapshot" in Chapter 1.

A summary of available synthesis tools is available in "Overview of Synthesis Tools" in Chapter 1.

To change the synthesis tool:

1. Select the targeted part in the Sources in Project window.
2. Select **Source** → **Properties**.
3. In the Project Properties dialog box, click the Synthesis Tool value and use the pull-down arrow to select the desired synthesis tool from the list.



Property Name	Value
Device Family	Virtex2
Device	xc2v40
Package	fg256
Speed Grade	-5
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Other
Generated Simulation Language	Verilog

Figure 2-15: **Specifying Synthesis Tool**

Next, perform design synthesis using one of the following tools:

- "Synthesizing the Design using XST"
- "Synthesizing the Design using Synplify / Synplify Pro"
- "Synthesizing the Design using LeonardoSpectrum"

Synthesizing the Design using XST

Now that you have created and analyzed the design, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

Processes available for synthesis using XST are as follows:

- **View Synthesis Report**

  Gives a mapping and timing summary as well as synthesis optimizations that took place.
- **View RTL Schematic**

  Generates a schematic of your HDL code and is accessible from the synthesis tool process hierarchy.
- **Analyze Hierarchy**

  Sets up the HDL in its hierarchical order.
- **Check Syntax**

  Verifies that the HDL code is entered properly.

Entering Constraints

XST supports a User Constraint File (UCF) style syntax to define synthesis and timing constraints. Xilinx strongly suggests that you use this syntax style for your new designs.

Note: ISE supports the old constraint syntax without any further enhancements for this release of XST. In future, support will be dropped.

This syntax style format is called the Xilinx Constraint File (XCF). The XCF must have an .xcf file extension. XST uses this extension to determine if the syntax is related to the new or old style. Please note that if the extension is not .xcf, XST will interpret it as the old constraint style.

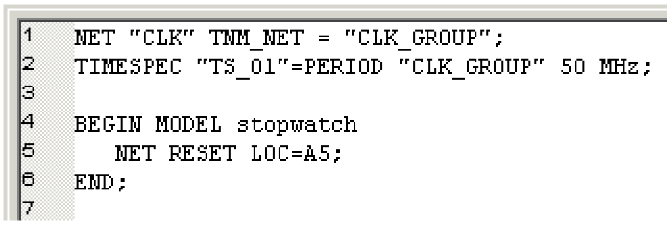
To create a new Xilinx Constraint File:

1. Select **Project** → **New Source**.
2. In the New Source dialog box, select **User Document** as the source type, and enter the file name 'stopwatch.xcf'.
3. Select **Next**, and **Finish**.

   The new XCF file launches in the ISE Text Editor.
4. In the new XCF document, type in the following:

```
NET "CLK" TNM_NET = "CLK_GROUP";
TIMESPEC "TS_01"=PERIOD "CLK_GROUP" 50 MHz;
BEGIN MODEL stopwatch
    NET RESET LOC = A5;
END;
```

5. Select **File** → **Save**.



```
NET "CLK" TNM_NET = "CLK_GROUP";
TIMESPEC "TS_01"=PERIOD "CLK_GROUP" 50 MHz;

BEGIN MODEL stopwatch
   NET RESET LOC=A5;
END;
```

Figure 2-16: **Contents of stopwatch.xcf**

Note: For more constraint options in the implementation tools, see "Editing Constraints in the Constraints Editor" and "Editing Constraints in the Pinout Area Constraints Editor (PACE)" in Chapter 5, "Design Implementation."

Entering Synthesis Options

Synthesis options enable you to modify the behavior of the synthesis tool to make optimizations according to the needs of the design. One commonly used option is to control synthesis to make optimizations based on area or speed. Other options include controlling the maximum fanout of a signal from a flip-flop or setting the desired frequency of the design.

To enter synthesis options:

1. Select *stopwatch.vhd* (or *stopwatch.v*) in the Sources in Project window.
2. Right-click on the **Synthesize** process and select **Properties**.
3. Under the **Synthesis Options** tab click in the **Synthesis Constraints File** property field and select *stopwatch.xcf*.
4. Check the **Write Timing Constraints** box.
5. Click **OK**.

Synthesizing the Design

Now, you are ready to synthesize your design. To take the HDL code and generate a compatible netlist:

1. Select *stopwatch.vhd* (or *stopwatch.v*).
2. Double-click the **Synthesize** process in the Processes for Current Source window.

   Note: This step can also be done by selecting *stopwatch.vhd* (or *stopwatch.v*), clicking **Synthesize** in the Processes for Current Source window, and selecting **Process** → **Run**.

The RTL Viewer

XST can generate a schematic representation of the HDL 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:

1. In Project Navigator, click + next to Synthesize to expand the process hierarchy.
2. Double-click **View RTL Schematic**.

The RTL Viewer (part of Engineering Capture System (ECS) tool) displays the schematic. Right-click on the schematic to view various options for the schematic viewer.

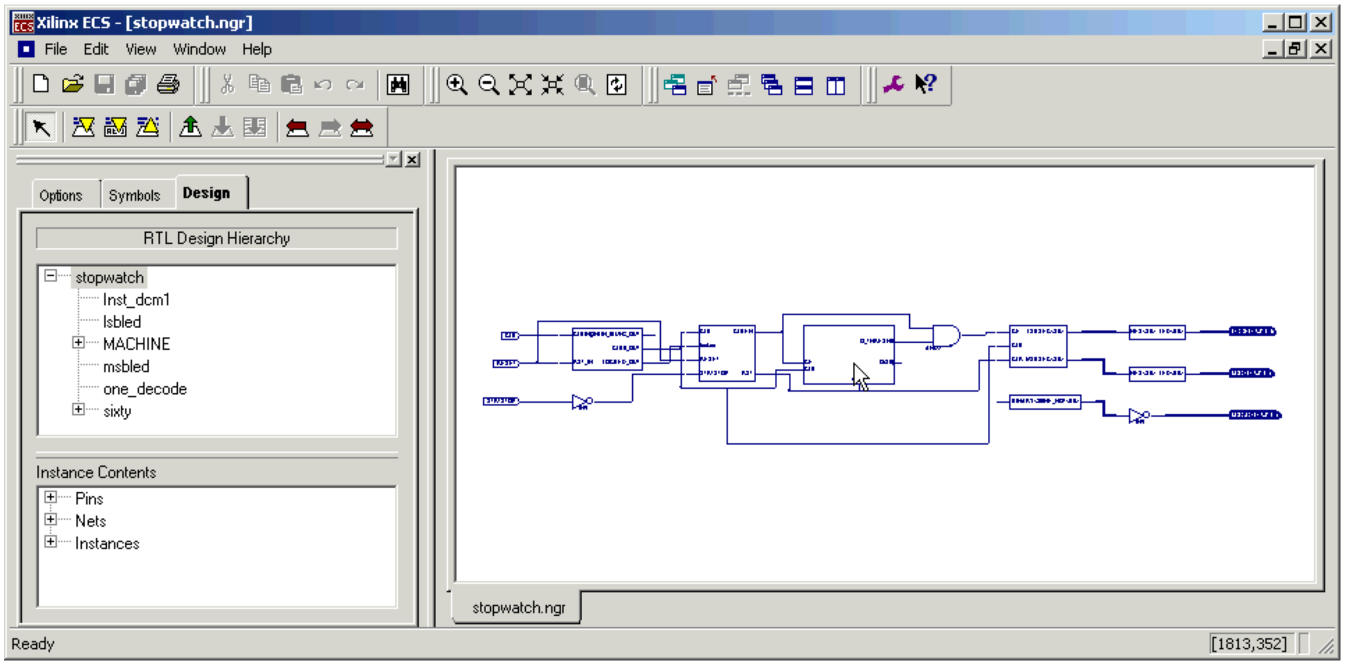


Figure 2-17: **ECS RTL Viewer**

You have completed XST synthesis. At this point, an NGC file exists for the Stopwatch design. Go to:

- Chapter 4, "Behavioral Simulation" to perform a pre-synthesis simulation of this design.
- Chapter 5, "Design Implementation" to place and route the design.
- Chapter 6, "Timing Simulation" for post-place and route simulation.

Note: For more information about XST constraints, options, reports, or running XST from the command line, see the *XST User Guide.* This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

Synthesizing the Design using Synplify/Synplify Pro

Now that you have entered and analyzed the design, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture. To access Synplify's RTL viewer and constraints editor you must run Synplify outside of ISE.

To synthesize the design, set the global synthesis options:

1. Select *stopwatch.vhd* (or *stopwatch.v*).
2. If you are following the Verilog flow, add the following to the top of *stopwatch.v*:

   `‘include "c:\<path_to_synp>\<synp_ver>\ ib\xilinx\virtex2.v"`
3. Right-click **Synthesize** in the Processes for Current Source window.
4. From the menu, select **Properties**.
5. Set the Default Frequency to **50MHz**, and check the **Write Vendor Constraint File** box.

6. Click **OK** to accept these values.
7. Select *stopwatch.vhd* (or *stopwatch.v*) and double-click the **Synthesize** process to run synthesis.

   Note: This step can also be done by selecting *stopwatch.vhd* (or *stopwatch.v*), clicking **Synthesize** in the Processes for Current Source window, and selecting **Process** → **Run**.

Examining Synthesis Results

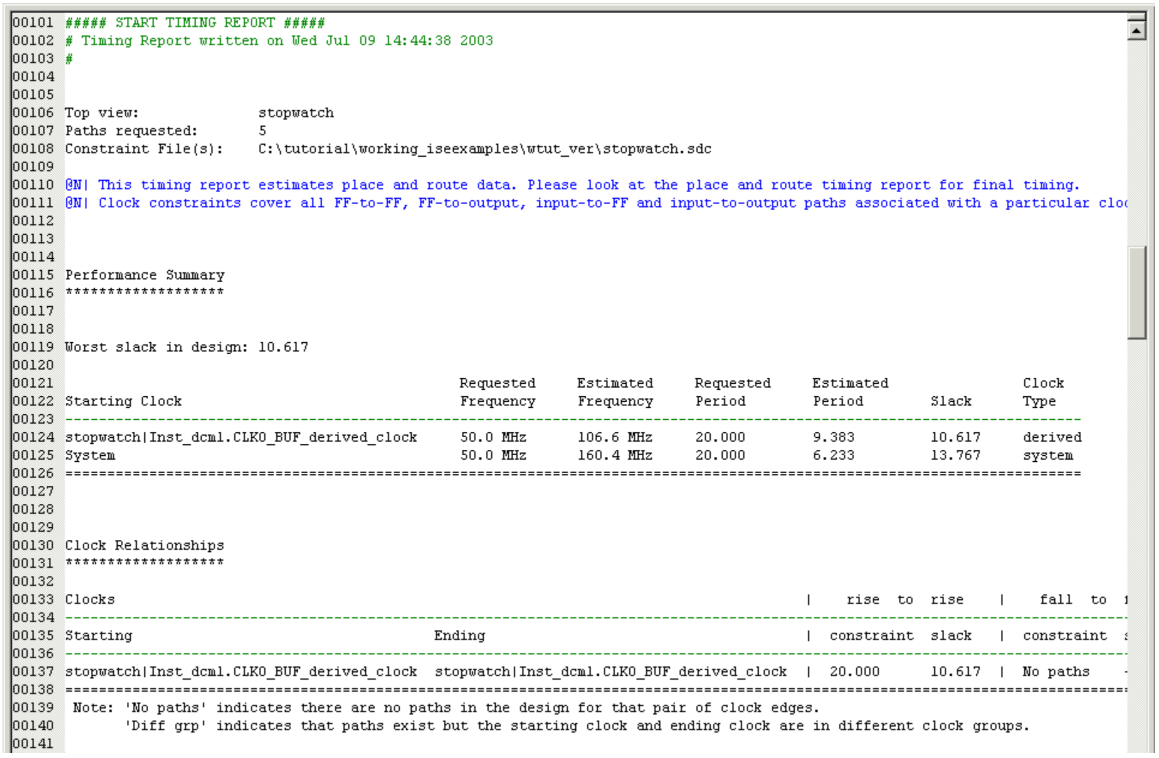
To view overall synthesis results, double-click **View Synthesis Report** under the **Synthesize** process. The report consists of the following three sections:

- "Compiler Summary"
- "Timing Report"
- "Mapping Report"

Compiler Summary

The compiler summary gives a brief report on the analysis, compile and mapping stages run by Synplify on the HDL design. Each of the summary reports provide the errors and warnings that are associated with each file.

Note: Black boxes (modules not read into Synplify's design environment) are always noted as Unbound in the Synplify reports. As long as the underlying netlist (NGO, NGC or EDN) for a black box exists in the project directory, the Implementation tools merge the netlist into the design during the Translate phase. Since the Tenths module was built using CORE Generator™ called from the project, the tenths EDN file is found.



```
##### START TIMING REPORT #####
# Timing Report written on Wed Jul 09 14:44:38 2003
#


Top view:                stopwatch
Paths requested:         5
Constraint File(s):      C:\tutorial\working_iseexamples\wtut_ver\stopwatch.sdc

@N| This timing report estimates place and route data. Please look at the place and route timing report for final timing.
@N| Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF and input-to-output paths associated with a particular cloc



Performance Summary
*******************


Worst slack in design: 10.617

                                              Requested     Estimated     Requested     Estimated                   Clock
Starting Clock                                Frequency     Frequency     Period        Period         Slack        Type
------------------------------------------------------------------------------------------------------------------------
stopwatch|Inst_dcml.CLK0_BUF_derived_clock    50.0 MHz      106.6 MHz     20.000        9.383          10.617       derived
System                                        50.0 MHz      160.4 MHz     20.000        6.233          13.767       system
========================================================================================================================



Clock Relationships
*******************

Clocks                                                                              |    rise  to  rise    |    fall  to
-------------------------------------------------------------------------------------------------------------------------
Starting                                    Ending                                  |  constraint  slack   |  constraint
-------------------------------------------------------------------------------------------------------------------------
stopwatch|Inst_dcml.CLK0_BUF_derived_clock  stopwatch|Inst_dcml.CLK0_BUF_derived_clock  |  20.000      10.617  |  No paths
=========================================================================================================================
 Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
       'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

```

Figure 2-18: **Synplify's Estimated Timing Data**

Timing Report

The timing report section details information on the constraints that you entered and on delays on parts of the design that had no constraints. The delay values are based on wireload models, and therefore, are considered preliminary. Consult the post-place and route timing reports discussed in Chapter 5, “Design Implementation,” for the most accurate delay information.

Mapping Report

The mapping report lists all of the components used for the design, such as LUTs, flip-flops, and block RAMs.

You have now completed Synplify synthesis. At this point, an netlist EDN file exists for the Stopwatch design.

- To perform a pre-synthesis simulation of this design, see Chapter 4, “Behavioral Simulation.”
- To place and route the design, see Chapter 5, “Design Implementation.”
- To perform post-place and route simulation, see Chapter 6, “Timing Simulation.”

Synthesizing the Design using LeonardoSpectrum

Now that you have entered and analyzed the design, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

Processes available in LeonardoSpectrum synthesis include:

- **Check Syntax**

  Checks the syntax of the HDL code.
- **Modify Constraints**

  Launches the LeonardoSpectrum tool to enable you to enter constraints.
- **View Synthesis Report**

  Lists the synthesis optimizations that were performed on the design and gives a brief timing and mapping report.
- **View Synthesis Summary**

  Gives a detailed map and timing report with no information on the synthesis optimizations.
- **View RTL Schematic**

  Accessible from the Launch Tools hierarchy, this process displays LeonardoSpectrum with a schematic-like view of your HDL code.
- **View Technology Schematic**

  Accessible from the Launch Tools hierarchy, this process displays LeonardoSpectrum with a schematic-like view of your HDL code mapped to the primitives associated with the target technology.

- **View Critical Path Schematic**

  Accessible from the Launch Tools hierarchy, this process displays LeonardoSpectrum with a schematic-like view of the critical path of your HDL code mapped to the primitives associated with the target technology.

Modifying Constraints

LeonardoSpectrum enables you to enter constraints to control optimization options and pass timing specifications to the implementation tools. All timing specifications are stored in the netlist constraints file (NCF) which is used by the implementation tools. Some of the timing constraints are used by the synthesis engine to produce better synthesis results for the place and route tools.

To modify constraints:

1. Expand the **Synthesize** process hierarchy.
2. Double-click on the **Modify Constraints** process.

   LeonardoSpectrum displays. First time users of the LeonardoSpectrum tool launch LeonardoSpectrum in ’Quick Setup’ mode.
3. Click on the Advanced Flow icon as shown below.

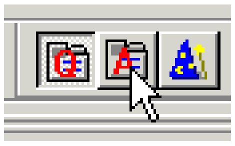


Figure 2-19: **LeonardoSpectrum Advanced Flow Icon**

4. Click the **Constraints** tab.

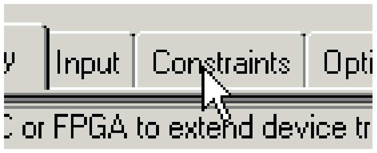


Figure 2-20: **LeonardoSpectrum Constraints Tab**

The constraints sub-tabs are as follows.

- **Global**

  Enables you to enter constraints that affect all of your design: PERIOD, OFFSETs and pad-to-pad type constraints. The constraints entered here modify LeonardoSpectrum’s run script only. A constraints file is not generated.
- **Clock**

  Enables you to enter a more detailed clock constraint accounting for pulse width and duty cycle as well as the period. The constraints entered here modify LeonardoSpectrum’s run script only. A constraints file is not generated.
- **Input**

  Enables you to specify constraints that affect the input ports such as arrival time, fanout, pin location, and pad type.

- **Output**

  Enables you to specify constraints that affect the output ports such as required time, pin location, and pad type.

- Signal

  Enables you to specify individual signal constraints such as preserve signal, a low skew constraint, and a max fanout constraint.

- **Module**

  Enables you to instruct the synthesis tool to synthesize a module differently then the rest of the design.

- **Path**

  Enables you to create false and multicycle paths.

- **Report**

  Enables you to generate a current report of constraints that have been entered.

For this tutorial, in the Constraints tab enter the following constraints:

1. Select the **Input** sub-tab.
2. Select the **Reset** input pad.
3. In the Pin Location field, enter A5.
4. Click **Apply**.
5. Select the **Report** sub-tab, and check that the constraints were applied.
6. In order to get LeonardoSpectrum to write out a constraints file (.ctr), select any tab (the Technology tab for example).

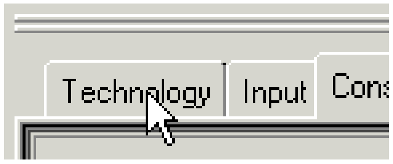


Figure 2-21: **LeonardoSpectrum Technology Tab**

7. Save the constraints file to the default name *stopwatch.ctr*.
8. Exit LeonardoSpectrum.

Note: For more constraint options in the implementation tools, see "Editing Constraints in the Constraints Editor" and "Editing Constraints in the Pinout Area Constraints Editor (PACE)" in Chapter 5, "Design Implementation."

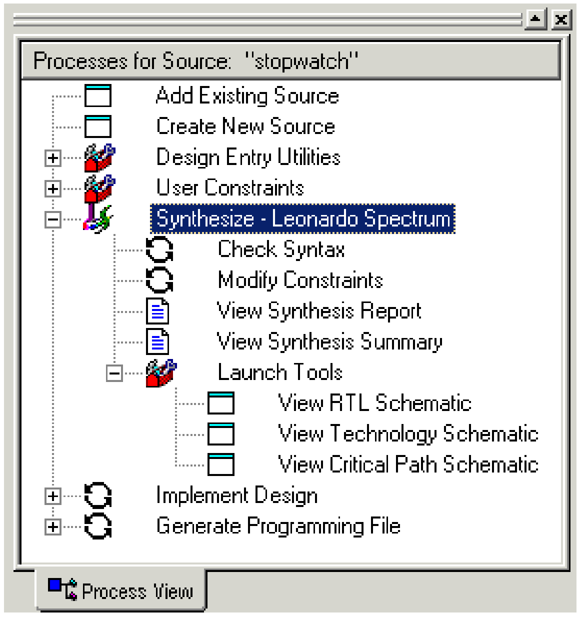
Entering Synthesis Options through ISE

Synthesis options enable you to modify the behavior of the synthesis tool to optimize according to the needs of the design. One option is to control synthesis by optimizing based on area or speed. Other options include controlling the maximum fanout of a signal from a flip-flop or setting the desired frequency of the design.

For this tutorial, set the global synthesis options:

1. Select *stopwatch.vhd* (or *stopwatch.v*).
2. Right-click the **Synthesis** process.
3. From the menu, select **Properties**.

4. Click the **Synthesis Options** tab, and set the Default Frequency to **50MHz**.
5. Click the **Netlist Options** tab, and ensure that the Do Not Write NCF box is unchecked.
6. Click the **Constraint File Options** tab, and select the *stopwatch.ctr* file created in LeonardoSpectrum, in the "Modifying Constraints" section above.
7. Click **OK** to accept these values.
8. Select *stopwatch.vhd* (or *stopwatch.v*) and double-click the **Synthesize** process in the Processes for Source window.



Processes for Source: "stopwatch"

- Add Existing Source
- Create New Source
- Design Entry Utilities
- User Constraints
- Synthesize - Leonardo Spectrum
  - Check Syntax
  - Modify Constraints
  - View Synthesis Report
  - View Synthesis Summary
  - Launch Tools
    - View RTL Schematic
    - View Technology Schematic
    - View Critical Path Schematic
- Implement Design
- Generate Programming File

Figure 2-22: **LeonardoSpectrum Synthesis Processes**

The RTL/Technology Viewer

LeonardoSpectrum can generate a schematic representation of the HDL 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 LeonardoSpectrum has inferred. To launch the design in LeonardoSpectrum's RTL viewer, double-click the **View RTL Schematic** process. The following figure displays the design in an RTL view.

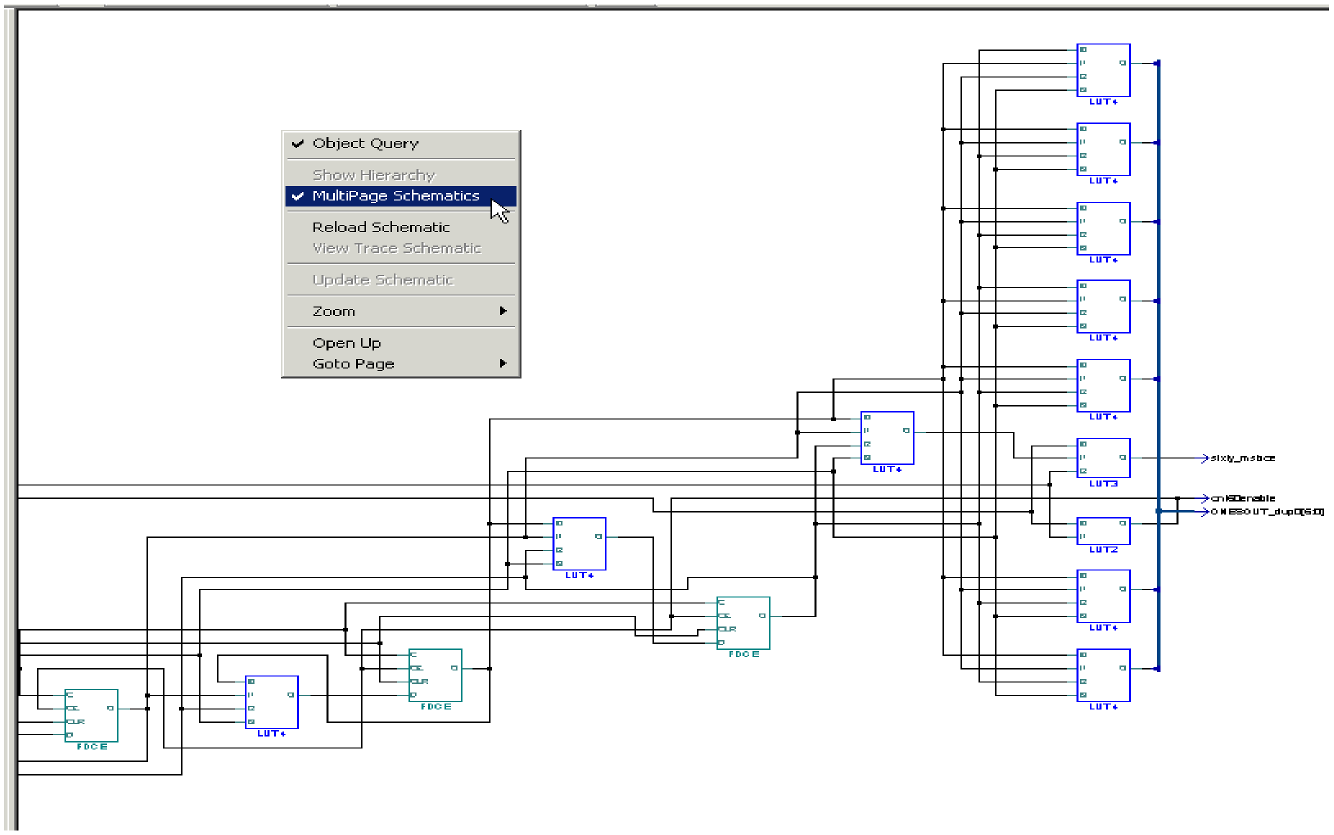


Figure 2-23: **Stopwatch Design in LeonardoSpectrum RTL Viewer**

LeonardoSpectrum also has the capability of generating a technology-specific view of the design after synthesis called the Technology Viewer. This schematic representation is useful for verifying that the inferred elements are what were intended to be for the design.

To launch the design in LeonardoSpectrum's Technology Schematic viewer, double-click the **View Technology Schematic** process.

Note: Viewing the technology schematic will most likely result in a multi-page schematic. To view a different page, right-click inside the schematic and select the appropriate option from the menu.

To view the path with the worst timing delay (the critical path) of the design, launch LeonardoSpectrum's Technology Viewer with LeonardoSpectrum's timing engine by double-clicking **View Critical Path Schematic**. Click the **View Trace** button in LeonardoSpectrum to display the critical path of the design.

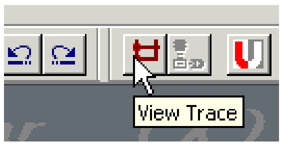


Figure 2-24: **LeonardoSpectrum View Trace Button**

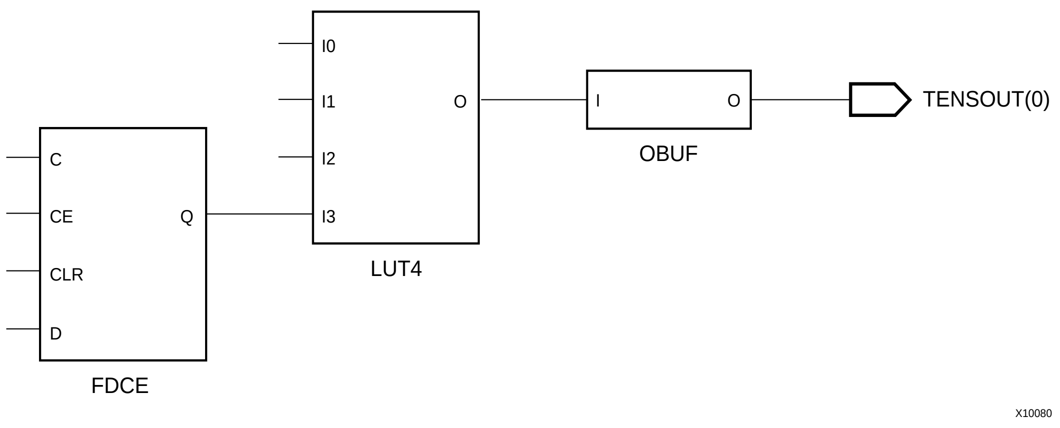


Figure 2-25: **LeonardoSpectrum Critical Path Schematic**

Double-click **View Synthesis Report** and **View Synthesis Summary** to see the details of the synthesis. The Synthesis Report summarizes the compilation, mapping and timing of the design. The Synthesis Summary provides more detail on the mapping and timing of the design.

