

# ISE 5 In-Depth Tutorial





"Xilinx" and the Xilinx logo shown above are registered trademarks of Xilinx, Inc. Any rights not expressly granted herein are reserved. CoolRunner, RocketChips, Rocket IP, Spartan, StateBENCH, StateCAD, Virtex, XACT, XC2064, XC3090, XC4005, and XC5210 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

ACE Controller, ACE Flash, A.K.A. Speed, Alliance Series, AllianceCORE, Bencher, ChipScope, Configurable Logic Cell, CORE Generator, CoreLINX, Dual Block, EZTag, Fast CLK, Fast CONNECT, Fast FLASH, FastMap, Fast Zero Power, Foundation, Gigabit Speeds...and Beyond!, HardWire, HDL Bencher, IRL, J Drive, JBits, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, NanoBlaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, Real-PCI, Rocket I/O, SelectI/O, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, SMARTswitch, System ACE, Testbench In A Minute, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex-II Pro, Virtex-II EasyPath, Wave Table, WebFITTER, WebPACK, WebPOWERED, XABEL, XACT-Floorplanner, XACT-Performance, XACTstep Advanced, XACTstep Foundry, XAM, XAPP, X-BLOX +, XC designated products, XChecker, XDM, XEPLD, Xilinx Foundation Series, Xilinx XDTV, Xinfo, XSI, XtremeDSP and ZERO+ are trademarks of Xilinx, Inc.

The Programmable Logic Company is a service mark of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx provides any design, code, or information shown or described herein "as is." By providing the design, code, or information as one possible implementation of a feature, application, or standard, Xilinx makes no representation that such implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of any such implementation, including but not limited to any warranties or representations that the implementation is free from claims of infringement, as well as any implied warranties of merchantability or fitness for a particular purpose. Xilinx, Inc. devices and products are protected under U.S. Patents. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

The contents of this manual are owned and copyrighted by Xilinx. Copyright 1994-2002 Xilinx, Inc. All Rights Reserved. Except as stated herein, none of the material may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of any material contained in this manual may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.



# About This Tutorial

---

## About the In-Depth Tutorial

This tutorial gives a description of the features and additions to Xilinx ISE 5. The primary focus of this tutorial is to show the relationship among the design entry tools, Xilinx and third-party tools, and the design implementation 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
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging <a href="http://support.xilinx.com/support/techsup/tutorials/index.htm">http://support.xilinx.com/support/techsup/tutorials/index.htm</a>
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at <a href="http://support.xilinx.com/support/searchtd.htm">http://support.xilinx.com/support/searchtd.htm</a>
Application Notes	Descriptions of device-specific design techniques and approaches <a href="http://support.xilinx.com/apps/appsweb.htm">http://support.xilinx.com/apps/appsweb.htm</a>
Forums	Discussion groups and chat rooms for Xilinx software users <a href="http://toolbox.xilinx.com/cgi-bin/forum">http://toolbox.xilinx.com/cgi-bin/forum</a>
Data Book	Pages from <i>The Programmable Logic Data Book</i> , which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging <a href="http://support.xilinx.com/partinfo/databook.htm">http://support.xilinx.com/partinfo/databook.htm</a>

Resource	Description/URL
Xcell Journals	Quarterly journals for Xilinx programmable logic users <a href="http://support.xilinx.com/xcell/xcell.htm">http://support.xilinx.com/xcell/xcell.htm</a>
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment <a href="http://support.xilinx.com/xlnx/xil_tt_home.jsp">http://support.xilinx.com/xlnx/xil_tt_home.jsp</a>

## 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, CoreGen, and HDL Editor.
- **Chapter 4, “Behavioral Simulation,”** explains how to use the ModelSim Simulator to simulate a design before design implementation 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 block and routing delay information from the routed design 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

---

## Preface: About This Tutorial

About the In-Depth Tutorial .....	5
Additional Resources .....	5
Tutorial Contents .....	6
Tutorial Flows .....	6
HDL Design Flow .....	6
Schematic Design Flow .....	7
Implementation-only Flow .....	7

## Chapter 1: Overview of ISE and Synthesis Tools

Overview of ISE .....	13
Project Navigator Interface .....	13
Sources in Project Window .....	14
Module View .....	14
Snapshot View .....	15
Library View .....	15
Processes for Current Source Window .....	15
Process View .....	15
Console Window .....	16
Error Navigation to Source .....	16
Error Navigation to Solution Record .....	16
Using Snapshots .....	16
Creating a Snapshot .....	16
Restoring a Snapshot .....	16
Viewing a Snapshot .....	17
Using Project Archives .....	17
Creating an Archive .....	17
Restoring an Archive .....	17
Overview of Synthesis Tools .....	17
Xilinx Synthesis Technology (XST) .....	17
Supported Devices .....	17
Process Properties .....	18
Synplify/Synplify Pro .....	18
Supported Devices .....	18
Process Properties .....	18
LeonardoSpectrum .....	18
Supported Devices .....	18
Process Properties .....	18

## Chapter 2: HDL-Based Design

Overview of HDL-Based Design .....	19
Getting Started .....	20
Required Software .....	20
Optional Software Requirements .....	20
VHDL or Verilog? .....	20

Installing the Tutorial Project Files . . . . .	20
Copying the Tutorial Files (Optional) . . . . .	21
Starting the ISE Software . . . . .	21
Stopping the Tutorial . . . . .	22
<b>Design Description . . . . .</b>	<b>22</b>
Inputs . . . . .	23
Outputs . . . . .	23
Functional Blocks . . . . .	23
<b>Design Entry . . . . .</b>	<b>24</b>
Adding Source Files . . . . .	24
Analyzing the Source Files . . . . .	25
Correcting HDL errors . . . . .	25
Creating an HDL-Based Module . . . . .	25
Using the New Source Wizard and HDL Editor . . . . .	26
Using the Language Templates . . . . .	27
Adding the Language Template to Your File . . . . .	28
Creating a CORE Generator Module . . . . .	29
Creating the CORE Generator Module . . . . .	29
Instantiating the CoreGen Module in the HDL Code . . . . .	31
Creating a DCM Module . . . . .	34
Using DCM Wizard . . . . .	34
Instantiating the DCM1 Macro - VHDL Design . . . . .	35
Instantiating the DCM1 Macro - Verilog . . . . .	36
<b>Synthesizing the Design . . . . .</b>	<b>36</b>
Synthesizing the Design using XST . . . . .	37
Entering Constraints . . . . .	37
Entering Synthesis Options . . . . .	38
Synthesizing the Design . . . . .	39
The RTL Viewer . . . . .	39
Synthesizing the Design using Synplify/Synplify Pro . . . . .	40
Examining Synthesis Results . . . . .	41
Synthesizing the Design using LeonardoSpectrum . . . . .	43
Modifying Constraints . . . . .	43
Entering Synthesis Options through ISE . . . . .	45
The RTL/Technology Viewer . . . . .	46

## Chapter 3: Schematic-Based Design

<b>Overview of Schematic-based Design . . . . .</b>	<b>49</b>
<b>Getting Started . . . . .</b>	<b>49</b>
Required Software . . . . .	49
Installing the Tutorial Project Files . . . . .	50
wtut_sc project . . . . .	50
watch_sc solution project . . . . .	50
Copying the Tutorial Files (Optional) . . . . .	50
Starting the ISE Software . . . . .	50
Stopping the Tutorial . . . . .	51
<b>Design Description . . . . .</b>	<b>51</b>
Inputs . . . . .	52
Outputs . . . . .	53
Functional Blocks . . . . .	53

<b>Design Entry</b> .....	54
Opening the Project File in the ECS Schematic Editor Tool .....	54
Manipulating the Window View .....	54
Creating a Schematic-Based Macro .....	55
Defining the CNT60 Schematic .....	55
Adding Components to CNT60 .....	56
Placing the Remaining Components .....	57
Correcting Mistakes .....	58
Drawing Wires .....	58
Adding Buses .....	58
Adding Bus Taps .....	60
Adding Net Names .....	61
Adding I/O Markers .....	61
Saving the Schematic .....	61
Creating the CNT60 symbol .....	62
Placing the CNT60 Macro .....	62
Creating a CORE Generator Module .....	63
Creating the CORE Generator Module .....	63
Creating a State Machine Module .....	65
Opening the State Editor .....	66
Adding New States .....	67
Adding a Transition .....	68
Adding a State Action .....	68
Adding a State Machine Reset Condition .....	70
Creating the State Machine Macro .....	71
Creating a DCM Module .....	72
Using DCM Wizard .....	72
Creating the DCM1 macro .....	73
Placing the STMACH, Tenths, DCM1, outs3, and decode symbols .....	73
Creating an HDL-Based Module .....	74
Using the New Source Wizard and HDL Editor .....	74
Using the Language Templates .....	76
Adding the Language Template to Your File .....	77
Creating the HEX2LED Symbol .....	78
Adding the HEX2LED Component to the Schematic .....	78
Specifying Device Inputs/Outputs .....	79
Hierarchy Push/Pop .....	79
Adding Input Pins .....	80
Adding I/O Markers and Net Names .....	81
Assigning Pin Locations .....	82
Completing the Schematic .....	83

## Chapter 4: Behavioral Simulation

<b>Overview of Behavioral Simulation Flow</b> .....	87
<b>Getting Started</b> .....	88
Required Files .....	88
Xilinx Simulation Libraries .....	88
Unisims Library .....	88
XilinxCoreLib Library .....	89
Viewing the Modelsim.ini File .....	89
Setting the Environment Variable .....	89

<b>Adding an HDL Testbench</b> .....	89
VHDL Design .....	90
Verilog Design .....	90
<b>Creating a Testbench Waveform Using HDL Bencher</b> .....	90
Creating a Testbench Waveform Source .....	90
Initializing Inputs .....	92
<b>Behavioral Simulation Using ModelSim</b> .....	93
Selecting Simulation Processes .....	93
Specifying Simulation Properties .....	93
Performing Simulation .....	95
Adding Signals .....	95
Saving the Simulation .....	96

## Chapter 5: Design Implementation

<b>Overview of Design Implementation</b> .....	97
<b>Getting Started</b> .....	98
Tutorial Option 1 .....	98
Tutorial Option 2 .....	98
<b>Creating an Implementation Project</b> .....	98
<b>Specifying Options</b> .....	100
<b>Translating the Design</b> .....	103
<b>Using the Constraints Editor</b> .....	103
<b>Using the Pin-out Area Constraints Editor (PACE)</b> .....	106
<b>Mapping the Design</b> .....	109
<b>Using Timing Analysis to Evaluate Block Delays After Mapping</b> .....	111
Estimating Timing Goals with the 50/50 Rule .....	111
Report Paths in Timing Constraints Option .....	111
<b>Placing and Routing the Design</b> .....	112
<b>Using FPGA Editor to Verify the Place and Route</b> .....	113
<b>Evaluating Post-Layout Timing</b> .....	115
<b>Creating Configuration Data</b> .....	116
<b>Creating a PROM File with iMPACT</b> .....	118

## Chapter 6: Timing Simulation

<b>Overview of Timing Simulation Flow</b> .....	123
<b>Getting Started</b> .....	123
Required Software .....	123
Required Files .....	123
Xilinx Simulation Libraries .....	124
<b>Starting ModelSim</b> .....	124
Specifying Simulation Process Properties .....	124
Simulation Properties .....	125
Display Properties .....	126
Simulation Model Properties .....	126
Performing Simulation .....	127
<b>Adding Signals</b> .....	128
<b>Saving the Simulation</b> .....	129

## 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, the user must 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 is updated during all project actions. The fourth window to the right is the HDL Editor window. HDL Editor enables you to edit source files and to access the Language Templates, which is a catalog of ABEL, Verilog and VHDL language templates that you can use and modify for your own design. These windows are discussed in more detail in the following sections.

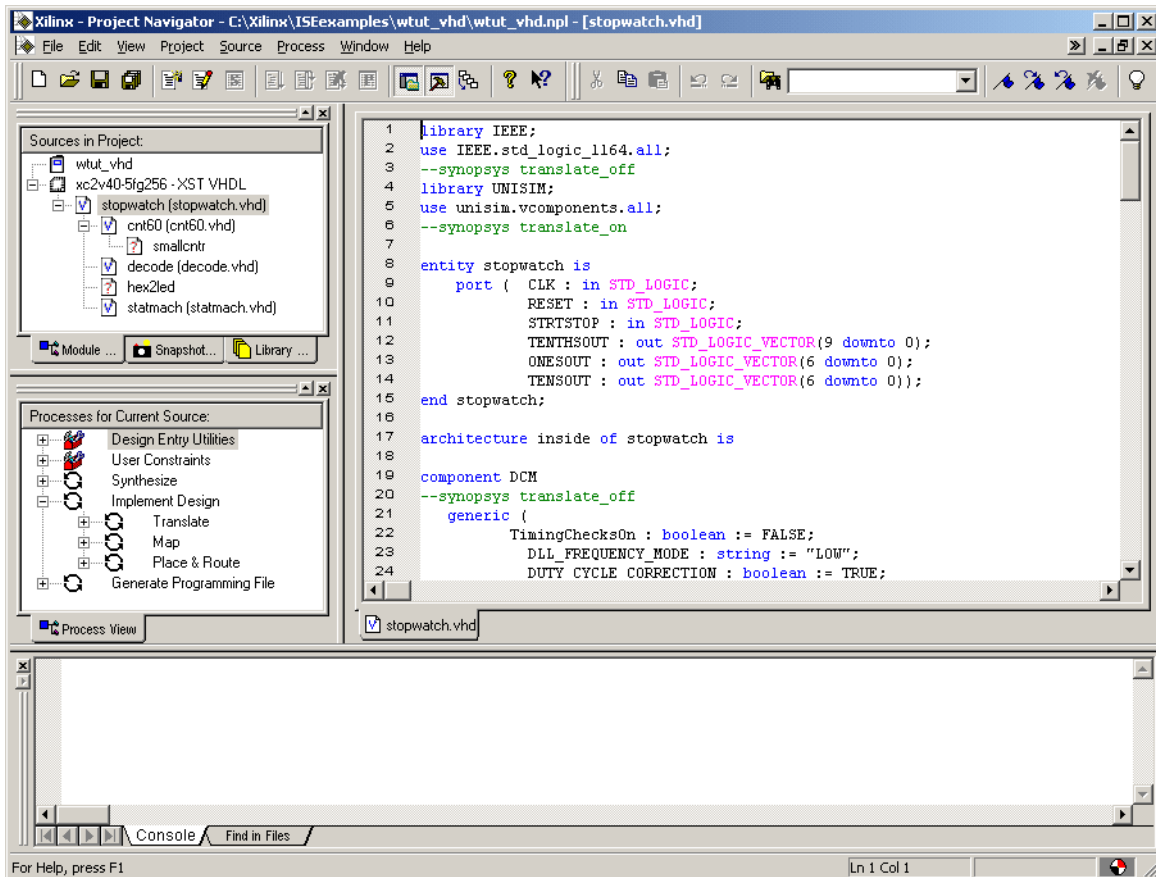


Figure 1-1: Project Navigator

## 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 more information about the file icons, see the Project Navigator online help. Select **Help** → **Project Navigator**, expand the About Projects and Sources section, 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 sub-directories. A snapshot is stored with the project for which it 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 and instantiation templates.
- **User Constraints**  
Provides access to editing location and timing and constraints.
- **Synthesis**  
Provides access to check syntax, synthesis, 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.

## Console Window

The Console window displays errors, warnings, and informational messages. Errors and warnings are signified by a red box next to the message, while warnings have a yellow box.

### 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.

## 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 displays.

### 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 open a snapshot to review a report or verify process status:

1. Select the snapshot.
2. Right-click the mouse.
3. From the menu, select **Open**.

## 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 /-II Pro
- Spartan™-II /-IIE
- XC9500™ /XL/XV
- Coolrunner™ /-II

## Process Properties

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

More detailed information is available in the *XST User Guide*, available in the collection of software manuals on the web, at

[http://support.xilinx.com/support/sw\\_manuals/xilinx5/](http://support.xilinx.com/support/sw_manuals/xilinx5/).

## 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 /-II Pro
- Spartan™ -II/-IIE
- XC9500™ /XL /XV
- Coolrunner™ /-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.

More detailed information about the specific synthesis options is available in 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. Through 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- and Schematic-based design flow.

### Supported Devices

- Virtex™/-E /-II /-IIPro
- Spartan™ -II/-IIE
- XC9500™ /XL /XV
- Coolrunner™ /-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.

# HDL-Based Design

---

This chapter includes the following sections:

- “Overview of HDL-Based Design”
- “Getting Started”
- “Design Description”
- “Design Entry”
- “Synthesizing the Design”

## 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. The simulation, implementation, and bitstream generation are described in subsequent chapters.

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 5.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 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` for your installation path.

### Optional Software Requirements

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

- Synplify/Synplify PRO 7.x
- LeonardoSpectrum 2002.1b (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.

### Installing the Tutorial Project Files

The Stopwatch tutorial projects can be downloaded from <http://support.xilinx.com/support/techsup/tutorials/tutorials5.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. The files downloaded from the web have the most recent updates.

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\ISEexamples`, and the tutorial files are copied into the directories.

These directories contain complete and incomplete versions of the design, done in VHDL and Verilog, respectively. The incomplete projects are used in this tutorial to step through the ISE flow. The completed projects are provided for reference.

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

*Table 2-1: Tutorial Project Directories*

Directory	Description
<i>wtut_vhd</i>	Incomplete Watch Tutorial - VHDL
<i>wtut_ver</i>	Incomplete Watch Tutorial - Verilog
<i>watchvhd_u</i>	Solution for Watch - VHDL (UNIX)
<i>watchver_u</i>	Solution for Watch - Verilog (UNIX)
<i>watchvhd</i>	Solution for Watch - VHDL
<i>watchver</i>	Solution for Watch - Verilog

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

The *watchvhd(\_u)* and *watchver(\_u)* 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.

## Copying the Tutorial Files (Optional)

You can either work within the project directory as it has been downloaded, or you can make a copy to work on. To make a working copy of the tutorial files, use Windows Explorer to copy the *wtut\_ver* or *wtut\_vhd* directory to another location. The project directory contains all of the necessary project files to follow the tutorial.

## 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 2-1: Project Navigator Desktop Icon*

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

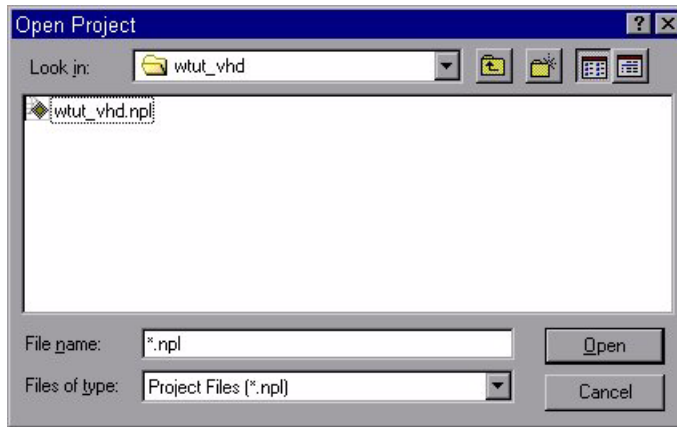


Figure 2-2: Getting Started Dialog Box

- In the Directories list, browse to `c:\xilinx\ISEexamples\wtut_vhd` or `c:\xilinx\ISEexamples\wtut_ver`.
- 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 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 Watch design is a simple runner's stopwatch. Throughout this tutorial, the design you'll work with is referred to as Watch. 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.

## Inputs

The following are input signals for the tutorial stopwatch 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 2 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**  
DCM 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.

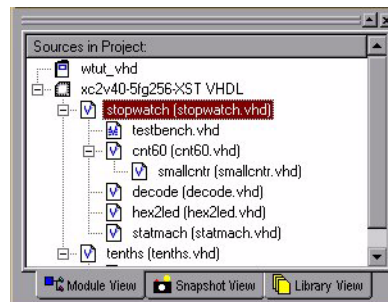


Figure 2-3: Sources in Project Window

## 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 process 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 **HDL Module**.
5. Click **OK**.



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

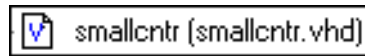


Figure 2-4: **smallcntr.vhd** file in **Source in Project** window

## Analyzing the Source Files

After adding the file to the project, the file is not automatically analyzed. To analyze the 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 **Analyze Hierarchy** in the Synthesize hierarchy to update the files.

## Correcting HDL errors

The SMALLCNTR design contains a syntax error that must be corrected. The red “x” beside the Analyze Hierarchy process indicates an error was found during analysis. The Project Navigator reports errors in red and warnings in yellow in the console.

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 **Analyze Hierarchy** 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 HDL 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 HDL Editor

In order to create the module, first create a file using the New Source Wizard specifying the name and ports of the component. The resulting “skeleton” HDL file is then modified further in the HDL 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](#).