Chapter 3

Schematic-Based Design

This chapter includes the following sections.

- "Overview of Schematic-based Design"
- "Getting Started"
- "Design Description"
- "Design Entry"

Overview of Schematic-based Design

This chapter guides you through a typical FPGA schematic-based design procedure using a design of a runner's stopwatch. The design example used in this tutorial demonstrates many device features, software features, and design flow practices that you can apply to your own designs. The Watch design targets a Virtex™-II device; however, all of the principles and flows taught are applicable to any Xilinx® device family, unless otherwise noted.

For an example of how to design with CPLDs, see the *ISE Software Interactive Tutorial for Xilinx CPLDs* http://support.xilinx.com/support/techsup/tutorials/index.htm.

This chapter is the first in the "Schematic Design Flow." In the first part of the tutorial, you will use the ISE design entry tools to complete the design. The design is composed of schematic elements, a state machine, a CORE Generator™ component, and an HDL macro. After the design is successfully entered in the Schematic Editor, you will perform a behavioral simulation with ModelSim (Chapter 4, "Behavioral Simulation"), implementation with the Xilinx Implementation Tools (Chapter 5, "Design Implementation"), and timing simulation with ModelSim (Chapter 6, "Timing Simulation."

Getting Started

The following sections describe the basic requirements for running the tutorial.

Required Software

You must have Xilinx ISE 6.x to perform this tutorial. For this design you must install the Virtex-II libraries and device files.

The Schematic Design Flow is supported on Windows and Linux platforms.

This tutorial assumes that the software is installed in the default location, *c:\xilinx*. If you have installed the software in a different location, substitute *c:\xilinx* with your installation path.

Note: For detailed instructions about installing the software, refer to the *ISE 6.1i Installation Guide and Release Notes.* This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

Installing the Tutorial Project Files

The tutorial project files can be downloaded to your local machine from http://support.xilinx.com/support/techsup/tutorials/tutorials6.htm.

Download the *wtut_sch.zip* file which contains two projects.

- `\ISEexamples\wtut_sc`
  (incomplete schematic tutorial)
- `\ISEexamples\watch_sc`
  (complete schematic tutorial)

Unzip the tutorial projects in the *c:\xilinx* directory (or any directory with Read/Write permissions), and replace any existing files. The files downloaded from the web have the most recent updates. The schematic tutorial files are copied into the directories when you unzip the project files.

wtut_sc project

The *wtut_sc* project contains an incomplete copy of the tutorial design. You will create the remaining files when you perform the tutorial. As described in a later step, you can copy this project to another area and perform the tutorial in this new area if desired.

watch_sc solution project

The *watch_sc* solution project contains the design files for the completed tutorial including schematics and the bitstream file. To conserve disk space, some intermediate files are not provided. Do not overwrite any files in the solutions directories.

Copying the Tutorial Files (Optional)

You can either work within the project directory where the tutorial files were downloaded, or you can make a copy of them to work on. To make a working copy of the tutorial files, use Windows Explorer to copy the *wtut_sc* directory to another location. The *wtut_sc* project directory contains all of the necessary project files.

Starting the ISE Software

To launch the ISE software package:

1. Double-click the ISE Project Navigator icon on your desktop or select **Start** → **Programs** → **Xilinx ISE** → **Project Navigator**.

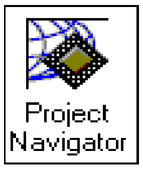
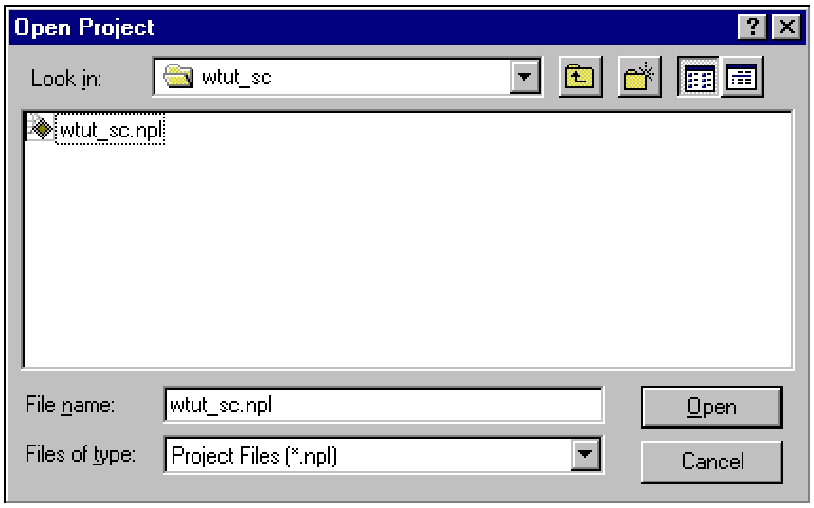


Figure 3-1: **Project Navigator Desktop Icon**

2. From Project Navigator, select **File** → **Open Project**.


Open Project

Look in: wtut_sc

wtut_sc.npl

File name: wtut_sc.npl

Files of type: Project Files (*.npl)

Figure 3-2: **Open Project Dialog Box**

3. Browse to the directory *c:\xilinx\ISEexamples\wtut_sc*.
4. Double-click the project file, *wtut_sc.npl*. If you cannot see this file, change the file type to **Project Files (*.npl)**.

Stopping the Tutorial

If you need to stop the tutorial at any time, save your work by selecting **File** → **Save.**

Design Description

The design used in this tutorial is a hierarchical, schematic-based design, which means that the top-level design file is a schematic sheet that references several other lower-level macros. The lower-level macros are a variety of different types of modules, including a schematic-based module, a CORE Generator™ module, a state machine module, an Architecture Wizard module, and an HDL module.

The design begins as an unfinished design. Throughout the tutorial, you will complete the design by creating some of the modules and by completing some others from existing files. A schematic of the completed Watch design is shown in the following figure. Through the course of this chapter, you will create these modules, instantiate and connect them

After the design is complete, you will simulate the design to verify its functionality. For more information about simulating your design, see Chapter 4, "Behavioral Simulation."

Note: Throughout this tutorial, the runner's stopwatch design that you are working on is referred to as Watch.

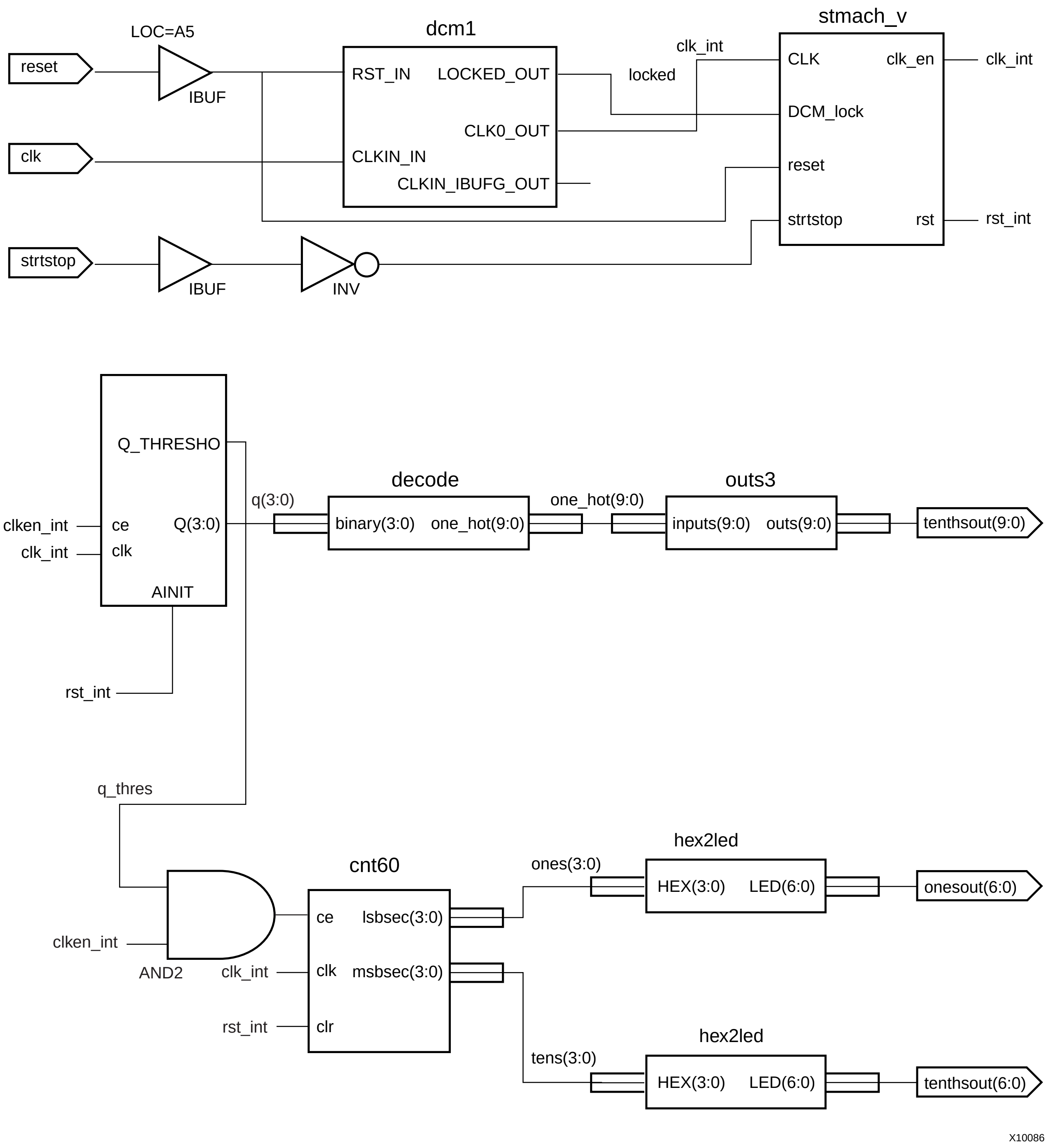


Figure 3-3: **Completed Watch Schematic**

There are three external inputs and three external outputs in the completed design. The following list summarizes the inputs and outputs, and their respective functions.

Inputs

The following are input signals for the Watch design:

- **STRTSTOP**

  Starts and stops the stopwatch. This is an active-low signal that acts like the start / stop button on a runner's stopwatch.

- **RESET**

  Resets the stopwatch to 00.0 after it has stopped.

- **CLK**

  System clock for the Watch design.

Outputs

The following are output signals for the design:

- **TENSOUT(6:0)**

  7-bit bus that represents the tens digit of the stopwatch value. This bus is in 7-segment display format to be viewable on the 7-segment LED display.

- **ONESOUT(6:0)**

  Similar to the TENSOUT bus above, but represents the ones digit of the stopwatch value.

- **TENTHSOUT(9:0)**

  10-bit bus which represents the tenths digit of the stopwatch value. This bus is one-hot encoded.

Functional Blocks

The completed design consists of the following functional blocks. Most of these blocks do not appear on the schematic sheet in the tutorial project until after you create and add them to the schematic in this tutorial.

- **STMACH_V**

  State Machine macro defined and implemented in StateCAD™.

- **CNT60**

  Schematic-based module which counts from 0 to 59 decimal. This macro has two 4-bit outputs, which represent the ones and tens digits of the decimal values, respectively.

- **TENTHS**

  CORE Generator™ 4-bit, binary encoded counter. This macro outputs a 4-bit code that is decoded to represent the tenths digit of the watch value as a 10-bit one-hot encoded value.

- **HEX2LED**

  HDL-based macro. This macro decodes the ones and tens digit values from hexadecimal to 7-segment display format.

- **OUTS3**

  Schematic-based macro containing inverters.
- **DECODE**

  Decodes the CORE Generator™ output from 4-bit binary to a 10-bit one-hot output.
- **DCM1**

  Clocking Wizard macro with internal feedback and duty-cycle correction.

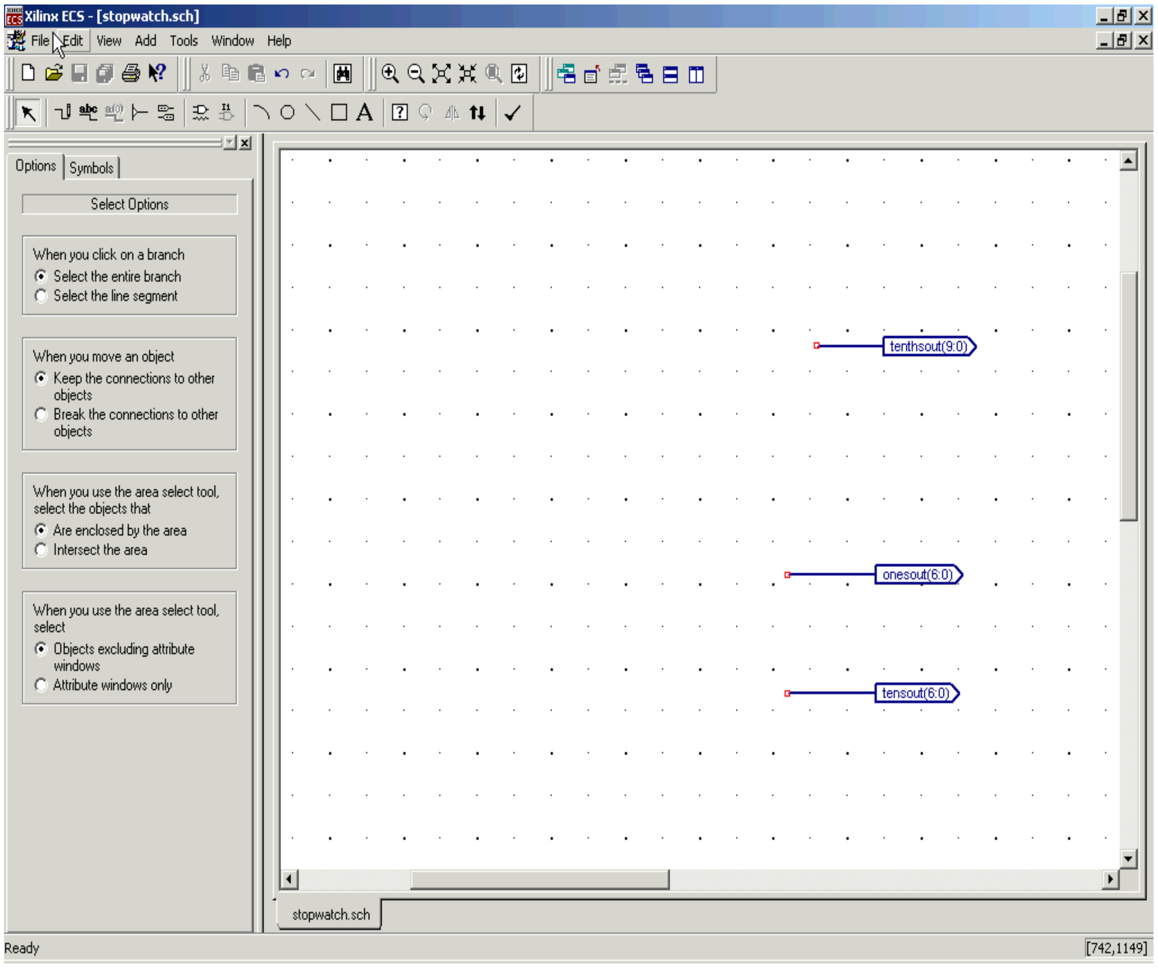
Design Entry

In this hierarchical design, you will create various types of macros, including schematic-based macros, HDL-based macros, state machine macros, and CORE Generator™ macros. You will learn the process for creating each of these types of macros, and connect them together to create the completed Watch design. All procedures used in the tutorial can be used later for your own designs.

Opening the Project File in the ECS Schematic Editor Tool

The Watch schematic available in the *wtut_sc* project is incomplete, and in this tutorial, will be updated in the Engineering Capture System (ECS) schematic editor tool. After you have opened the project in ISE, you can now open the *stopwatch.sch* file in the ECS tool. To do so, double-click the file *stopwatch.sch* in the Sources in Project window.

The Watch schematic diagram opens in ECS. You will see the unfinished design as shown in the figure below.



Select Options

When you click on a branch
(●) Select the entire branch
( ) Select the line segment

When you move an object
(●) Keep the connections to other objects
( ) Break the connections to other objects

When you use the area select tool, select the objects that
(●) Are enclosed by the area
( ) Intersect the area

When you use the area select tool, select
(●) Objects excluding attribute windows
( ) Attribute windows only

Figure 3-4: **Incomplete Watch Schematic in Engineering Capture System (ECS)**

Manipulating the Window View

The View menu commands enable you to manipulate how the schematic is displayed. Select **View** → **Zoom** → **In** until you can comfortably view the schematic.

Creating a Schematic-Based Macro

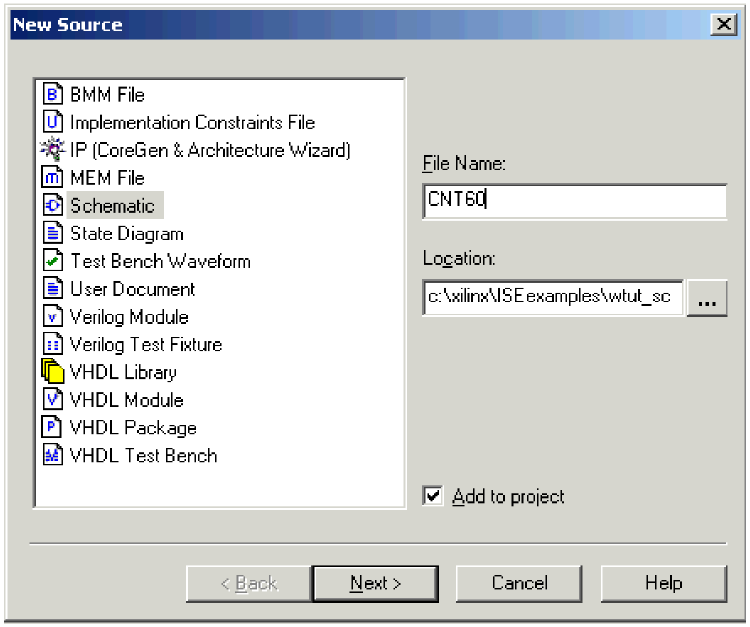
A schematic-based macro consists of a symbol and an underlying schematic. You can create either the underlying schematic or the symbol first. ECS then generates the corresponding symbol or schematic file.

In the following steps, you will create a schematic-based macro by using the New Source Wizard in Project Navigator. An empty schematic file is then created, and you can define in ECS with the appropriate logic. The created macro is then automatically added to the project’s library.

The macro you will create is called CNT60. CNT60 is a binary counter with two 4-bit outputs, which represent the Ones and Tens values of the stopwatch. The counter counts from 0 to 59, in decimals.

To create a schematic-based macro:

1. In Project Navigator, select **Project** → **New Source**. The New Source dialog opens.



File Name: CNT60

Location: c:\xilinx\ISEexamples\wtut_sc

☑ Add to project

Figure 3-5: **New Source Dialog Box**

The New Source dialog provides a list of all available source types.

2. Select **Schematic** as the source type.
3. Enter **CNT60** as the file name.
4. Click **Next** and click **Finish**.

This creates a new schematic named CNT60 and adds the schematic file to the project.

Defining the CNT60 Schematic

You have now created an empty schematic for CNT60. The next step is to add the components that make up the CNT60 macro. You can then reference this macro symbol by placing it on a schematic sheet.

Adding I/O Markers

I/O markers are used to determine the ports on a macro or the top level schematic. The name of each pin on the symbol must have a corresponding connector in the underlying schematic. Add I/O markers to the CNT60 schematic to determine the macro ports.

To add the I/O markers:

1. Select **Tools** → **Create I/O Markers**.

   The Create I/O Markers Options window opens.

2. In the Inputs box type: **ce,clk,clr**.

3. In the Outputs box type: **lsbsec(3:0),msbsec(3:0)**.

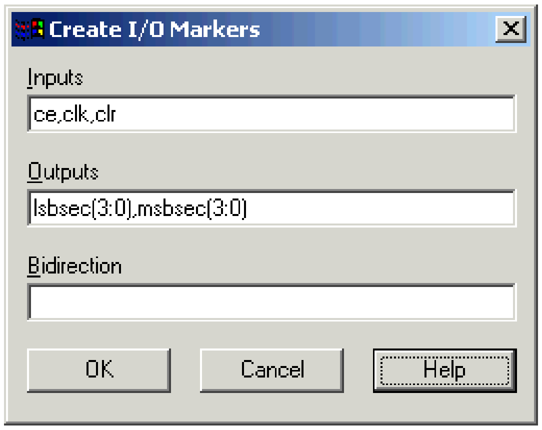


Figure 3-6: **Creating I/O Markers**

4. Click **OK** and the five pins are added to the schematic sheet.

Note: The Create I/O Marker function is available only for an empty schematic sheet. However, I/O markers may be added to nets at any time by selecting **Add** → **I/O Marker** and selecting the desired net.

Adding Components to CNT60

Components from the device and project libraries for the given project are available from the Symbol Libraries toolbox, and the component symbol can be placed on the schematic. The available components listed in this toolbox are arranged alphabetically within each library.

1. From the menu bar, select **Add** → **Symbol** or click the Add Symbol icon from the Tools toolbar.

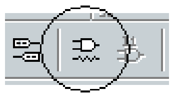


Figure 3-7: **Add Symbol Icon**

This opens the Symbol Browser dialog box to the left of the schematic editor, which displays the libraries and their corresponding components.

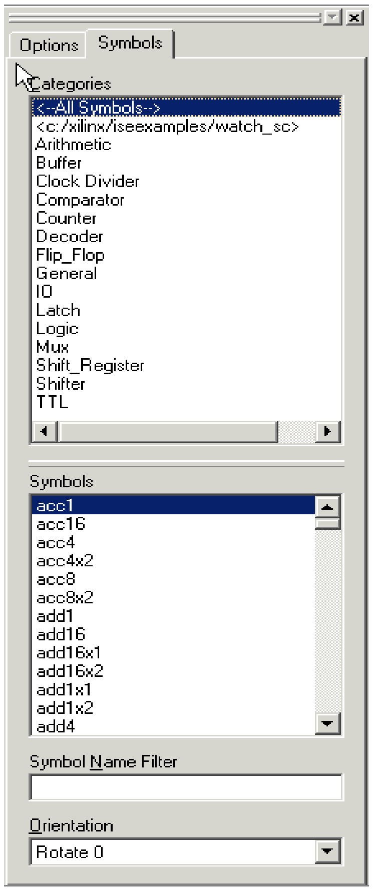


Figure 3-8: **Symbol Browser Dialog Box**

The first component you will place is an AND2, a 2-input AND gate.

2. Select the AND2 component using one of two ways:
   - Highlight the **Logic** category from the Symbol Browser dialog box and select the component **AND2** from the symbols list.

     or

   - Select **All Symbols** and type **AND2** in the Symbol Name Filter at the bottom of the Symbol Browser window.
3. Move the mouse back into the schematic window.

   You will notice that the cursor has changed to represent the AND2 symbol.
4. Move the symbol outline to the location shown in Figure 3-9 and click the left mouse button to place the object.

   Note: You can rotate new components being added to a schematic by selecting CTRL+R. You can rotate existing components by selecting the move mode, selecting the component, and then selecting CTRL+R.

5. Place the second AND2 symbol on the schematic by moving the cursor with attached symbol outline to the desired location, and click the left mouse button. See Figure 3-9.

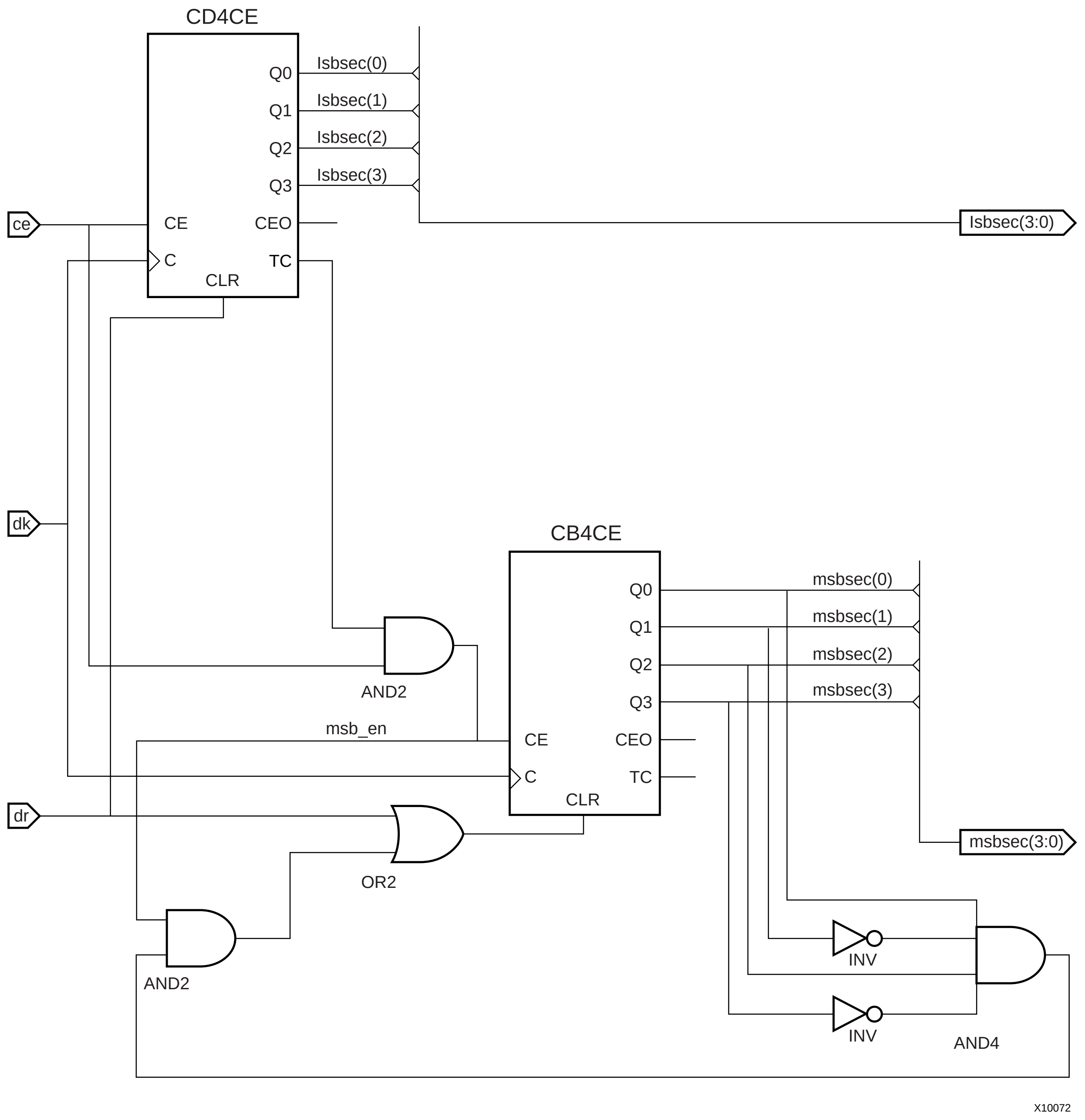


Figure 3-9: **Completed CNT60 Schematic**

Placing the Remaining Components

Follow the steps above in "Adding Components to CNT60" to place the CD4CE, OR2, CB4CE, INV, and AND4 components on the schematic sheet. Refer to Figure 3-9 for placement of all components.

To exit the Symbols Mode, press the **Esc** key on the keyboard.

For a detailed description of the functionality of each of these components, right-click on the component and select **Object Properties**. In the Object Properties window, select **Symbol Information**. Symbol information is also available in the *Libraries Guide* which is found in the collection of software manuals. To access the software manuals, select **Help** → **Online Documentation**.

Correcting Mistakes

If you make a mistake when placing a component, you can easily move or delete the component.

To move the component, click the component and drag the mouse around the window.

Delete a placed component in one of two ways:

- Click the component and press the **Delete** key on your keyboard.

  or

- Right-click the component and select **Delete**.

Drawing Wires

Use the Add Wire icon in the Tools toolbar to draw wires (also called nets) to connect the components placed in the schematic.

Signals can be logically connected by naming multiple segments identically. In this case, the nets do not need to be physically connected on the schematic to make the logical connection. In the CNT60 schematic, draw wires to connect the components together. The nets for the LSBSEC and MSBSEC buses are drawn in the next section.

Perform the following steps to draw a net between the AND2 and CB4CE components on the CNT60 schematic.

1. Select **Add** → **Wire** or click the Add Wires icon in the Tools toolbar.

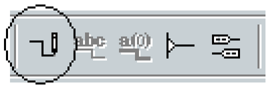


Figure 3-10: **Add Wires Icon**

2. Click the output pin of the AND2 and then click the destination pin, CE on the CB4CE component. ECS draws a net between the two pins.

Draw the nets to connect the remaining components as shown in the Figure 3-9. To specify the shape of the net:

1. Move the mouse in the direction you want to draw the net.
2. Click the mouse to create a 90-degree bend in the wire.

To draw a net between an already existing net and a pin, click once on the component pin and once on the existing net. A junction point is drawn on the existing net.

You should now have all the nets drawn except those connected to the LSBSEC and MSBSEC buses. You will draw these in the next section. Net names will be added in a later section.

Adding Buses

In ECS, a bus is simply a wire which has been given a multi-bit name. To add a bus, use the methodology for adding wires and then add a multi-bit name. Once a bus has been created, you have the option of “tapping” this bus off to use each signal individually.

In this CNT60 schematic, create two buses called LSBSEC(3:0) and MSBSEC(3:0), each consisting of the 4 output bits of each counter by connecting a wire to each of the output I/O markers. The results can be found in the completed schematic.

To add the buses LSBSEC(3:0) and MSBSEC(3:0) to the schematic, perform the following steps:

1. Select **Add** → **Wire** or click the Add Wires icon in the Tools toolbar.
2. Click to the right of the CB4CE and then click again on pin of the msbsec(3:0) I/O marker. The wire should automatically be drawn as a bus with the name matching that of the I/O marker.
3. To verify this, zoom in. The bus is represented visually by a thicker wire.

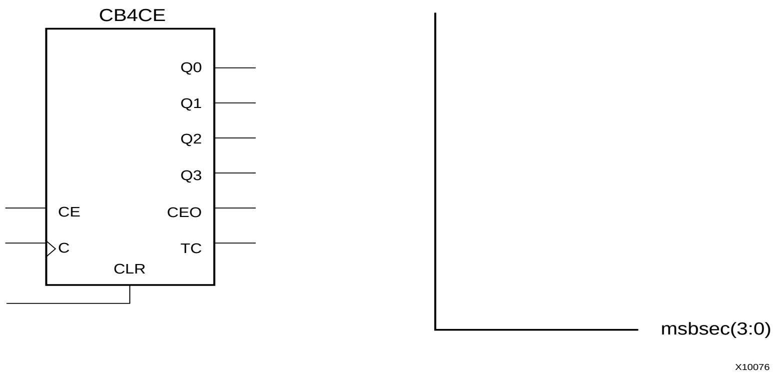


Figure 3-11: **Adding a Bus**

4. Repeat Steps 1 through 3 for the lsbsec(3:0) bus, referring to Figure 3-9 for placement of the wire and the bus name.
5. After adding the two buses, press **esc** or right-click to exit the Add Wire mode.

Adding Bus Taps

Next, add nets to attach the appropriate pins from the CB4CE and CD4CE counters to the buses. Use Bus Taps to tap off a single bit of a bus and connect it to another component.

Note: Zooming in on the schematic will enable greater precision when drawing the nets.

To tap off a single bit of each bus:

1. Select **Add** → **Bus Tap** or click the Add Bus Tap icon in the Tools toolbar.

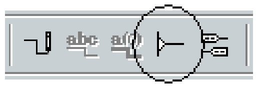


Figure 3-12: **Add Bus Tap Icon**

The cursor changes, indicating that you are now in Draw Bus Tap mode.

2. From the options window to the left of the schematic, choose the correct orientation for the bus.
3. Place the tap on the bus so that the wire side of the bus tap is pointing to an unconnected pin.

   Repeat steps 1 to 3 to tap off the other three bits of the bus.

To connect each of the tap off bits:

1. Select **Add** → **Wire** or click the Add Wire icon in the Tools toolbar.
2. Draw a wire from the other end of the bus taps to the corresponding pins.
3. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.
4. Type **msbsec(0)** in the Name field of the options toolbar.

   The net name is now at the end of your cursor.
5. Select **Increment the Name** in the Add Net Names options dialog box.
6. With the Increment the Name option selected, start at the top net and continue clicking down until you have named the fourth and final net msbsec(3). This option appends a higher number to the net name for each pin you click.

Note: ECS names the bus taps incrementally as they are drawn. Alternatively, you can click **Decrement the Name** option, name the bottom net msbsec(3) and continue clicking to decrement the nets names.

   Repeat Steps 1 through 6 for the lsbsec(3:0) bus.

7. Press **Esc** to exit the Add Net Name mode.
8. Draw the nets to connect the msbsec bus taps to the INV and AND4 components. If necessary, refer to "Drawing Wires" for guidance.
9. Compare your CNT60 schematic again with Figure 3-9 to ensure that all connections are made properly.

Note: If the nets appear disconnected, select **View** → **Refresh** to refresh the screen.

Adding Net Names

Next, add net names to the *msb_en* net (wire).

1. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.
2. Type *msb_en* in the Name box of the Add Net Name options dialog box.

Note: The Options window changes depending on which tool you have selected in the Tools toolbar.

   The net name *msb_en* is now attached to the cursor.

3. Click the *msb_en* net.

   The name is then attached to the net. The net name will appear above the net if you do not place the name at an end point.

Saving the Schematic

The CNT60 schematic is now complete.

1. Save the schematic by selecting **File** → **Save** or by clicking the Save icon in the toolbar.

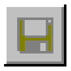


Figure 3-13: **Save Icon**

When you save a macro, the ECS schematic editor checks the I/O markers against the corresponding symbol. If there is a discrepancy, you can let the software update the symbol automatically, or you can modify the symbol manually. You should use I/O markers to connect signals between levels of hierarchy and to specify the ports on top-level schematic sheets.

2. Exit ECS.

Creating and Placing the CNT60 Symbol

The next step is to create a "symbol" that represents the CNT60 macro. The symbol is an instantiation of the macro. After you create a symbol for CNT60, you will place (or add) the symbol to a top-level schematic of the Watch design. In the top-level schematic, the symbol of the CNT60 macro will be connected to other components in a later section in this chapter.

Creating the CNT60 symbol

You can create a symbol in either Project Navigator or ECS. For this tutorial, you can follow either method.

In Project Navigator, create a symbol that represents the CNT60 schematic as follows.

1. In the Sources in Project window, select *cnt60.sch*.
2. In the Processes for Current Source window, click the + beside **Design Entry Utilities** to expand the hierarchy.
3. Double-click **Create Schematic Symbol**.

In ECS, create a symbol that represents the CNT60 schematic as follows.

1. Select **Tools** → **Symbol Wizard**.
2. In the Symbol wizard, select **Using Schematic** and select CNT60 in the schematic value field.
3. Click **Next**, click **Next**, click **Next**, and click **Finish** to use the wizard defaults.

Placing the CNT60 symbol

Next, place the symbol that represents the macro on the top-level Watch schematic sheet (stopwatch.sch).

1. In Project Navigator, double-click *stopwatch.sch* in the Sources in Project window to open the Watch schematic sheet.
2. Select the Add Symbol icon to open the Symbol Browser dialog box.

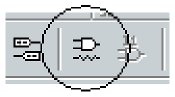


Figure 3-14: **Add Symbol Icon**

3. Select the Local Symbols library (*c:/xilinx/ISEexamples/wtut_sc*), and locate and select the newly created CNT60 symbol from this list.

4. Place the CNT60 symbol in the schematic as shown below.

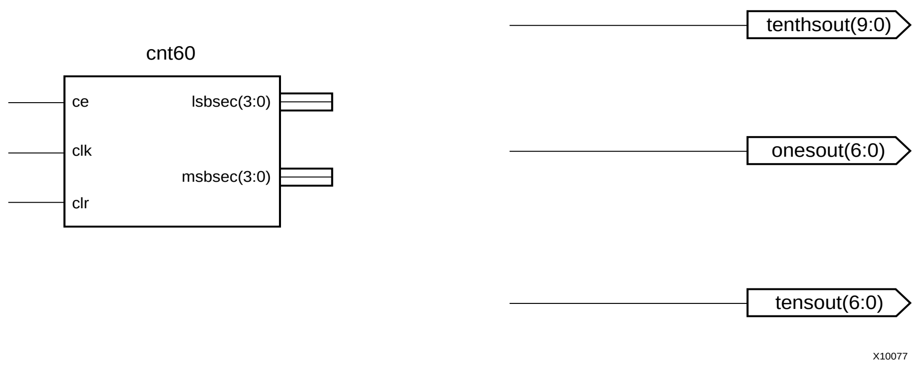


Figure 3-15: **Placing the CNT60 Symbol in the Top-Level Schematic**

Note: Do not worry about connecting nets to the pins of the CNT60 symbol. You will do this after you have added other components to the Watch schematic.

5. Save changes and exit ECS.

Creating a CORE Generator Module

CORE Generator™ is a graphical interactive design tool you use to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and pre-optimize 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 create a CORE Generator module called Tenths. Tenths is a 4-bit binary encoded counter. The 4-bit number is then decoded to count the tenths digit of the stopwatch's time value.

Creating a CORE Generator Module

Create the CORE Generator module using the New Source Wizard in Project Navigator. This invokes CORE Generator in which you can select and define the type of module you want.

To create the module:

1. In Project Navigator, select **Project** → **New Source**.
2. Select **IP(Coregen & Architecture Wizard)**.
3. Type *tenths* in the File Name field.
4. Click **Next**.
5. Double-click **Basic Elements - Counters**.
6. Select **Binary Counter**, click **Next** and click **Finish** to open the Binary Counter dialog box. This dialog box enables you to customize the counter to the design specifications.
7. Fill in the Binary Counter dialog box with the following settings:
   - Component Name: **tenths**

     Defines the name of the module.
   - Output Width: **4**

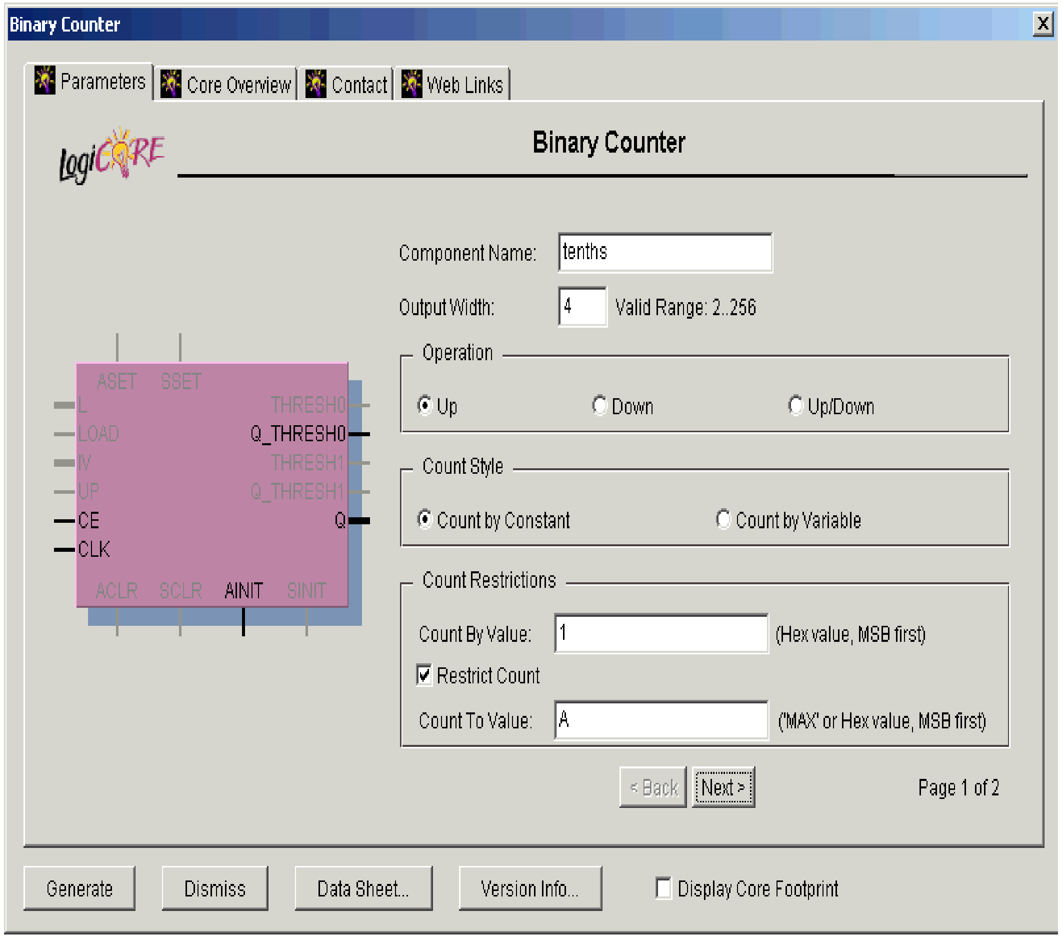
     Defines the width of the output bus.

- Operation: **Up**

  Defines how the counter will operate. This field is dependent on the type of module you select.
- Count Style: **Count by Constant**

  Allows counting by a constant or a user-supplied variable.
- Count Restrictions:
  - Count by value: **1**
  - Select **restricted count**
  - Count to value: **A**

  This dictates the maximum count value.



Component Name: tenths

Output Width: 4 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: A ('MAX' or Hex value, MSB first)

☐ Display Core Footprint

Figure 3-16: **CORE Generator Module Selector**

8. Click **Next**.
   - Threshold Options: **Enable Threshold 0 and set to A**

     Signal goes high when the value specified has been reached.
   - Select **Registered**
   - Click the **Register Options** button to open the Register Options dialog box.

9. Enter the following settings and then click **OK**.
   - Clock Enable: **Selected**
   - Asynchronous Settings: **Init with a value of 1**
   - Synchronous Settings: **None**
10. Check that *only* the following pins are used (used pins will be highlighted on the model symbol to the left side of the CORE Generator™ window):
    - **AINIT**
    - **CE**
    - **Q**
    - **Q_THRESH0**
    - **CLK**
11. Click **Generate**.

    The module is created and automatically added to the project library.

    Note: A number of other files are added to the project directory. These files are:
    - *tenths.sym*

      This is a schematic symbol file.
    - *tenths.edn*

      This file is the netlist that is used during the Translate phase of implementation.
    - *tenths.vho* or *tenths.veo*

      This is the instantiation template that is used to incorporate the CORE Generator module in your source HDL.
    - *tenths.vhd* or *tenths.v*

      These are simulation-only files.
    - *tenths.xco*

      This file stores the configuration information for the Tenths module and is used as a project source.
    - *coregen.prj*

      This file stores the CORE Generator configuration for the project.
12. Click **Dismiss** to exit CORE Generator.

Creating a State Machine Module

With Xilinx StateCAD™, you can graphically create finite state machines—states, inputs / outputs, and state transition conditions. Transition conditions and state actions are typed into the diagram using language independent syntax. The State Editor then exports the diagram to either VHDL, Verilog or ABEL code. The resulting HDL file is finally synthesized to create a netlist and / or macro for you to place on a schematic sheet.

For this tutorial, a partially complete state machine diagram is provided. In the next section, you will complete the diagram and synthesize the module into a macro to place on the Watch schematic. A completed VHDL State Machine diagram has been provided for you in the *watch_sc* directory.

Opening the State Editor

You can invoke StateCAD™ from Project Navigator. The tutorial utilizes an existing diagram which you will complete.

To open the diagram, double-click *stmach_v.dia* in the Sources in Project window. The state machine file is launched in StateCAD.

In the incomplete state machine diagram below:

- The circles represent the various states.
- The black expressions are the transition conditions, defining how you move between states.
- The output expressions for each state are contained in the circle representing the state.

In the state machine diagrams, the transition conditions and the state actions are written in language independent syntax and then exported to Verilog, VHDL, or ABEL.

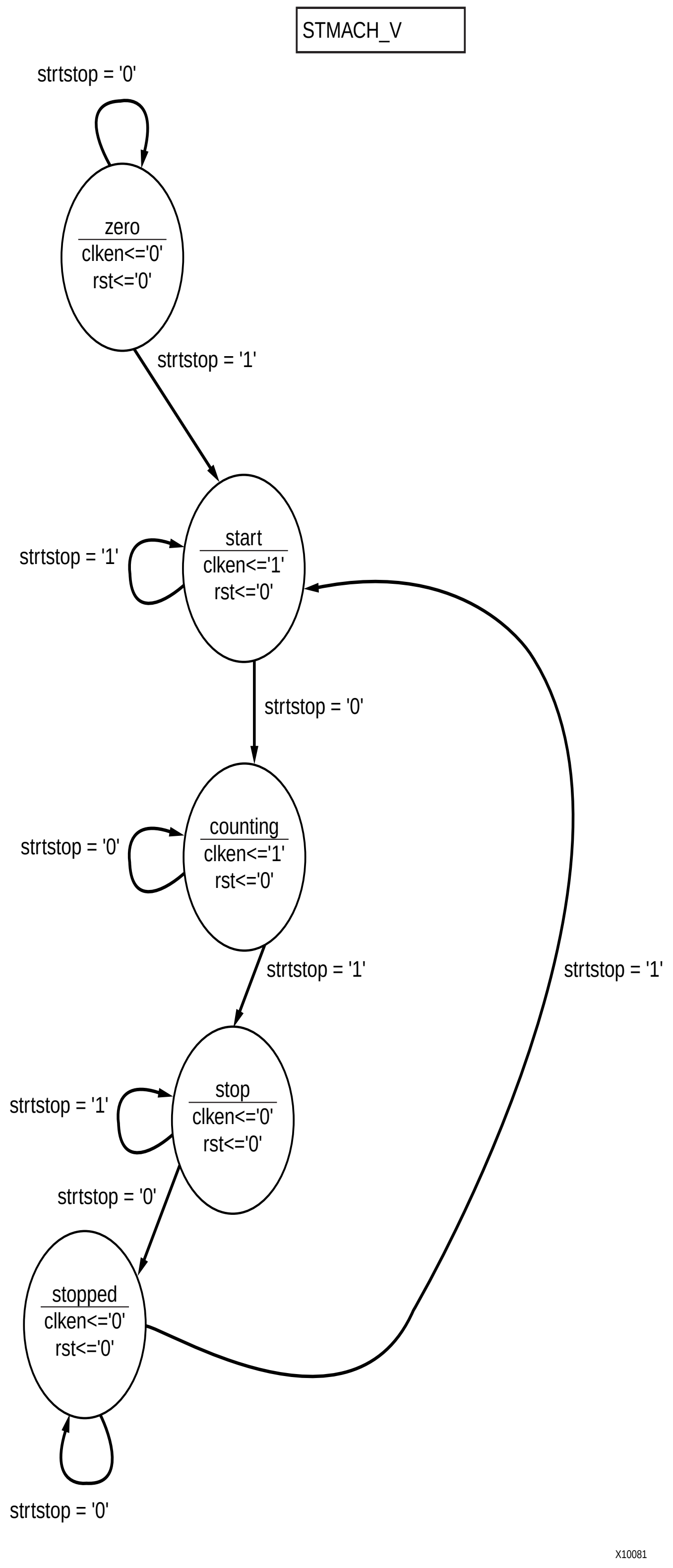


Figure 3-17: **Incomplete State Machine Diagram**

In the following section, add the remaining states, transitions, actions, and a reset condition to complete the state machine.

Adding New States

Complete the state machine by adding a new state called *clear.* To do so:

1. Click the Add State icon in the vertical toolbar.

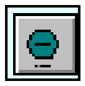


Figure 3-18: **Add State Icon**

   The state bubble is now attached to the cursor.

2. Place the new state on the left-hand side of the diagram as shown in Figure 3-19. Click the mouse to place the state bubble.

   The state is given the default name, STATE0.

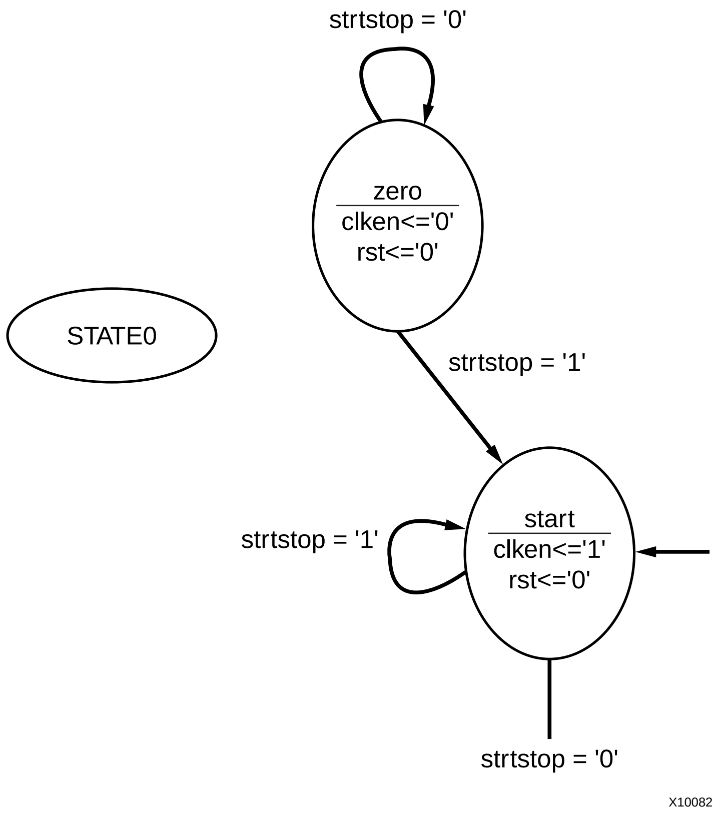


Figure 3-19: **Adding the CLEAR State**

3. Double-click **STATE0** in the state bubble, and change the name of the state to *clear.*

   Note: The name of the state is for your use only and does not affect synthesis. Any name is fine.

4. Click **OK**.

To change the shape of the state bubble, click the bubble and drag it in the direction you wish to “stretch” the bubble.

Adding a Transition

A transition defines the movement between states of the state machine. Transitions are represented by arrows in the editor. You will be adding a transition from the *clear* state to the *zero* state in the following steps. Because this transition is unconditional, there is no transition condition associated with it.

1. Click the Add Transitions icon in the vertical toolbar.

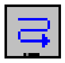


Figure 3-20: **Add Transitions Icon**

2. Double-click the **clear** state (one click to select it, and one click to start the transition.)
3. Click the **zero** state to complete the transition arrow.
4. To manipulate the arrow's shape, click and drag it in any directory.

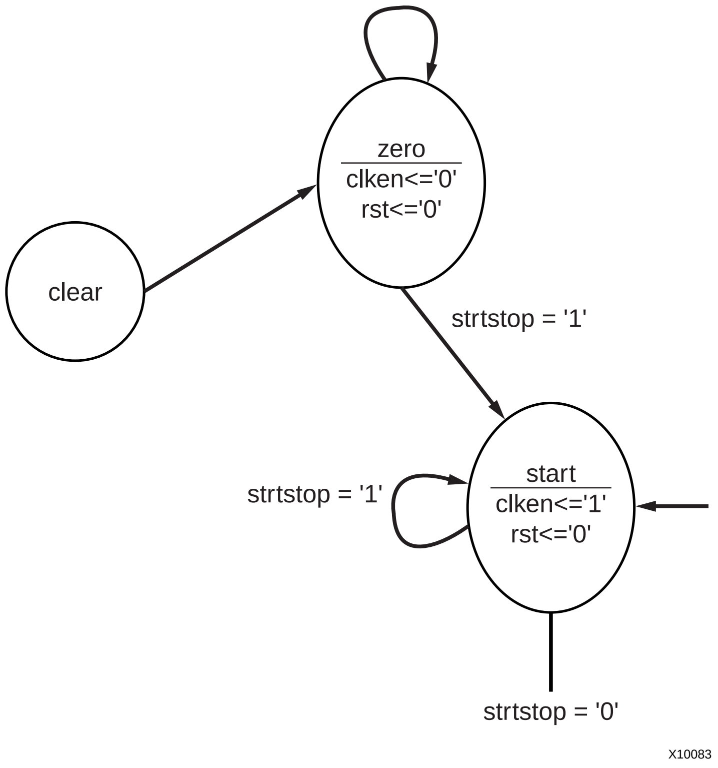


Figure 3-21: **Adding State Transition**

5. Click the Select Objects icon in the vertical toolbar to exit the Add Transition mode.

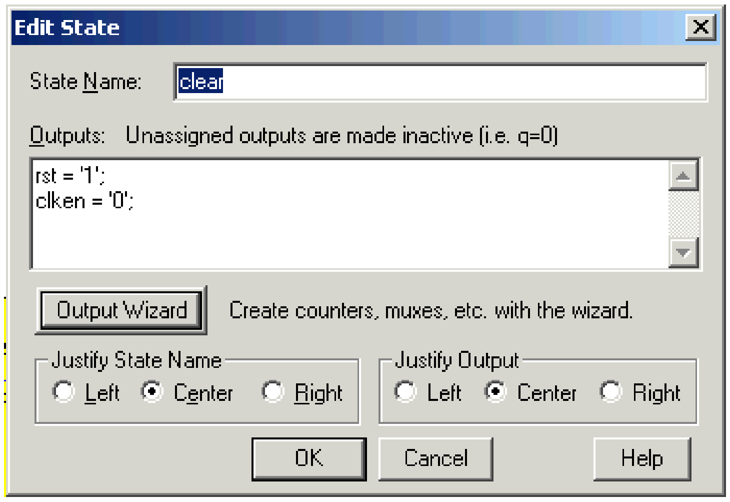
Adding a State Action

A State Action dictates how the outputs should behave in a given state. You will add two state actions to the *clear* state, one to drive the clken output to 0, and one to drive the RST output to 1.

To add a State Action:

1. Double-click the **clear** state.

   The Edit State dialog box opens and you can begin to create the desired outputs.



State Name: clear

Outputs: Unassigned outputs are made inactive (i.e. q=0)

```
rst = '1';
clken = '0';
```

Output Wizard — Create counters, muxes, etc. with the wizard.

Justify State Name: ( ) Left [✓ Center] ( ) Right

Justify Output: ( ) Left [✓ Center] ( ) Right

Figure 3-22: **Edit State Dialog Box**

2. Select the **Output Wizard** button.
3. In the Output Wizard, select the following values:

   ```
   DOUT = rst, CONSTANT = ‘1’;
   DOUT = clken, CONSTANT = ‘0’;
   ```

4. Click **OK** to enter each individual value.
5. Click **OK** to exit the Edit State dialog box. The outputs are now added to the state.