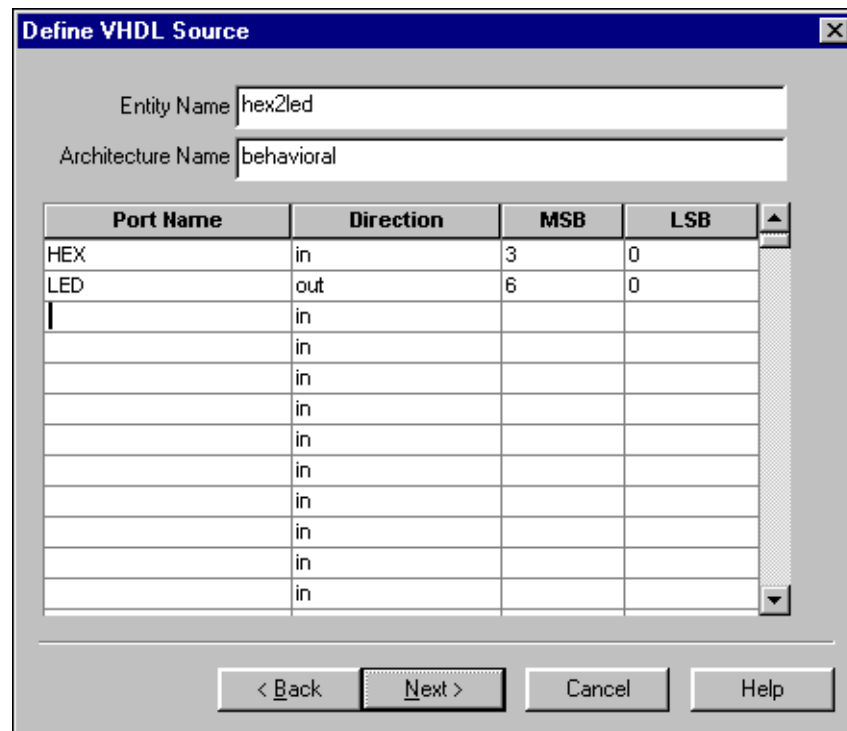


Figure 2-5: New Source Wizard

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 HDL Editor.

The VHDL file is found in [Figure 2-6](#). The Verilog HDL file is found in [Figure 2-7](#).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  -- Uncomment the following lines to use the declarations that are
7  -- provided for instantiating Xilinx primitive components.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11  entity hex2led is
12      Port ( HEX : in std_logic_vector(3 downto 0);
13            LED : in std_logic_vector(6 downto 0));
14  end hex2led;
15
16  architecture Behavioral of hex2led is
17
18  begin
19
20
21  end Behavioral;
22
```

Figure 2-6: Skeleton VHDL File in HDL Editor

```
1  module HEX2LED(HEX,LED);
2      input [3:0] HEX;
3      input [6:0] LED;
4
5
6
7  endmodule
8
```

Figure 2-7: Skeleton Verilog File in HDL Editor

In the HDL Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are printed in blue, data types in red, comments in green, and values are 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 template for components or constructs you use often.

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

1. 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 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.

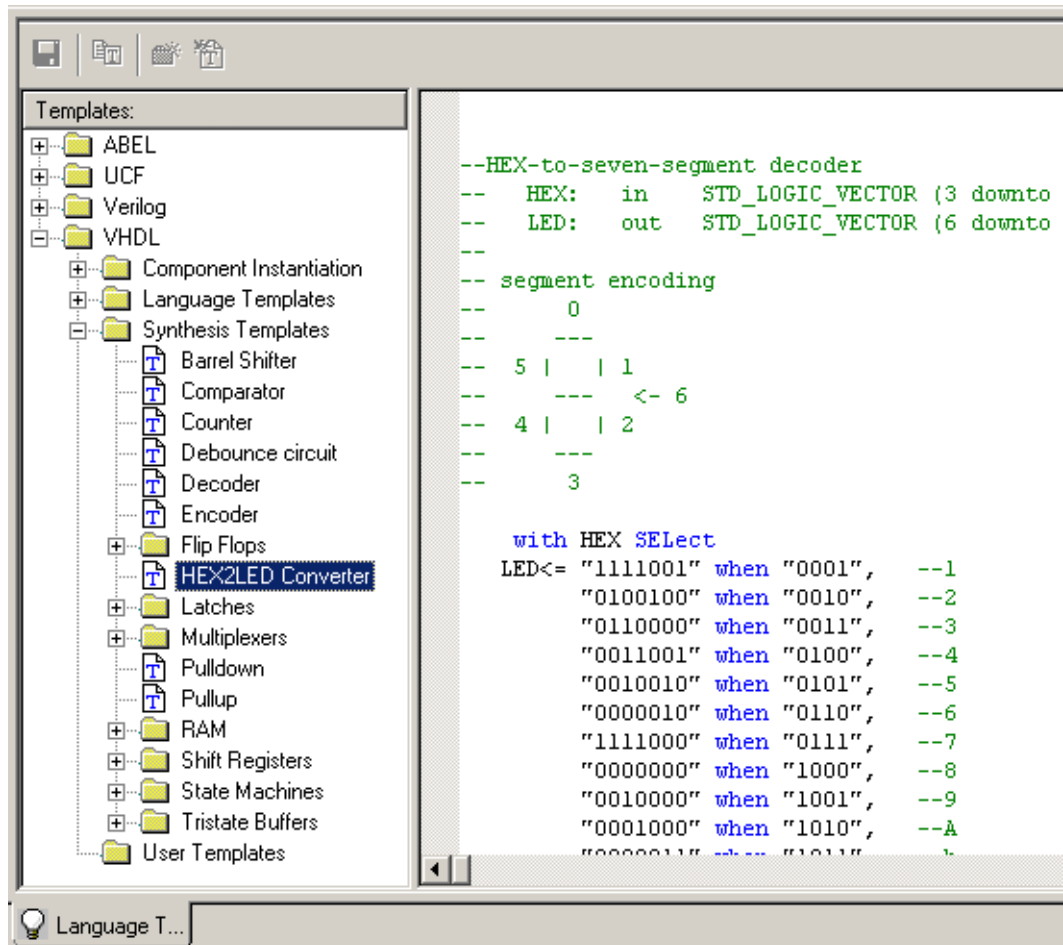


Figure 2-8: Language Templates

## 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 Template 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 Template, 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 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 HDL 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 **Coregen IP** as the source type.
3. Enter 'tenths' in the File Name field.
4. Click **Next** and then **Finish**.

The Xilinx CORE Generator opens and displays a list of possible COREs available.

5. Double-click on **Basic Elements - Counters**.
6. Double-click on **Binary Counter** 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 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.

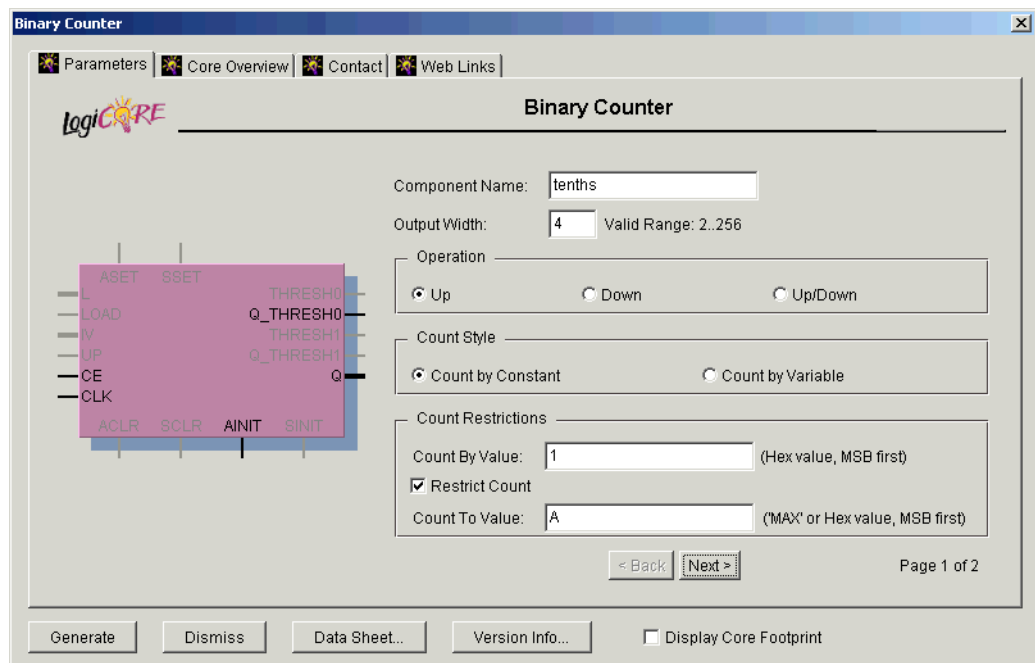


Figure 2-9: CoreGen Module Selector

8. Select the **Next** button.
9. 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**
10. Click the **Register Options** button to open the Register Options dialog box.
11. In the Register Options dialog box, enter the following settings:
  - ◆ Clock Enable: **Selected**
  - ◆ Asynchronous Settings: **Init with a value of 1**
  - ◆ Synchronous Settings: **None**

12. Click **OK**.
13. 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**
14. 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 Coregen configuration for the project.

15. Click **Cancel** and close Core Generator.

## Instantiating the CoreGen Module in the HDL Code

Next, instantiate the Coregen Module in the HDL code using either a VHDL flow or a Verilog flow.

### VHDL Flow

To instantiate the Coregen Module using a VHDL flow:

1. In Project Navigator, double-click *stopwatch.vhd* to open the file in HDL 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 Coregen instantiation is inserted.

**Note:** The Component Declaration does not need to be modified.

```

67 component statmach
68     port ( CLK : in STD_LOGIC;
69           RESET : in STD_LOGIC;
70           STRTSTOP : in STD_LOGIC;
71           CLKEN : out STD_LOGIC;
72           RST : out STD_LOGIC);
73 end component;
74
75 -- Insert Coregen Counter Component Declaration.
76 ----- Begin Cut here for COMPONENT Declaration ----- COMP_TAG
77 component tenths
78     port (
79         Q: OUT std_logic_VECTOR(3 downto 0);
80         CLK: IN std_logic;
81         Q_THRESH0: OUT std_logic;
82         CE: IN std_logic;
83         AINIT: IN std_logic);
84 end component;
85
86 -- XST black box declaration
87 attribute box_type : string;
88 attribute box_type of tenths: component is "black_box";
89
90 -- FPGA Express Black Box declaration
91 attribute fpga_dont_touch: string;
92 attribute fpga_dont_touch of tenths: component is "true";
93
94 -- Synplicity black box declaration
95 attribute syn_black_box : boolean;
96 attribute syn_black_box of tenths: component is true;
97
98 -- COMP_TAG_END ----- End COMPONENT Declaration -----
99

```

Figure 2-10: VHDL Component Declaration of Coregen 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 Coregen module as shown in Figure 2-11.

```

164 MACHINE:statmach port map(CLK=>clk_dcm,
165                             RESET=>RESET,
166                             STRTSTOP=>strtstopinv,
167                             CLKEN=>clkenable,
168                             RST=>rstint);
169
170 -- Insert Coregen Counter Instantiation.
171 xcounter : tenths
172     port map (
173         Q => Q,
174         CLK => CLK_dcm,
175         Q_THRESHO => xtermcnt,
176         CE => clkenable,
177         AINIT => rstint);
178
179
180
181 decoder: decode port map (
182     binary => Q,
183     one_hot => xcountout);
184

```

Figure 2-11: VHDL Component Instantiation of Coregen Module

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

### Verilog Flow

To instantiate the Coregen Module using a Verilog flow:

1. In Project Navigator, double-click *stopwatch.v* to open the file in HDL Editor.
2. Select **File** → **Open** and open the *tenths.veo* file.
3. Highlight the inserted code in *tenths.veo* from  

```

    "Tenths YourInstanceName"
to
    "AINIT=(AINIT) );"

```
4. Select **Edit** → **Copy**.
5. Place the cursor after the line in *stopwatch.v* that states:  

```

    "// Place the Coregen Component Instantiation for Tenths here."

```
6. Select **Edit** → **Paste** to place the instantiation here.
7. Change "YourInstanceName" to XCOUNTER.

8. Edit this code to connect the signals in the Stopwatch design to the ports of the Coregen module as shown in Figure 2-12.

```

31 statmach MACHINE(.CLK(clk_dcm),
32     .RESET(RESET),
33     .STRTSTOP(strtstopinv),
34     .CLKEN(clkenable),
35     .RST(rstint));
36
37 //Place the CORE Generator Component Instantiation for Tenths here
38 //----- Begin Cut here for INSTANTIATION Template ----// INST_TAG
39 tenths xcounter (
40     .Q(Q),
41     .CLK(clk_dcm),
42     .Q_THRESHO(xtermcnt),
43     .CE(clkenable),
44     .AINIT(rstint));
45
46 // INST_TAG_END ----- End INSTANTIATION Template -----
47
48 decode one_decode (.BINARY(Q), .ONE_HOT(xcountout));
49
50 cnt60 sixty(.CE(cnt60enable),
51     .CLK(clk_dcm),
52     .CLR(rstint),
53     .LSBSEC(lsbcnt),
54     .MSBSEC(msbcnt));

```

Figure 2-12: Verilog Component Instantiation of the CoreGen Module

9. Save the design using **File** → **Save** and close *stopwatch.v* in HDL Editor.

## Creating a DCM Module

The DCM Wizard, one part of the Xilinx Architecture Wizard, enables a user 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 **Architecture Wizard** and type 'DCM1' for the File Name.
3. Click **Next**, then **Finish**.  
The Xilinx Architecture Wizard is launched.
4. In the Xilinx Architecture Wizard selection box, select **DCM Wizard** and click **OK**.  
The DCM Wizard is launched.
5. Deselect RST and LOCKED.
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**
8. Select the **Advanced** button.
9. Change Wait for DCM lock before DONE signal goes high to **Yes**.
10. Select **OK** and **Next**.
 

An informational message displays the locked signal and the STARTUP\_WAIT option.
11. Select **OK** and **Finish**.

*DCM1.xaw* is added to the list of project source files in the Sources in Project window.

**Note:** The newly created *DCM1\_arwz.ucf* does not need to be added to the projects as all of the constraints are passed into the relevant source file(s).

## 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 on **View HDL Instantiation Template** under the Design Entry Utilities in the Processes for Current Source window.
3. From the newly opened HDL Instantiation Template copy the component declaration template:

```
COMPONENT DCM1
PORT (
  clk_in_in : IN std_logic;
  clk0_out : OUT std_logic
);
END COMPONENT;
```

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

```
Inst_DCM1: DCM1 PORT MAP (
  clk_in_in => ,
  clk0_out =>
);
```
6. Paste the instantiation template into the section in *stopwatch.vhd* labeled -- Insert DCM1 instantiation here.
7. Connect the DCM1 port *clk\_in\_in* to signal *clk*.
8. Connect the DCM1 port *clk0\_out* to signal *clk\_dcm*.

9. If you encounter an error when trying to view the HDL instantiation template, the design will not synthesize properly. To resolve the potential synthesis problem:
  - a. Select **File** → **Open**, and select *DCM1.vhd*.
  - b. In HDL Editor, scroll down to the port map of the DCM.
  - c. Add `DSSSEN => GND`, after `CLKFB => CLKFB_IN`,
  - d. Select **File** → **Save**.

## 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 on **View HDL Instantiation Template** under the Design Entry Utilities in the Processes for Current Source window.
3. From the newly opened HDL Instantiation Template, copy the instantiation template:

```
DCM1 instance_name (  
    .CLKIN_IN (CLKIN_IN) ,  
    .CLK0_OUT (CLK0_OUT)  
);
```

4. Paste the instantiation template into the section in *stopwatch.v* labeled `//Insert DCM1 instantiation here.`

## Synthesizing the Design

So far, the design in the tutorial has been using XST for syntax checking and analysis. The next portion of the design is the synthesis of the HDL code that you entered into the project. The job of a synthesis tool is to take HDL code and generate 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**  
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**  
The components from the compile stage are translated into the target technology's primitive components.

The synthesis tool can be changed 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.”](#)

A summary of available synthesis tools is available in [“Overview of Synthesis Tools.”](#)

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 Design Flow column and use the pulldown arrow to select the desired synthesis tool from the list.

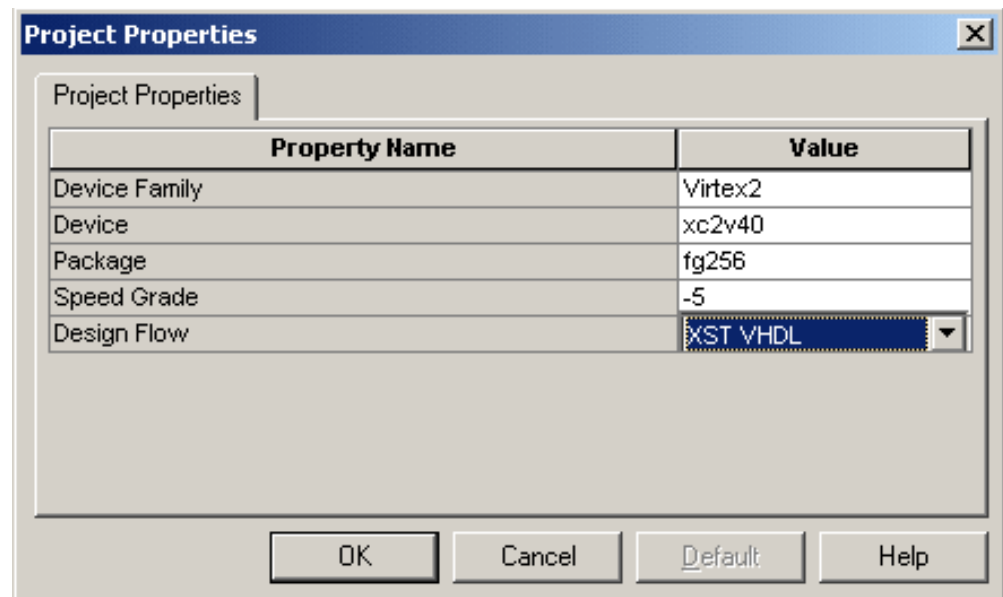


Figure 2-13: Specifying Synthesis Tool

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 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 for synthesis using XST:

- View Synthesis Report gives a mapping and timing summary as well as synthesis optimizations that took place.
- View RTL Schematic, accessible from the Launch Tools hierarchy, generates a schematic of your HDL code.
- Analyze Hierarchy will set up the HDL in its hierarchical order.
- Check Syntax verifies that the HDL code is entered properly.

## Entering Constraints

Starting in the 5.1i release, XST supports a new 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:** Xilinx supports the old constraint syntax without any further enhancements for this release of XST, but eventually support will be dropped.

This new syntax style format is called the Xilinx Constraint File (XCF). The XCF must have an extension of .xcf. 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:

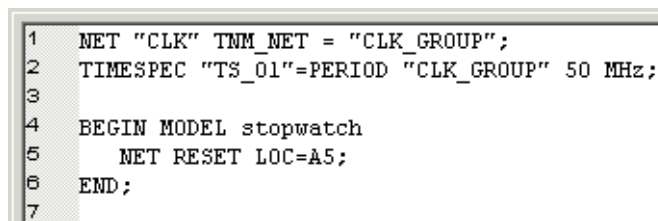
1. Select **Project** → **New Source**.
2. In the New Source dialog box, select **User Document** as the source type, and enter the file name 'stopwatch.xcf'.
3. Select **Next**, and **Finish**.

The new XCF file launches in HDL Editor.

4. In the new XCF document, type in the following:

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

5. Select **File** → **Save**.



```
1 NET "CLK" TMM_NET = "CLK_GROUP";  
2 TIMESPEC "TS_01"=PERIOD "CLK_GROUP" 50 MHz;  
3  
4 BEGIN MODEL stopwatch  
5     NET RESET LOC=A5;  
6 END;  
7
```

Figure 2-14: Contents of stopwatch.xcf

**Note:** For more constraint options in the implementation tools, see [“Using the Constraints Editor”](#) and [“Using the Pin-out 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. Highlight *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** field and select *stopwatch.xcf*.
4. Check the **Write Timing Constraints** box.
5. Select the **OK** button.

## 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, expand the Synthesize process.
2. Double-click on the **View RTL Schematic**.

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

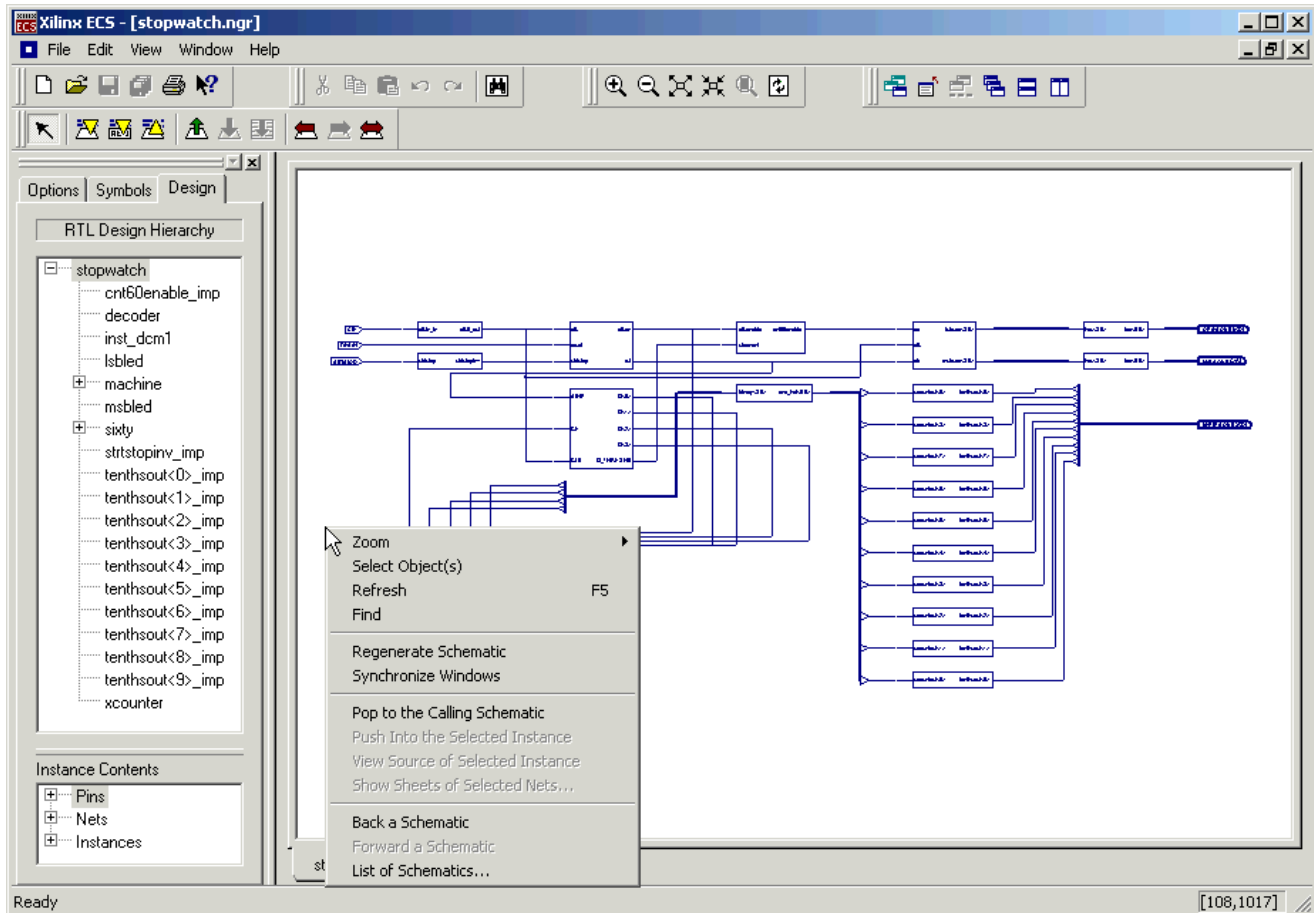


Figure 2-15: XST's RTL Viewer

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*, available in the collection of software manuals on the web, at [http://support.xilinx.com/support/sw\\_manuals/xilinx5/](http://support.xilinx.com/support/sw_manuals/xilinx5/).

## 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. Right-click **Synthesis** in the Processes for Current Source window.
3. From the menu, select **Properties**.
4. Set the Default Frequency to **50MHz**, and check the **Write Vendor Constraint File** box.
5. Click **OK** to accept these values.
6. 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 that Synplify does to the HDL design. Each of the summaries report on the files in the project, giving 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 (.xnf, .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 Tenth module was built using CORE Generator called from the project, the tenths EDN file is found.