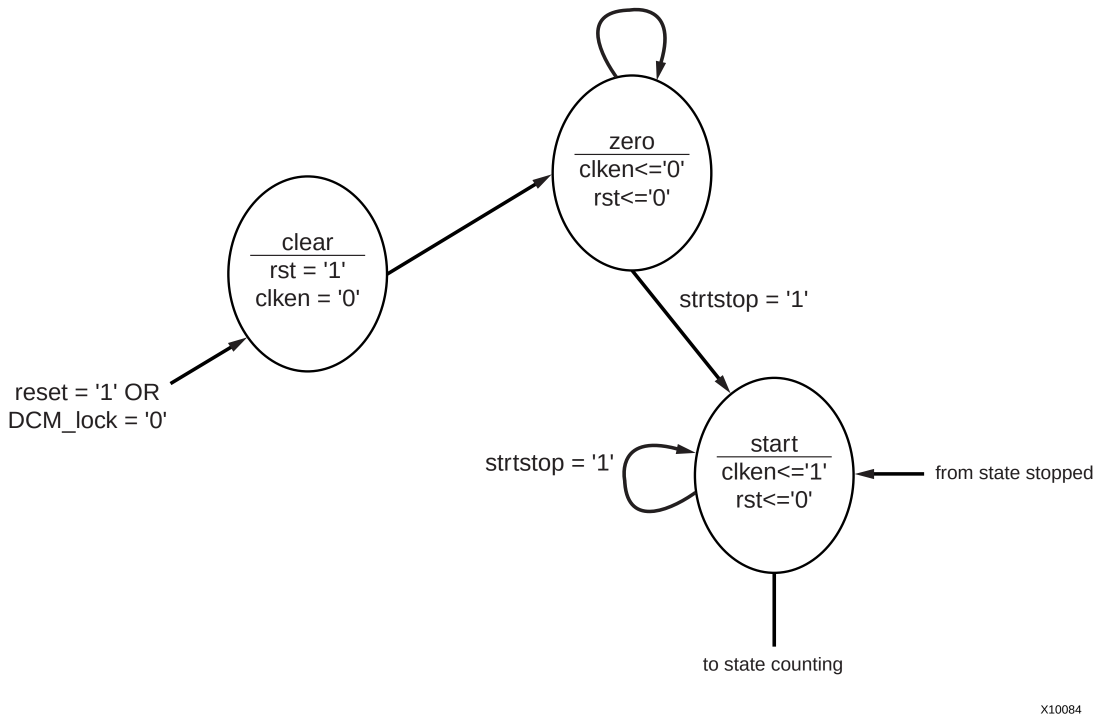


Figure 3-23: **Adding State Outputs**

Adding a State Machine Reset Condition

Using the State Machine Reset, specify a reset condition for the state machine. The state machine initializes to this specified state and enters the specified state whenever the reset condition is met. In this design, add a Reset condition which sends the state machine to the *clear* state whenever either the *reset* signal is asserted or the *DCM_lock* signal is deasserted.

1. Click the Add Reset icon in the vertical toolbar.

Figure 3-24: **Add Reset Icon**

2. Click the diagram near the *clear* state, as shown in the diagram below.
3. The cursor is automatically attached to the transition arrow for this reset. Move the cursor to the *clear* state, and click the **state bubble**.
4. A question is then asked, "Should this reset be asynchronous(Yes) or synchronous(No)?" Answer **Yes**.

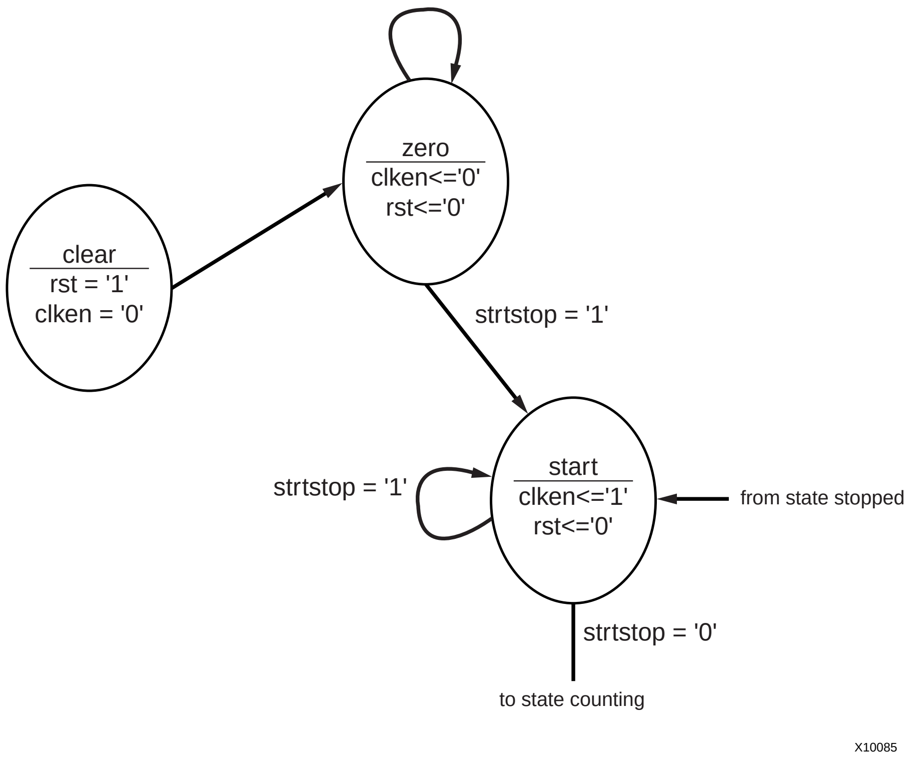


Figure 3-25: **Adding Reset**

5. Double-click the newly created RESET condition and edit the condition field to read: reset='1' OR DCM_lock='0'. Then click **OK**.
6. Save your changes by selecting **File** → **Save**.

Creating the State Machine Symbol

In this section, you will create the HDL code used to create a macro symbol that you can place on the Watch schematic. The macro symbol is added to the project library. When you create the macro, StateCAD™ creates HDL code representing the macro from the state machine diagram.

1. Select **Options** → **Compile (Generate HDL)**.

   StateCAD verifies the state machine and displays the results.
2. Review the results and exit the dialog box.

   StateCAD will then create the HDL code and open a browser displaying the code.
3. Exit the browser when you have finished examining the code.
4. Exit StateCAD.
5. In Project Navigator, select **Project** → **Add Source**.
6. Select *STMACH_V.vhd*, which is the VHDL file generated by StateCAD.
7. Click **Open**.
8. Select **VHDL Design File** as the source type.
9. Click **OK**.

The file *STMACH_V.vhd* is added to the project in Project Navigator.

10. In the Sources in Project window, select *stmach_v.vhd*.
11. In the Processes for Current Source window, double-click **Create Schematic Symbol** from the Design Entry Utilities hierarchy.

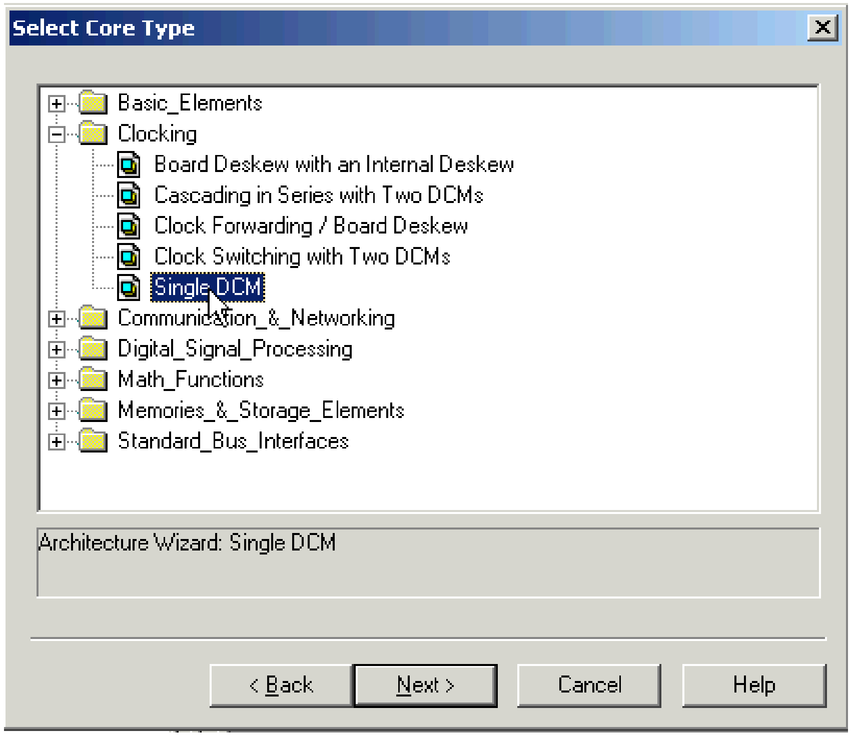
Creating a DCM Module

The Clocking Wizard, a Xilinx® Architecture Wizard, enables you to graphically select Digital Clock Manager (DCM) features that you wish to use. In this section, you will create a basic DCM module with CLK0 feedback and duty-cycle correction.

Using DCM Wizard

Create the DCM1 module.

1. Select **Project** → **New Source**.
2. In the New Source dialog box, select the **IP (Coregen & Architecture Wizard)** source type, and type the filename *DCM1*. Click **Next**.
3. Select **Single DCM** in the Clocking hierarchy.

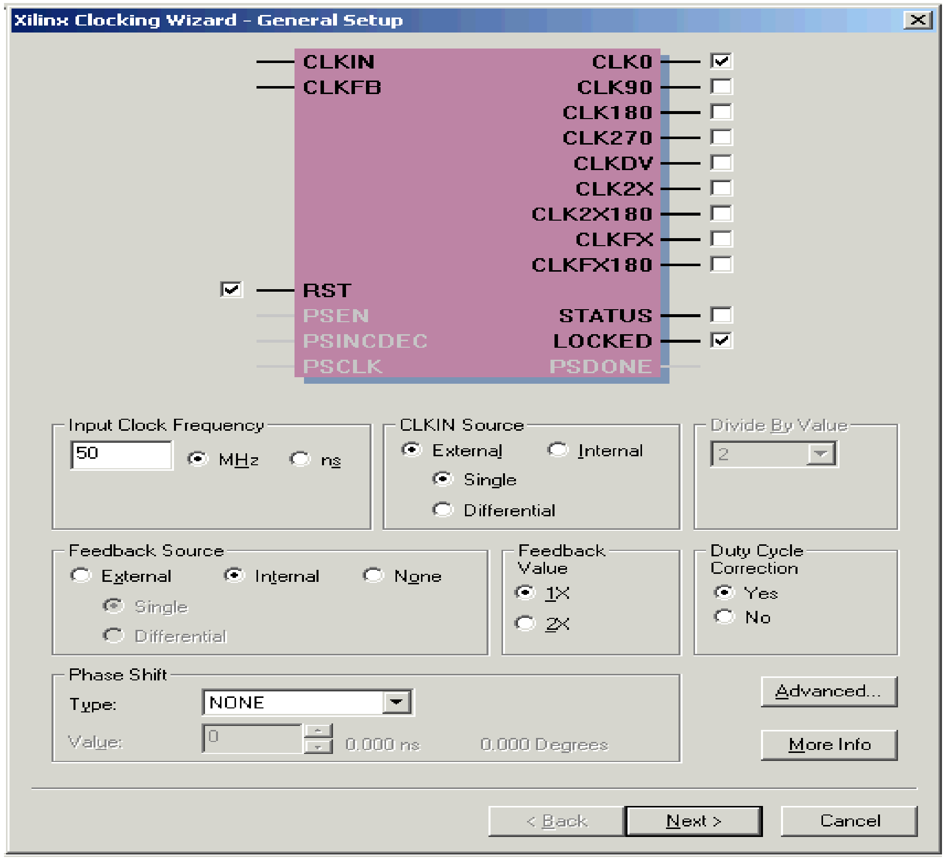

Select Core Type
Basic_Elements
Clocking
Board Deskew with an Internal Deskew
Cascading in Series with Two DCMs
Clock Forwarding / Board Deskew
Clock Switching with Two DCMs
Single DCM
Communication_&_Networking
Digital_Signal_Processing
Math_Functions
Memories_&_Storage_Elements
Standard_Bus_Interfaces
< Back
Next >
Cancel
Help

Architecture Wizard: Single DCM

Figure 3-26: **Selecting Single DCM core type**

4. Click **Next** and click **Finish**.
5. Verify that **RST**, **CLK0** and **Locked** are selected.
6. Type **50** for the Input Clock Frequency.

7. Verify the following settings:
   - Clkin Source: **External**
   - Feedback Source: **Internal**
   - Feedback Value: **1X**
   - Phase Shift: **None**
   - Duty Cycle Correction: **Yes**



Xilinx Clocking Wizard - General Setup

✓ RST; ✓ CLK0; CLK90; CLK180; CLK270; CLKDV; CLK2X; CLK2X180; CLKFX; CLKFX180; STATUS; ✓ LOCKED

Input Clock Frequency: 50 [✓ MHz] ns

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

Divide By Value: 2

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

Feedback Value: [✓ 1X] 2X

Duty Cycle Correction: [✓ Yes] No

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

Figure 3-27: **Xilinx Clocking Wizard - General Setup**

8. Click the **Advanced** button.
9. Select the **Wait for DCM Lock before DONE Signal goes high** option.
10. Click **OK**.

    An informational message about the LCK_cycle and the STARTUP_WAIT Bitgen options appears.
11. Click **OK**, click **Next** and click **Finish**.

DCM1.xaw is added to your project sources.

Creating the DCM1 Symbol

Next, create a symbol representing the DCM1 macro. This symbol will be later added to the top-level schematic (stopwatch.sch).

1. In Project Navigator, in the Sources in Project window, select *DCM1.xaw*.
2. In the Processes for Current Source window, double-click **Create Schematic Symbol** from the Design Entry Utilities hierarchy.

Note: The newly created *DCM1_arwz.ucf* file does not need to be added to the project, as all of the constraints are passed into the relevant source file(s).

Placing the STMACH, Tenths, DCM1, outs3, and decode symbols

You can now place the STMACH, Tenths, DCM1, outs3, and decode symbols on the Watch schematic (stopwatch.sch). If this file is already open in ECS, ignore step 1.

1. In Project Navigator, double-click stop*watch.sch*. The schematic file opens in the ECS schematic editor.
2. View the list of available library components in the Symbol Browser window.
3. Locate the macros in the Local Symbols library.
4. Select the appropriate symbol, and add it to the Watch schematic as shown in Figure 3-28.

Note: Do not worry about drawing the wires to connect this symbol. You will connect components in the schematic later in the tutorial.

5. Save the schematic.

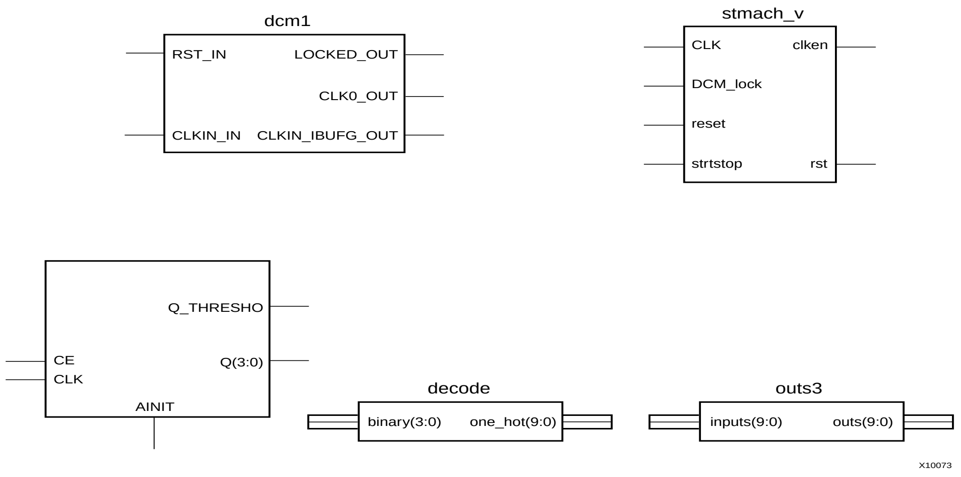


Figure 3-28: **Placing Design Macros**

Creating an HDL-Based Module

With ISE, you can easily create modules from HDL code. The HDL code is connected to your top-level HDL design through instantiation and compiled with the rest of the design.

Next you will create a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 7-segment LED display format.

Using the New Source Wizard and ISE Text Editor

Enter the name and ports of the component in the New Source Wizard, and the wizard creates a "skeleton" HDL file which you can complete with the remainder of your code.

1. In Project Navigator, select **Project** → **New Source**.

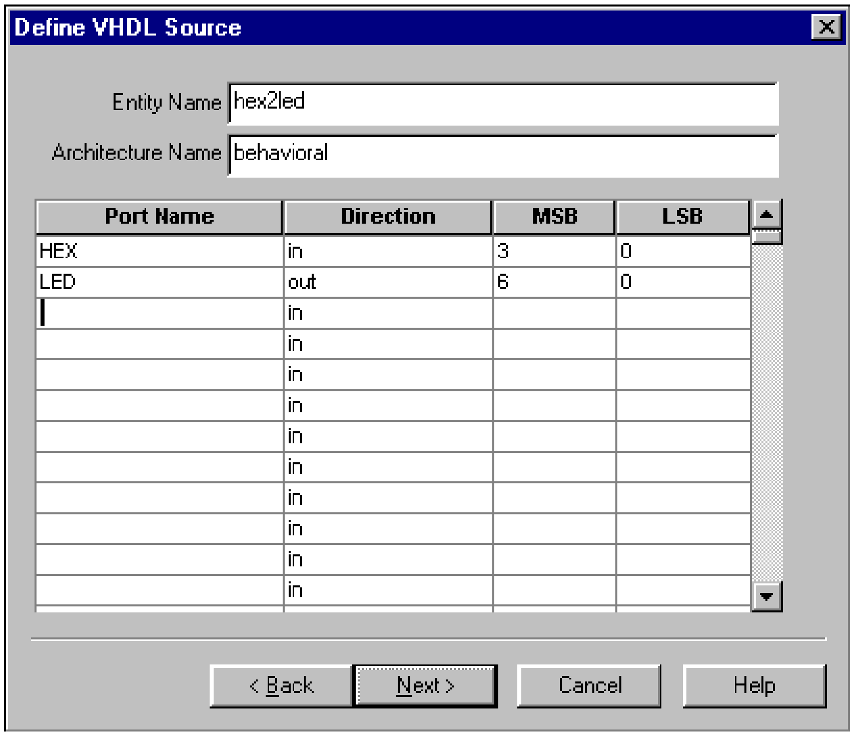
   The New Source dialog box opens.

2. Select the source type **VHDL Module** or **Verilog Module**, depending on your coding preference.
3. In the File Name field, type **hex2led**.
4. Click **Next**.

   The hex2led component has a 4-bit input port named HEX and a 7-bit output port named LED. First enter the port named HEX as follows:

5. Click in the Port Name field and type **HEX**.
6. Click in the Direction field and set the direction to **in**.
7. In the MSB field, enter **3**, and in the LSB field, enter **0**.
8. Repeat the previous steps for the LED(6:0) output bus. Be sure that the direction is set to out.

   The dialog box entries are displayed in Figure 3-29.



Entity Name: hex2led
Architecture Name: behavioral

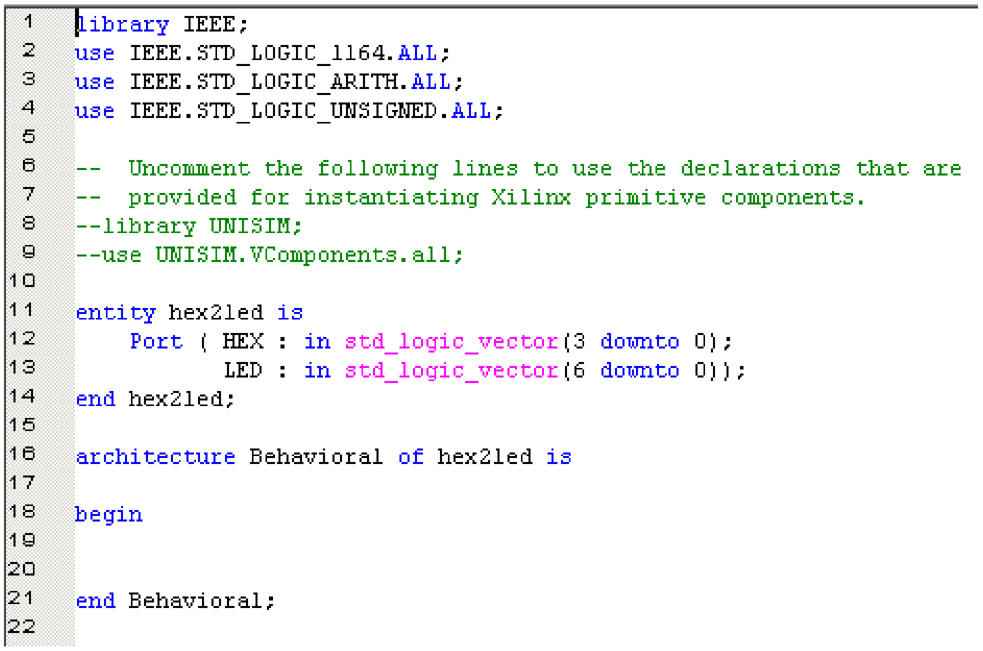
Port Name	Direction	MSB	LSB
HEX	in	3	0
LED	out	6	0
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Figure 3-29: **New Source Wizard**

9. Select **Next** to complete the Wizard session.

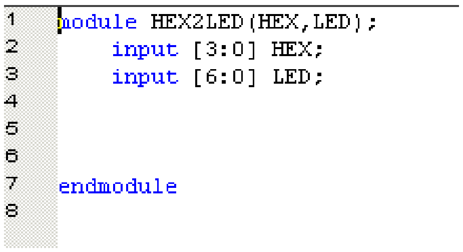
   A description of the module displays.

10. Select **Finish**. The "skeleton" HDL file opens in the ISE Text Editor.



```
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 hex2led is
    Port ( HEX : in std_logic_vector(3 downto 0);
           LED : in std_logic_vector(6 downto 0));
end hex2led;

architecture Behavioral of hex2led is

begin


end Behavioral;

```

Figure 3-30: **Skeleton VHDL File**



```
module HEX2LED(HEX,LED);
    input [3:0] HEX;
    input [6:0] LED;



endmodule

```

Figure 3-31: **Skeleton Verilog File**

In the HDL file, the ports are already declared and some of the basic file structure is already in place. Keywords are displayed in blue, data types in red, comments in green, and values in black. This color-coding enhances readability and recognition of typographical errors.

Using the Language Templates

The ISE Language Templates are HDL constructs and synthesis templates that represent commonly used logic components, such as counters, D flip-flops, multiplexers, and primitives.

Note: You can add your own templates to the Language Templates for components or constructs that you use often.

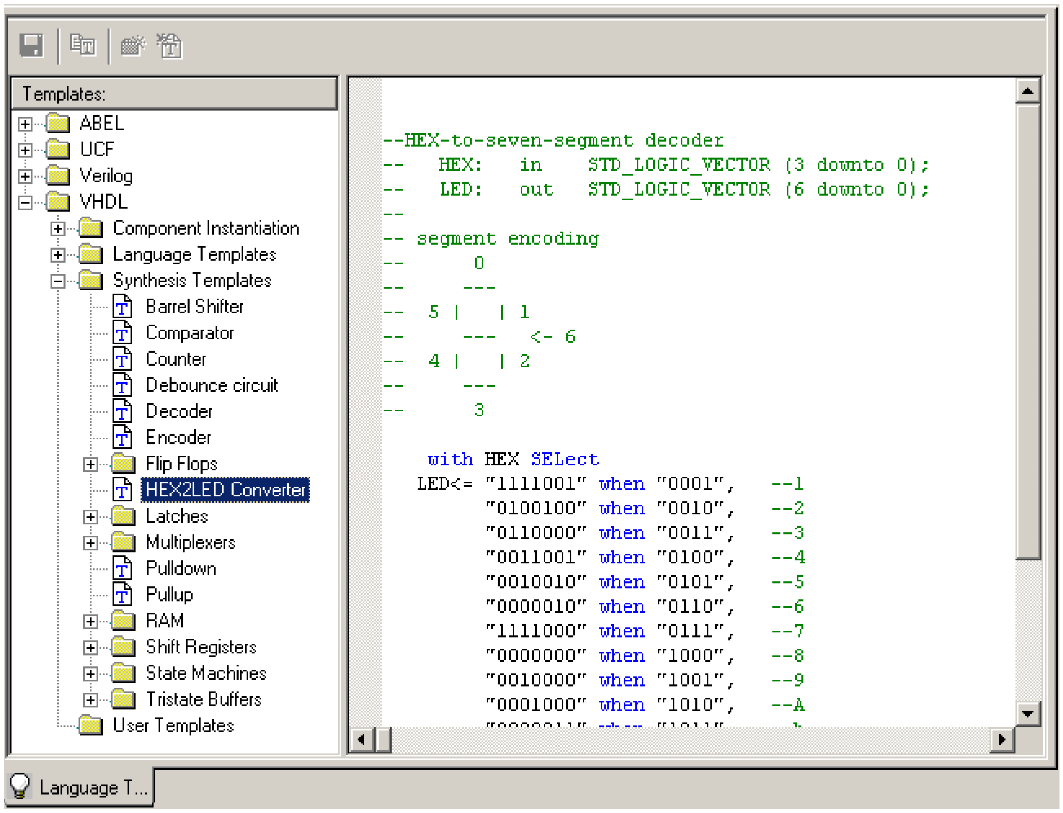
To invoke the Language Templates window and select a template for this tutorial:

1. In Project Navigator, select **Edit → Language Templates**.

   Each HDL language in the Language Template is divided into four sections: Component Instantiations, Language Templates, Synthesis Templates, and User Templates. To expand the view of any of these sections, click the + next to the topic. Click any of the listed templates to view the template contents in the right-hand pane.

2. Locate the template called **HEX2LED Converter** for VHDL or Verilog located under the Synthesis Templates heading. Use the appropriate template for the language you are using.
3. To preview the HEX2LED Converter template, click the template in the hierarchy. The contents display in the right-hand pane.

   This template provides source code to convert a 4-bit value to 7-segment LED display format.



```
--HEX-to-seven-segment decoder
--   HEX:   in    STD_LOGIC_VECTOR (3 downto 0);
--   LED:   out   STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoding
--        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
```

Figure 3-32: **Language Templates**

Adding the Language Template to Your File

Next, using the drag and drop method, add a template to your HDL file. A copy and paste function is also available from the Language Template Edit Menu and right-click menu.

To add the HEX2LED language template to your file:

1. In the Language Templates, click and drag the **HEX2LED Converter** name into
   - the *hex2led.vhd* file under the architecture begin statement.

   or

   - the *hex2led.v* file under the module declaration.
2. Close the Language Templates window.
3. (Verilog only) After the input and output statements and before the HEX2LED converter that you just added, add the following line of code to the HDL file to allow an assignment.

```
reg [6:0] LED;
```

You now have complete and functional HDL code.

4. Save the file by selecting **File** → **Save**.
5. In Project Navigator, select *hex2led.vhd* or *hex2led.v* in the Sources in Project window.
6. Double-click **Check Syntax** located in the Synthesize hierarchy in the Processes for Current Source window. This launches the ISE Text Editor.
7. Exit ISE Text Editor.

Creating and Placing the HEX2LED Symbol

Next, create the schematic symbol representing the HEX2LED HDL in Project Navigator.

1. In the Sources in Project window, select *hex2led.vhd* or *hex2led.v*.
2. In the Processes for Current Source window, click the + beside Design Entry Utilities to expand the hierarchy.
3. Double-click **Create Schematic Symbol**.

You are now ready to place the HEX2LED symbol on the Watch schematic.

1. In the Sources in Project window, double-click *watch.sch*. The schematic file opens in the ECS schematic editor.
2. Select **Add** → **Symbol** or click the Add Symbol icon from the Tools toolbar.

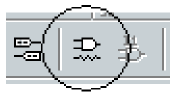


Figure 3-33: **Add Symbol Icon**

This opens the Symbol Browser dialog box to the left of the schematic editor, which displays the libraries and their corresponding components to view the list of available library components.

Locate the HEX2LED macro in this list.

3. Select **HEX2LED**, and add two instances of this symbol to the Watch schematic, as shown in Figure 3-34. You will connect the entire schematic later in the tutorial.

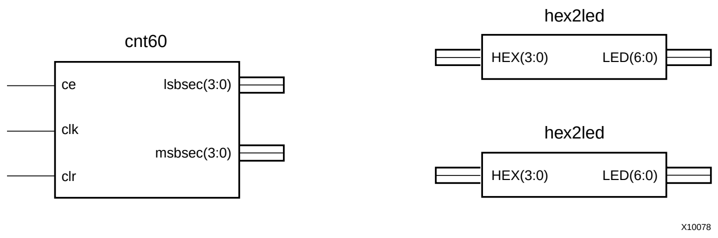


Figure 3-34: **Placing the HEX2LED Component**

Specifying Device Inputs/Outputs

Use the I/O marker to specify device I/O on a schematic sheet. All ECS schematics are netlisted to VHDL or Verilog and then synthesized by the synthesis tool of choice. When the synthesis tool synthesizes the top-level HDL, the I/O markers are replaced with the appropriate pads and buffers.

Hierarchy Push/Pop

First, perform a hierarchy "push down" which enables you to focus in on a lower-level of the schematic hierarchy to view the underlying file. Push down into the OUTS3 macro, which is a schematic-based user-created macro, and examine its components.

To push down into OUTS3, in ECS:

1. Click **OUTS3 symbol** in the schematic and select the Hierarchy Push icon. You can also right-click the macro and select **Push into Symbol**.

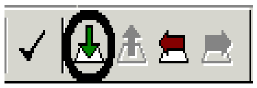


Figure 3-35: **Hierarchy Push Icon**

In the OUTS3 schematic, you see a series of inverters (INV) and output buffers (OBUF). This macro illustrates that you can place I/O buffers in a lower level macro. The output buffers are not required because the synthesis tool inserts a buffer when one is not found.

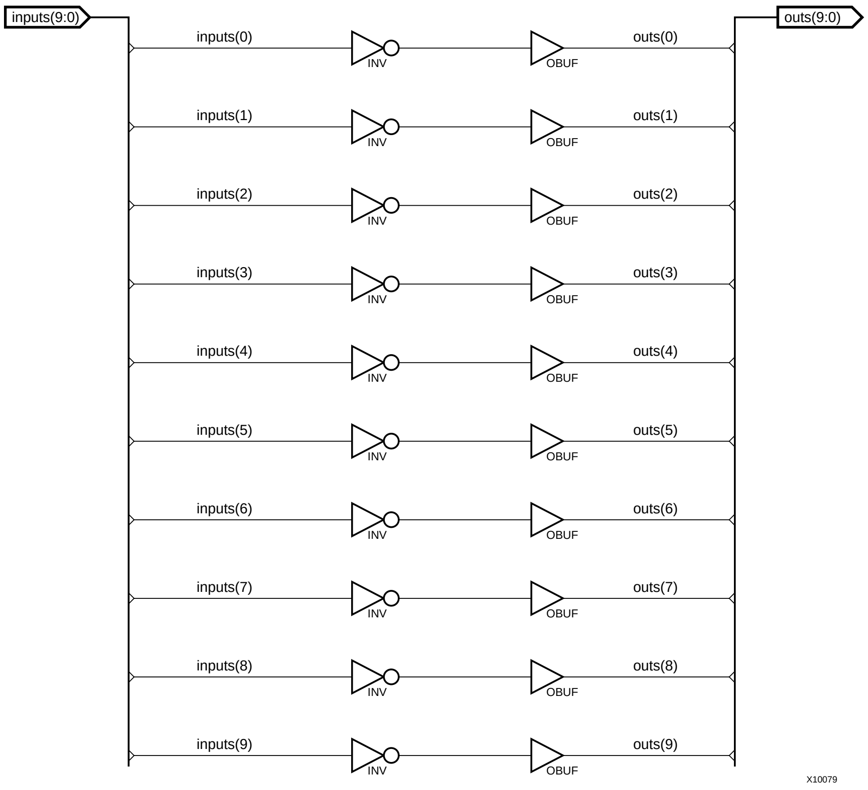


Figure 3-36: **OUTS3 Schematic Symbol**

2. After examining the macro, exit the *outs3.sch* view.

Adding Input Pins

Next, add three more input pins to the Watch schematic: CLK, RESET and STRTSTOP. Add an IBUF component for two of the new input pins: RESET and STRTSTOP.

To add these components:

1. Click the Add Symbol icon in the toolbar to open the Symbol Browser dialog box.
2. Browse to locate the IBUF and INV components in the library.
3. Drag and drop these two components onto the schematic, as shown below.
4. Draw a hanging wire to the input of the IBUFs and DCM1. Refer to the "Drawing Wires" for detailed instructions.
5. Draw a net between the output of the IBUF and input of the INV. Refer to "Drawing Wires" for detailed instructions.

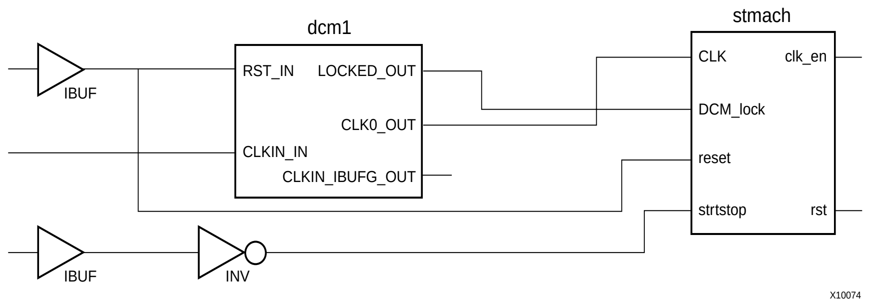


Figure 3-37: **Placing CLK, RESET and STRTSTOP I/O Components**

Adding I/O Markers and Net Names

It is important to label nets and buses for several reasons:

- It aids in debugging and simulation, as you can more easily trace nets back to your original design.
- Any nets that remain unnamed in the design will be given generated names that will mean nothing to you later in the implementation process.
- Naming nets also enhances readability and aids in documenting your design.

Label the three input nets you just drew. Refer to the completed schematic below. To label the RESET net:

1. Select **Add** → **Net Name**.
2. Type **reset** into the Name box.

   The net name is now attached to the cursor.
3. Place the name on the leftmost end of the net as illustrated in Figure 3-38.
4. Repeat Steps 1 through 3 for the STRTSTOP and CLK pins.

   Once all of the nets have been labeled, add the I/O marker.
5. Select **Add** → **I/O Marker**.
6. In the Add I/O Marker Options dialog box, select **Add an input Marker** for an input signal direction.

7. Click and drag a box around the three labeled nets to place an input signal around each net name.

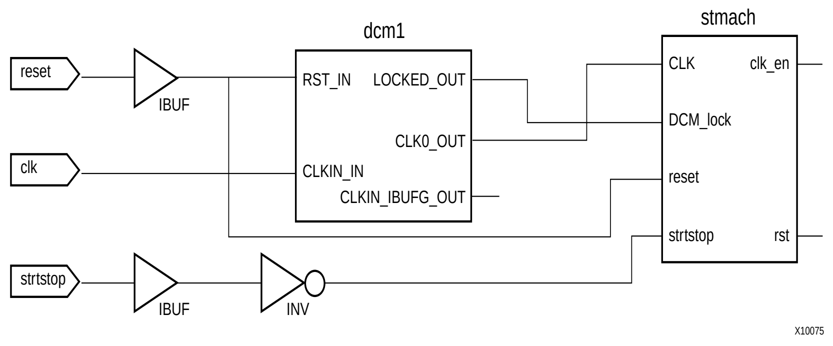


Figure 3-38: **Labeled Nets with I/O Markers**

Assigning Pin Locations

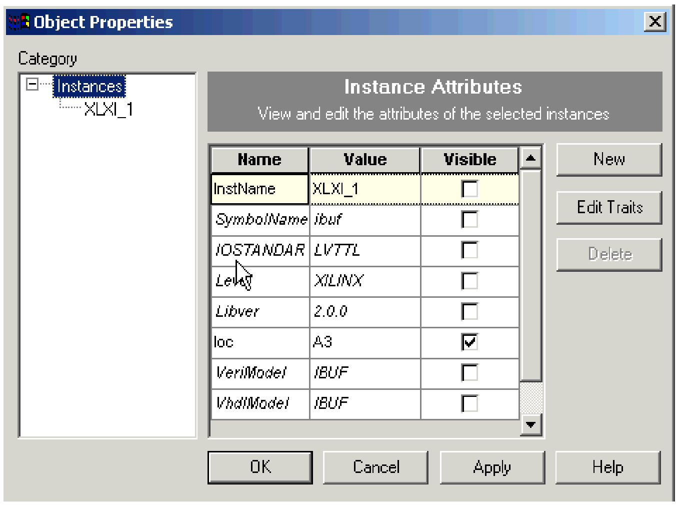
Xilinx® recommends that you let the automatic placement and routing (PAR) program define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place-and-route tools. However, it may be necessary at some point to lock the pinout of a design so that it can be integrated into a Printed Circuit Board (PCB).

For this tutorial, you will define the initial pinout by running the place-and-route tools without pin assignments. Then you will lock down the pin placement so that it reflects the locations chosen by the tools. Because the tutorial Watch design is simple and timing is not critical, the example pin assignments will not adversely affect the ability of PAR to place and route the design.

Specify pin locations by attaching a LOC parameter to a buffer component. In ECS, assign a LOC parameter to the RESET net on the Watch schematic as follows:

1. Right-click on the IBUF component connected to the RESET I/O marker, and from the menu, select **Object Properties**.
2. Click the **New** button under Instance Attributes to add a new property.
3. Enter *loc* for the Attribute Name and *A5* for the Attribute Value.

4. Click **OK** to return to the Object Properties dialog box.



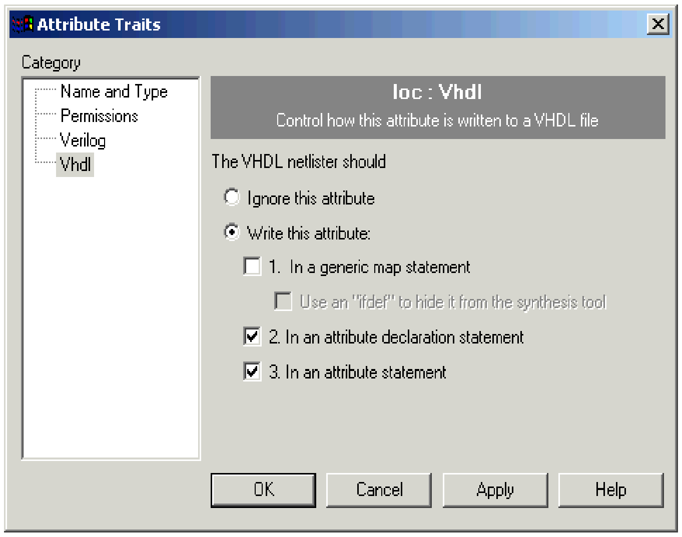
Name	Value	Visible
InstName	XLXI_1	☐
SymbolName	ibuf	☐
IOSTANDAR	LVTTL	☐
Level	XILINX	☐
Libver	2.0.0	☐
loc	A3	☑
VeriModel	IBUF	☐
VhdlModel	IBUF	☐

Figure 3-39: **Assigning Pin Locations**

5. Check to make sure the visible box is selected.

   This will display the LOC attribute on the schematic.

6. With the loc attribute selected, click **Edit Traits**.
7. Select **VHDL** and select **Write this attribute** as well as options **2** and **3** as shown below.



The VHDL netlister should

- ○ Ignore this attribute
- ● Write this attribute:
  - ☐ 1. In a generic map statement
    - ☐ Use an "ifdef" to hide it from the synthesis tool
  - ☑ 2. In an attribute declaration statement
  - ☑ 3. In an attribute statement

Figure 3-40: **Writing attribute to HDL file**

8. Click **OK** twice to return to schematic.

Note: For more constraint options in the implementation tools, see "Editing Constraints in the Constraints Editor" and "Editing Constraints in the Pinout Area Constraints Editor (PACE)" in Chapter 5, "Design Implementation."

Completing the Schematic

Complete the schematic by wiring the components you have created and placed, adding any additional necessary logic, and labeling nets appropriately. The following steps guide you through the process of completing the schematic, or you may want to use the completed schematic shown below for guidance. Each of the actions referred to in this section has been discussed in detail in earlier sections of the tutorial. Please see the earlier sections for detailed instructions.

The finished schematic is shown in the following figure as a guide.

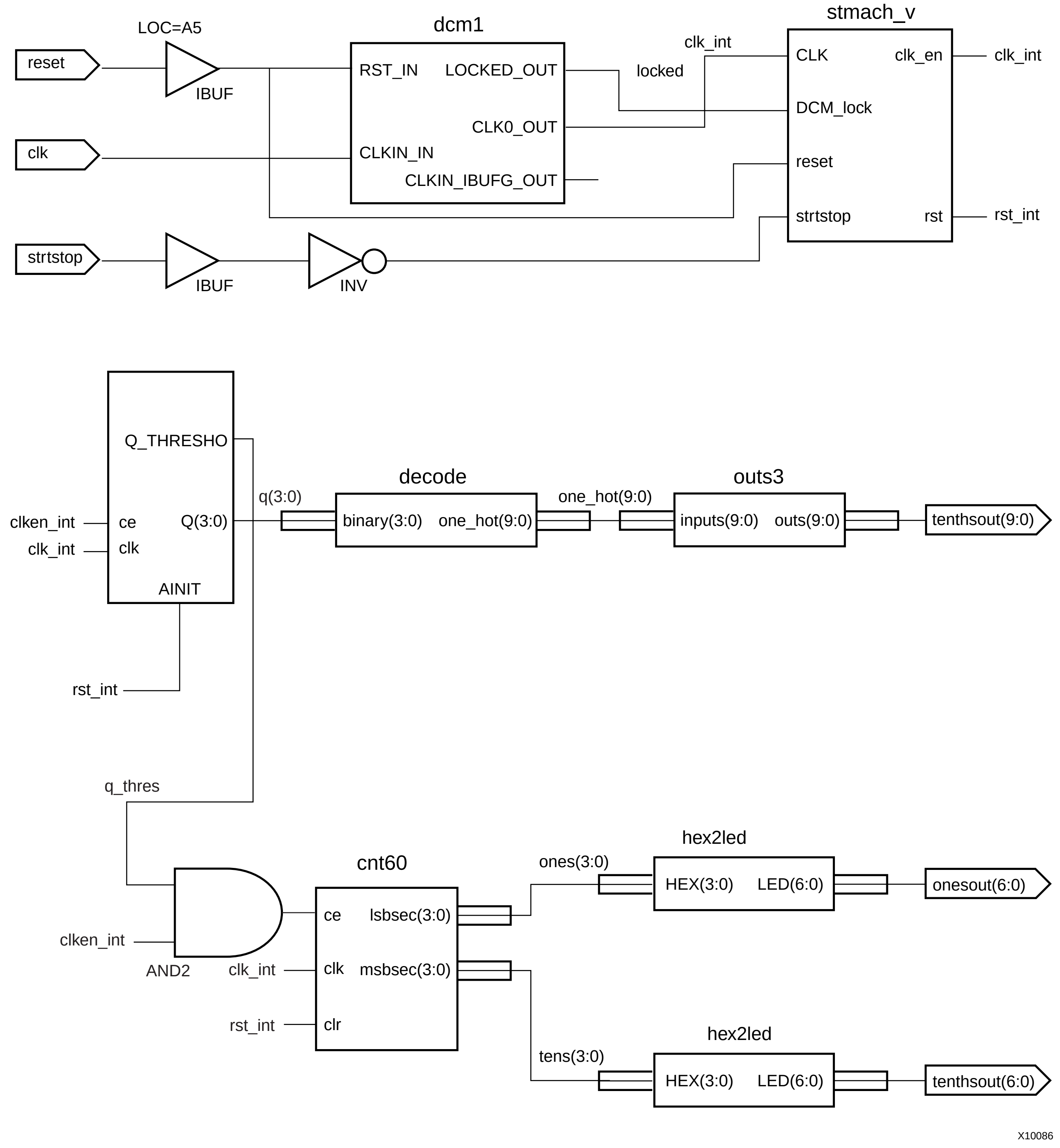


Figure 3-41: **Completed Watch Schematic**

To complete the schematic diagram:

1. Draw a wire between the CLK0_OUT pin of DCM1 and the CLK pin of the STMACH state machine macro (see “Drawing Wires.”)
2. Draw a wire between the LOCKED_OUT pin of DCM1 and the DCM_lock pin of the STMACH state machine macro (see “Drawing Wires.”)
3. Label this net CLK_INT.
4. Draw wires between the IBUF of the RESET input and the RESET pins of the STMACH and DCM1 macros (see “Drawing Wires.”)
5. Place an INV (inverter) component from the Virtex library between the IBUF of the STRTSTOP input and the STRTSTOP pin of the STMACH state machine macro (see “Adding Components to CNT60.”)
6. Draw wires to connect the INV to both the IBUF and the STMACH state machine macro (see “Drawing Wires.”)
7. Place an AND2 component to the left of the CNT60 macro (see “Adding Components to CNT60.”)
8. Draw a wire to connect the output of the AND2 with the CE pin of the CNT60 macro (see “Drawing Wires.”)
9. Draw a wire to connect the Q_THRES0 pin of the TENTHS macro to one of the inputs to the AND2 (see “Drawing Wires.”)
10. Draw a hanging net from the clken pin of the STMACH macro. To terminate a hanging wire, double-click it (see “Drawing Wires.”)
11. Name the new added net CE.
12. Draw a hanging net at the CLK_EN input pin of the TENTHS macro. Label this net CLKEN_INT (see “Adding I/O Markers and Net Names.”)
13. Draw a hanging wire (see “Drawing Wires”) at the other input of the AND2 component. Label this net CLKEN_INT again (see “Adding I/O Markers and Net Names.”)

    Note: Remember that nets are logically connected if their names are the same, even if the net is not physically drawn as a connection in the schematic. This method is used to make the logical connection of the RST_INT, CLKEN_INT and CLK_INT signals.
14. Draw a hanging wire from the RST output pin of the STMACH macro (see “Drawing Wires.”)
15. Label this net RST_INT.
16. Draw two more hanging wires, also named RST_INT, from the AINIT pin of the TENTHS macro and from the CLR pin of the CNT60 macro (see “Drawing Wires.”)
17. Draw two hanging wires, each named CLK_INT, from the CLOCK pin of the TENTHS macro and from the CLK pin of the CNT60 macro (see “Drawing Wires.”)
18. Draw buses to complete the schematic. Label them as shown on the preceding schematic diagram (see “Adding Buses.”)

The schematic is now complete.

Save the design by selecting **File** → **Save**.

Behavioral Simulation

This chapter contains the following sections.

- "Overview of Behavioral Simulation Flow"
- "ModelSim Setup"
- "Getting Started"
- "Adding an HDL Test Bench"
- "Behavioral Simulation Using ModelSim"

Overview of Behavioral Simulation Flow

Behavioral simulation is done before the design is synthesized to verify that the logic you have created is correct. This allows a designer to find and fix any bugs in the design before spending time with Synthesis or Implementation.

Xilinx® ISE provides an integrated flow with the ModelTech ModelSim simulator that allows simulations to be run from the Xilinx Project Navigator graphical user interface (GUI). The examples in this tutorial show how to use this integrated flow. For additional information about simulation and for a list of the other supported simulators, refer to Chapter 6 of *Synthesis and Verification Guide*. This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

ModelSim Setup

In order to follow this tutorial, you need to install ModelSim on your machine. The next sections discuss requirements and setup for ModelSim PE, ModelSim SE and ModelSim XE.

ModelSim PE and SE

ModelSim PE and ModelSim SE are the full versions of the ModelSim product that can be purchased directly from ModelTech. In order to simulate with the ISE 6 libraries, use ModelSim 5.6 or later. Older versions may work but are not supported.

Note: For more information on purchasing ModelSim PE or SE version 5.6 or later, contact ModelTech.

ModelSim Xilinx Edition

ModelSim Xilinx® Edition II (MXE II) is the Xilinx version of ModelSim which is based on ModelSim PE. Two versions exists: a starter version that is free, and a full version that can be purchased from Xilinx.

MXE II 5.7c must be used with the ISE 6.1i software, as this is the only version for which the latest 6.1i libraries have been compiled.

Note: When ISE 6.2i is released, a newer version of MXE II will also be released. If you are using ISE 6.2i, you will need to use the newer version of MXE II as well.

- For information on how to obtain MXE II, go to the *Getting Started* section of the MXE II Tech Tips page:

  http://support.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=MXE+II

- For general information about MXE II, go to the *FAQ* section of the MXE II Tech Tips page:

  http://support.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=MXE+II

Getting Started

The following sections outline the requirements to perform behavioral simulation in this tutorial.

Required Files

The behavioral simulation flow requires design files, a test bench file and Xilinx simulation libraries.

- **Design Files (VHDL, Verilog, or Schematic)**

  This chapter assumes that you have completed the tutorial design entry by following Chapter 2, "HDL-Based Design," or Chapter 3, "Schematic-Based Design." After you have completed one of these chapters, your design includes the required design files and is ready for simulation.

- **Test Bench File**

  In order to simulate the design, a test bench is required to provide stimulus to the design. A VHDL and Verilog test bench are available with the tutorial files. Alternatively, you may choose to create you own test bench from scratch, for which instructions are found in "Creating a Test Bench Waveform Using HDL Bencher" in this chapter.

  Note: The HDL Bencher flow is only available on Windows platforms.

- **Xilinx Simulation Libraries**

  Xilinx simulation libraries are required when any Xilinx primitive is instantiated in the design. The design in this tutorial requires the use of simulation libraries because it contains instantiations of a digital clock manager (DCM) and a CORE Generator™ component. Information on simulations libraries and how to compile them is provided in the next section.

Xilinx Simulation Libraries

To simulate designs that contain instantiated Xilinx® primitives or CORE Generator™ components, you need to use the Xilinx simulation libraries. These libraries contain models for each component. These models reflect the functions of each component, and provide the simulator with the information required to perform simulation.

Note: For a detailed description of each library, refer to Chapter 6 of the *Synthesis and Verification Design Guide*. This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

Updating the Xilinx Simulation Libraries

The Xilinx simulation libraries contain models that are updated on a regular basis. Model updates occur as follows:

- The XilinxCoreLib models are updated each time an IP Update is installed.
- All other models are updated each time a service pack is installed.

When the models are updated, your libraries need to be re-compiled. The compiled Xilinx simulation libraries are then available during the simulation of any design.

ModelSim Xilinx Edition II

If you are using ModelSim Xilinx Edition II (MXE II), the updated models are precompiled and available on the Xilinx Support Website for download. To get the latest precompiled models for MXE II, go to http://support.xilinx.com/support/mxelibs/index.htm

ModelSim PE or SE

If you are using ModelSim PE or SE, you will need to compile the simulation libraries with the updated models. Refer to Xilinx Answer Record # 15338 for instructions on how to compile the libraries. To locate Answer Record # 15338:

1. Go to http://support.xilinx.com.
2. Enter **compile libraries** in the search box.
3. Check to see that the search engine is pointing to **Answer Database**.
4. Click **OK**.

   Answer Record # 15388 displays on the next page.
5. Click on Answer Record # 15338.

Mapping Simulation Libraries in the Modelsim.ini File

ModelSim uses the *modelsim.ini* file to determine the location of the compiled libraries. For instance, if you compiled the UNISIM library to c:\lib\UNISIM, the following mapping should appear in the *modelsim.ini* file:

```
UNISIM = c:\lib\UNISIM
```

ModelSim searches for a *modelsim.ini* file in the following order:

- The modelsim.ini file pointed to by the MODELSIM environment variable if it exists.
- The modelsim.ini file in the current working directory if one exists.
- The modelsim.ini file in the directory where ModelSim or MXE is installed.

If the MODELSIM environment variable is not set and the modelsim.ini file has not been copied to the working directory, the modelsim.ini file in the installation directory will be used.

For this tutorial, verify the mapping for your edition of ModelSim:

ModelSim Xilinx Edition II

If you are using ModelSim Xilinx Edition II (MXE II), open the *modelsim.ini* file in the directory where MXE II was installed. You will see that all of the Xilinx simulation libraries are already mapped to the proper location.

ModelSim PE or SE

If you are using ModelSim PE or SE, you should have gone through Answer Record #15338 and used COMPXLIB to compile the libraries. During that process, COMPXLIB will also update the *modelsim.ini* file with the correct mapping. Open the modelsim.ini file and make sure that the library mappings are correct.

Note: In future, you can copy the modelsim.ini file to the working directory and make changes that are specific to that project or you could use the MODELSIM environment variable to point to the desired modelsim.ini file.

Adding an HDL Test Bench

In order to add an HDL test bench to your design project, you have the option of either adding the tutorial test bench files provided with this tutorial, or creating your own test bench files and adding them to your project.

Adding Tutorial Test Bench File

This section demonstrates how to add pre-existing test bench files to the project. A Verilog and VHDL test bench have been provided with this tutorial. You can either use the provided test bench, or skip to the next section and create a test bench using HDL Bencher™.

VHDL Design

To add your test bench for a VHDL design:

1. Select **Project** → **Add Source**.
2. Select the test bench file *stopwatch_tb.vhd*.
3. Click **Open**.

   The Choose Source Type dialog box opens.
4. Select **VHDL Test Bench File**.
5. Click **OK**.

ISE recognizes the top-level design file associated with the test bench, and adds the test bench in the correct order.

Verilog Design

To add your test bench for a Verilog design:

1. Select **Project** → **Add Source**.
2. Select the test bench file *stopwatch_tb.v*.
3. Click **Open**.

   The Choose Source Type dialog box opens.
4. Select **Verilog Test Fixture File**.
5. Click **OK**.

ISE recognizes the top-level design file associated with the test bench, and adds the test bench in the correct order.

Creating a Test Bench Waveform Using HDL Bencher

This section demonstrates how to use the HDL Bencher™ tool, which is a PC-based test bench and test fixture creation tool in ISE 6. You can use HDL Bencher to graphically enter stimuli and to generate a VHDL test bench or Verilog test fixture. It is not necessary to follow this section if you have added the tutorial test bench to the project already.

There are two parts to this section:

- "Creating a Test Bench Waveform Source"
- "Applying Stimulus"

Note: HDL Bencher is available for PC only.

Creating a Test Bench Waveform Source

To create a test bench or test fixture with HDL Bencher:

1. Select *stopwatch* in the Sources in Project window.
2. Select **Project** → **New Source** from the Project Navigator menu.
3. In the New dialog box, select **Test Bench Waveform** as the source type.
4. Type the name *stopwatch_tb*.
5. Click **Next**.