```

115 ##### START TIMING REPORT #####
116 # Timing Report written on Mon Jul 08 12:43:47 2002
117 #
118 #
119 #
120 Top view:          stopwatch
121 Slew propagation mode: worst
122 Paths requested:   5
123 Constraint File(s): C:\xilinx\iseexamples\wtut_vhd\stopwatch.sdc
124                   C:\xilinx\iseexamples\wtut_vhd\stopwatch_fsm.sdc
125
126 @N| This timing report estimates place and route data. Please look at the place and route timing
127 @N| Clock constraints cover all FF-to-FF, FF-to-output, input-to-FF and input-to-output paths as
128
129
130
131 Performance Summary
132 *****
133
134
135 Worst slack in design: 14.200
136
137
138 Starting Clock      Requested      Estimated      Requested      Estimated      Slack      Clock
139                   Frequency      Frequency      Period         Period         Type
140 -----
141 CLK                 50.0 MHz      340.3 MHz      20.000         2.939          17.061     inferred
142 System             50.0 MHz      172.4 MHz      20.000         5.800          14.200     system
143 -----
144
145 Clock Relationships
146 *****
147
148
149 Clocks             | rise to rise | fall to fall | rise to fall | fall
150 -----
151 Starting Ending | constraint slack | constraint slack | constraint slack | constrai
152 -----
153 CLK      CLK   | 20.000   17.061 | No paths -   | No paths -   | No paths
154 -----
155 Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
156       'Diff grp' indicates that paths exist but the starting clock and ending clock are in diff
157
158

```

Figure 2-16: Synplify's Estimated Timing Data

## Timing Report

The timing report section details information on the constraints that you entered along with delays on portions 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, a 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.
2. Double-click on the **Modify Constraints** process.

LeonardoSpectrum displays. First time users of the LeonardoSpectrum tool launch LeonardoSpectrum in 'Quick Setup' mode.

- Click on the Advanced Flow icon as shown below.



Figure 2-17: LeonardoSpectrum Advanced Flow Icon

- Click the **Constraints** tab.

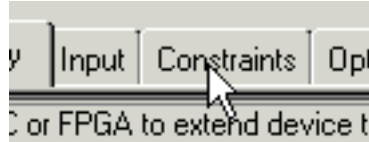


Figure 2-18: LeonardoSpectrum Constraints Tab

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

Constraints that affect the input ports such as arrival time, fanout, pin location, and pad type.

#### Output

Constraints that affect the output ports such as required time, pin location, and pad type.

#### Signal

Individual signal constraints such as preserve signal, a low skew constraint and a max fanout constraint.

#### Module

Tell the synthesiS tool to synthesize a module differently then the rest of the design.

#### Path

Create false and multicycle paths.

#### Report

A current report of constraints that have been entered.

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 2-19: 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 “Using the Constraints Editor” and “Using the Pin-out 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 make optimizations according to the needs of the design. One commonly used options 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.

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 on the **Synthesize** process in the Process Window.

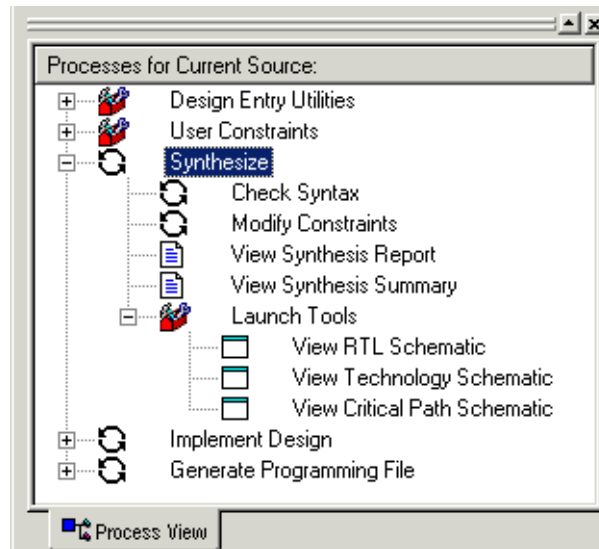


Figure 2-20: LeonardoSpectrum Synthesis/Implementation Window

## 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 **View RTL Schematic**.



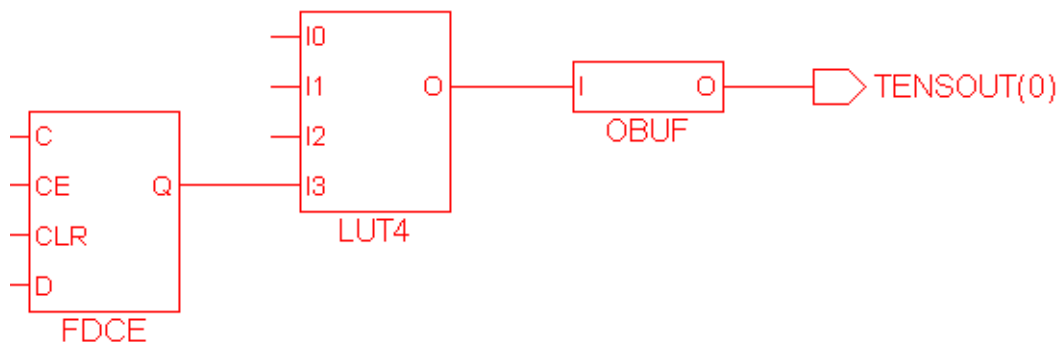


Figure 2-23: LeonardoSpectrum Critical Path Schematic

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 goes into more detail on the mapping and timing of the design.



# 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 5.x to perform this tutorial. For this design you must install the Virtex-II libraries and device files.

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* for your installation path.

**Note:** For detailed instructions about installing the software, refer to the *ISE 5.1i Installation Guide and Release Notes*.

## Installing the Tutorial Project Files

The tutorial project files can be downloaded to your local machine from <http://support.xilinx.com/support/techsup/tutorials/tutorials5.htm>.

You can download and install the following schematic project directories with the tutorial.

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

Unzip the tutorial projects in the `c:\xilinx` directory, 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 as it has been downloaded, or you can make a copy 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

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

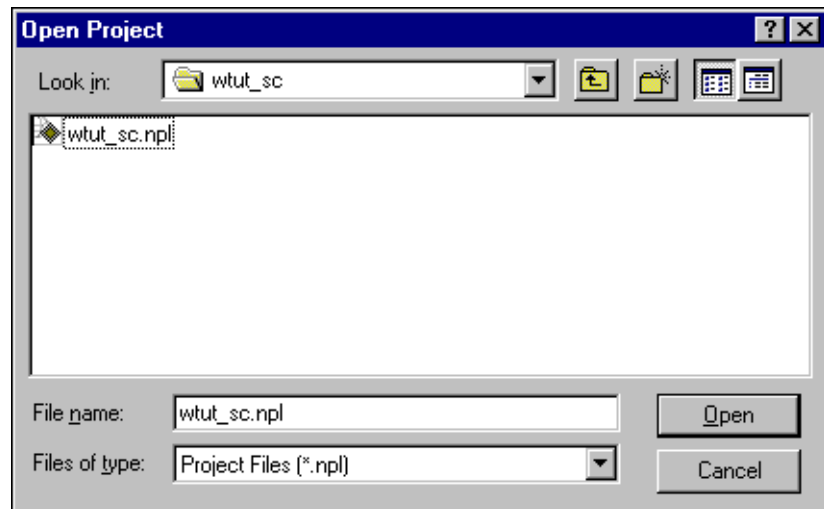


Figure 3-2: Open Project Dialog Box

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

## Stopping the Tutorial

If you need to stop the tutorial at any time, first 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 refers to several other lower-level macros. The lower-level macros are a variety of different types of modules, including a schematic-based module, CORE Generator module, state machine module, DCM Wizard Module, and HDL module.

Throughout this tutorial, the runner's stopwatch design you'll work with is referred to as Watch.

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. 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."](#)

Watch is a simple runner's stopwatch. The completed schematic is shown in the following figure.

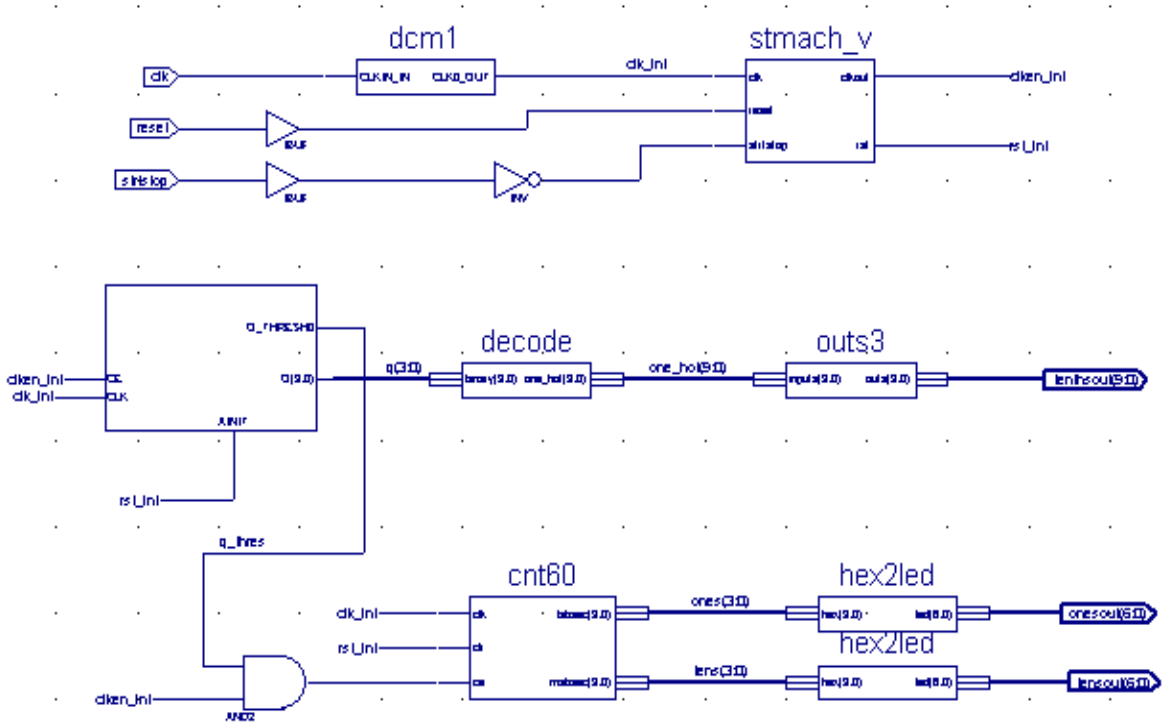


Figure 3-3: Completed Watch Schematic

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 runner's stopwatch 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**  
DCM 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

To open the *stopwatch.sch* file in the Engineering Capture System (ECS) schematic editor tool, double-click the file name *stopwatch.sch* in the Sources in Project window.

The Watch schematic diagram opens in ECS. The Watch schematic is incomplete at this point. The unfinished design is shown in the figure below.

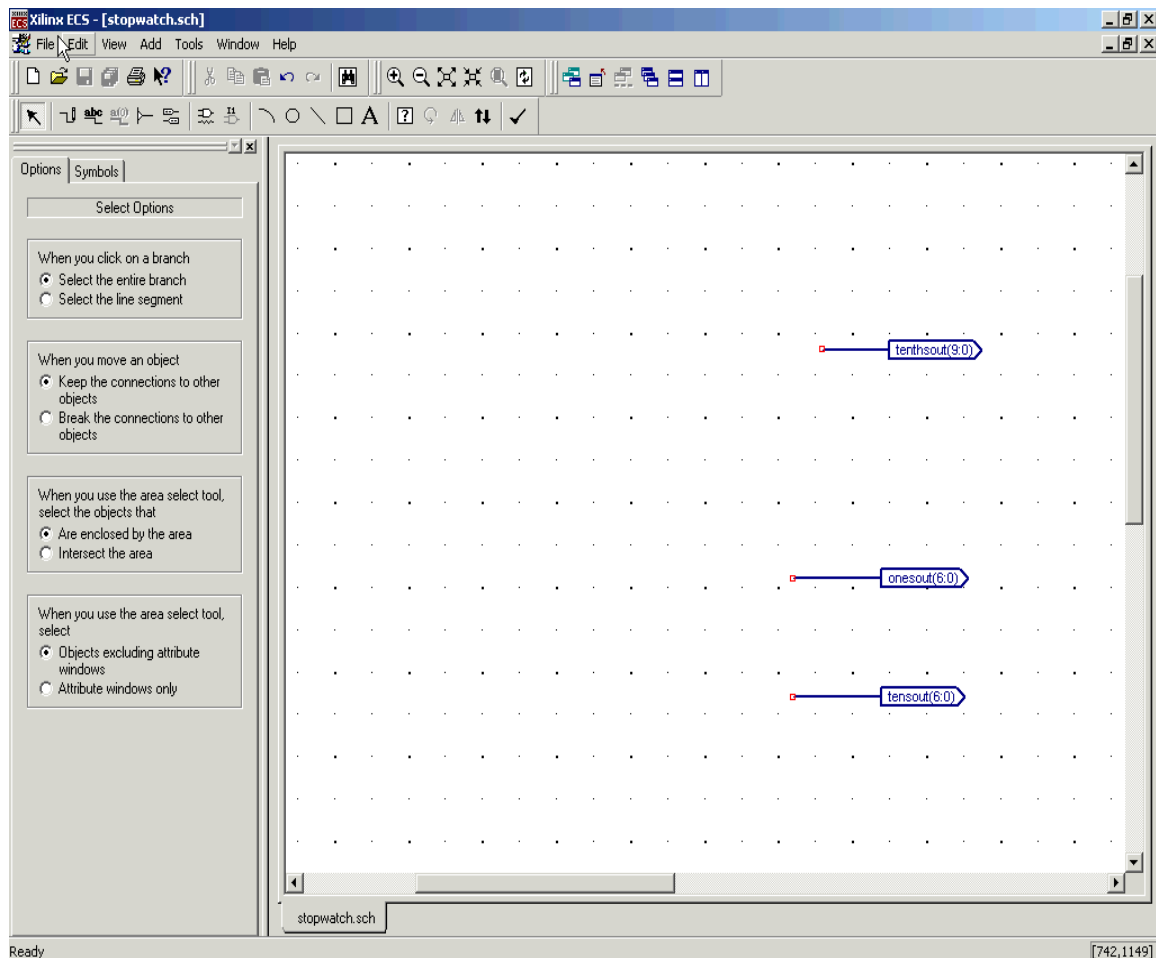


Figure 3-4: Incomplete Watch Schematic in Engineering Capture System (ECS)

### 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 by ISE that 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, decimal.

To create a schematic-based macro:

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

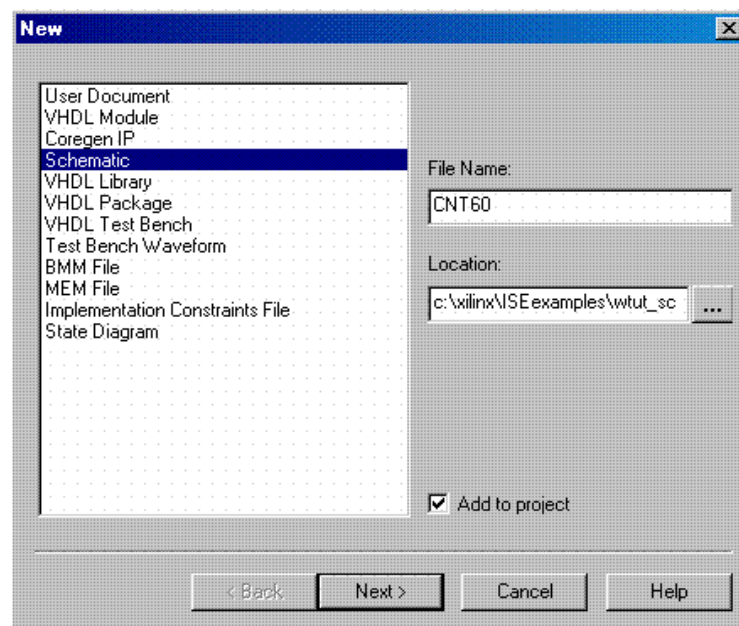


Figure 3-5: New Source Dialog Box

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 make up the CNT60 macro. You can then reference this macro symbol by placing it on a schematic sheet.

## Adding Components to CNT60

Components from the device and project libraries for the given project are available from the Symbol Libraries toolbox to place 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 3-6: Add Symbol Icon

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

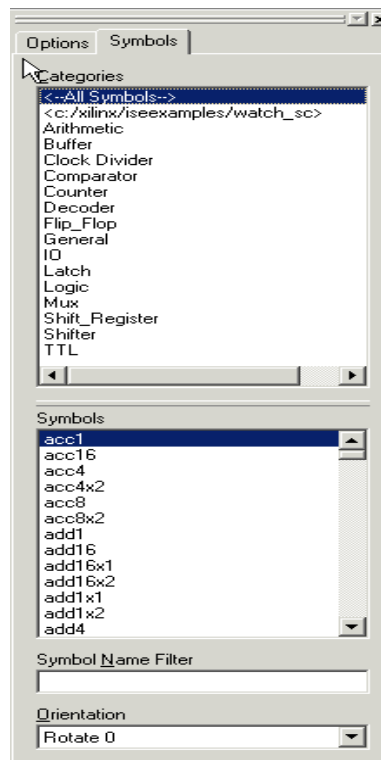


Figure 3-7: Symbol Browser Dialog Box

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

2. Select this component one of two ways:
  - ◆ Highlight the **Logic** category from the Symbol Browser 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 3-8](#) 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 3-8](#).

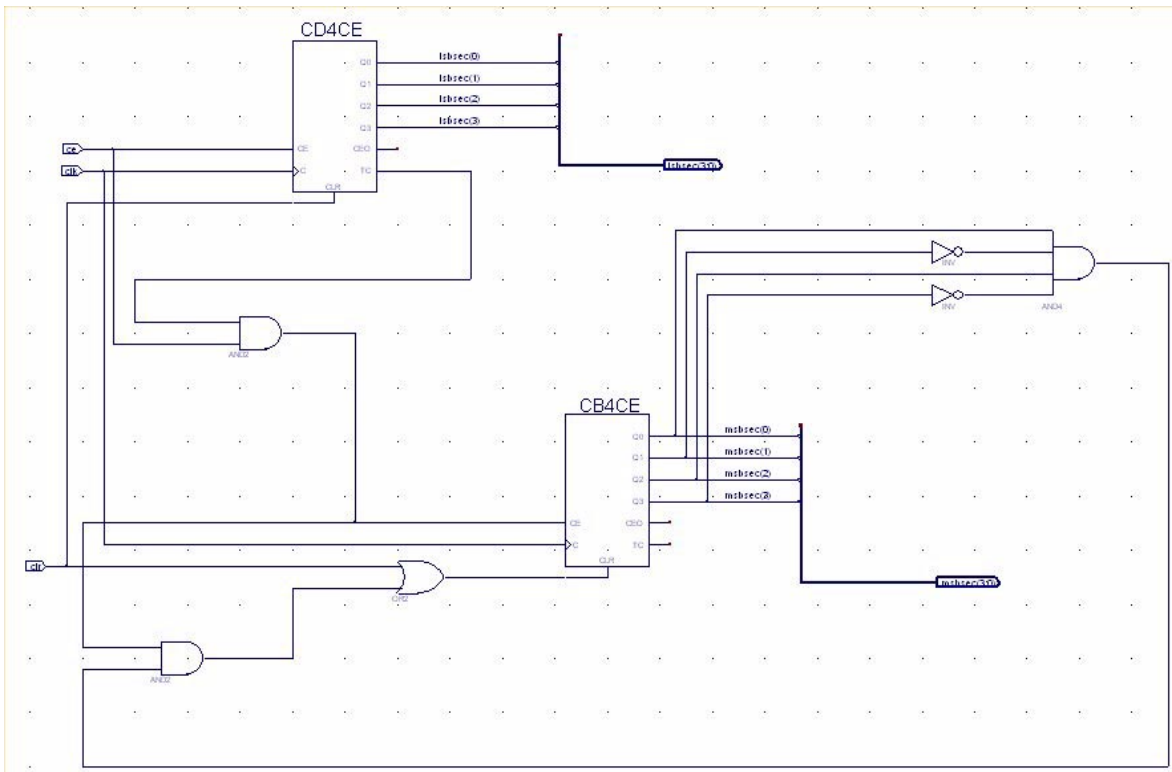


Figure 3-8: Completed CNT60 Schematic

### 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 3-8](#) 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, refer to the *Libraries Guide*, available in the collection of software manuals on the web, at [http://support.xilinx.com/support/sw\\_manuals/xilinx5/](http://support.xilinx.com/support/sw_manuals/xilinx5/).

## 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 click **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 the CB4CE components on the CNT60 schematic.

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



Figure 3-9: 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 3-8](#). 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. In order to add a bus follow the same methodology for adding wires and then add the proper 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. Then connect an I/O marker to each bus in order to connect them to the CNT60 symbol. 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 draw a wire down and to the right of the symbol.
3. Terminate the wire with a double-click on the left mouse button.

To change the wire into a bus, the wire must be named. To name a wire:

4. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.



Figure 3-10: Add Net Name Icon

5. With the keyboard enter 'msbsec(3:0)' and press **Enter**.  
This will attach the bus name to the cursor.
6. Click the mouse cursor, which now displays the bus name, at the end of the bus to apply the name.  
The wire changes to a bus.
7. To verify this, zoom in. The bus is represented visually by a thicker wire.

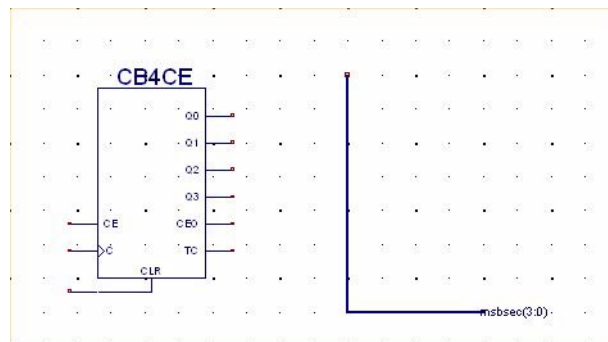


Figure 3-11: Creating a Bus by Name

8. Repeat Steps 1 through 7 for the LSBSEC(3:0) bus, referring to [Figure 3-8](#) for placement of the wire and the bus name.
9. After adding the two buses, press **Esc** or right-click to exit the Draw Buses 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 Taps 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 in 'msbsec(0)' in the blank area 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 window.
6. With the Increment Name option selected, start at the top net and continue clicking down until you have named the fourth and final net msbsec(3).

**Note:** ECS names the bus taps incrementally as they are drawn. Alternatively, name the first net msbsec(3) and decrement as nets are named from the bottom up.

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 3-8 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 clk, clr, and ce nets (wires).

1. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.
2. Type 'clk' in the Name box of the Add Net Name Options window.

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

The net name clk is now attached to the cursor.

3. Click the end of the clk net.

The name is then attached to the end of the net. The net name will appear above the net if the name is placed at any point of the net other than the endpoint.

Repeat steps 1-3 for ce and clr.

## 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 **Add** → **I/O Marker** or click the I/O Marker icon from the Tools toolbar.  
The Add I/O Marker Options window opens to the left of the schematic.
2. Select **Add an input marker**.
3. Click and drag a box around the following nets: clk, ce, and clr.
4. Select **Add an output marker** in the Options window and add a marker to the msbsec(3:0) and lsbsec(3:0) nets.

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 also to specify the ports on top-level schematic sheets.

## 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

2. Close ECS.

## Creating the CNT60 symbol

You will now create the symbol representing the CNT60 schematic in Project Navigator.

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**.

## Placing the CNT60 Macro

So far, you have created the CNT60 macro. The next step is to place this macro on the top-level Watch schematic sheet, where it will then be connected to other components in the design.

1. Open the Watch schematic sheet by double-clicking *stopwatch.sch* in the Sources in Project window.
2. Select the Add Symbol icon to open the Symbol Browser dialog box.

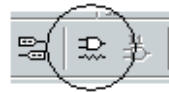


Figure 3-14: Add Symbol Icon

3. Select the Local Symbols library (*c:/xilinx/ISEexamples/wtut\_sc*), and locate and select the newly created CNT60 macro from this list.
4. Place the CNT60 macro in the schematic as shown below.

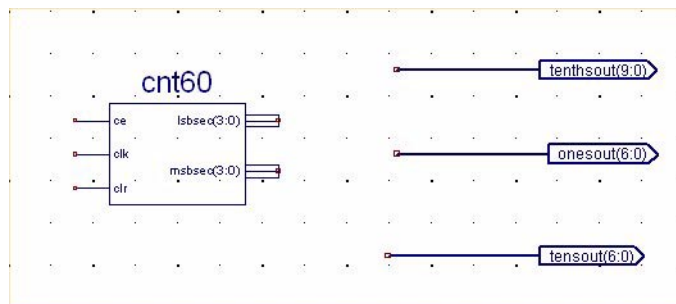


Figure 3-15: Placing the CNT60 Macro

**Note:** The Symbol Browser window remains open to enable you to quickly place additional symbols without having to click the Add Symbol icon again. To close the Symbols window, click the X in the upper right corner of the window or redock it to the right of the application.

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

5. Close 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 the CORE Generator Module

Select the type of module you want and the specific features of the module in the CORE dialog box. To create the CORE Generator module using this dialog box:

1. In Project Navigator, select **Project** → **New Source**.
2. Select **Coregen IP**, enter 'tenthS' in the File Name field.
3. Click **Next** and click **Finish**.

The Xilinx CORE Generator opens and displays a list of available COREs.

4. Double-click **Basic Elements - Counters**.
5. Double-click **Binary Counter** to open the Binary Counter dialog box. This dialog box enables you to customize the counter to the design specifications.
6. 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.
7. Select **Next**.
  - ◆ Threshold Options: **Enable Threshold 0 and set to A**  
Signal goes high when the value specified has been reached.
  - ◆ Select **Registered**

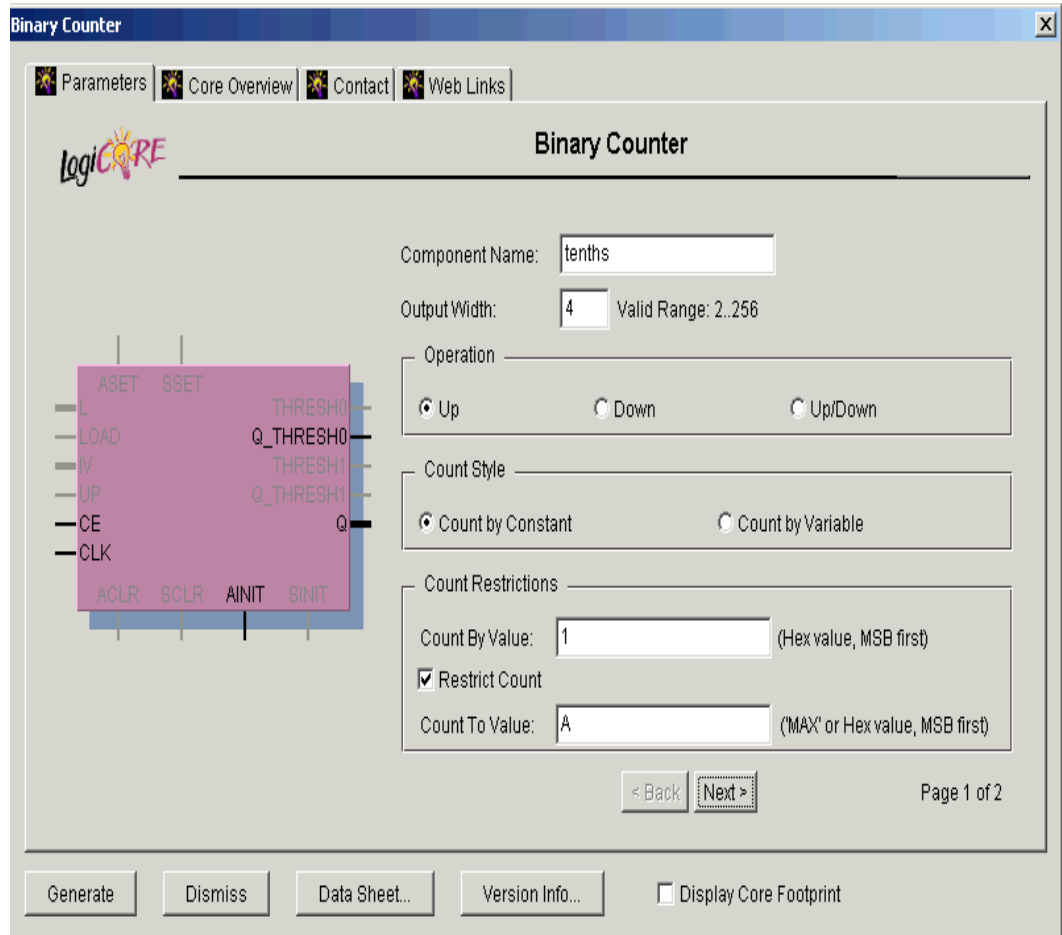


Figure 3-16: CORE Generator Module Selector

8. Click the **Register Options** button to open the Register Options dialog box.
9. Enter the following settings.
  - ◆ 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 Coregen 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 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.vhd* or *tenths.v*  
This is the instantiation template that is used to incorporate the CORE Generator module into your source HDL.
- ◆ *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 Coregen configuration for the project.

12. Select **Dismiss** and close 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.

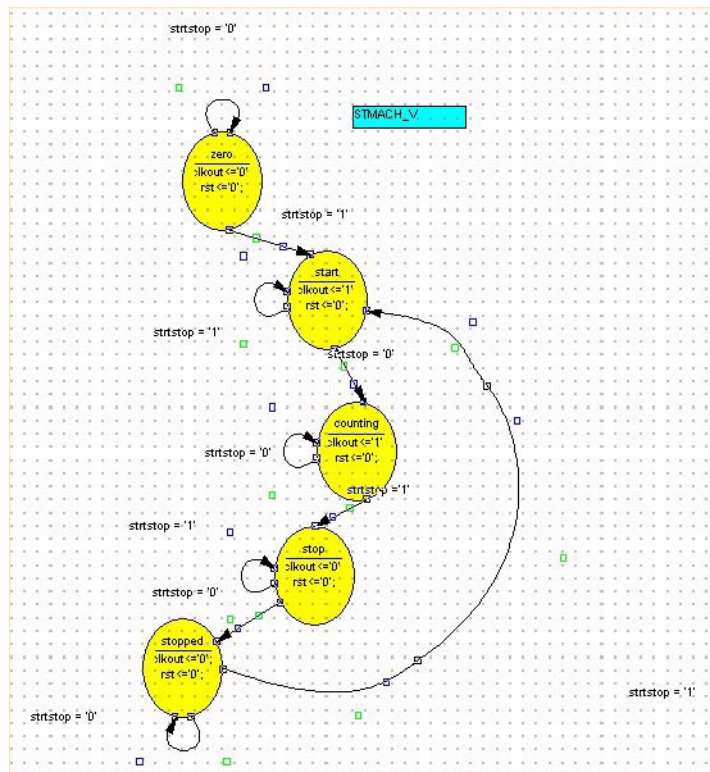


Figure 3-17: Incomplete State Machine Diagram

In the incomplete state machine diagram above:

- 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.

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.

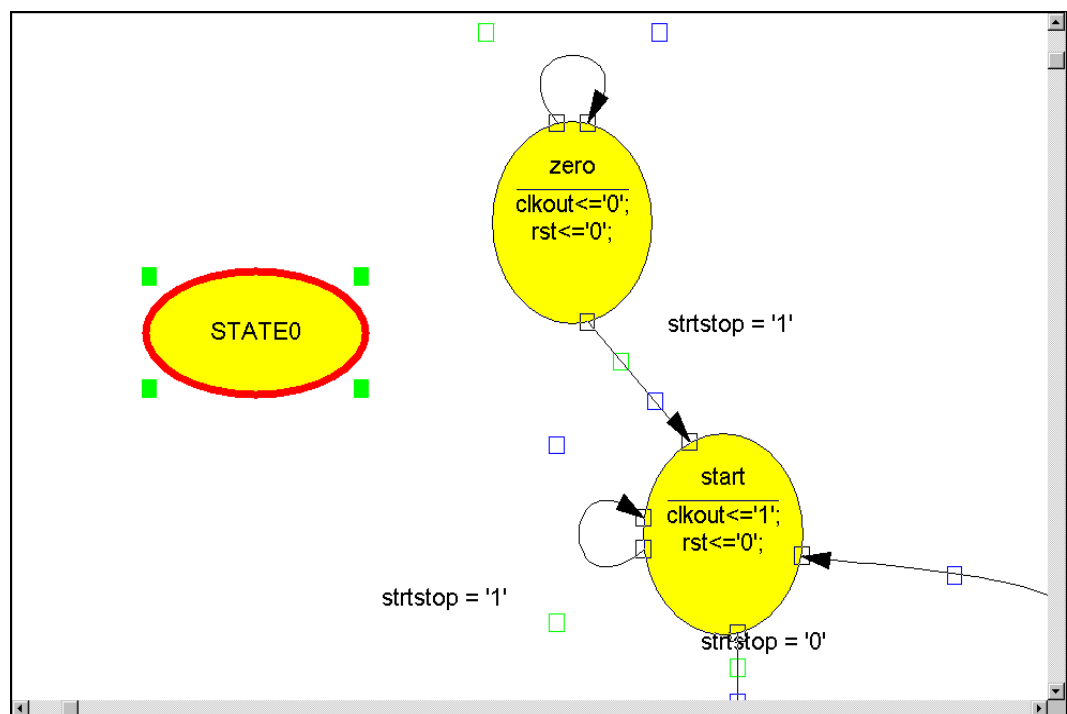


Figure 3-19: Adding the CLEAR State

3. The state is given a default name, in this case **STATE0**. 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 (once to select it, and once 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.

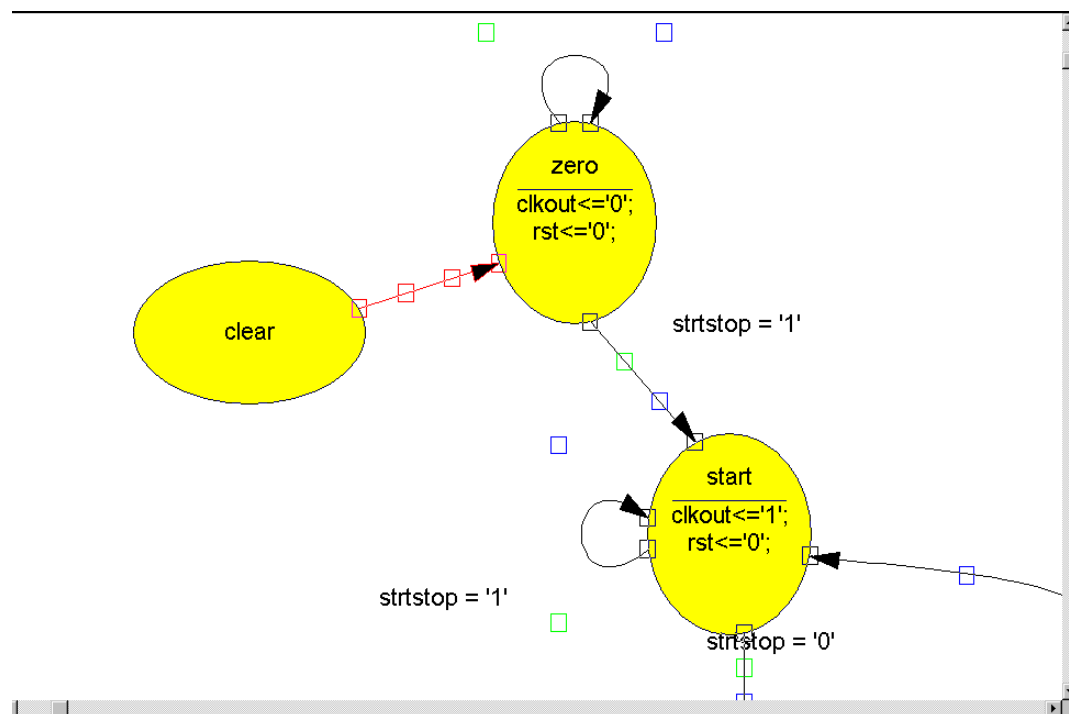


Figure 3-21: Adding State Transition

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 *clkout* 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.

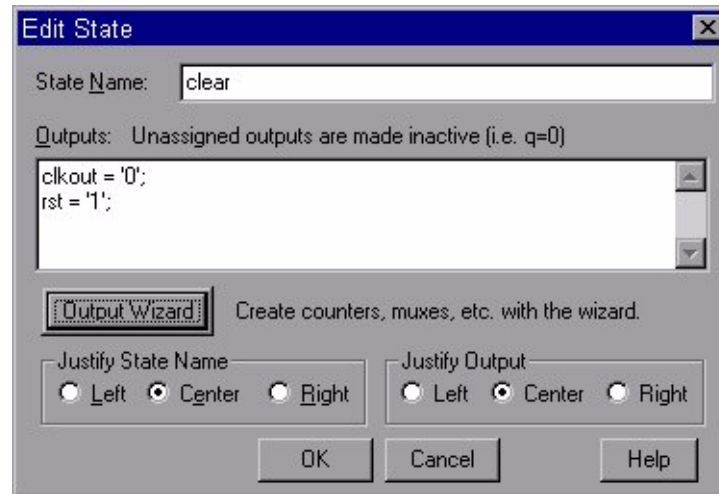


Figure 3-22: Edit State Dialog Box

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