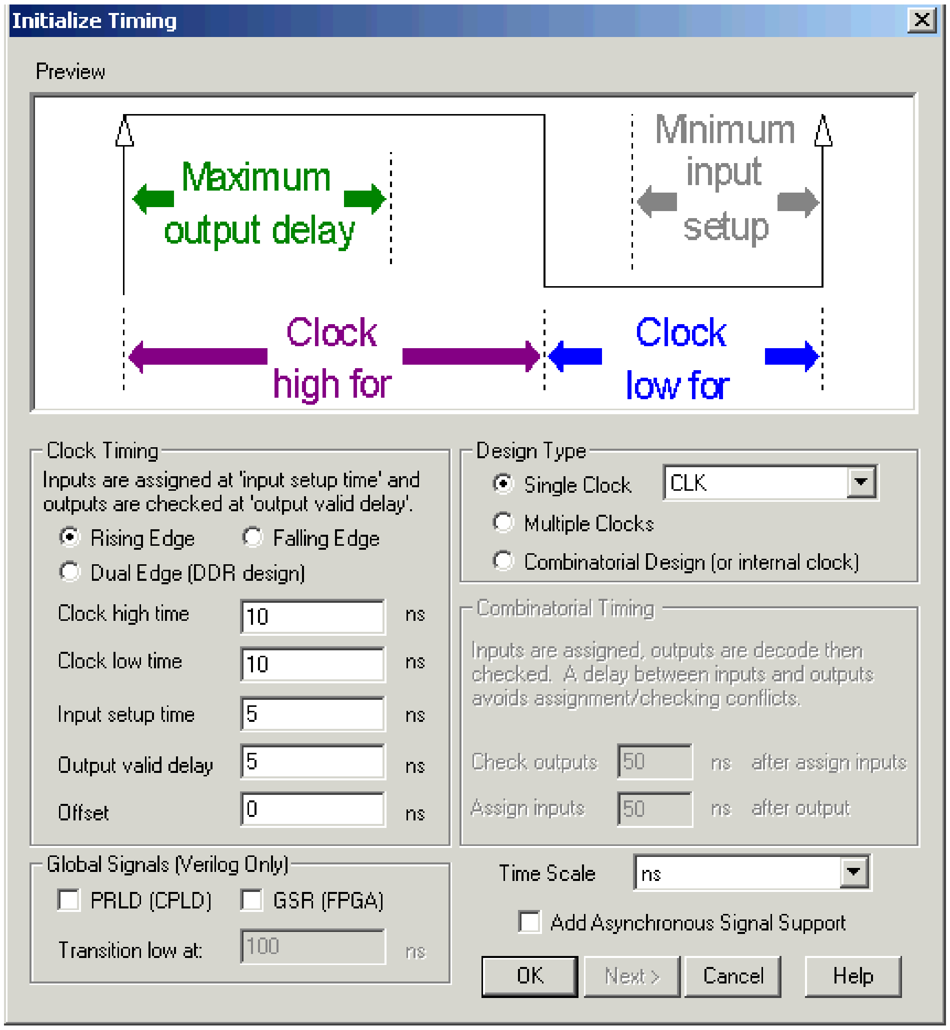
Note: In the Select dialog box, the stopwatch file is the default source file because it is selected in the Sources in Project window (step 1).

6. Click **Next.**
7. Click **Finish**.

HDL Bencher launches in ISE. You are prompted to specify the timing parameters used during simulation. The clock high time and clock low time together define the clock period for which the design must operate. The Input setup time defines when inputs must be valid. The Output valid delay defines the time after active clock edge when the outputs must be valid.

For this tutorial, use the settings in Figure 4-1.

If you are using the Verilog project, select **GSR** under Global Signals.



Clock Timing

Inputs are assigned at 'input setup time' and outputs are checked at 'output valid delay'.

[✓ Rising Edge] Falling Edge
Dual Edge (DDR design)

Clock high time	10	ns
Clock low time	10	ns
Input setup time	5	ns
Output valid delay	5	ns
Offset	0	ns

Global Signals (Verilog Only)

PRLD (CPLD) ☐ GSR (FPGA) ☐

Transition low at: 100 ns

Design Type

[✓ Single Clock] CLK
Multiple Clocks
Combinatorial Design (or internal clock)

Combinatorial Timing

Inputs are assigned, outputs are decode then checked. A delay between inputs and outputs avoids assignment/checking conflicts.

Check outputs 50 ns after assign inputs

Assign inputs 50 ns after output

Time Scale ns

☐ Add Asynchronous Signal Support

Figure 4-1: **HDL Bencher Initialization**

Click **OK**, and the HDL Bencher™ window displays in the right-pane of ISE.

Applying Stimulus

Enter the following input stimuli:

1. Click the RESET cell at time 0 to set it high (RESET is active high).
2. Click the STRTSTOP cell at time 0 to set it high (STRTSTOP active low).
3. Click the RESET cell under CLK cycle 6 to set it low.

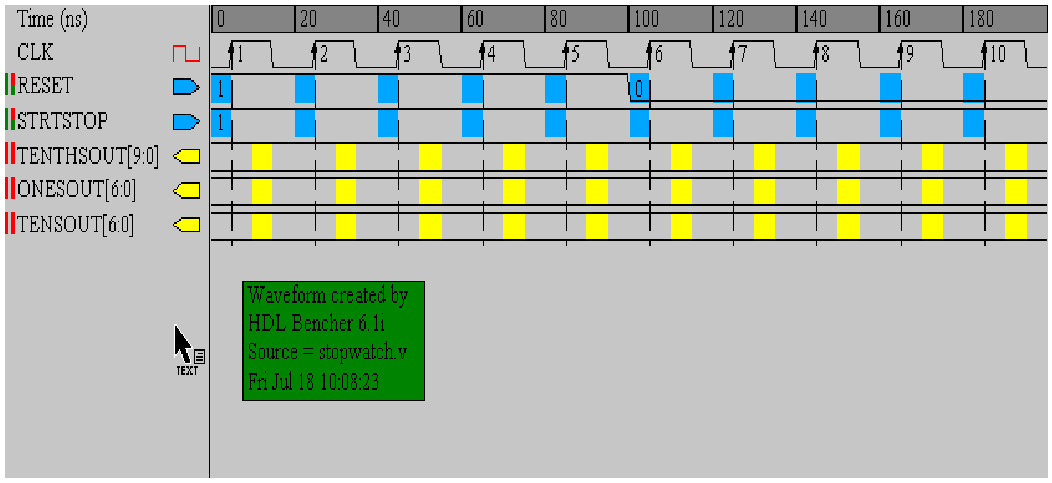


Figure 4-2: **Applying Stimulus in HDL Bencher Window**

4. Click the STRTSTOP cell under CLK cycle 51 to set it low.
5. Grab the blue line (end of test bench) and drag it to CLK cycle 80.
6. Click the Save icon in the toolbar.

Note: STRTSTOP is not asserted until CLK cycle 50 to give the DCM time to lock.

The new test bench waveform source (*stopwatch_tb.tbw*) is automatically added to the project.

Behavioral Simulation Using ModelSim

Now that you have a test bench in your project, you can perform behavioral simulation on the design. ISE has full integration with the ModelSim Simulator. ISE enables ModelSim to create the work directory, compile the source files, load the design, and perform simulation based on simulation properties.

Locating the Simulation Processes

The simulation processes in ISE enables you to run simulation on the design using ModelSim. To locate the ModelSim simulator processes:

1. In the Sources in Project window, select the test bench file (*stopwatch_tb*).
2. Click the + beside **ModelSim Simulator** to expand the process hierarchy.

Note: If the ModelSim Simulator processes do not appear, it means that either ModelSim is not installed or Project Navigator cannot find modelsim.exe.

If ModelSim is installed but the processes are not available, the Project Navigator preferences may not to set correctly. To set the ModelSim location, select **Edit** → **Preferences** and click the **Integrated Tools** tab. Under Model Tech Simulator, browse to the location of *modelsim.exe* file. For example, c:\modeltech_xe\win32xoem\modelsim.exe.

The following simulation processes are available:

- **Simulate Behavioral Model**

  This process will start the design simulation.

- **Generate Expected Simulation Results**

  This process is available only if you have a TBW file from HDL Bencher™. If you double-click on this process, ModelSim will run in the background to generate expected results and display them in HDL Bencher.

- **Simulate Post-Translate VHDL (or Verilog) Model**

  Simulates the netlist after the NGDBuild implementation stage.

- **Simulate Post-Map VHDL (or Verilog) Model**

  Simulates the netlist after the Map implementation stage.

- **Simulate Post-Place & Route VHDL (or Verilog) Model**

  Simulates the back-annotated netlist after Place & Route, which contains the detailed timing information as well.

Specifying Simulation Properties

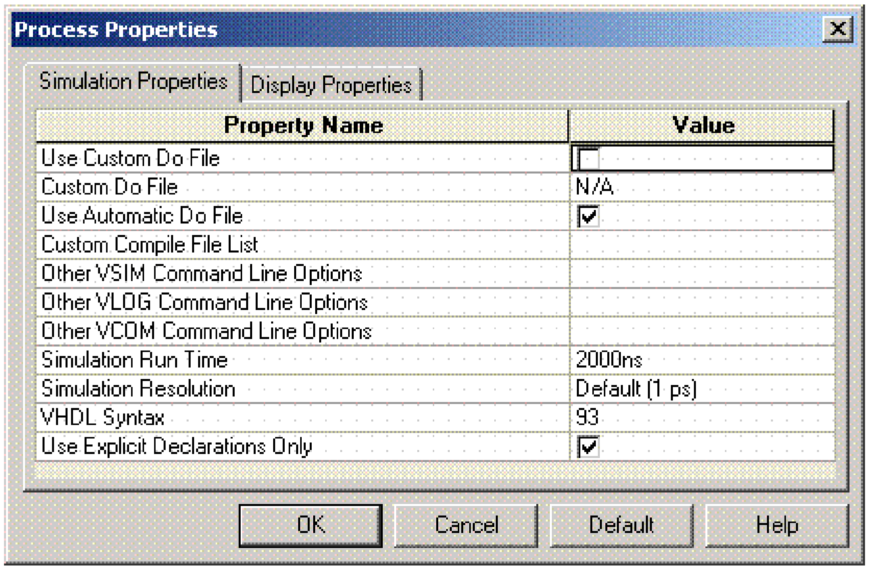
In this chapter, you will perform a behavioral simulation on the Watch design. You must first specify the process properties for simulation as shown in the following section.

ISE allows you to set several ModelSim Simulator properties in addition to the simulation netlist properties. To see which properties are available for behavioral simulation:

1. In the Sources in Project window, select the test bench file (*stopwatch_tb*).
2. Click the + sign next to **ModelSim Simulator** to expand the hierarchy in the Processes For Current Source window.
3. Right-click on **Simulate Behavioral Model**.
4. Select **Properties**.

The Process Properties dialog box (Figure 4-3) displays.

5. Change the Simulation Run Time to **2000 ns**



Property Name	Value
Use Custom Do File	☐
Custom Do File	N/A
Use Automatic Do File	✓
Custom Compile File List	
Other VSIM Command Line Options	
Other VLOG Command Line Options	
Other VCOM Command Line Options	
Simulation Run Time	2000ns
Simulation Resolution	Default (1 ps)
VHDL Syntax	93
Use Explicit Declarations Only	✓

Figure 4-3: **Behavioral Simulation Process Properties**

6. Click **OK** to continue.

Note: For a detailed description of each property available in a Process Property dialog box, click the **Help** button.

Performing Simulation

Once the process properties have been set, you are ready to run ModelSim. To start the behavioral simulation, double-click **Simulate Behavioral Model**. ModelSim creates the work directory, compiles the source files, loads the design, and performs simulation for the time specified.

Adding Signals

To view signals during the simulation, you must add them to the Wave window. ISE automatically adds all the top-level ports to the Wave window. Additional signals are displayed in the Signal window based upon the selected structure in the Structure window.

There are two basic methods for adding signals to the Simulator Wave window.

- Drag and drop from the Signal window.
- Highlight signals in the wave window and then select **Add** → **Wave** → **Selected Signals** from the Signal window.

The following procedure explains how to add additional signals in the design hierarchy. For the purpose of this example, add the lsbsec and msbsec signals in the cnt60 macro.

1. In the Structure window, click the + next to **UUT** to expand the hierarchy.

Figure 4-4 shows the Structure window for the Verilog flow. The graphics and the layout of the Structure window for a schematic or VHDL flow may appear different.

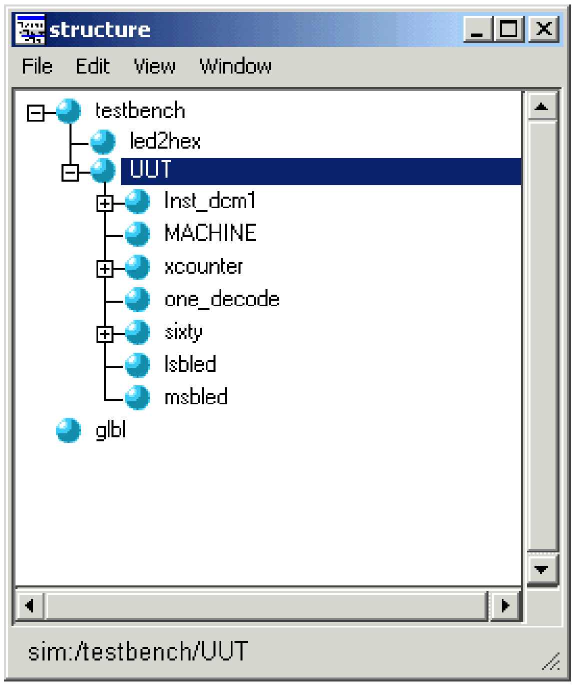


Figure 4-4: **Structure Window - Verilog flow**

2. Select **sixty** in the Structure window.

   Notice that the signals listed in the Signal window are updated.

3. Click and drag **lsbsec** from the Signal window to the Wave window.
4. Select **msbsec** in the Signal window and select **Add** → **Wave** → **Selected Signals** to add the signal to the Wave window.

Notice that the waveforms have not been drawn for lsbsec or msbsec. This is because ModelSim did not record the data for these signals. By default, ModelSim will only record data for the signals that have been added to the waveform window while the simulation is running. Therefore, when new signals are added to the waveform window, the simulation needs to be restarted and re-run for the desired amount of time.

To restart and re-run the simulation:

1. Click **Restart Simulation**.

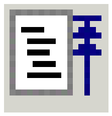


Figure 4-5: **Restart Simulation Icon**

The Restart dialog box opens

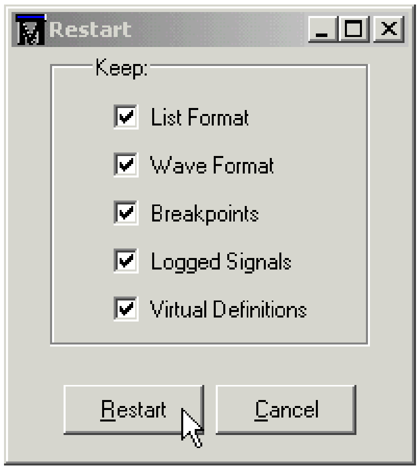
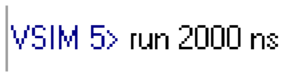


Figure 4-6: **Restart Dialog Box**

2. Click **Restart**.
3. At the ModelSim command prompt, enter “run 2000 ns” and hit enter.



```
VSIM 5> run 2000 ns
```

Figure 4-7: **Entering the run command at the ModelSim command prompt**

4. The simulation will run for 2000 ns. The waveforms for lsbsec and msbsec should now be visible in the Waveform Window.

Saving the Simulation

The ModelSim Simulator provides the capability of saving the signals list in the Wave window. This can be important when additional signals or stimuli are added, and the simulation is restarted. The saved signals list can easily be loaded each time the simulation is started.

In the Wave window, select **File** → **Save Format**.

5. In the Save Format dialog box, rename the default filename *wave.do* to *sec_signal.do*.

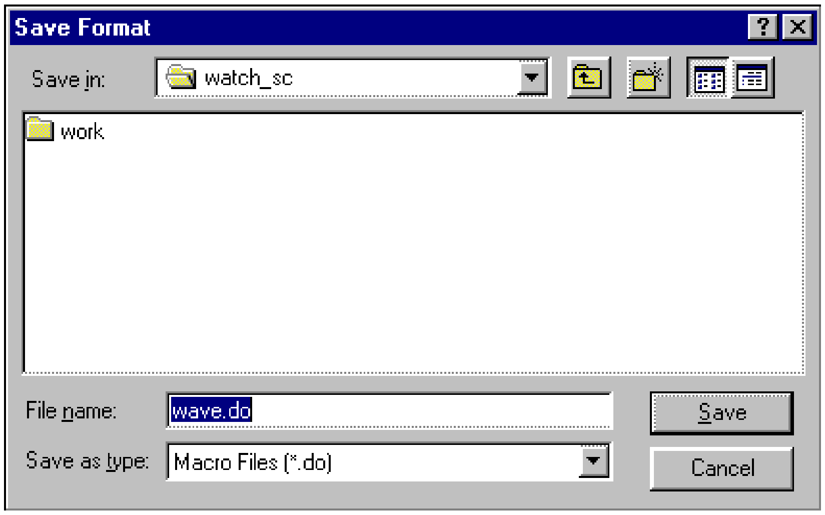


Figure 4-8: **Save Format Dialog Box**

6. Click **Save**.

After restarting the simulation, you can select **File** → **Load Format** in the Wave window to load this file.

Chapter 5

Design Implementation

This chapter contains the following sections.

Overview of Design Implementation

Design implementation is the process of translating, mapping, placing, routing, and generating a BIT file for your design. The design implementation tools are embedded in ISE for easy access and project management.

This chapter is the first in the "Implementation-only Flow" and is an important chapter for the "HDL Design Flow" and the "Schematic Design Flow".

This chapter demonstrates the ISE Implementation flow. The front-end design has already been compiled in an EDA interface tool. For details about compiling the design, see Chapter 2, "HDL-Based Design" or Chapter 3, "Schematic-Based Design." In this chapter, you will be passing an input netlist (EDN, NGC) from the front-end tool to the back-end design implementation tools, and incorporating placement constraints through a user constraints file (UCF). You will add timing constraints later through the Constraints Editor and the Pinout Area Constraints Editor (PACE).

Getting Started

The tutorial Watch design emulates a runner's stopwatch. There are two inputs to the system: RESET and SRTSTP. The configuration clock on the device is used as a ten-hertz clock signal. This system generates three seven-bit outputs for output to three seven-segment LED displays. There are two options in this tutorial for design implementation.

Tutorial Option 1

Go through the previous chapters and synthesize the design to create the EDIF/NGC Netlist File. If you don't have a *stopwatch.ucf* file, you will need to create one.

To add a UCF file to the design:

1. Select **xc2v40-5fg256**.
2. Select **Project** → **New Source**.
3. Select **Implementation Constraints File**.
4. Type **stopwatch.ucf** as the file name.
5. Click **Next**.
6. Select **stopwatch** from the list.
7. Click **Next**.
8. Click **Finish.**
9. Go to the "Specifying Options" section in this chapter.

Tutorial Option 2

Use the EDIF Netlist Files that are provided. If you choose this option, create a working directory with the tutorial files as follows.

1. Create an empty working directory named Watch.
2. Copy the Required Tutorial Files listed in the following table from the http://support.xilinx.com/support/techsup/tutorials/tutorial6.html directory into your newly created working directory.

Table 5-1: **Required Tutorial Files**

File Name	Description
stopwatch.edn, *stopwatch.edf*, or *stopwatch.ngc*	Input netlist file (EDIF)
tenths.edn	Counter netlist file (EDIF)
stopwatch.ucf	User Constraints File

Creating an Implementation Project

This section describes how to create a project with ISE. The process is the same for either Schematic or HDL designs.

To create a project:

1. Open ISE.
   a. On a UNIX workstation, enter `ise &`

   Note: The implementation-only flow of this tutorial is the only flow available for a UNIX environment.

   b. On a PC, select **Start** → **Programs** → **Xilinx ISE6** → **Project Navigator**.

If you are continuing this project from the previous chapters, go to "Specifying Options."

If you are using the pre-synthesized design, create a new project and add the (stopwatch) EDIF netlist as follows:

1. Select **File** → **New Project**.
2. Type **EDIF Flow** for the Project Name.
3. Select **EDIF** for the top_level Module Type.
4. Click **Next**.
5. Select *stopwatch.edn* for the Input Design file
6. Select *stopwatch.ucf* for the Constraints file.
7. Click **Next**.
8. Select the following:
   ♦ **Virtex2** for the Device Family
   ♦ **xc2v40** for the Device
   ♦ **-5** for the Speed Grade, **fg256** for the Package
9. Click **Next**.
10. Click **Finish**.

When you create a new project, specify a design to open and a directory for the project. You can create as many projects as you want, but you can only work with one at a time. See "Snapshot View" in Chapter 1 for strategies for project revision control using snapshots.

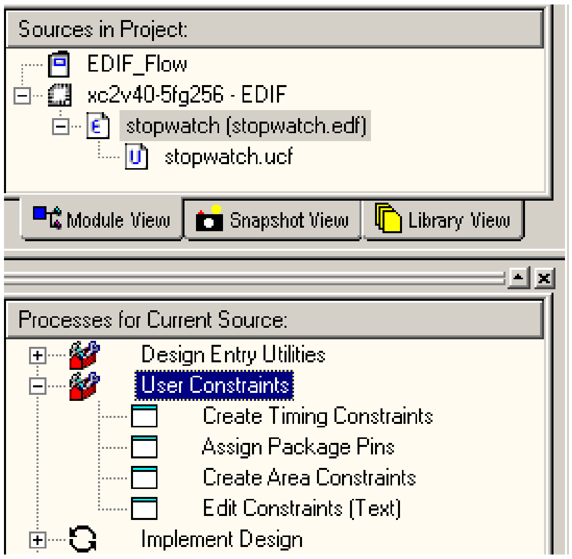


Figure 5-1: **Selecting File in Sources in Project Window**

In the Sources in Project window, select the top-level module, *stopwatch.edf* or *stopwatch.edn*. This enables the design to be implemented.

Specifying Options

This section describes property options and sets two options for this tutorial. The options associated with the **Implement** process control how the software maps, places, routes and optimizes a design. To access these options, right-click on any process in the Implement process hierarchy and select **Properties** from the menu. The list of options is found in the Process Properties dialog box that displays.

ISE offers two property display levels: Standard and Advanced.

- **Standard**

  Standard options are the most commonly used options. By default, Standard property options are displayed in ISE.

- **Advanced**

  The Advanced options provide access to all options, including options not needed for a pushbutton flow. Advanced options are intended for experienced users. By default, Advanced property options are not displayed in ISE. You must select the Advanced property display level in the Preferences dialog box.

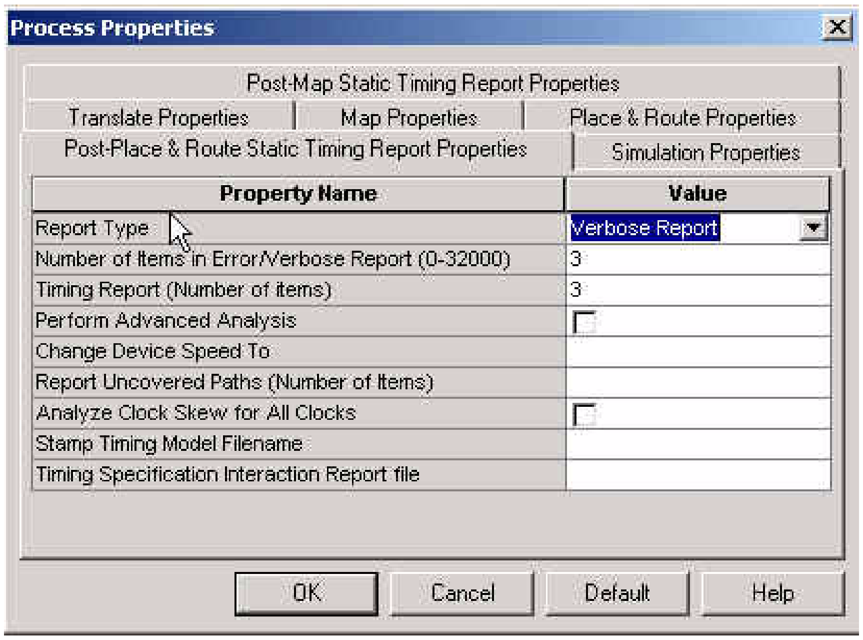
While not necessary for the tutorial, you can select the Advanced property display level. With this property display level specified, you will see both Advanced and Standard properties.

1. Select **Edit** → **Preferences**.
2. In the Preferences dialog box, click the **Processes** tab.
3. Change the Property Display Level from Standard to **Advanced**.
4. Click **OK**.

Next, you will change the settings for two options: the Report Type option and the Place & Route Effort Level (Overall):

1. In the Processes for Project window, right-click **Implement Design**.
2. Select **Properties**.

The Process Properties dialog box provides access to the Translate, Map, Place and Route, Simulation, and Timing Report options.



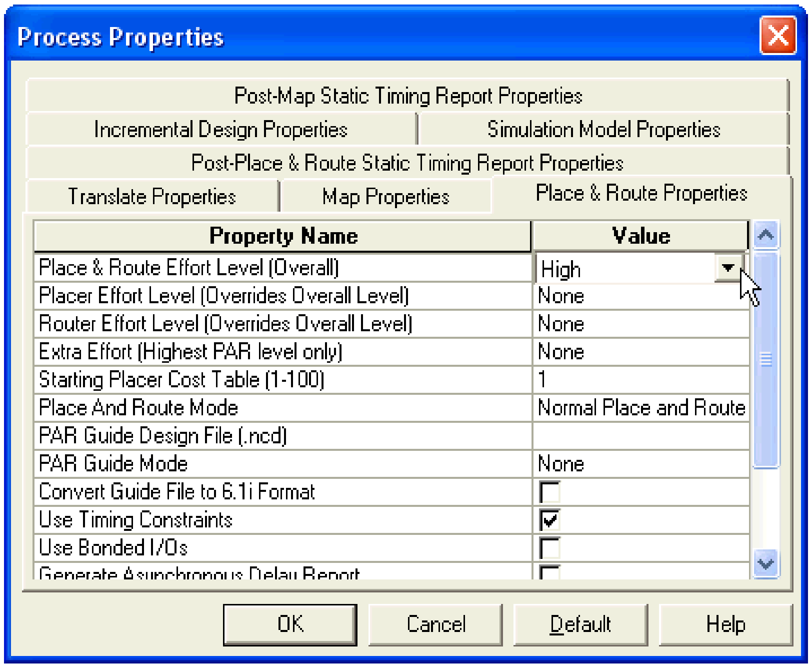
Property Name	Value
Report Type	Verbose Report
Number of Items in Error/Verbose Report (0-32000)	3
Timing Report (Number of items)	3
Perform Advanced Analysis	☐
Change Device Speed To	
Report Uncovered Paths (Number of Items)	
Analyze Clock Skew for All Clocks	☐
Stamp Timing Model Filename	
Timing Specification Interaction Report file	

Figure 5-2: **Post-Place & Route Static Timing Report Properties**

3. In the Post-Place & Route Static Timing Report Properties tab, change Report Type to **Verbose Report**.

   This option changes the type of report from an error report to a verbose report. This report is created after Place and Route is completed.

4. In the Place & Route Properties tab, change the Place & Route Effort Level (overall) to **High**. This option increases the overall effort level of Place and Route during implementation.