```
DOUT = clkout, CONSTANT = '0';  
DOUT = rst, CONSTANT = '1';
```
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.

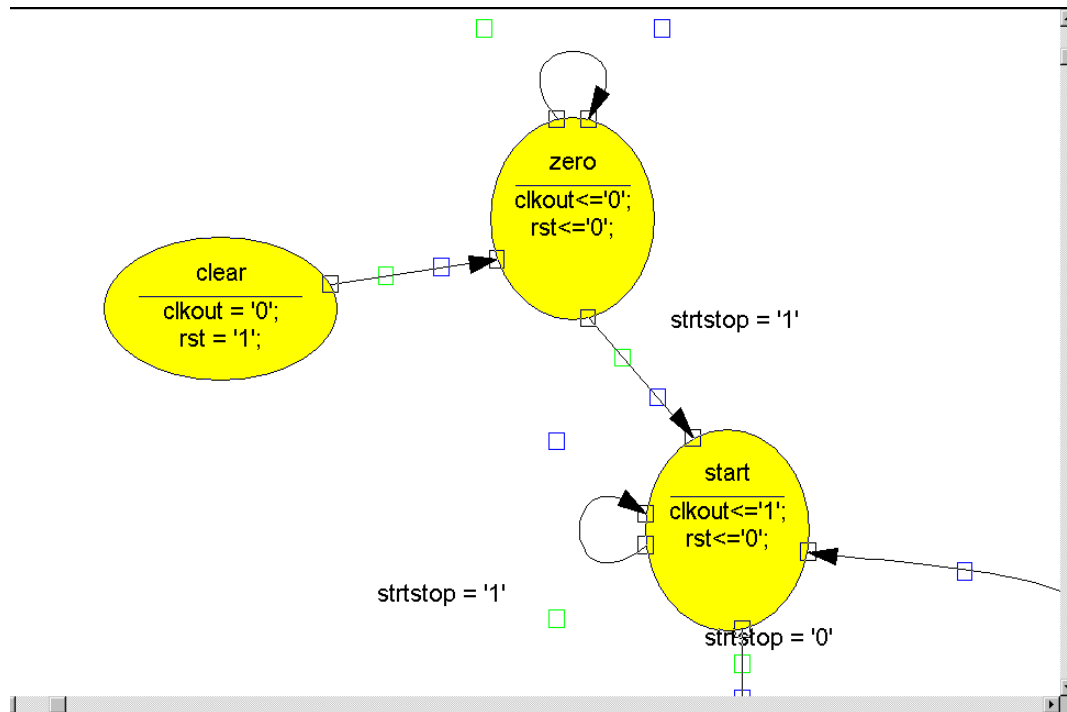


Figure 3-23: Adding State Outputs

### 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 the *reset* signal is asserted.

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**.

- A question is then asked, “Should this reset be asynchronous(Yes) or synchronous(No)?” Answer **Yes**.

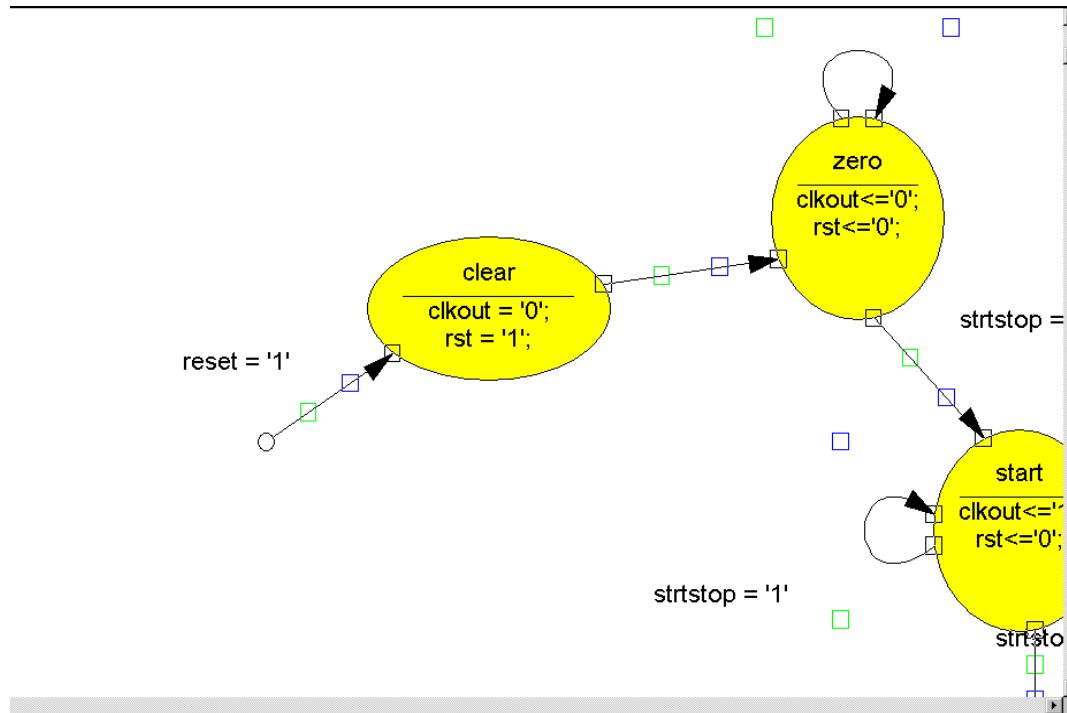


Figure 3-25: Adding Reset

- Save your changes by selecting **File** → **Save**.

## Creating the State Machine Macro

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.

- Select **Options** → **Compile (Generate HDL)**.  
StateCAD verifies the state machine and displays the results.
- Review the results and close the dialog box.  
StateCAD will then create the HDL code and open a browser displaying the code.
- Close the browser when you have finished examining the code.
- Close StateCAD.
- In Project Navigator, select **Project** → **Add Source**.
- Select *stmach\_v.vhd*, which is the VHDL file generated by StateCAD.
- Click **Open**.
- Select **VHDL module** as the source type.
- Click **OK**.

The file *stmach\_v.vhd* is added to the project.

10. 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 DCM Wizard, one part of the 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 **Architecture Wizard** source type, and enter the filename 'DCM1'.
3. Click **Next** and click **Finish**.
4. In the Xilinx Architecture Wizard Selection Box select **DCM Wizard**.
5. Click **OK**.
6. Deselect RST and Locked.
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**



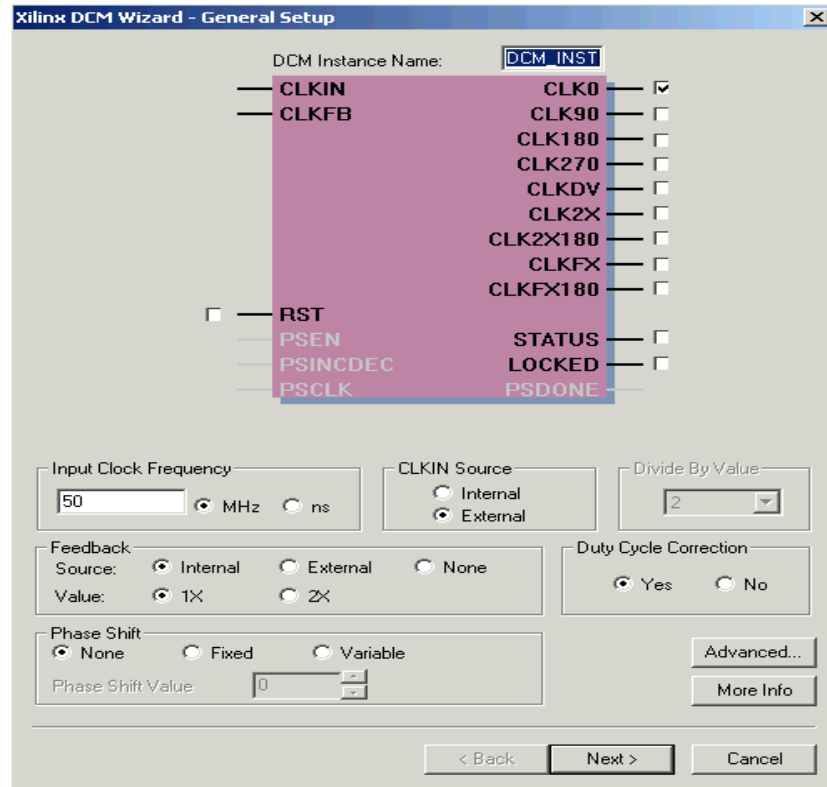


Figure 3-26: Xilinx DCM Wizard

9. Select the **Advanced** button.
10. Change Wait for DCM Lock before DONE Signal goes high to **Yes**.
11. Select **OK** and click **Next**.

An informational message about the Locked signal and the STARTUP\_WAIT option appears.

12. Select **OK** and then **Finish**.

*DCM1.xaw* is added to your project sources.

## Creating the DCM1 macro

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.

1. In Project Navigator, double-click *watch.sch*. The schematic file opens in the ECS schematic editor. If the file is already open in ECS, ignore this step.
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 3-27](#).

**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.

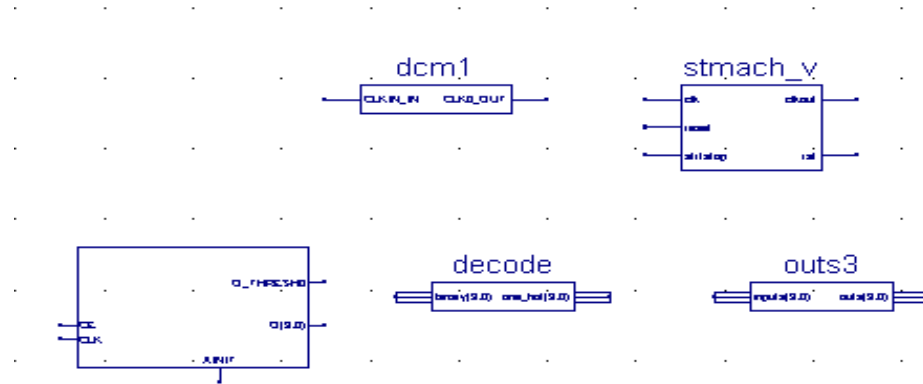


Figure 3-27: Placing Design Macros

## 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 HDL 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 3-28](#).

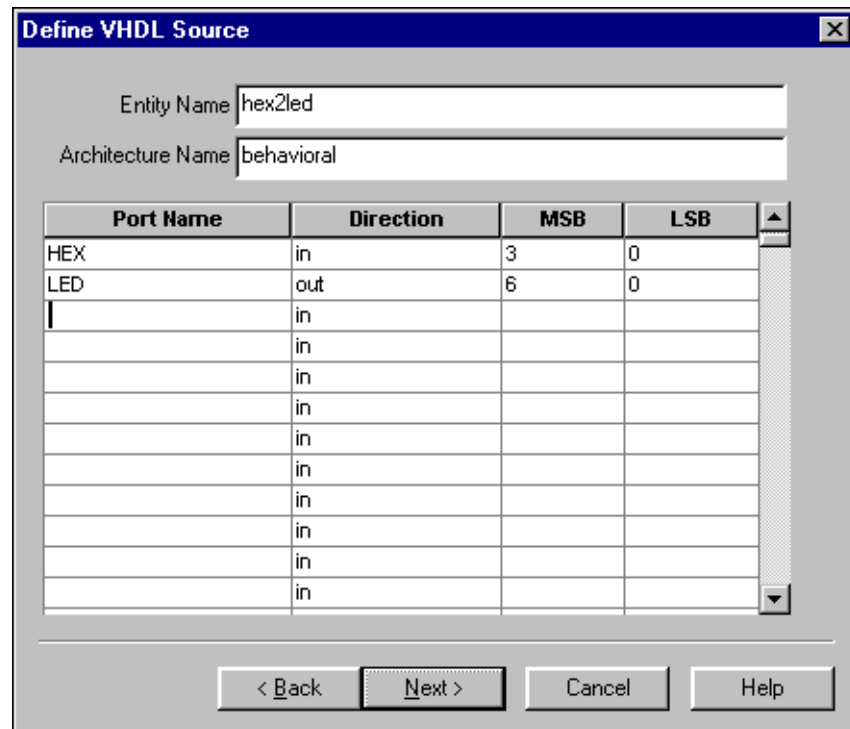


Figure 3-28: New Source Wizard

9. Select **Next** to complete the Wizard session.  
A description of the module displays.
10. Select **Finish**. The “skeleton” HDL file opens in the HDL Editor.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 -- Uncomment the following lines to use the declarations that ar
7 -- provided for instantiating Xilinx primitive components.
8 --library UNISIM;
9 --use UNISIM.VComponents.all;
10
11 entity hex2led is
12     Port ( HEX : in std_logic_vector(3 downto 0);
13           LED : in std_logic_vector(6 downto 0));
14 end hex2led;
15
16 architecture Behavioral of hex2led is
17
18 begin
19
20
21 end Behavioral;
22
```

Figure 3-29: Skeleton VHDL File

```
1 module HEX2LED(HEX,LED);
2     input [3:0] HEX;
3     input [6:0] LED;
4
5
6
7 endmodule
8
```

Figure 3-30: 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 printed in blue, data types in red, comments in green, and values are 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. You can add your own templates to the Language Templates for components or constructs 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.

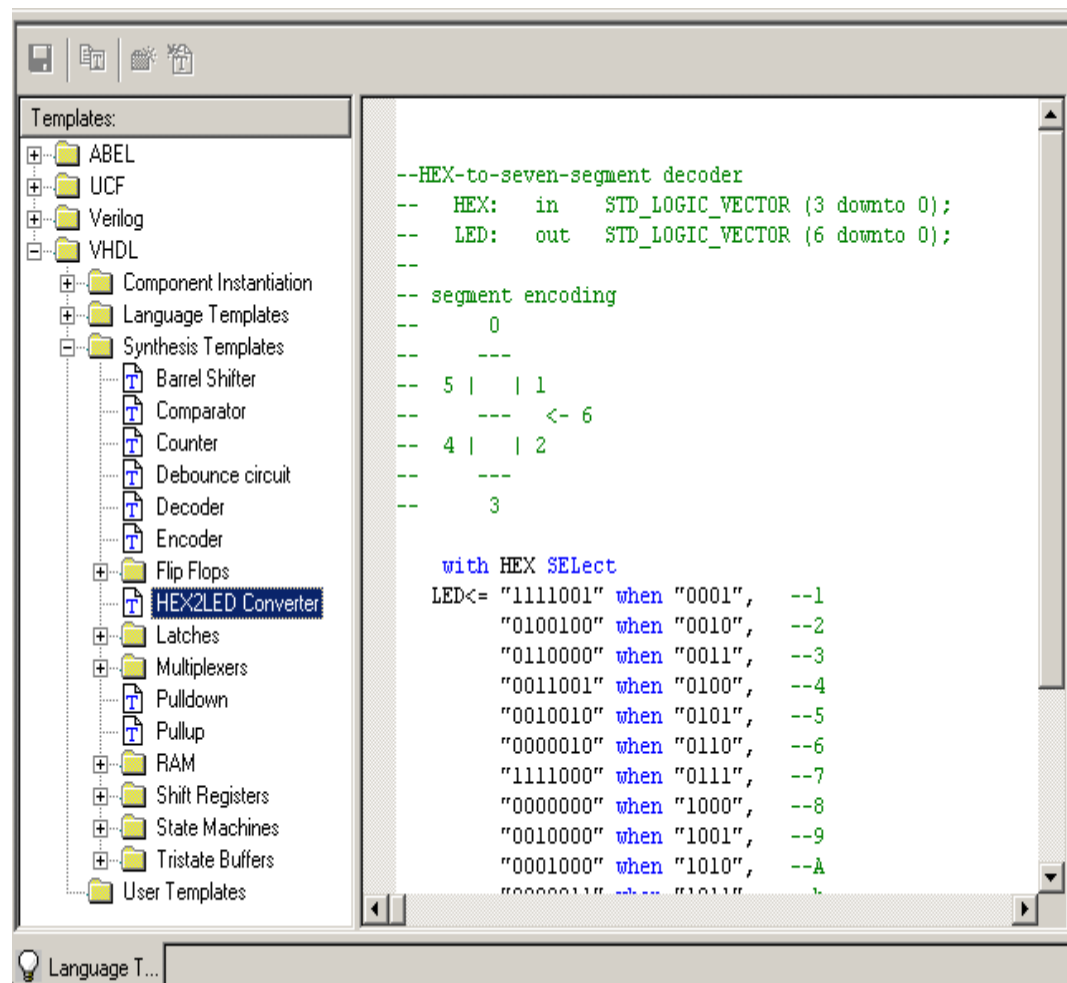


Figure 3-31: Language Templates

## Adding the Language Template to Your File

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

To add the HEX2LED language template to your file:

1. In the Language Template, click and drag the **HEX2LED Converter** name into
  - ◆ *hex2led.vhd* under the architecture begin statementor
  - ◆ *hex2led.v* file under the module declaration.
2. Close the Language Template 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 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.
7. Close HDL Editor.

## Creating 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**.

## Adding the HEX2LED Component to the Schematic

You are now ready to place the HEX2LED macro (or 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. Open the **Symbols Libraries dialog box** (refer to “[Adding Components to CNT60](#)”) 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 3-32](#). You will connect the entire schematic later in the tutorial.

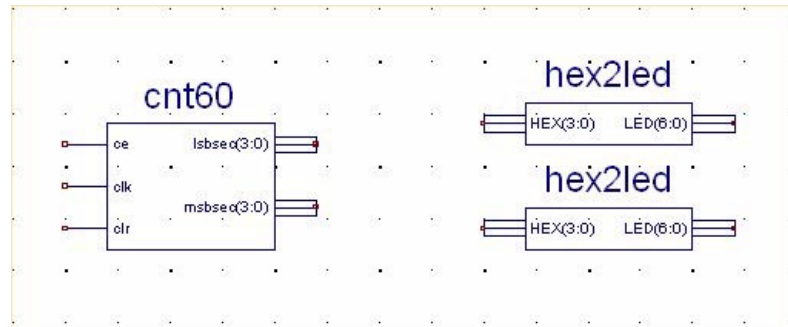


Figure 3-32: Placing the HEX2LED Component

## 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:

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



Figure 3-33: Hierarchy Push/Pop 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.

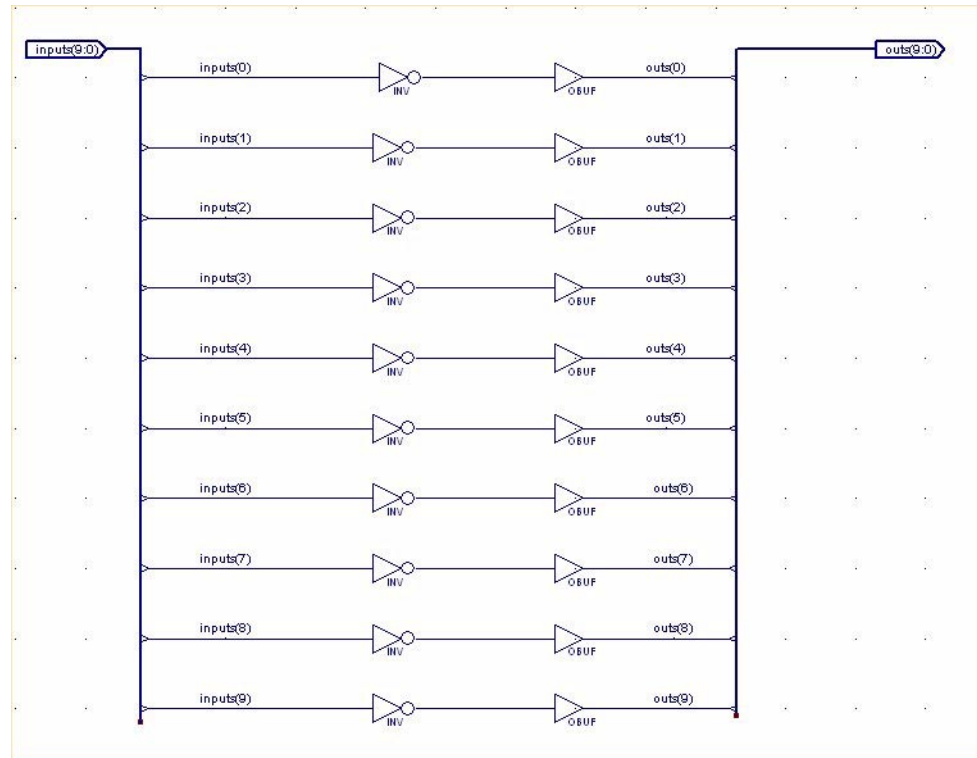


Figure 3-34: OUTS3 Schematic Macro

2. After examining the macro, “pop out” of the OUTS3 component by clicking the Hierarchy Push/Pop icon.

### 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.



- Draw a net between the output of the IBUF and input of the INV. Refer to “Drawing Wires” for detailed instructions.

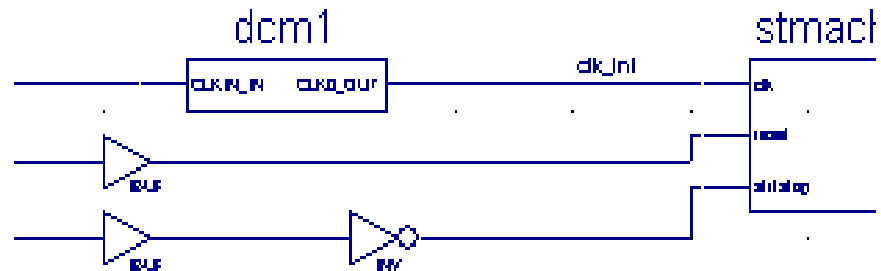


Figure 3-35: Placing CLK, RESET and STARTSTOP I/O Components

## 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, which 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:

- Select **Add** → **Net Name**.
- Type 'reset' into the Name box.  
The net name is now attached to the cursor.
- Place the name on the leftmost end of the net as illustrated in Figure 3-36.
- Repeat Steps 1 through 3 for the STARTSTOP and CLK pins.  
Once all of the nets have been labeled, add the I/O marker.
- Select **Add** → **I/O Marker**.
- In the Add I/O marker Options window, select **Add an input** for an input signal direction.
- Click and drag a box around the three labeled nets to place an input signal around each net name.

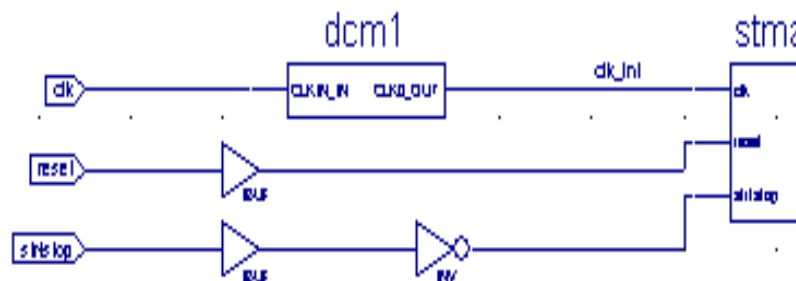


Figure 3-36: Labeled Nets with I/O Markers

## Assigning Pin Locations

Xilinx recommends that you let the automatic placement and routing program, PAR, 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 is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a Printed Circuit Board (PCB).

Define the initial pinout by running the place-and-route tools without pin assignments, then locking 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. 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.

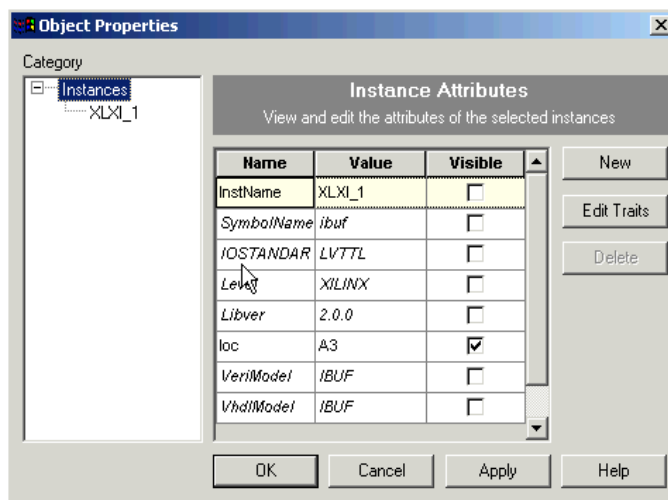


Figure 3-37: Assigning Pin Locations

To view the location constraint on the net, add a net attribute window.

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.