Process Properties
Post-Map Static Timing Report Properties
Incremental Design Properties
Simulation Model Properties
Post-Place & Route Static Timing Report Properties
Translate Properties
Map Properties
Place & Route Properties
OK
Cancel
Default
Help

Property Name	Value
Place & Route Effort Level (Overall)	High
Placer Effort Level (Overrides Overall Level)	None
Router Effort Level (Overrides Overall Level)	None
Extra Effort (Highest PAR level only)	None
Starting Placer Cost Table (1-100)	1
Place And Route Mode	Normal Place and Route
PAR Guide Design File (.ncd)	
PAR Guide Mode	None
Convert Guide File to 6.1i Format	☐
Use Timing Constraints	☑
Use Bonded I/Os	☐
Generate Asynchronous Delay Report	☐

Figure 5-3: **Place & Route Properties**

5. Click **OK** to exit the Process Properties dialog box.

Translating the Design

ISE manages the files created during implementation. The ISE tools use the settings you supply in the Process Properties dialog box. This gives you complete control over how a design is processed. Typically, you set your options first. You then run through the entire flow by clicking **Implement Design**, and selecting **Process** → **Run**. This tutorial illustrates the implementation one step at a time.

Note that during translation, the program NGDBuild performs the following functions:

- Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
- Performs timing specification and logical design rule checks.
- Adds a user constraints file (UCF) to the merged netlist.

Creating and Editing Timing Constraints

The user constraints file (UCF) provides a mechanism for constraining a logical design without returning to the design entry tools. However, without the design entry tools, you must understand the exact syntax needed to define constraints. In the Xilinx® Development System, the Constraints Editor and the Pinout Area Constraints Editor (PACE) are graphical tools that enable you to enter timing and pin location constraints.

Launch the Constraints Editor as follows:

1. Expand the **User Constraints** hierarchy.
2. Double-click **Create Timing Constraints**.

   This automatically runs the Translate process and launches the Constraints Editor.

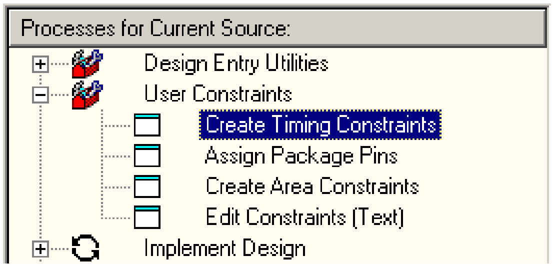


Figure 5-4: **Create Timing Constraints**

Editing Constraints in the Constraints Editor

In the Constraints Editor you can:

- Edit constraints previously defined in a UCF file.
- Create new constraints for your design.

Input files to the Constraints Editor are:

- **NGD (Native Generic Database) File**

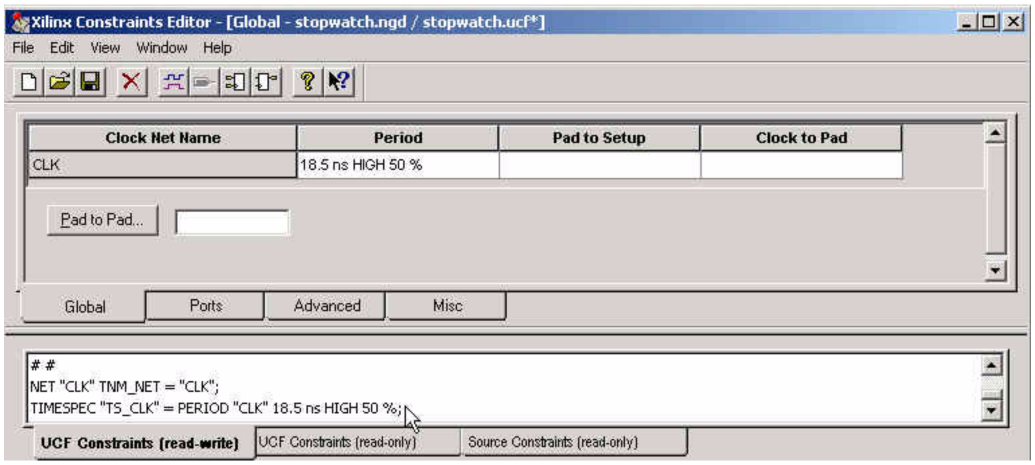
  The NGD file serves as input to the mapper, which then outputs the physical design database, an NCD (Native Circuit Description) file.

- **Corresponding UCF (User Constraint File)**

  By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

The Constraints Editor generates a valid UCF file. The Translate step (NGDBuild) uses the UCF file and the design source netlists to produce a newer NGD file which incorporates the changes made. The Map program (the next section in the design flow) then reads the NGD. In this design, the *stopwatch.ngd* file and *stopwatch.ucf* files are automatically read into the Constraints Editor.

The Global tab appears in the foreground of the Constraints Editor window. This window automatically displays all the clock nets in your design, and enables you to define the associated period, pad to setup, and clock to pad values.


Xilinx Constraints Editor - [Global - stopwatch.ngd / stopwatch.ucf*]
File Edit View Window Help
Pad to Pad...
Global
Ports
Advanced
Misc
UCF Constraints (read-write)
UCF Constraints (read-only)
Source Constraints (read-only)

Clock Net Name	Period	Pad to Setup	Clock to Pad
CLK	18.5 ns HIGH 50 %		

```
# #
NET "CLK" TNM_NET = "CLK";
TIMESPEC "TS_CLK" = PERIOD "CLK" 18.5 ns HIGH 50 %;
```

Figure 5-5: **Constraints Editor**

In the Constraints Editor, edit the constraints as follows:

1. Double-click the Period cell on the row associated with the clock net CLK. The Clock Period dialog box opens.
2. For the Clock Signal Definition, keep the default (Specific Time) selected to define an explicit period for the clock.
3. Enter a value of **18.5** in the Time text box.
4. Verify that **ns** is selected from the Units pull-down list.
5. Click **OK**.

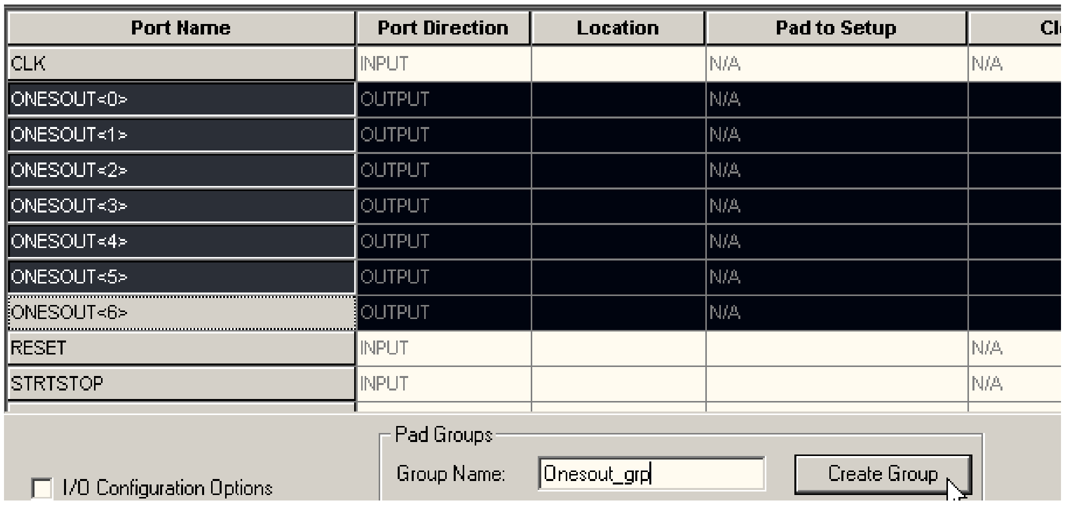
   The period cell is updated with the global clock period constraint that you just defined (with a default 50% duty cycle).

   Note: For the purpose of this tutorial, you have opened a secondary dialog box by double-clicking a cell to specify your constraint values. You can also click once in a cell and enter constraints directly.
6. Select the **Ports tab** from the Constraints Editor main window.

   The left hand side displays a listing of all the current ports as defined by the user.
7. Select **OnesOut<0>** in the Port Name Column.
8. Hold the **Shift** key and select **OnesOut<6>**.

   This selects the group of elements.

9. In the Group Name text box, type **OnesOut_grp**, and click **Create Group** to create the group.



Port Name	Port Direction	Location	Pad to Setup	Cl
CLK	INPUT		N/A	N/A
ONESOUT<0>	OUTPUT		N/A	
ONESOUT<1>	OUTPUT		N/A	
ONESOUT<2>	OUTPUT		N/A	
ONESOUT<3>	OUTPUT		N/A	
ONESOUT<4>	OUTPUT		N/A	
ONESOUT<5>	OUTPUT		N/A	
ONESOUT<6>	OUTPUT		N/A	
RESET	INPUT			N/A
STRTSTOP	INPUT			N/A

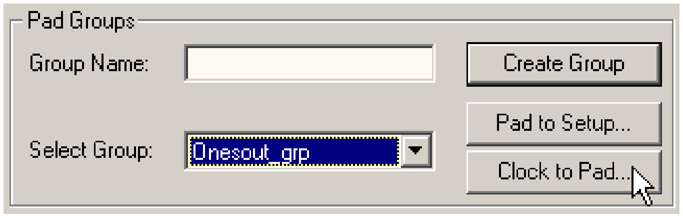
I/O Configuration Options

Pad Groups

Group Name: Onesout_grp

Figure 5-6: **Selected Elements of a Grouped OFFSET**

10. In the Select Group pull-down list, select the group you just created.
11. Click **Clock to Pad**.



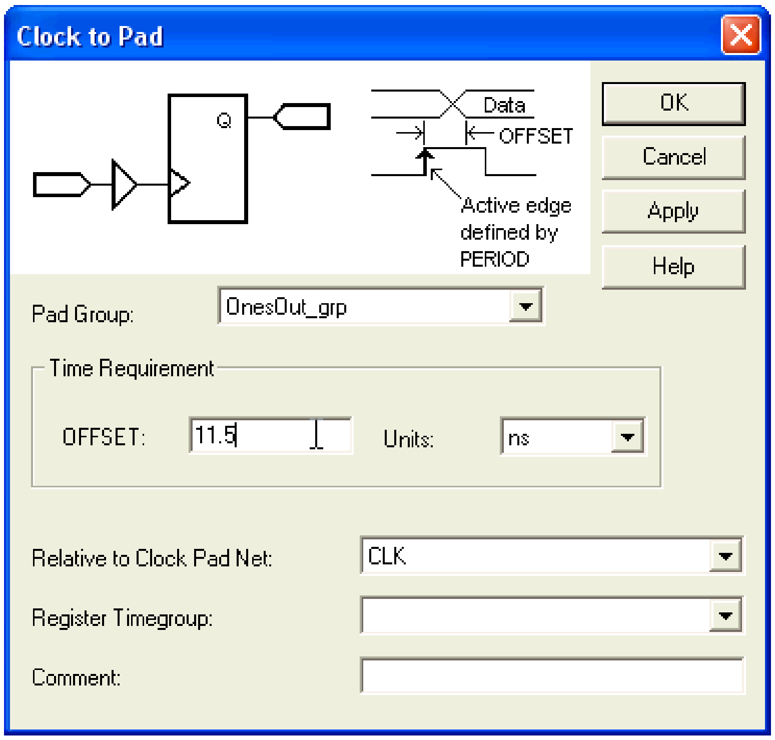
Pad Groups

Group Name:

Select Group: Onesout_grp

Figure 5-7: **Selecting the Group that was created to use in an OFFSET**

The Clock to Pad dialog box opens.



Pad Group: OnesOut_grp

Time Requirement

OFFSET: 11.5 Units: ns

Relative to Clock Pad Net: CLK

Register Timegroup:

Comment:

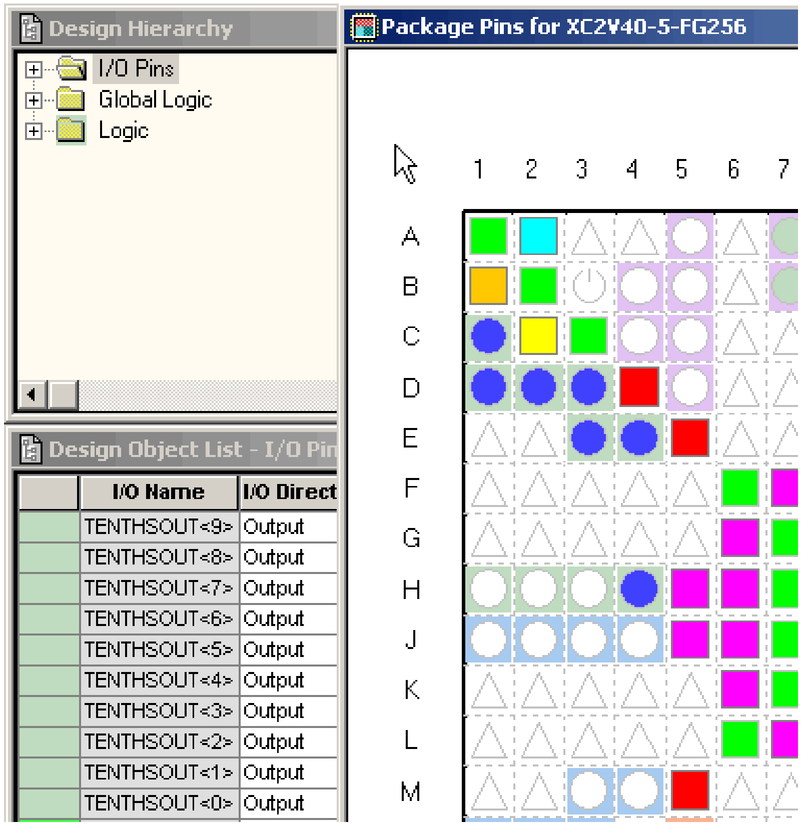
Figure 5-8: **Clock to Pad Dialog**

12. Enter **11.5 ns** for the Timing Requirement.
13. Click **OK**.
14. Select **File** → **Save**.

    The changes made by the Constraints Editor are now saved in the *stopwatch.ucf* file in your current revision directory.
15. Exit the Constraints Editor.

Editing Constraints in the Pinout Area Constraints Editor (PACE)

Use the Pinout Area Constraints Editor (PACE) to add and edit the pin locations and area group constraints defined in the NGD file. PACE generates a valid UCF file. The Translate step uses this UCF file, along with the design source netlists, to produce a newer NGD file. The NGD file incorporates the changes made in the design and the UCF file from the previous section. PACE also places Global Logic at the Slice level with Block RAM, Digital Clock Managers (DCMs), Gigabit Transceivers (GTs), and BUFG buffers.



	I/O Name	I/O Direct
	TENTHSOUT<9>	Output
	TENTHSOUT<8>	Output
	TENTHSOUT<7>	Output
	TENTHSOUT<6>	Output
	TENTHSOUT<5>	Output
	TENTHSOUT<4>	Output
	TENTHSOUT<3>	Output
	TENTHSOUT<2>	Output
	TENTHSOUT<1>	Output
	TENTHSOUT<0>	Output

Figure 5-9: **The PACE tool**

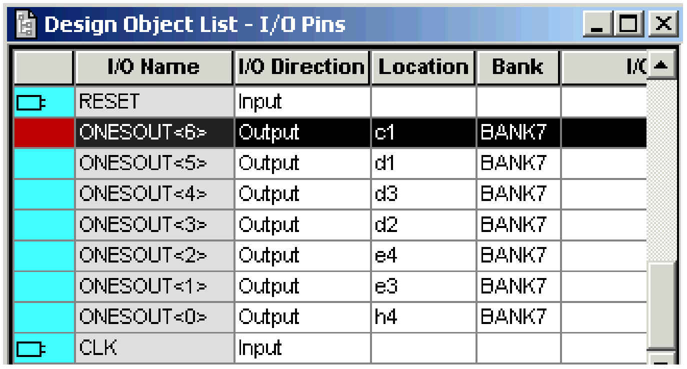
This section describes the creation of IOB assignments for several signals. PACE edits the UCF file by adding the newly created placement constraints.

Launch PACE and enter pin locations as follows:

1. In Project Navigator, double-click **Assign Package Pins** under the User Constraints hierarchy.
2. Select the Package Pin window.

   This window shows the graphical representation of the device package.
3. Select the Design Object List window.

   This window displays all the IO pins in the design.
4. In the Design Object List window, scroll down to the Onesout nets.
5. To enter the pin locations, click and type in pin locations in the Pin Location field associated with each of the following signals:
   - onesout<0> → H4
   - onesout<1> → E3
   - onesout<2> → E4
   - onesout<3> → D2
   - onesout<4> → D3
   - onesout<5> → D1
   - onesout<6> → C1


Design Object List - I/O Pins

	I/O Name	I/O Direction	Location	Bank	I/(
	RESET	Input			
	ONESOUT<6>	Output	c1	BANK7	
	ONESOUT<5>	Output	d1	BANK7	
	ONESOUT<4>	Output	d3	BANK7	
	ONESOUT<3>	Output	d2	BANK7	
	ONESOUT<2>	Output	e4	BANK7	
	ONESOUT<1>	Output	e3	BANK7	
	ONESOUT<0>	Output	h4	BANK7	
	CLK	Input			

Figure 5-10: **Pin Locations Typed in PACE**

To place some IO pins in the Package Pin window using the drag and drop method:

1. In the Design Object List window, click, drag and drop the following signals to the specific location in the Package Pins window:
   - Tenthsout<9> → A7
   - Tenthsout<8> → B7
   - Tenthsout<7> → A8
   - Tenthsout<6> → B8
   - Tenthsout<5> → C8
   - Tenthsout<4> → D8
   - Tenthsout<3> → D9
   - Tenthsout<2> → C9
   - Tenthsout<1> → B9
   - Tenthsout<0> → A9

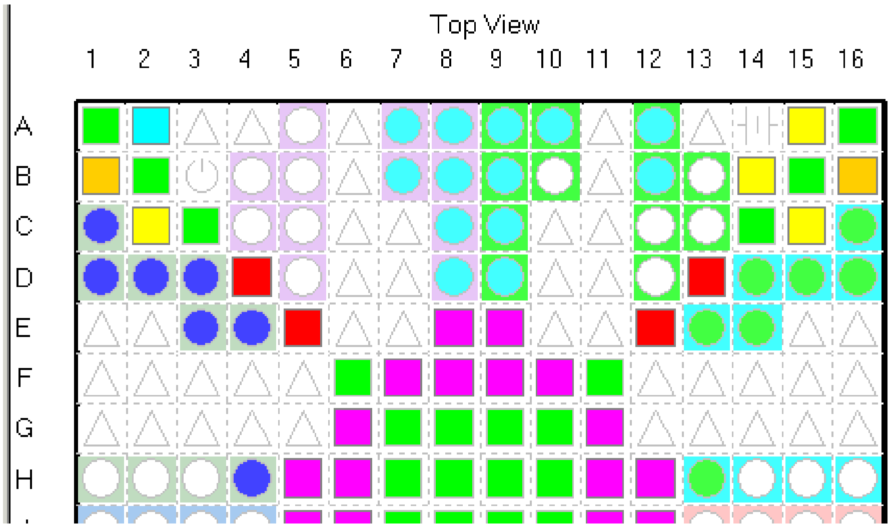


Figure 5-11: **Drag and Drop IOs in the Package Pins Window**

2. Once the pins are locked down, select **File** → **Save**. The changes made in PACE are now saved in the *stopwatch.ucf* file in your current working directory.
3. Exit PACE.

Mapping the Design

Now that all implementation strategies have been defined (options and constraints), continue with the implementation of the design.

1. In the Processes for Current Source window, right-click on **Map**.
2. Select **Run** from the menu.
3. Expand the Implement Design hierarchy to see the progress through implementation.

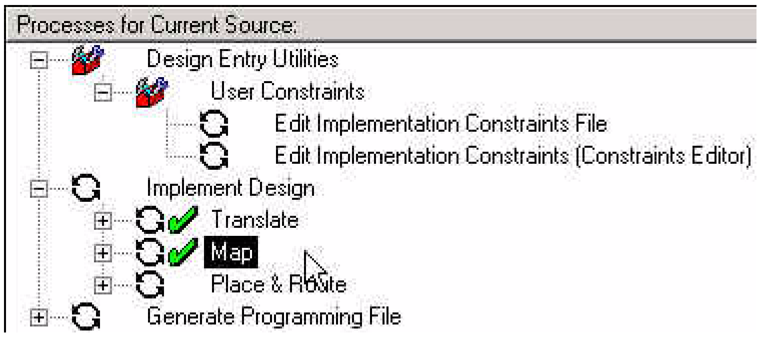

Processes for Current Source:
Design Entry Utilities
User Constraints
Edit Implementation Constraints File
Edit Implementation Constraints (Constraints Editor)
Implement Design
Translate
Map
Place & Route
Generate Programming File

Figure 5-12: **Mapping the Design**

The design is mapped into CLBs and IOBs. Map performs the following functions:

- Allocates CLB and IOB resources for all basic logic elements in the design.
- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

Each step generates its own report as shown in the following table.

Table 5-2: **Reports Generated By Map**

Report Type	Description
Translation Report	Includes warning and error messages from the translation process.
Map Report	Includes information on how the target device resources are allocated, references to trimmed logic, and device utilization. For detailed information on the Map report, refer to the *Development System Reference Guide*.

To view a report:

1. Expand the Translate or Map hierarchy.

2. Double-click a report, such as Translation Report or Map Report.

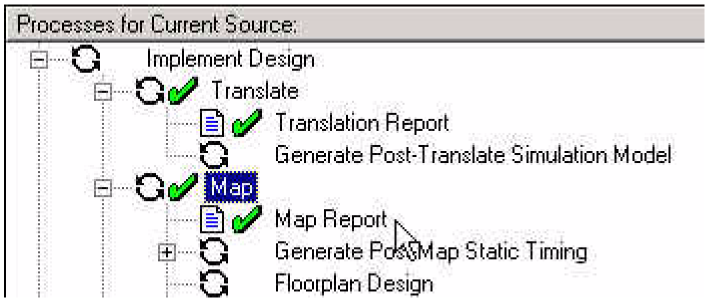

Processes for Current Source:
Implement Design
Translate
Translation Report
Generate Post-Translate Simulation Model
Map
Map Report
Generate Post-Map Static Timing
Floorplan Design

Figure 5-13: **Translation Report and Map Report**

3. Review the report for Warnings, Errors, and Information (INFO).

Using Timing Analysis to Evaluate Block Delays After Mapping

After the design is mapped, use the Logic Level Timing Report to evaluate the logical paths in the design. Because the design is not yet placed and routed, actual routing delay information is not available. The timing report describes the logical block delays and estimated routing delays. The net delays provided are based on an optimal distance between blocks (also referred to as *unplaced floor*s).

Estimating Timing Goals with the 50/50 Rule

For a preliminary indication of how realistic your timing goals are, evaluate the design after the map stage. The 50/50 rule offers a rough guideline to follow. The 50/50 rule specifies that the block delays in any single path are approximately 50% of the total path delay after the design is routed. For example, a path with 10ns of block delay should meet a 20ns timing constraint after it is placed and routed.

If your design is extremely dense, the Logic Level Timing Report provides a summary analysis of your timing constraints based on block delays and estimates of route delays that can help to determine if your timing constraints are going to be met. This report is produced after Map and prior to Place and Route (PAR).

Report Paths in Timing Constraints Option

Use the Logic Level Timing Report to determine timing violations that may occur prior to running PAR. For the tutorial design, timing constraints were defined earlier in this tutorial, and, as a result, the Report Paths in Timing Constraints option is selected. When Implement Design is run with the Report Paths in Timing Constraints option selected, a Logic Level Timing Report is generated with a period and path analysis for each constraint specified.

To view the Logic Level Timing Report and review the PERIOD constraints that were entered earlier:

1. In the Processes for Current Source, expand the Map hierarchy.
2. Double-click **Generate Post-Map Static Timing**.

3. To open the Post-Map Static Timing Report, double-click **Post-Map Static Timing Report**. Timing Analyzer automatically launches and shows the report.

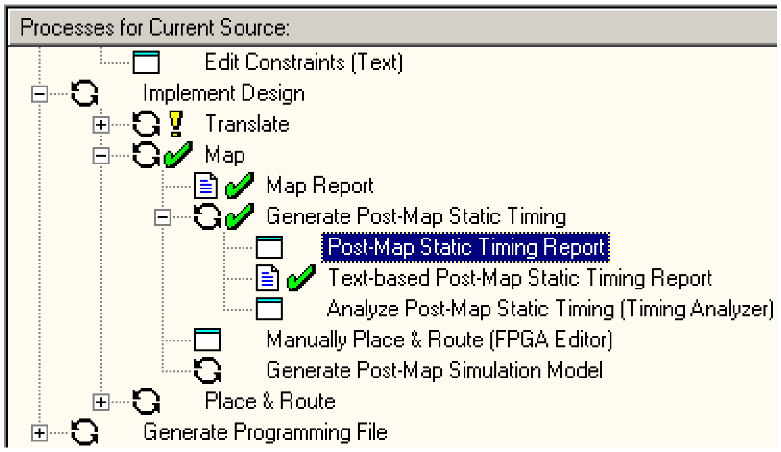

Processes for Current Source:
Edit Constraints (Text)
Implement Design
Translate
Map
Map Report
Generate Post-Map Static Timing
Post-Map Static Timing Report
Text-based Post-Map Static Timing Report
Analyze Post-Map Static Timing (Timing Analyzer)
Manually Place & Route (FPGA Editor)
Generate Post-Map Simulation Model
Place & Route
Generate Programming File

Figure 5-14: **Post-Map Static Timing Report**

At the top of this report, you will find the period timing constraint and the minimum period obtained by the tools after mapping. The report contains only one path per timing constraint, and as a result, you can see a breakdown of a single path that contains four levels of logic. Notice the percentage of block / logic delay versus routing delay is calculated in the report. The delay for the unplaced nets / blocks that are listed, are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

4. Exit Timing Analyzer.

Even if you do not generate a Logical Level Timing Report, PAR still processes a design based on the relationship between the block delays and timing specifications for the design. For example, if a PERIOD constraint of 8 ns is specified for a path, and there are block delays of 7 ns and unplaced net delays of 3 ns, PAR stops and generates an error message. In this example, PAR fails because it determines that the total delay (10 ns) is greater than the constraint placed on the design (8 ns). Therefore, we recommend using the Logic Level Timing Report to determine timing violations that may occur prior to running PAR.

Placing and Routing the Design

The design can be placed and routed after the mapped design is evaluated. Evaluation verifies that block delays are reasonable given the design specifications.

The Flow Engine performs the following place and route algorithms:

- **Timing Driven**

  Run PAR with timing constraints specified from within the input netlist or from a constraints file.

- **Non-Timing Driven**

  Run PAR and ignore all timing constraints.

Timing constraints were defined earlier in the tutorial design, and, as a result, the Place and Route (PAR) process performs timing driven placement and timing driven routing.

To run PAR in the Design Implement hierarchy, double-click **Place & Route**.

To review the reports generated to ensure that the place and route process finished as expected:

1. Expand the Place & Route hierarchy.
2. Double-click **Place & Route Report**.

Follow this by displaying and examining the **Pad Report** and **Asynchronous Delay Report**.

Table 5-3: **Reports Generated by PAR**

Report Type	Description
Place & Route Report	Provides a device utilization and delay summary. Use this report to verify that the design successfully routed and that all timing constraints were met.
Pad Report	Contains a report of the location of the device pins. Use this report to verify that pins locked down were placed in the correct location.
Asynchronous Delay Report	Lists all nets in the design and the delays of all loads on the net.

Using FPGA Editor to Verify the Place and Route

Use the FPGA Editor to display and configure Field Programmable Gate Arrays (FPGAs).

The FPGA Editor reads and writes:

- Native Circuit Description (NCD) files
- Macro files (NMC)
- Physical Constraints Files (PCF)

Use FPGA Editor to:

- Place and route critical components before running the automatic place-and-route tools.
- Finish placement and routing if the routing program does not completely route your design.
- Add probes to your design to examine the signal states of the targeted device. Probes are used to route the value of internal nets to an IOB (Input/Output Block) for analysis during debugging of a device.
- Run the Bitgen program and download the resulting bitstream file to the targeted device.
- View and change the nets connected to the capture units of an Integrated Logic Analyzer (ILA) core in your design.