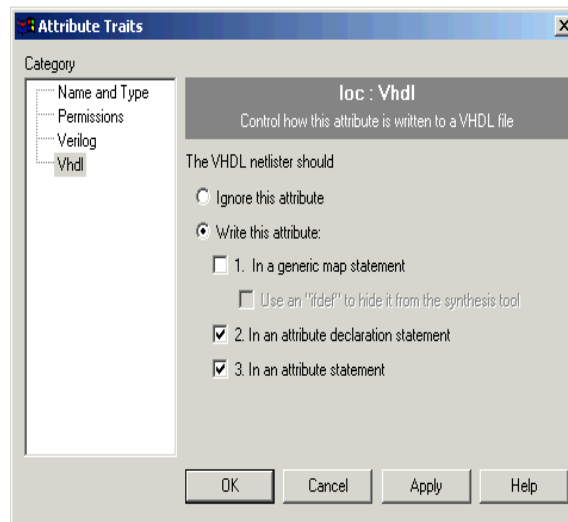


Figure 3-38: Writing attribute to HDL file

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

**Note:** For details on adding Pin LOCs and other constraints, see [“Using the Constraints Editor”](#) and [“Using the Pin-out 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 in this section has been discussed in detail in earlier sections of the tutorial. If you need to review these sections, you may return to them.

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

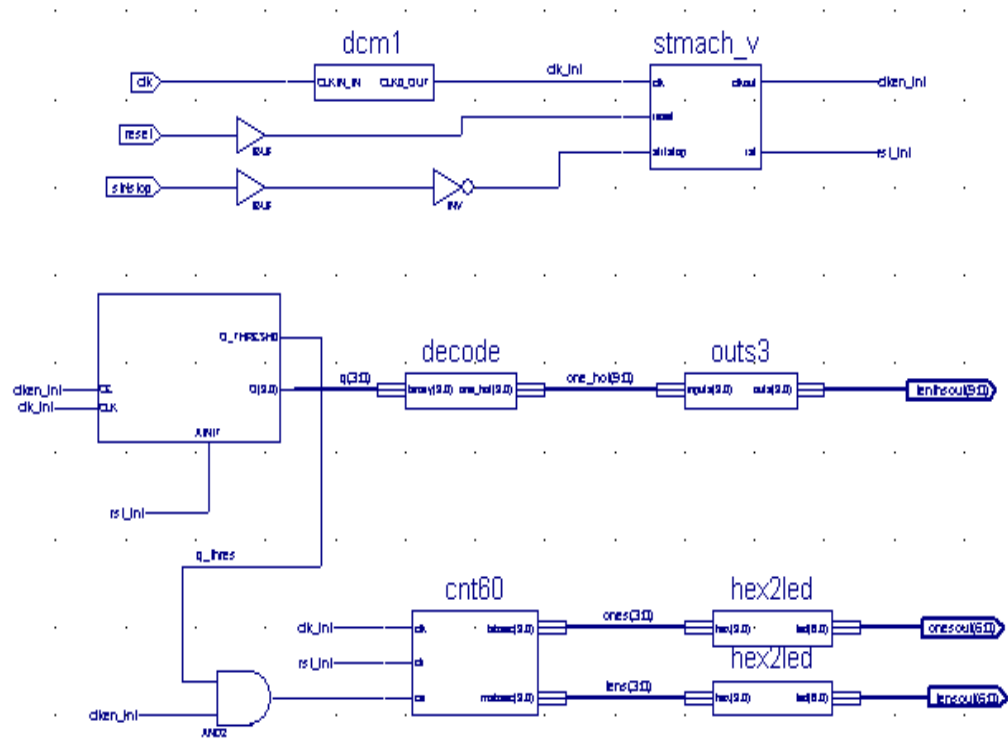


Figure 3-39: Completed Watch Schematic

To complete the schematic diagram:

1. Draw a wire between the output of DCM1 and the CLK pin of the STMACH state machine macro (see “Drawing Wires”).
2. Label this net CLK\_INT.
3. Draw a wire between the IBUF of the RESET input and the RESET pin of the STMACH state machine macro (see “Drawing Wires”).
4. 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”).
5. Draw wires to connect the INV to both the IBUF and the STMACH state machine macro (see “Drawing Wires”).
6. Place an AND2 component to the left of the CNT60 macro (see “Adding Components to CNT60”).
7. Draw a wire to connect the output of the AND2 with the CE pin of the CNT60 macro (see “Drawing Wires”).
8. Draw a wire to connect the Q\_THRES0 pin of the TENTHS macro to one of the inputs to the AND2 (see “Drawing Wires”).
9. Draw a hanging net from the CLKOUT pin of the STMACH macro. To terminate a hanging wire, double-click it (see “Drawing Wires”).
10. Name the new added net CLKEN\_INT.

11. 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”](#)).
12. 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.
13. Draw a hanging wire from the RST output pin of the STMACH macro (see [“Drawing Wires”](#)).
14. Label this net RST\_INT.
15. 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.”](#))
16. 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.”](#))
17. 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”
- “Getting Started”
- “Adding an HDL Testbench”
- “Creating a Testbench Waveform Using HDL Bencher”
- “Behavioral Simulation Using ModelSim”

## Overview of Behavioral Simulation Flow

You can perform behavioral simulation before design implementation to verify that the logic you have created is correct. Behavioral simulation is also called functional simulation.

Xilinx ISE provides integration with any ModelSim Simulator. ISE produces generic HDL netlists that work with most HDL simulators. However, the integrated flow is only available with MTI ModelSim.

You can perform behavioral simulation on a schematic-based or HDL-based design. In a later section, you can perform timing simulation, which takes place after the design is implemented (placed and routed) with the Xilinx implementation tools.

Behavioral simulation is an integral part of any HDL design flow. It enables a logical (functional) check of the design before any additional time is invested in synthesis and implementation. Xilinx ISE 5 provides a tightly integrated functional simulation flow with any version of ModelSim (release 5.4 and newer). ModelSim 5.6a XE is used for the examples in this tutorial.

Using ISE, behavioral simulation can be conducted using either a hand-written HDL testbench (PC and Unix), or one generated automatically by Xilinx HDL Bencher (PC only).

This chapter assumes that the ModelSim CD has been installed on your computer. Since Xilinx HDL Bencher is only available on the PC platform, Unix machine users should skip the automated testbench generation section which uses HDL Bencher. PC users can choose either flow.

## Getting Started

The following sections outline the requirements to perform this part of the tutorial flow.

### Required Files

The functional simulation flow requires the following files:

- **Design Files (VHDL, Verilog, or Schematic)**  
These files are produced by the designer, or from an HDL generation tool such as Xilinx StateCAD.
- **Stimulus File**  
The stimulus files are known as the testbench (VHDL, Verilog). A testbench can be hand-written or produced using Xilinx HDL Bencher. To produce a testbench using HDL Bencher, see “[Adding an HDL Testbench](#)”. The HDL Bencher flow is only available on Windows platforms.
- **ModelSim Script File**  
Use this file to run the simulation (optional). Alternatively, you can enter the commands one-by-one into the simulator. Xilinx ISE creates the script file needed to run simulation in ModelSim.
- **Xilinx Simulation Libraries**  
Xilinx Simulation Libraries are required if any Xilinx primitive is instantiated in the design. More details on the libraries and how to compile them is provided in the next section, “[Xilinx Simulation Libraries](#)”.

### Xilinx Simulation Libraries

To simulate designs containing Xilinx primitives with ModelSim in VHDL or Verilog, you need the simulation libraries listed below, which you must compile. If you are using ModelSim Xilinx Edition, the models are already precompiled. To get the latest models for ModelSim XE go to <http://support.xilinx.com/support/mxlibs/index.htm>

#### Unisims Library

The Unisims library is used for behavioral (RTL) simulation with instantiated components in the netlist, and for post-synthesis simulation.

- The recommended mapping name for the VHDL Unisims library is UNISIM.
- The recommended mapping name for the Verilog Unisims library is UNISIMS\_VER.
- Additionally, there is a separate Unisims library in Verilog for simulating CPLD designs. This library is called UNI9000. The recommended mapping name for this library is UNI9000.



## XilinxCoreLib Library

The XilinxCoreLib library must be used if a CoreGEN component is instantiated in the design.

- The recommended mapping name for the VHDL library is XILINXCORELIB.
- The recommended mapping name for Verilog is XILINXCORELIB\_VER.
- All VHDL simulation libraries are provided at \$XILINX/vhdl/src.
- All Verilog simulation libraries are provided at \$XILINX/verilog/src.

For detailed instructions on compiling these libraries, see Xilinx Solution # 2561:

1. Go to <http://support.xilinx.com>.
2. Enter **2561** in the search box.
3. Check to see that the search engine is pointing to **Answer Records**.
4. Click **OK**.  
Solution 2561 displays on the next page.
5. Click Solution 2561 on the next page.

## Viewing the Modelsim.ini File

Before compiling the libraries, view the *modelsim.ini* file in the ModelSim install directory. The upper portion of the file defines the locations of the compiled libraries. When doing a simulation, the *modelsim.ini* file must be provided either by:

- Copying the file directly to the directory where the HDL files are to be compiled and the simulation is to be run, or
- Setting the MODELSIM environment variable to the location of your *master.ini* file. You must set this variable since the ModelSim installation does not initially declare the path for you. (See below.)

## Setting the Environment Variable

For UNIX, type the following environment variable:

```
setenv MODELSIM /path/to/the/modelsim.ini
```

For PCs, set the MODELSIM environment variable to the path where the *modelsim.ini* file is located. To set the environment variable, go to **Start** → **Settings** → **Control Panel** → **System** → **Environment**.

## Adding an HDL Testbench

This section demonstrates how to add pre-existing testbench files to the project. This design flow is for users of UNIX machines who cannot generate testbenches using HDL Benchers. PC machine users can use hand-written testbenches for their design. The testbench must be associated with the top-level design file in order to successfully simulate the design.

## VHDL Design

To add your testbench for a VHDL design:

1. Select **Project** → **Add Source**.
2. Select the testbench file *stopwatch\_tb.vhd*.
3. Click **Open**.

The Choose Source Type dialog box opens.

4. Select **VHDL Testbench**.
5. Click **OK**.

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

## Verilog Design

To add your testbench for a Verilog design:

1. Ensure that the extension of the testbench file is *.tf* rather than *.v*.
2. Select **Project** → **Add Source**.
3. Select the testbench file *stopwatch\_tb.tf*.
4. Click **Open**.

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

## Creating a Testbench Waveform Using HDL Bencher

HDL Bencher is a PC-based testbench and test fixture creation tool that is part of ISE 5. Use HDL Bencher to graphically enter stimuli and the expected response, then generate a VHDL testbench or Verilog test fixture.

### Creating a Testbench Waveform Source

To create a testbench 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 and 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).

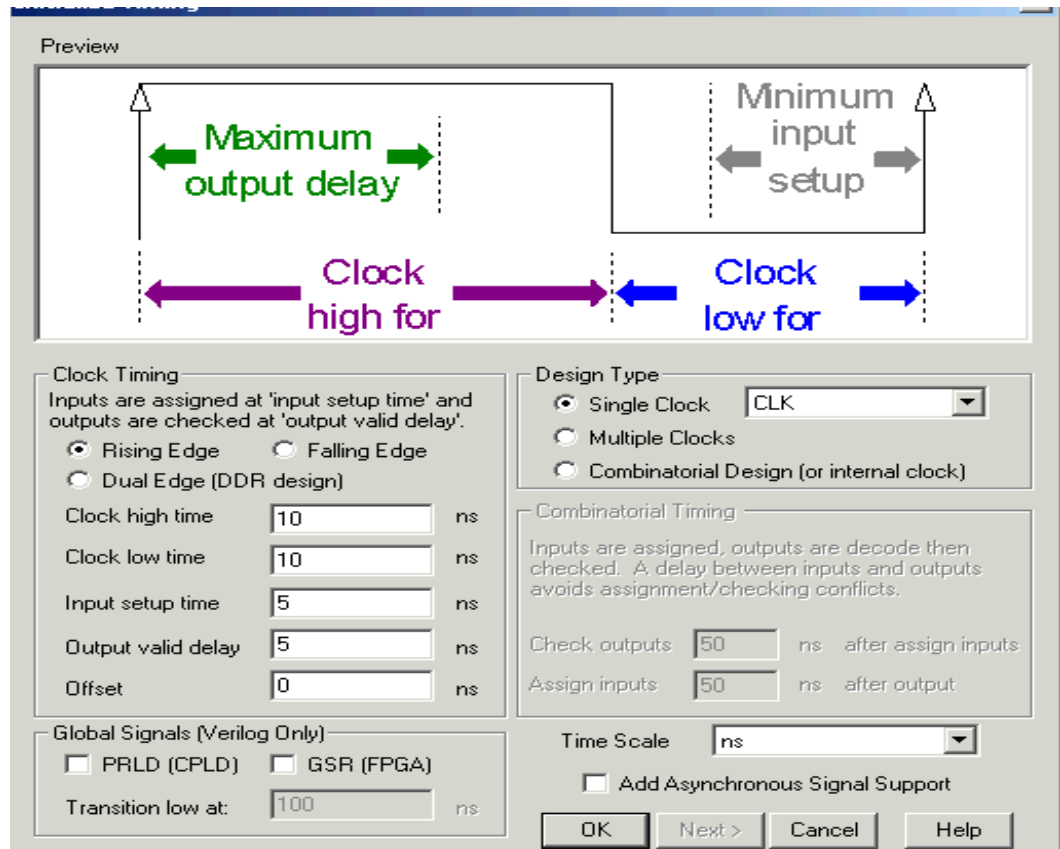


Figure 4-1: HDL Bencher Initialization

**Note:** Click OK. If the Verilog project is being used, make sure that GSR is selected before clicking OK.

HDL Bencher now opens with two main windows. The top window is the Waveform window. In this window, enter graphical depictions of the stimuli and expected response. The bottom window is the currently loaded HDL file.

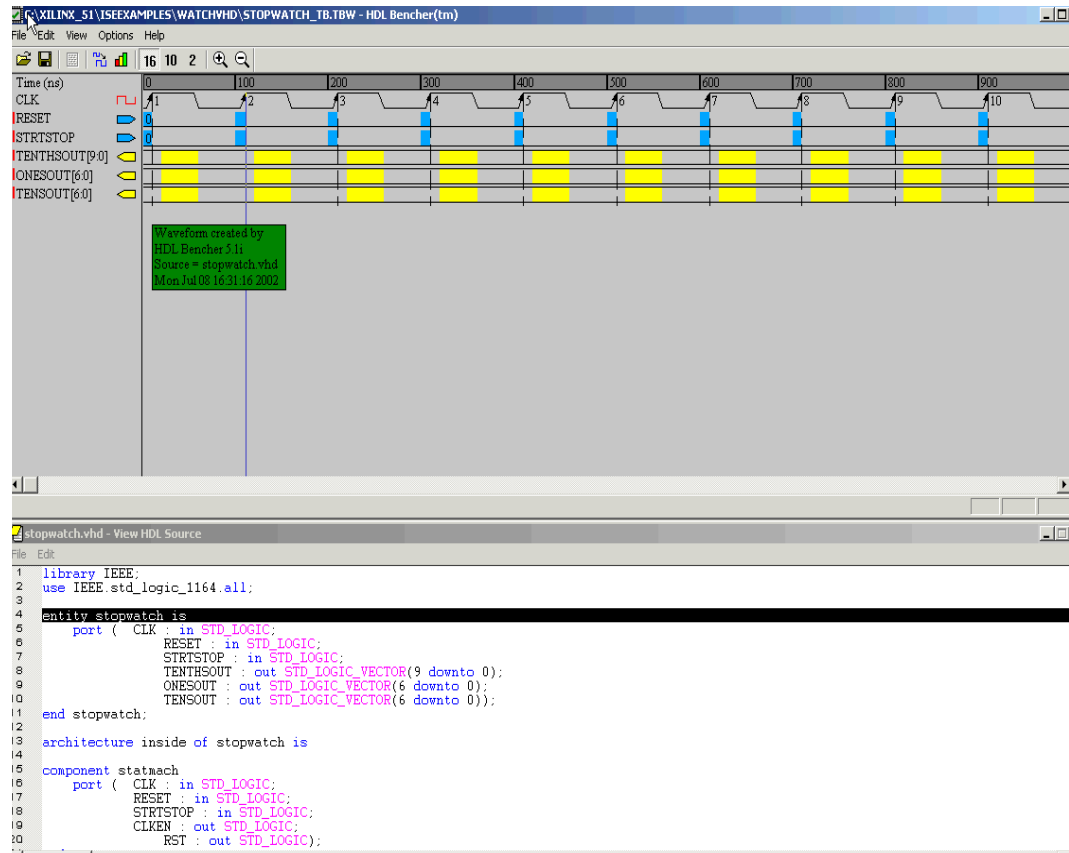


Figure 4-2: HDL Bencher Windows

## Initializing Inputs

Enter the following input stimuli:

1. Click the RESET cell at time 0 to set it high.
2. Click the RESET cell under CLK cycle 5 to set it low.
3. Click the STRTSTOP cell under CLK cycle 50 to set it high.
4. Click the STRTSTOP cell under CLK cycle 55 to set it low.
5. Click the STRTSTOP cell under CLK cycle 65 to set it high.
6. Click the STRTSTOP cell under CLK cycle 70 to set it low.
7. Grab the blue line (end of testbench) and drag it to CLK cycle 80.
8. Click the Save Waveform icon in the toolbar.
9. Exit HDL Bencher.

**Note:** STRTSTOP is set to high only at CLK cycle 50 to give the DCM time to lock, and to be able to give out a valid clock output.

The new testbench waveform source (*stopwatch\_tb.tbw*) is automatically added to the project. In the future, you can open HDL Bencher from Project Navigator by double-clicking this file.

## Behavioral Simulation Using ModelSim

ISE has full integration with any version of the ModelSim Simulator. ISE provides work directory creation, source file compilation, simulation initialization, and control over simulation properties.

### Selecting Simulation Processes

To display the ModelSim Simulator Processes:

1. In the Sources in Project window, select *stopwatch\_tb.tbw*.
2. Click the + beside ModelSim Simulator to expand the process hierarchy.

The following simulation processes are available:

- **Simulate Behavioral VHDL (or Verilog) Model**  
This process will start the design simulation.
- **Generate Expected Simulation Results**  
This option is available only if you have a TBW file from HDL Benchner. If you double-click on this, ModelSim will run in the background to generate expected results and display them in HDL Benchner.
- **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.

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

### Specifying Simulation Properties

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

1. In the Sources in Project window, select *stopwatch\_tb.tbw*.
2. Click the + sign next to ModelSim Simulator in the Processes For Current Source window.
3. Right-click **Simulate Behavioral VHDL (or Verilog) Model**.
4. Select **Properties**.

The Process Properties dialog box (Figure 4-3) contains the following simulation properties which can be specified or changed as indicated below:

- ◆ **Custom Do File**  
Enables users to select a user-created .do file.
- ◆ **Use Automatic Do File**  
When unchecked, this option will bring up ModelSim but not automatically run the processes required to simulate the design. You will have to manually run the .do file from ModelSim or enter the commands one-by-one to run simulation.
- ◆ **Simulation Run Time**  
Specifies default time for which simulation is run. The “-all” setting tells ModelSim to run the simulation until it encounters a break specified in the test bench. Alternatively, you can enter a setting of “1000ns” so that the simulation only runs for 1000ns.
- ◆ **Simulation Resolution**  
This is set to 1 ps by default. All Xilinx simulations should be run with the resolution set to 1ps.
- ◆ **Design Unit Name**  
Enables you to specify the top-level model to be loaded in ModelSim. This property must be changed if the testbench, module, or entity is named something different than testbench.

The Second tab in this GUI is for selecting Display Properties. Using this tab, you can select which ModelSim Simulation windows will automatically be invoked. By default, the Signals, Structure and Wave windows are invoked. For more details on MTI ModelSim Simulator windows, refer to the *ModelSim User Manual*.

5. For the purpose of this tutorial, none of the defaults need to be changed. Click OK to continue.

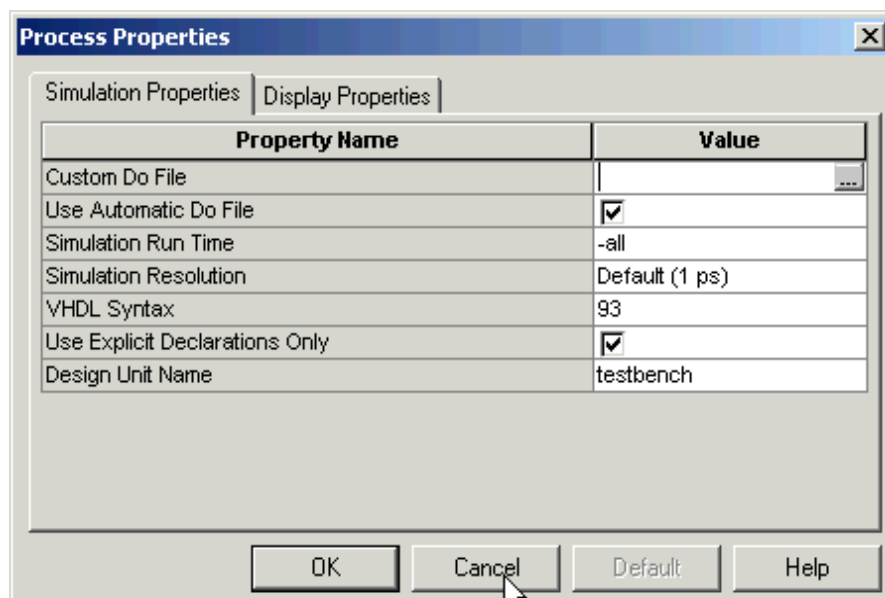


Figure 4-3: Process Properties Dialog Box

## Performing Simulation

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

## Adding Signals

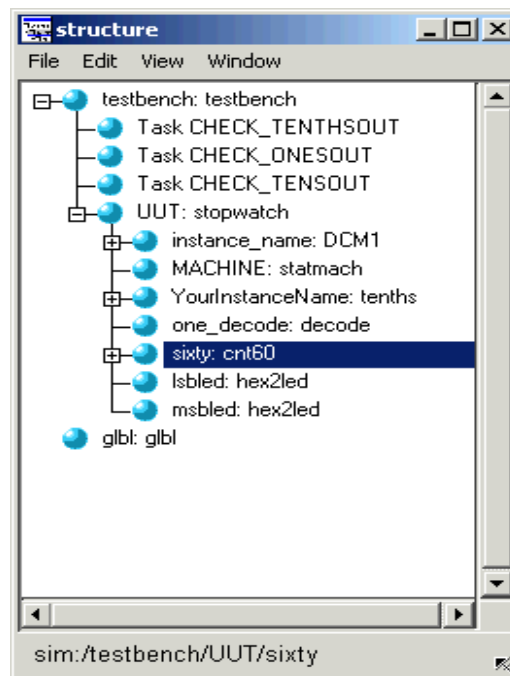
To view signals during the simulation, you must add them to the Wave window. ISE automatically adds all the top-level ports to the Wave window. Additional signals are displayed in the Signal window based upon the selected structure in the Structure window.

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:stopwatch`.



*Figure 4-4: Structure Window - Verilog flow (In schematic / VHDL flow this may be different.)*

2. Select `sixty:cnt60`(inside) 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.

When new signals are added to the waveform window, the simulation needs to be re-run. To restart and re-run the simulation:

1. Click **Restart Simulation**.



Figure 4-5: Restart Simulation Icon

The Restart dialog box opens.

2. Click **Restart**.
3. Click **Run-all** or **Run** to re-run the simulation.

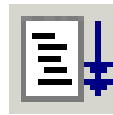


Figure 4-6: Run-All Icon

## 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 have been added and the simulation must be restarted. The saved signals list can easily be loaded each time the simulation is started.

1. In the Wave window, select **File** → **Save Format**.
2. In the Save Format dialog box, change the default filename *wave.do* to *sec\_signal.do*.

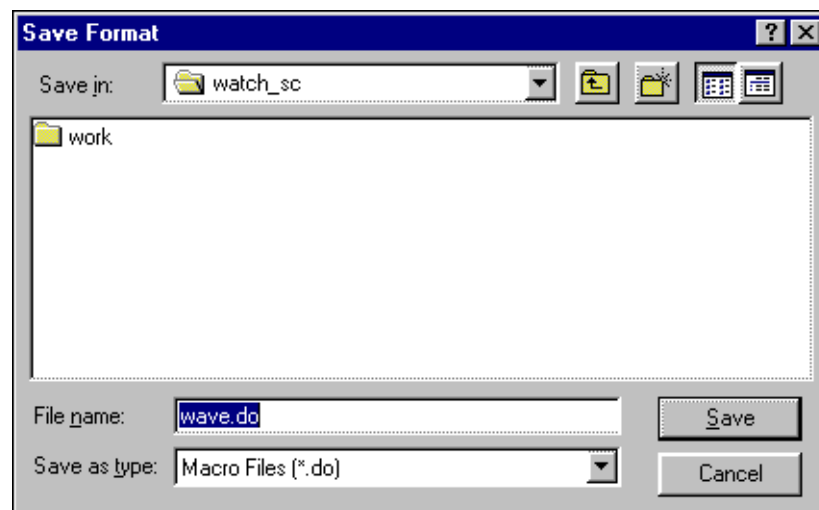


Figure 4-7: Save Format Dialog Box

3. Click **Save**.



# Design Implementation

---

This chapter contains the following sections.

- “Overview of Design Implementation”
- “Getting Started”
- “Creating an Implementation Project”
- “Specifying Options”
- “Translating the Design”
- “Using the Constraints Editor”
- “Using the Pin-out Area Constraints Editor (PACE)”
- “Mapping the Design”
- “Using Timing Analysis to Evaluate Block Delays After Mapping”
- “Placing and Routing the Design”
- “Using FPGA Editor to Verify the Place and Route”
- “Evaluating Post-Layout Timing”
- “Creating Configuration Data”
- “Creating a PROM File with iMPACT”

## 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 into 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”](#) and [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 Pin-out Area Constraints Editor (PACE).

## Getting Started

The tutorial design (Watch) emulates a track coach'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 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** twice.
6. Click **Finish**.
7. Go to the "Specifying Options" section.

### 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/tutorial5> directory into your newly created working directory.

Table 5-1: Required Tutorial Files

File Name	Description
<i>stopwatch.edn</i> , <i>stopwatch.edf</i> , or <i>stopwatch.ngc</i>	Input netlist file (EDIF)
<i>tenths.edn</i>	Counter netlist file (EDIF)
<i>stopwatch.ucf</i>	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 workstation, enter `ise &`
  - b. On a PC, select **Start** → **Programs** → **Xilinx ISE 5** → **Project Navigator**.

2. If you are continuing this project from the previous chapters, go to “[Specifying Options.](#)”
3. If you are using the pre-synthesized design, create a new project and add the (stopwatch) EDIF netlist as follows:
  - a. Select **File** → **New Project**.
  - b. Type **EDIF Flow** for the Project Name.
  - c. Select the following:
    - **Virtex2** for the Device Family
    - **xc2v40** for the Device
    - **-5** for the Speed Grad, **fg256** for the Package
    - **EDIF** for the Design Flow.
  - d. Click **OK**.

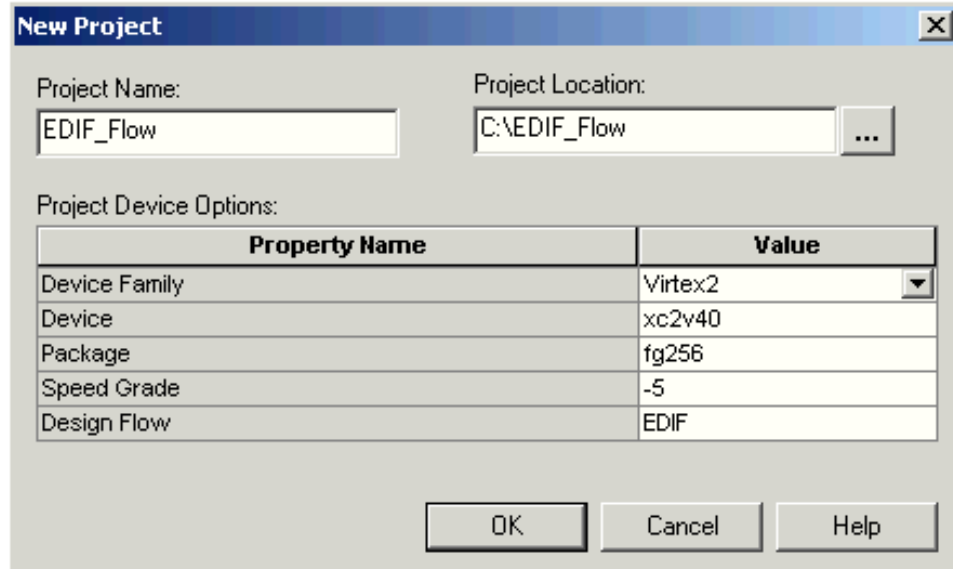


Figure 5-1: New Project Dialog Box

After the project is created:

1. Right-click **xc2v40-5fg256**.
2. Select **Add** → **Sources**.
3. Select *stopwatch.edf* or *stopwatch.edn*, and *stopwatch.ucf*.
4. Click **Open**.

When you create a new project, you 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.

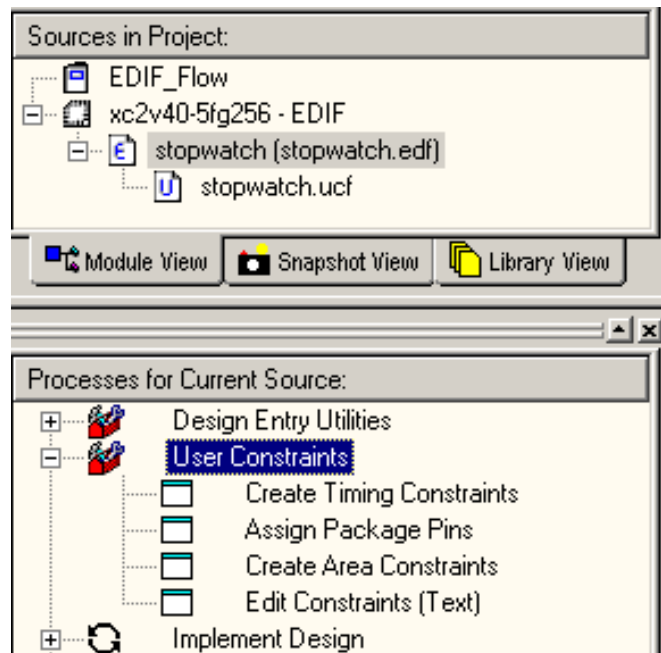


Figure 5-2: Selecting File in Sources in Project Window

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 how to set some options for implementation. The implementation options control how the software maps, places, routes, and optimizes a design.

The implementation options for ISE are divided into two groups, Standard and Advanced

- The default setting is Standard, which enables access to the most commonly used options.
- The Advanced settings provide access to all implementation options.

To enable the Advanced Options:

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**.

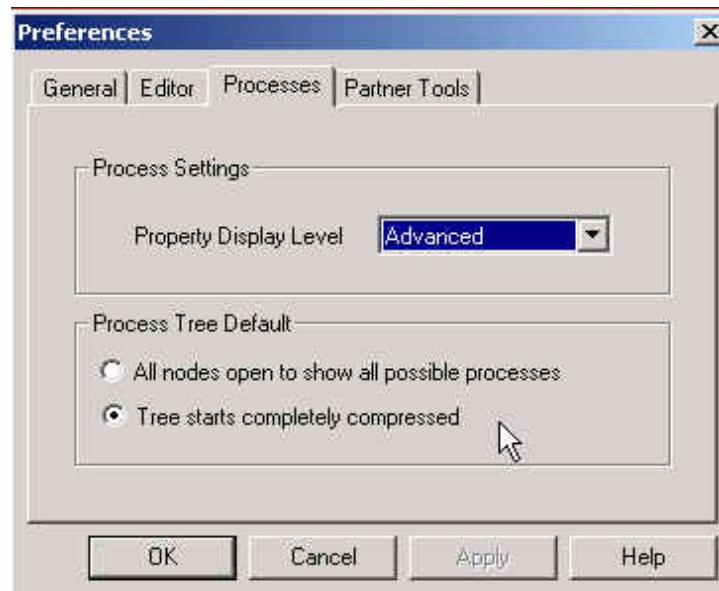


Figure 5-3: Preferences Dialog Box

To set more implementation options:

1. 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.

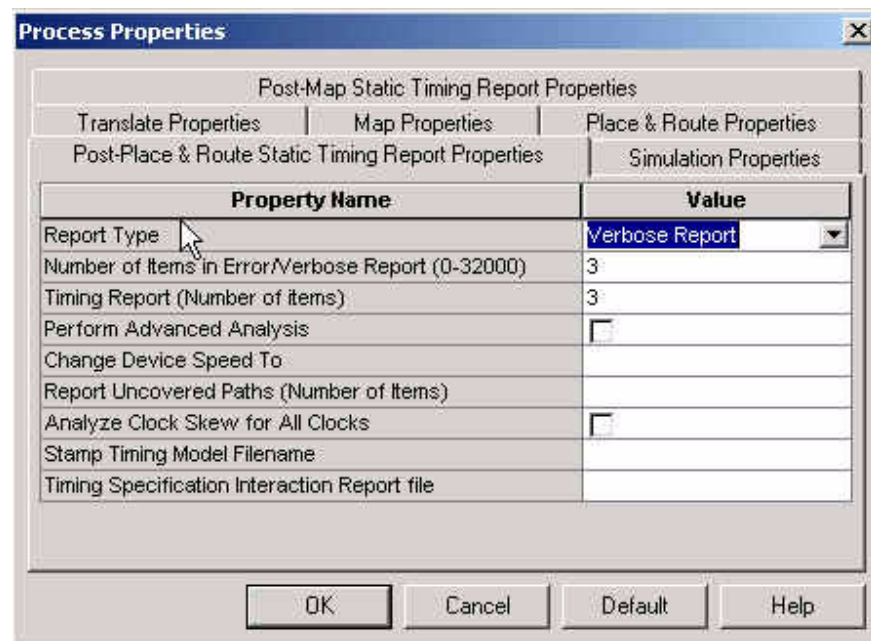


Figure 5-4: Post-Place & Route Static Timing Report Properties

- 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.

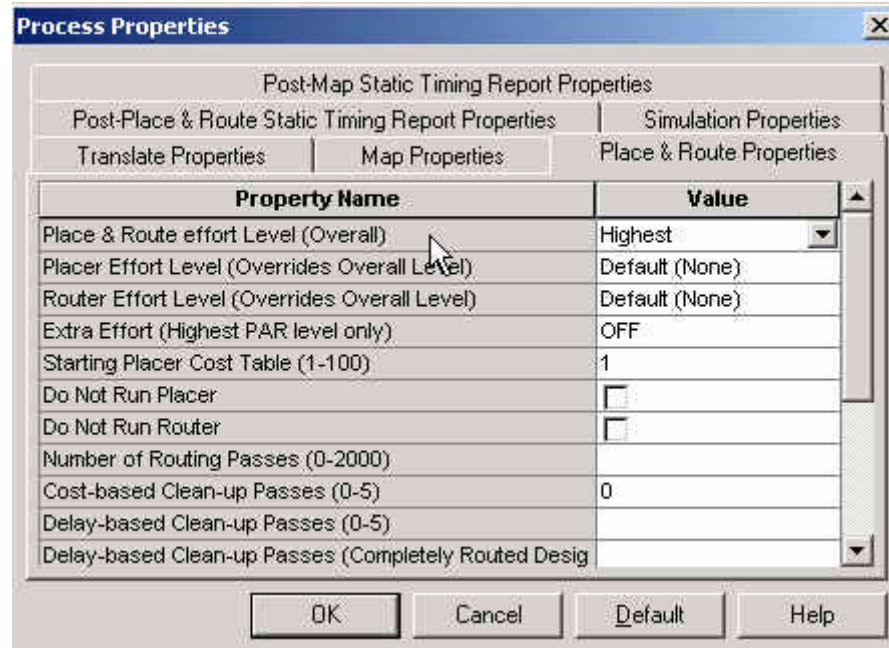


Figure 5-5: Place & Route Properties

- In the Place & Route Properties tab, change the Place & Route effort level (overall) to **Highest**. This option increases the overall effort level of Place and Route during implementation.
- Click **OK** to exit the Process Properties dialog box.

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 Pinout Area Constraints Editor are graphical tools that enables you to enter timing and pin location constraints.

To launch the Constraints Editor:

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

This automatically runs the Translate step, which is discussed in the following section.

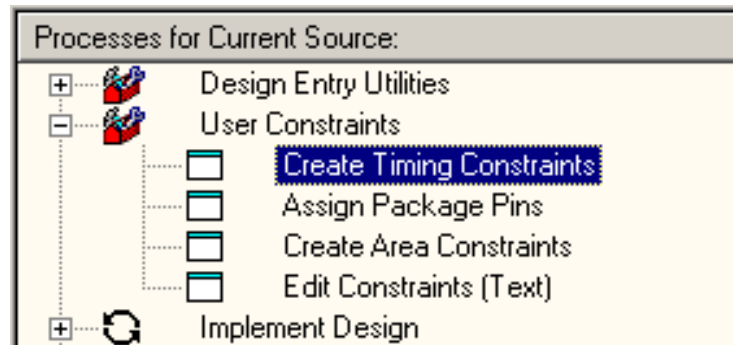


Figure 5-6: Edit Implementation Constraints

## Translating the Design

ISE manages the files created during implementation. The ISE tools use the settings you supply in the Options 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 Run. This tutorial illustrates the implementation one step at a time.

During translation, the program NGDBuild executes and 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 the User Constraints File (UCF) to the merged netlist.

Once these processes are complete, ISE launches the Constraints Editor.

## Using the Constraints Editor

The Constraints Editor enables you to:

- Edit constraints previously defined in a UCF file.
- Add new constraints to 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, along with 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.

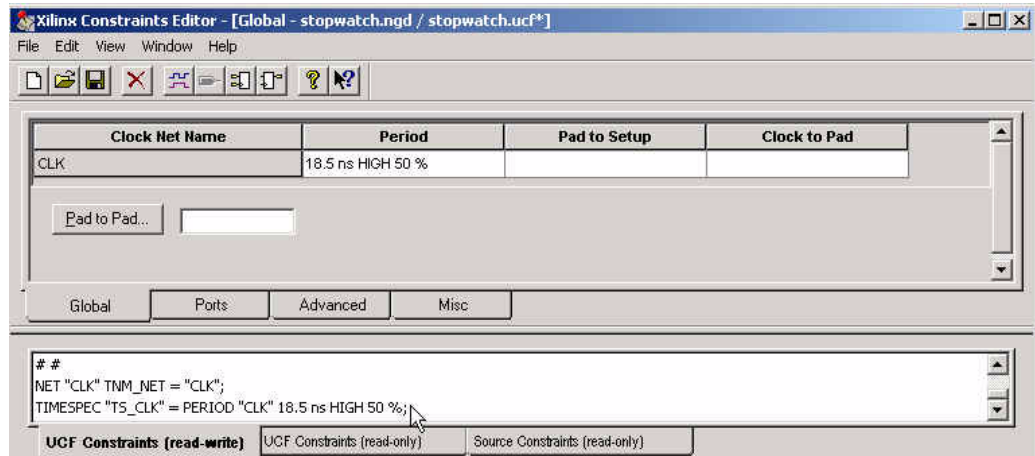


Figure 5-7: Constraints Editor

The Constraints Editor, edit the constraints as follows:

1. Select the Period cell on the row associated with the clock net CLK.
2. Double-click the left mouse button. This opens the Clock Period dialog box.
3. Within the Clock Signal Definition, keep the default (Specific Time) selected to define an explicit period for the clock rather than designate a period which is relative to another timing specification.
4. Enter a value of 18.5 in the Time text box.
5. Verify that ns is selected from the Units pull-down list.
6. 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 open a secondary dialog box by double-clicking a cell to specify your constraint values. Another feature is to do direct entry of constraints into cells by simply clicking once.

7. 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.
8. Select **OnesOut<0>** in the Port Name Column.
9. Hold the **Shift Key** and select **OnesOut<6>**.  
This selects the elements for creating a grouped offset.
10. In the Group Name text box, type *OnesOut\_grp* and click **Create Group**.  
This creates the group.



Port Name	Port Direction	Location	Pad to Setup	Clock to Pad
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
STRSTOP	INPUT			N/A

I/O Configuration Options
 
 Pad Groups  
 Group Name:

Figure 5-8: Selected Elements of a Grouped OFFSET

11. In the Select Group pulldown list, select the group you just created.
12. Click **Click to Pad**.

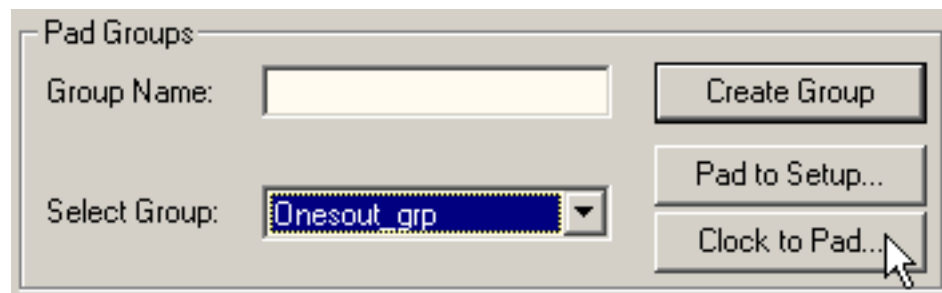


Figure 5-9: Selecting the Group that was created to use in an OFFSET

The Clock to Pad dialog box opens.

13. Enter 11.5 ns for the Timing Requirement.
14. Click **OK**.

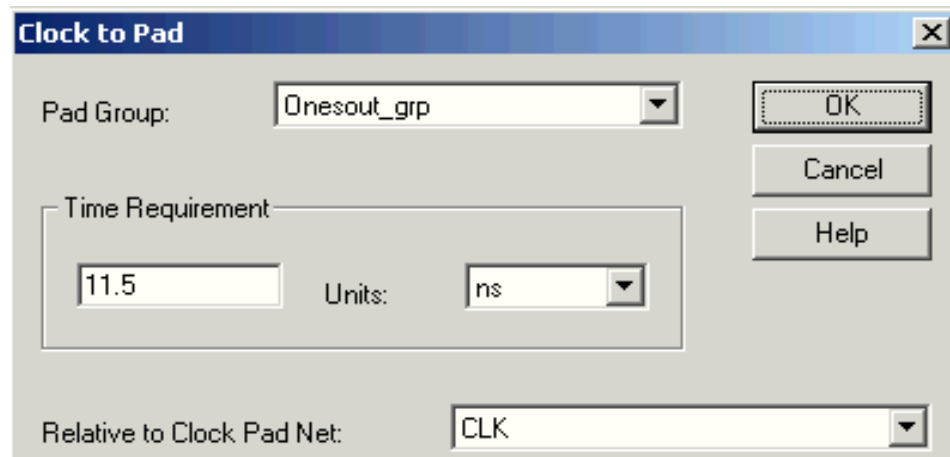


Figure 5-10: Clock to Pad Dialog

15. Select **File** → **Save**.

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

16. Select **File** → **Exit**.

## Using the Pin-out Area Constraints Editor (PACE)

Use the Pin-out 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, DCMs, GTs, and BUFGs.