To view the actual design layout on the FPGA using FPGA Editor:

1. Launch FPGA Editor in the expanded Place & Route hierarchy by double-clicking **View/Edit Routed Design (FPGA Editor)**.

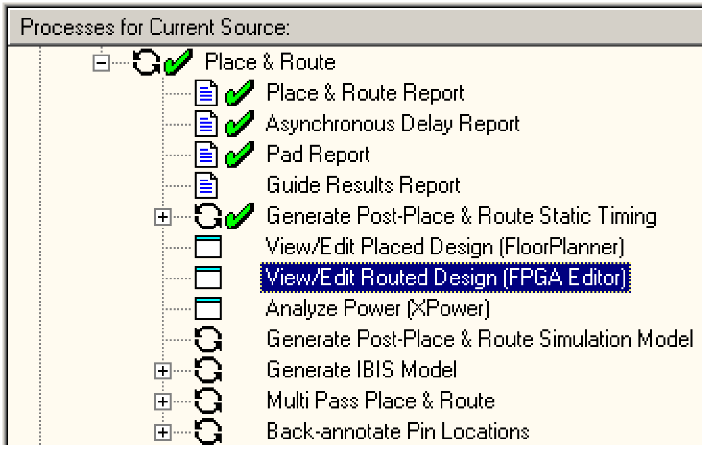

Processes for Current Source:
- Place & Route
  - Place & Route Report
  - Asynchronous Delay Report
  - Pad Report
  - Guide Results Report
  - Generate Post-Place & Route Static Timing
  - View/Edit Placed Design (FloorPlanner)
  - View/Edit Routed Design (FPGA Editor)
  - Analyze Power (XPower)
  - Generate Post-Place & Route Simulation Model
  - Generate IBIS Model
  - Multi Pass Place & Route
  - Back-annotate Pin Locations

Figure 5-15: **View/Edit Routed Design (FPGA Editor) Process**

2. In FPGA Editor, change the List Window from All Components to **All Nets**. This enables you to view all of the possible nets in the design.

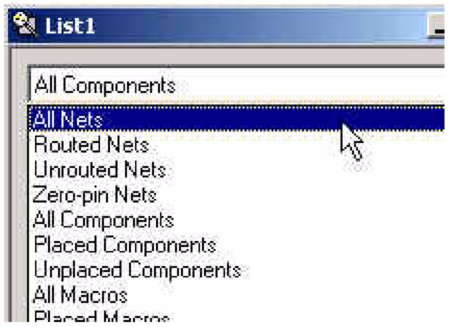


Figure 5-16: **List Window in FPGA Editor**

3. Select the **clk_dcm** (Clock) net and see the fanout of the clock net.

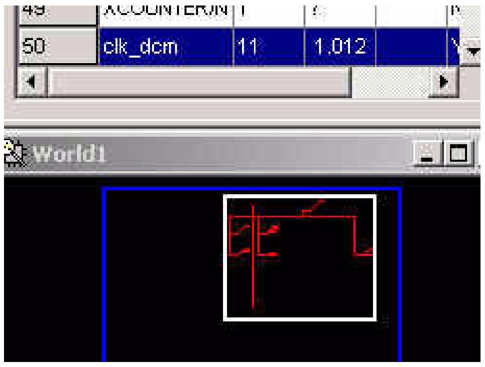


Figure 5-17: **Clock Net**

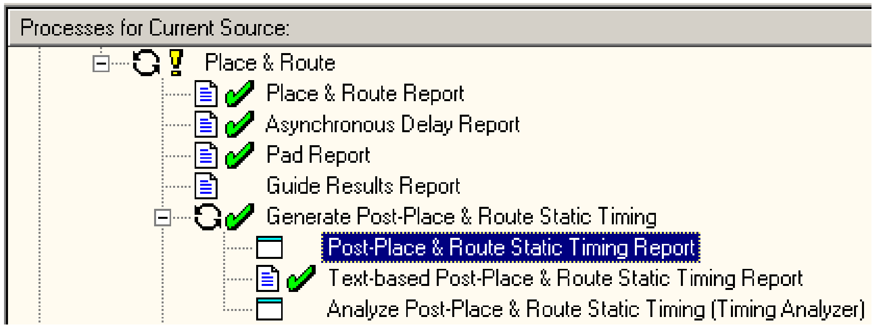
4. Exit FPGA Editor.

Evaluating Post-Layout Timing

After the design is placed and routed, the Post-Place & Route Static Timing Report, a post-layout timing report, is generated by default to verify that the design meets your specified timing goals. This report evaluates the logical block delays and the routing delays. The net delays are now reported as actual routing delays after the Place and Route process (indicated by the letter “R” next to the net delay).

To display the report:

1. Expand the Generate Post-Place & Route Timing hierarchy.
2. Double-click **Post-Place & Route Static Timing Report** to open the report in Timing Analyzer.



Processes for Current Source:

- Place & Route
  - Place & Route Report
  - Asynchronous Delay Report
  - Pad Report
  - Guide Results Report
  - Generate Post-Place & Route Static Timing
    - Post-Place & Route Static Timing Report
    - Text-based Post-Place & Route Static Timing Report
    - Analyze Post-Place & Route Static Timing (Timing Analyzer)

Figure 5-18: **Post-Place & Route Static Timing Report**

The following is a summary of the Post-Place & Route Static Timing Report.

- The minimum period value increased due to the actual routing delays.
- After the Map step, logic delay contributed to about 80% of the minimum period attained. The post-layout report indicates that the logical delay value decreased somewhat. The total unplaced floors estimate changed as well. Routing delay after PAR now equals about 31% of the period.
- The post-layout result does not necessarily follow the 50/50 rule previously described because the worst case path primarily includes component delays. After the design is mapped, block delays constitute about 80% of the period.

After place and route, the majority of the worst case path is still made up of logic delay. Since total routing delay makes up only a small percentage of the total path delay spread out across three nets, expecting the routing delay to be reduced any further is unrealistic. In general, you can reduce excessive block delays and improve design performance by decreasing the number of logic levels in the design.

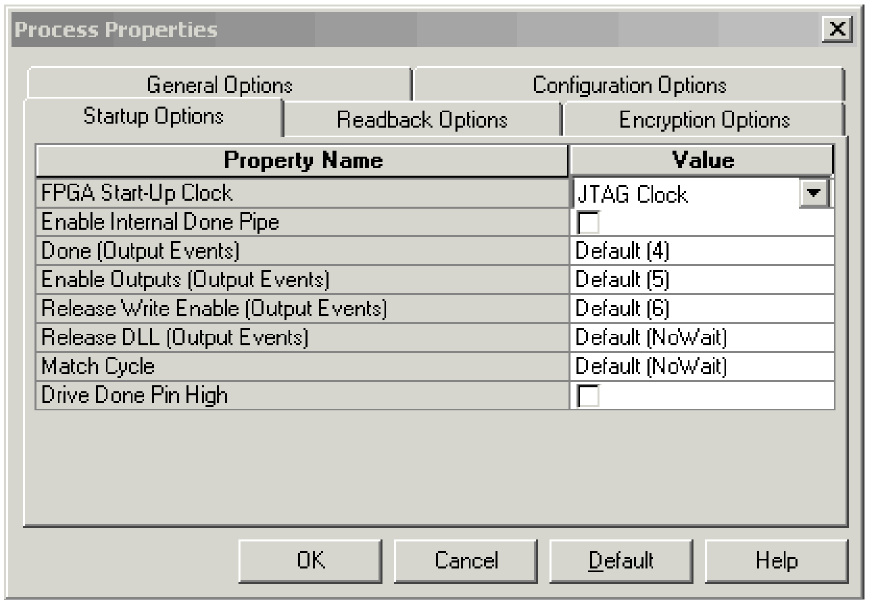
3. Exit Timing Analyzer.

Creating Configuration Data

After analyzing the design through timing constraints in Timing Analyzer, you can then create configuration data. Creating configuration data is the final step in the design flow. A configuration bitstream is created for downloading to a target device or for formatting into a PROM programming file.

In this tutorial, we are creating configuration data for a Xilinx® Serial PROM. Create a bitstream for the target device as follows:

1. Right-click on **Generate Programming File.**
2. Select **Properties**. The Process Properties dialog box opens.



Property Name	Value
FPGA Start-Up Clock	JTAG Clock
Enable Internal Done Pipe	☐
Done (Output Events)	Default (4)
Enable Outputs (Output Events)	Default (5)
Release Write Enable (Output Events)	Default (6)
Release DLL (Output Events)	Default (NoWait)
Match Cycle	Default (NoWait)
Drive Done Pin High	☐

Figure 5-19: **Process Properties' Startup Options Tab**

3. Select the **Startup Options** tab.
4. Change the FGPA Startup Clock property from CCLK to **JTAG Clock**.

   Note: You can use CCLK if you are configuring Select Map or Serial Slave.
5. Leave the remaining options in the default setting.
6. Click **OK** to apply the new properties.
7. Double-click **Generate Programming File** to create a bitstream of this design.

   The bitstream comes from the BitGen program and creates the *design_name.bit* file (in this tutorial, the *watch.bit* file). The *design_name.bit* file is the actual configuration data.
8. Verify that the file is in the project directory.
9. To review the Programming File Generation Report, double-click **Programming File Generation Report**. Verify that the specified options were used when creating the configuration data.

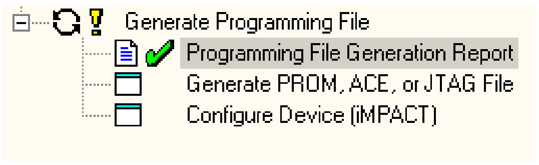

Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)

Figure 5-20: **Programming File Generation Report**

Creating a PROM File with iMPACT

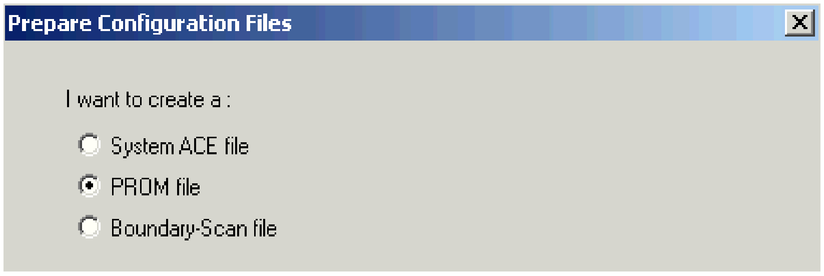
A BIT file is required to program a single device using iMPACT. A PROM file is required to program several devices in a daisy chain configuration or if you intend on using a PROM. iMPACT accepts any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations.

In iMPACT, a wizard enables you to create a PROM file and to:

- Add additional bitstreams to the daisy chain.
- Create additional daisy chains.
- Remove the current bitstream and start over, or immediately save the current PROM file configuration.

For this tutorial, launch iMPACT and create a PROM file as follows:

1. In Project Navigator, double-click **Generate PROM, ACE, or JTAG File**. A wizard opens.
2. In the Prepare Configuration Files dialog box, under "I want to create a:", select **PROM File**.
3. Click **Next**.

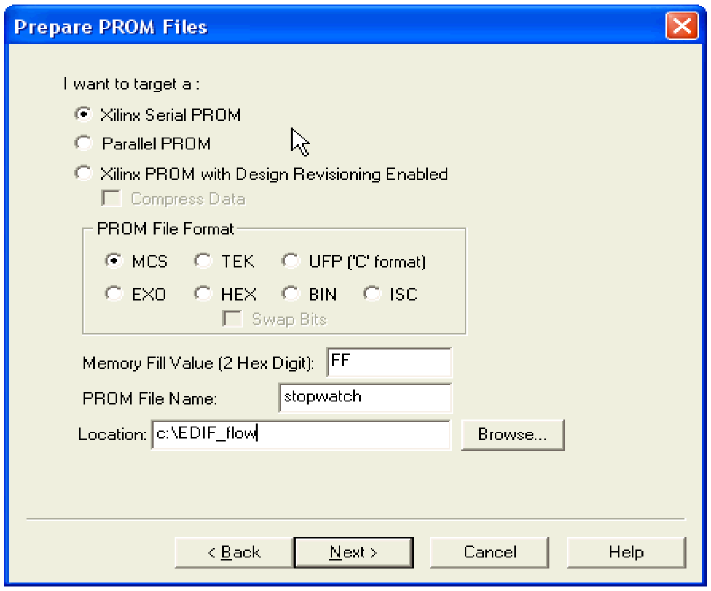

Prepare Configuration Files

I want to create a :

System ACE file [✓ PROM file] Boundary-Scan file

Figure 5-21: **Prepare Configuration Files Dialog**

4. In the Prepare PROM Files dialog, select the following:
   a. Under "I want to target a", select **Xilinx Serial PROM**.
   b. Under PROM File Format, select **MCS**.
   c. For PROM file name, type **stopwatch1**.



I want to target a : (●) Xilinx Serial PROM

PROM File Format: (●) MCS

Memory Fill Value (2 Hex Digit): FF

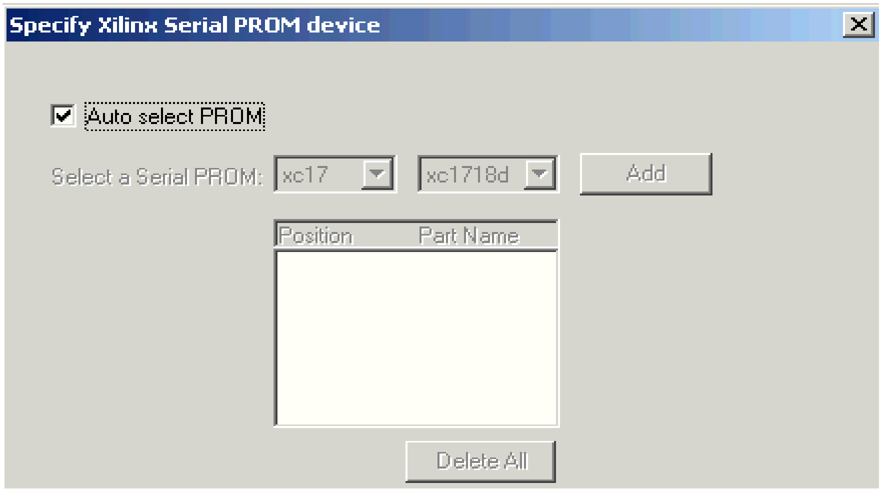
PROM File Name: stopwatch

Location: c:\EDIF_flow

Figure 5-22: **Prepare PROM Files Dialog**

5. Click **Next**.

   Note: In the Specify Xilinx Serial PROM Device dialog box, check the box associated with **Auto Select PROM.**



[✓] Auto select PROM

Figure 5-23: **Specify Xilinx Serial PROM Device Dialog Box**

6. Click **Next**.

   Note: If you have more data than space available in the PROM, you must split the data into several individual PROMs with the Split PROM option. In this case, only a single PROM is needed.

7. In the File Generation Summary dialog box, click **Next**.

8. In the Add Device File dialog box, click **Add File** and select the *stopwatch.bit* file.

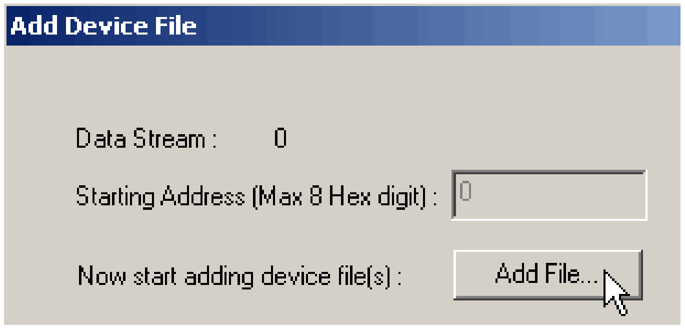

Add Device File
Data Stream : 0
Starting Address (Max 8 Hex digit) : 0
Now start adding device file(s) :
Add File...

Figure 5-24: **Add Device File Dialog Box**

9. Click **No** when you are asked if you would like to add another design file to the datastream.
10. Click **Finish**.

    iMPACT displays the PROM associated with your bit file.
11. When asked to generate a file now, click **Yes**. This creates the PROM file.
12. Select **File** → **Save** to save the supporting files associated with iMPACT.
13. Exit iMPACT.

This completes this chapter of the tutorial. For more information on this design flow and implementation methodologies, see the iMPACT Help, available from the iMPACT application by clicking **Help** → **Help Topics**.

Command Line Implementation

ISE allows you to view and extract the command line arguments for the various steps of the **Implement Design** process. By viewing the command line arguments, you can verify the options being used or create a command batch file to replicate the design flow.

Viewing The Command Line Log File

At any stage of the design flow you can look at the command line arguments for completed processes. To do so:

1. Expand the Design Entry Utilities hierarchy in the Processes for Source window
2. Double-click **View Command Line Log File**.

This process opens a file named *<source_name>.cmd_log* in read-only mode.

Creating Your Own Command Batch File

Since the Command Line Log file can be viewed in read-only mode, you can save the data to an editable batch file. Select **File** → **Save As** and enter the desired file name.

The ISE Text Editor provides many ways to manipulate the contents of a file. These include the copy and paste method (both functions are available from the **Edit** menu) or the drag and drop method, both of which allow you to copy the contents to another open text file.

Command Line Reference Information

For a complete listing of command line options for most Xilinx® executables, refer to the *Development System Reference Guide*. Command line options are organized according to implementation tools, such as Map and PAR, which you will see in the Table of Contents. This Guide is available with the collection of software manual and is accessible from ISE by selecting **Help** → **Online Documentation**, from the web at http://support.xilinx.com/support/sw_manuals/xilinx6/.

Command line options may also be obtained by typing the executable name followed by the **-h** option at a command prompt.

Another useful tool for automating design implementation is XFLOW. XFLOW is a Xilinx command line tool that automates the Xilinx implementation and simulation flows. XFLOW reads a design file as input as well as a flow file and option files. For more information on XFLOW, refer to the "XFLOW" section in the *Development System Reference Guide*.

XILINX®

Chapter 6

Timing Simulation

This chapter includes the following sections.

- "Overview of Timing Simulation Flow"
- "Getting Started"
- "Timing Simulation Using ModelSim"

Overview of Timing Simulation Flow

Timing simulation uses the block and routing delay information from a routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. For this reason, timing simulation is performed after the design has been placed and routed.

Timing (post-place and route) simulation is a recommended part of the HDL design flow for Xilinx® devices. Timing simulation uses the detailed timing and design layout information that is available after place and route. This enables simulation of the design, which closely matches the actual device operation.

Getting Started

The following sections outline the requirements to perform this part of the tutorial flow.

Required Software

In addition to Xilinx ISE 6, you must have ModelSim installed. Refer to Chapter 4, "Behavioral Simulation" for information on installing and setting up ModelSim.

Required Files

The timing simulation flow requires the following files:

- **Design Files (VHDL or Verilog)**

  This chapter assumes that you have completed Chapter 5, "Design Implementation," and thus, have a placed and routed design. A tool called NetGen will be used in this chapter to create a simulation netlist from the placed and routed design which will be used to represent the design during the Timing Simulation.

- **Test Bench File (VHDL or Verilog)**

  In order to simulate the design, a test bench is needed to provide stimulus to the design. You should use the same test bench that was used to perform the behavioral simulation. Please refer to the "Adding an HDL Test Bench" in Chapter 4 if you do not already have a test bench in your project.

- **Xilinx Simulation Libraries**

  For timing simulation, the SIMPRIM library is used to simulate the design.

To perform timing simulation of Xilinx® designs in any HDL simulator, the SIMPRIM library must be set up correctly. The timing simulation netlist created by Xilinx is composed entirely of instantiated primitives, which are modeled in the SIMPRIM library.

If you completed Chapter 4, "Behavioral Simulation", the SIMPRIM library should already be compiled. For more information on compiling and setting up the Xilinx simulation libraries, see to "Xilinx Simulation Libraries" in Chapter 4.

Timing Simulation Using ModelSim

Xilinx ISE is fully integrated with the ModelSim Simulator. ISE enables work directory creation, source file compilation, simulation initialization, and simulation property control for ModelSim.

ISE also runs NetGen to generate a simulation netlist from the placed and routed design.

Specifying Simulation Process Properties

To set the simulation process properties:

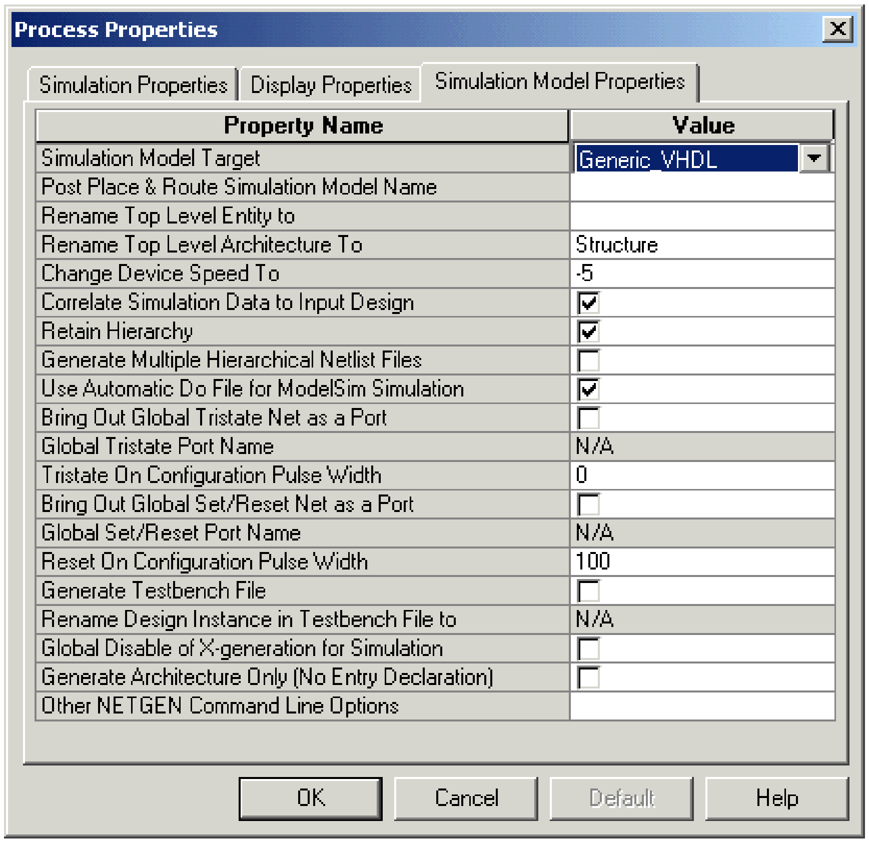
1. In the Sources in Project window, select the test bench file.
2. Click the + to expand the ModelSim Simulator hierarchy.

   Note: If the ModelSim Simulator processes do not appear, it means that either ModelSim is not installed or Project Navigator cannot find modelsim.exe. If ModelSim is installed, select **Edit** → **Preferences** in Project Navigator and select the **Integrated Tools** tab. Under Model Tech Simulator, browse to the location of *modelsim.exe*. For example, c:\modeltech_xe\win32xoem\modelsim.exe
3. Select and right-click **Simulate Post-Place & Route VHDL (Verilog) Model**.
4. Select **Properties**.

Simulation Model Properties

Click the **Simulation Model Properties** tab. The properties should appear as shown in Figure 6-1. These properties set the options that NetGen uses when generating the simulation netlist. For a description of each property, click **Help** in the Simulation Model Properties tab.

For this tutorial, the default Simulation Model Properties are used.



Property Name	Value
Simulation Model Target	Generic_VHDL
Post Place & Route Simulation Model Name	
Rename Top Level Entity to	
Rename Top Level Architecture To	Structure
Change Device Speed To	-5
Correlate Simulation Data to Input Design	✓
Retain Hierarchy	✓
Generate Multiple Hierarchical Netlist Files	☐
Use Automatic Do File for ModelSim Simulation	✓
Bring Out Global Tristate Net as a Port	☐
Global Tristate Port Name	N/A
Tristate On Configuration Pulse Width	0
Bring Out Global Set/Reset Net as a Port	☐
Global Set/Reset Port Name	N/A
Reset On Configuration Pulse Width	100
Generate Testbench File	☐
Rename Design Instance in Testbench File to	N/A
Global Disable of X-generation for Simulation	☐
Generate Architecture Only (No Entry Declaration)	☐
Other NETGEN Command Line Options	

Figure 6-1: **Simulation Model Properties**

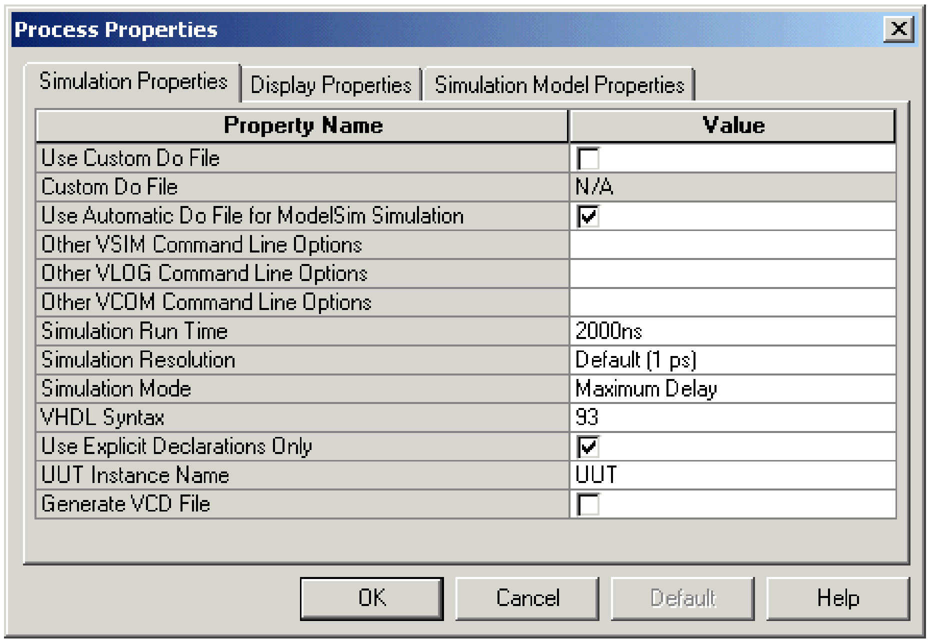
Display Properties

Click the **Display Properties** tab. This tab gives you control over the MTI (ModelSim) simulation windows. By default, three windows open when timing simulation is launched from ISE. They are the Signal window, the Structure window, and the Wave window. For more details on ModelSim Simulator windows, refer to the *ModelSim User Manual*.

Simulation Properties

Click the **Simulation Properties** tab. The properties should appear as shown in Figure 6-2. These properties set the options that ModelSim uses to run the timing simulation. For a description of each property, click **Help**.

Set the **Simulation Run Time** to **2000 ns**. The default options will be used for the other properties.



Property Name	Value
Use Custom Do File	☐
Custom Do File	N/A
Use Automatic Do File for ModelSim Simulation	☑
Other VSIM Command Line Options	
Other VLOG Command Line Options	
Other VCOM Command Line Options	
Simulation Run Time	2000ns
Simulation Resolution	Default (1 ps)
Simulation Mode	Maximum Delay
VHDL Syntax	93
Use Explicit Declarations Only	☑
UUT Instance Name	UUT
Generate VCD File	☐

Figure 6-2: **Simulation Properties**

Click **OK** to close the Process Properties dialog box.

Performing Simulation

To start the timing simulation, double-click **Simulate Post-Place and Route VHDL Model** or **Simulate Post-Place and Route Verilog Model** in the Processes for Current Source window.

ISE will run NetGen to create the timing simulation model. ISE will then call ModelSim and create the working directory, compile the source files, load the design, and run the simulation for the time specified.