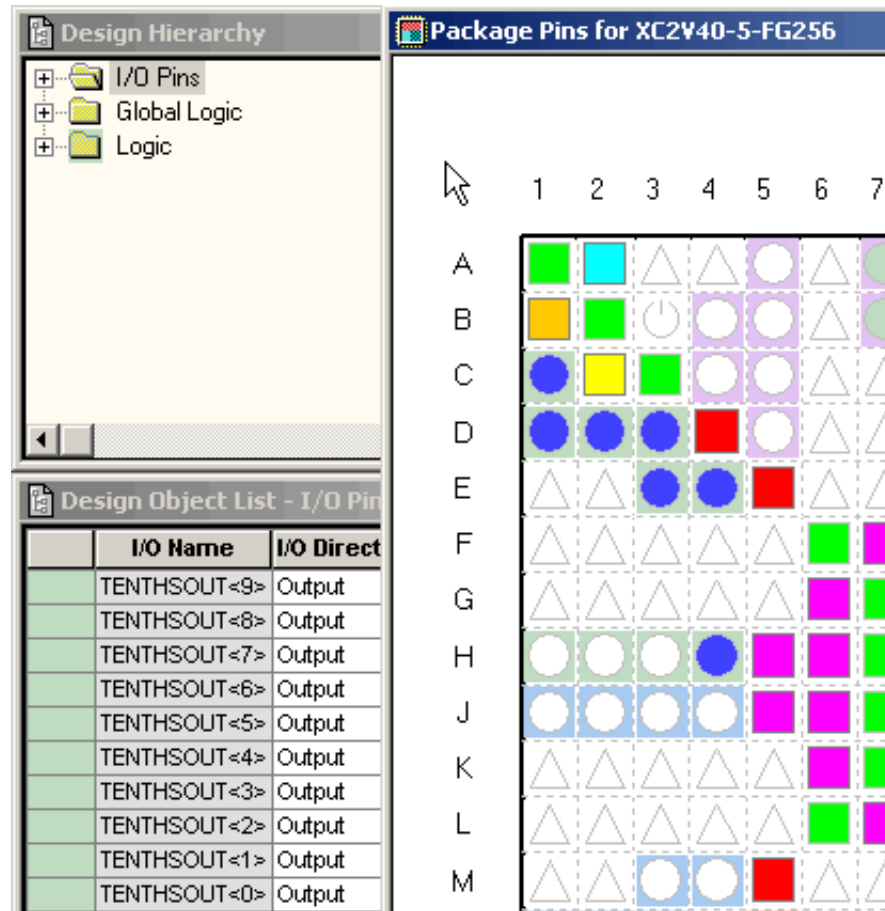
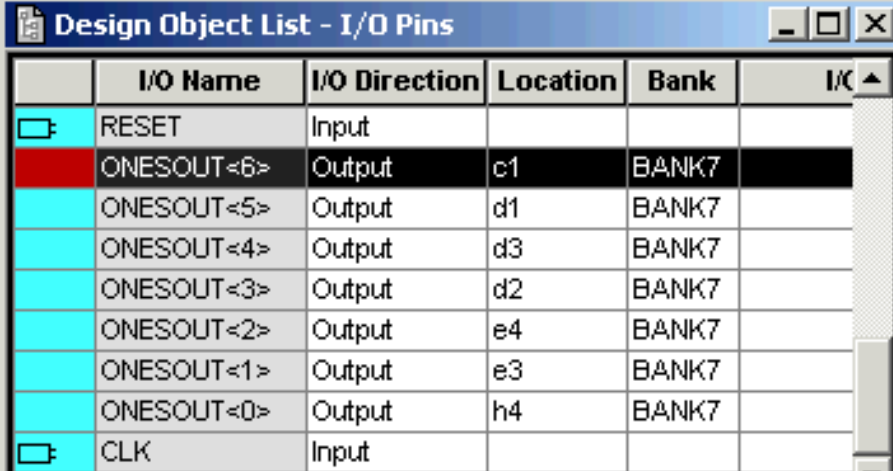


Figure 5-11: Pin Out Area Constraints Editor tool when launched

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

1. To launch PACE, select **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 IOs in the Design.
4. In the Design Object List window, scroll down until you find the Onesout nets.
5. To enter the pin locations, select the Pin Location text box 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












	I/O Name	I/O Direction	Location	Bank	I/C ▲
	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 5-12: Pin Locations Typed in PACE

To place some IOs in the Package Pin Window with the Drag and Drop functionality:

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

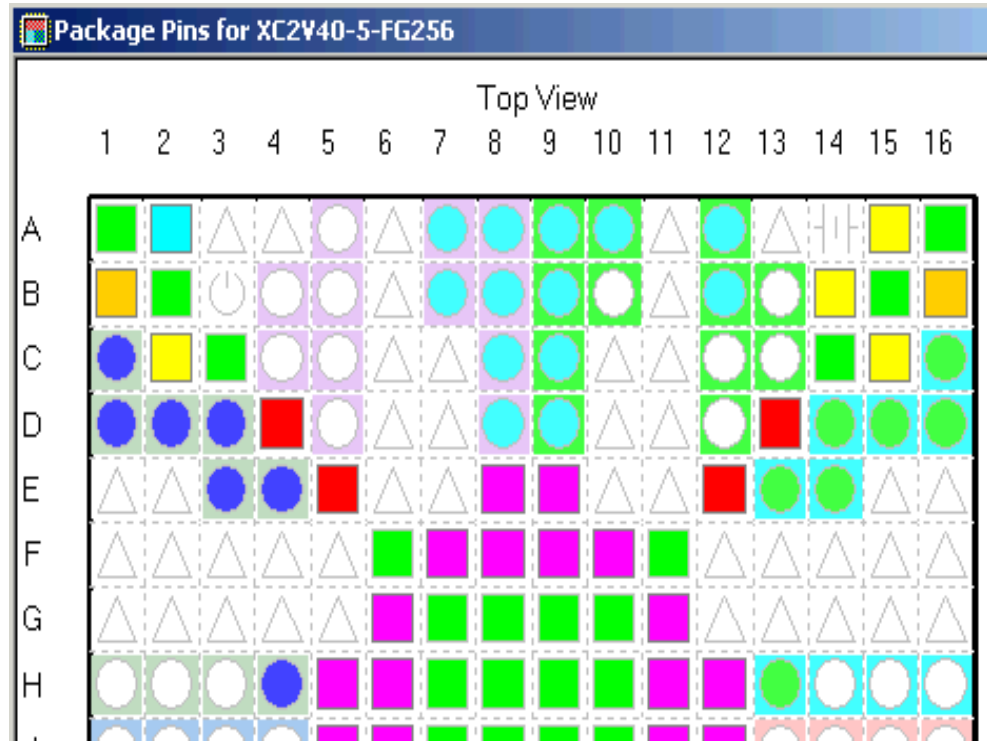


Figure 5-13: Drag and Drop IOs in the Package Pins Window

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. Select **File** → **Exit**.

## Mapping the Design

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

1. Right-click **Map**.
2. Select **Run** from the menu.
3. Expand the Implement Design hierarchy to see the progress through implementation.

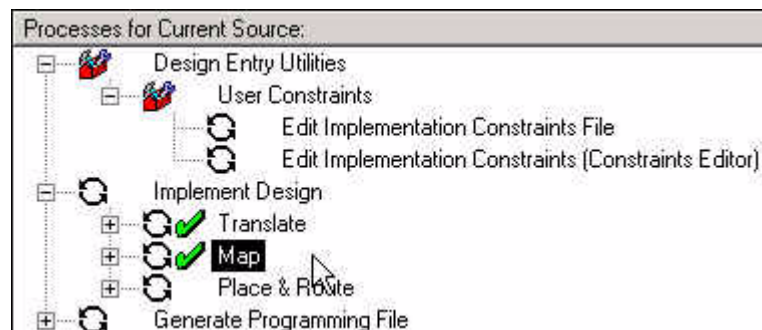


Figure 5-14: Mapping the Design

The design is being mapped into CLBs and IOBs. After mapping, the design is placed and routed. The final step in the design flow is Configure. In Configure, a configuration bitstream is created for downloading to a target device, or for formatting into a PROM programming file.

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 Through MAP

<b>Translation Report</b>	Includes warning and error messages from the translation process.
<b>Map Report</b>	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 <i>Development System Reference Guide</i> .

To view a report:

1. Expand the Translate or Map hierarchy.
2. Double-click a report.

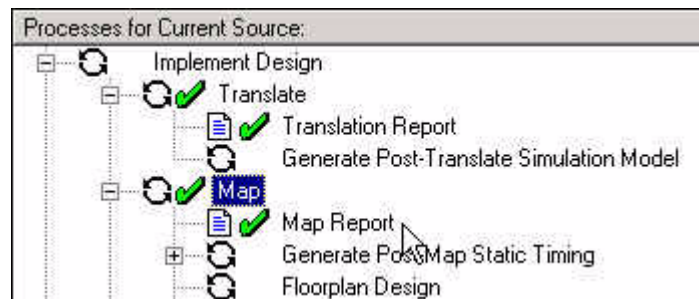


Figure 5-15: Translation Report and Map Report

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 that are provided are based on an optimal distance between blocks (also referred to as *unplaced floors*).

### 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. A rough guideline (known as the 50/50 rule) specifies that the block delays in any single path make up 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 PAR (Place And Route).

### Report Paths in Timing Constraints Option

Because timing constraints were defined for the design illustrated in this tutorial, the Report Paths in Timing Constraints option is selected. This option forces the Logic Level Timing Report to provide a period and path analysis on the constraints specified. The period timing constraint is listed on top, as is the minimum period obtained by the tools after mapping.

Because the report was limited to one path per timing constraint, you see a breakdown of a single path that contains 4 levels of logic. Notice the percentage of block (logic) delay versus routing delay for this calculation. The unplaced floors listed are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

If you do not generate a Logical Level Timing Report, PAR still processes a design based on the relationship between the block delays, floors, 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 floor 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). Use the Logic Level Timing Report to determine timing violations that may occur prior to running PAR.

To open the Logic Level Timing Report and review the PERIOD Constraints that were entered earlier:

1. Expand the Map hierarchy.
2. Double-click **Generate Post-Map Static Timing Report**.

- To open the Post-Map Static Timing Report, double-click **Post-Map Static Timing Report**. Timing Analyzer automatically launches and shows the report.

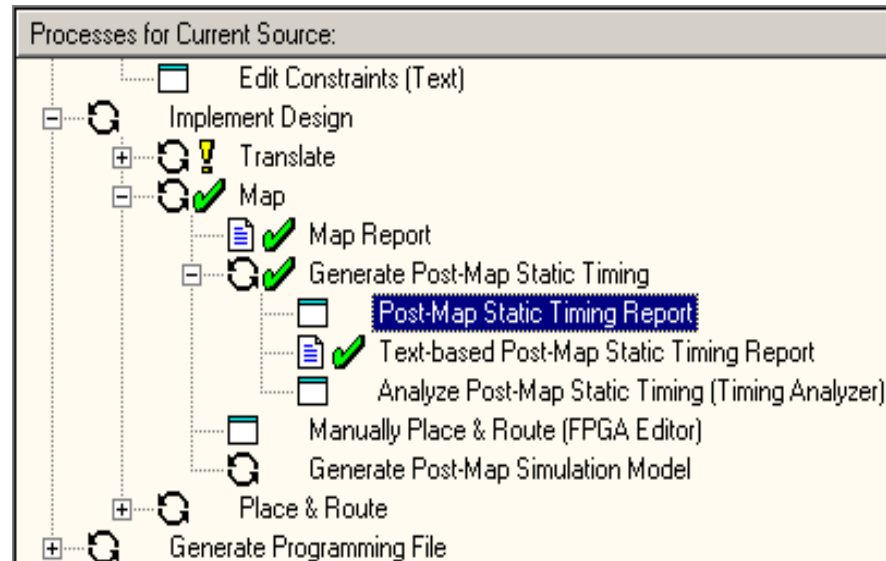


Figure 5-16: Post-Map Static Timing Report

- To exit Timing Analyzer, select **File** → **Exit**.

## 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.

Because timing constraints were specified for the design illustrated in this tutorial, PAR automatically performs timing driven placement and timing driven routing.

To run Place and Route 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:

- Expand the Place & Route hierarchy.
- Double-click the **Place & Route Report**.



*Table 5-3: Reports Generated by PAR*

Report	Description
<b>Place &amp; Route Report</b>	Provides a device utilization and delay summary. Use this report to verify that the design successfully routed and that all timing constraints were met.
<b>Pad Report</b>	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.
<b>Asynchronous Delay Report</b>	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:

1. To launch FPGA Editor in the expanded Place & Route hierarchy, double-click **View/Edit Routed Design (FPGA Editor)**.

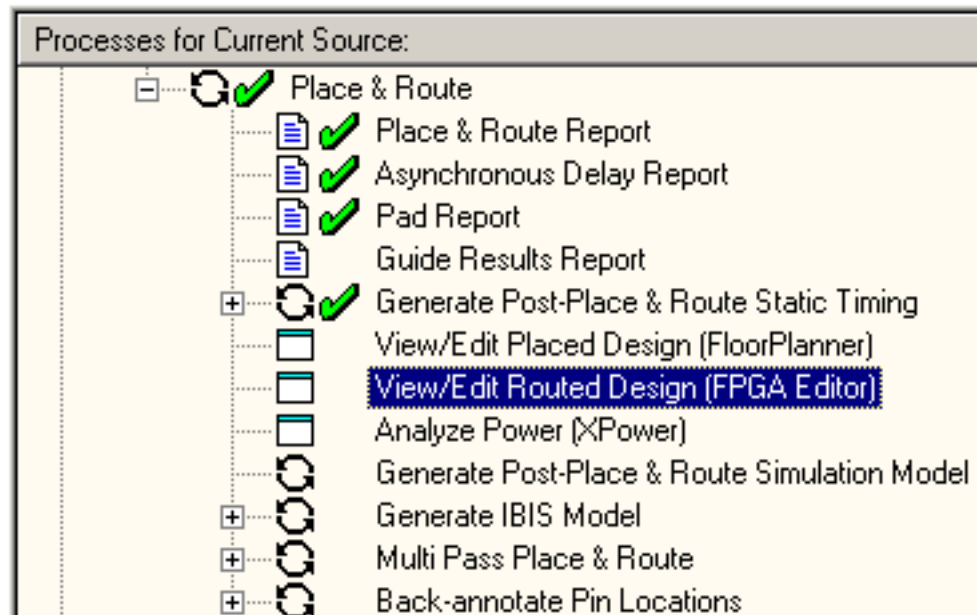


Figure 5-17: **View/Edit Routed Design (FPGA Editor) Process**

2. After FPGA Editor is open, change the List Window from All Components to All Nets. This enables you to view all of the possible nets in the design.

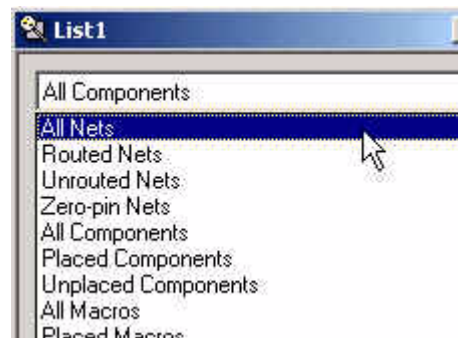


Figure 5-18: **List Window in FPGA Editor**

3. Select the clk\_dcm (Clock) net and see the fanout of the clock net.

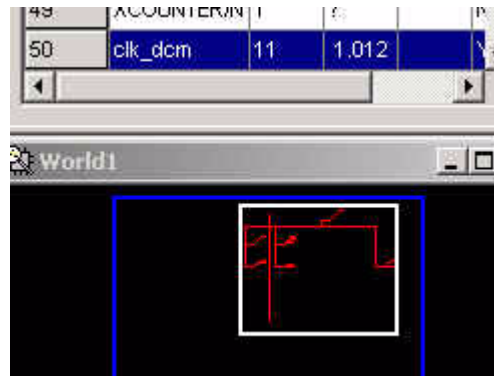


Figure 5-19: Clock Net

4. To exit FPGA Editor, select **File** → **Exit**.

## Evaluating Post-Layout Timing

After the design is placed and routed, 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).

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.

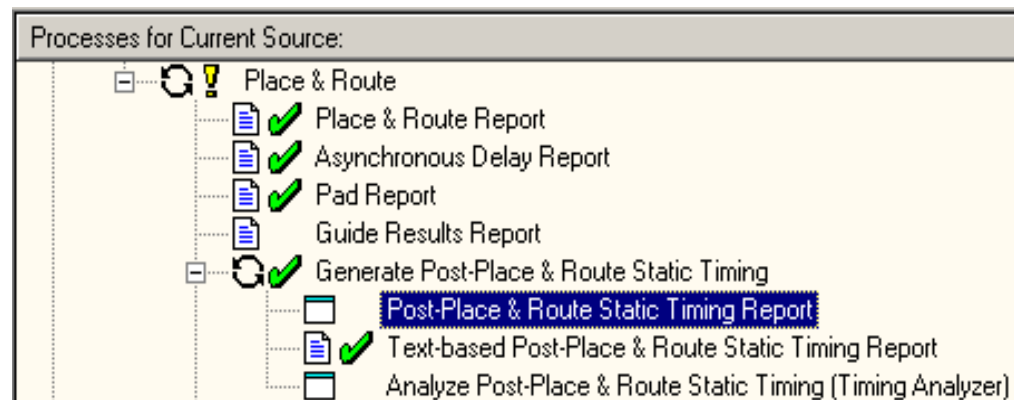


Figure 5-20: Post-Place & Route Static Timing Report

Following is a summary of this 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; a true report of net delays after the place and route step.

- ◆ 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 this 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. To exit Timing Analyzer, select **File** → **Exit**.

## Creating Configuration Data

After analyzing the design through timing constraints in Timing Analyzer, you need to create configuration data. To create a bitstream for the target device, run the Configure step:

1. Right-click Generate Programming File.
2. Select **Properties**.

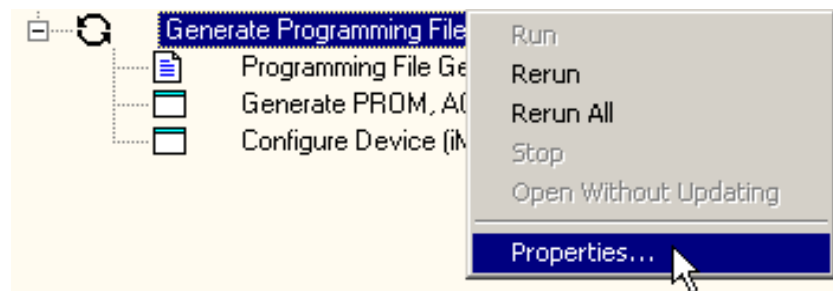


Figure 5-21: Selecting Properties

The Process Properties dialog box opens.

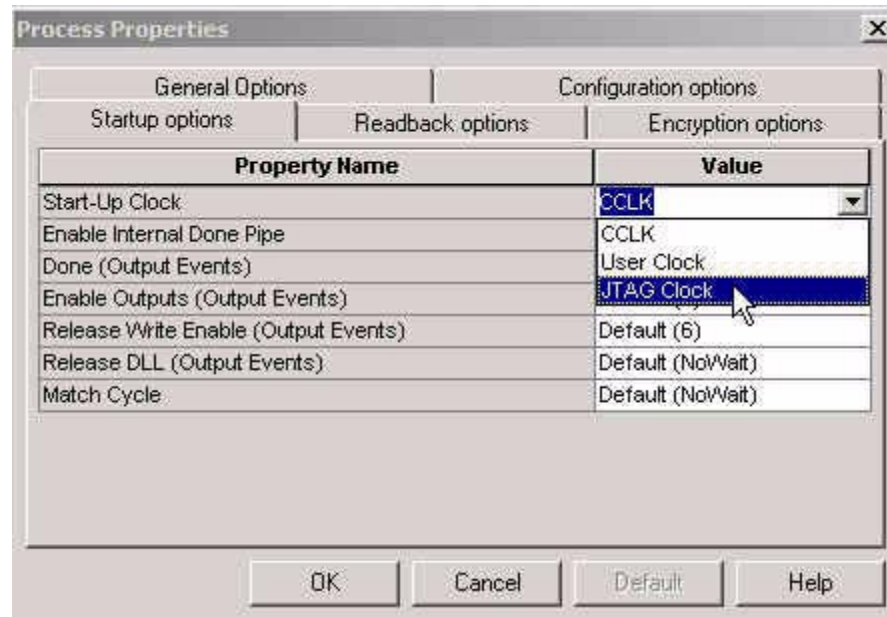


Figure 5-22: Process Properties' Startup Options Tab

3. Select the **Startup Options** tab.
4. Change the Startup Clock from CCLK to **JTAG**, since you are going to configure this device via Boundary Scan. This can remain at CCLK, if you were doing Select Map or Serial Slave configuration.
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* and *design\_name.ll* files (in this tutorial, the *watch.bit* and *watch.ll* files). The *design\_name.bit* file is the actual configuration data. The *design\_name.ll* file is the logical allocation file that is used during iMPACT to determine the location of the probable points in the design. These files are automatically copied to your working directory.

8. Verify that these files are in this directory. The *design\_name.ll* file is used to perform device readback with the iMPACT tool. For more information, see iMPACT online help.
9. To review the Programming File Generation Report, double-click the report. Verify that the specified options were used when creating the configuration data.

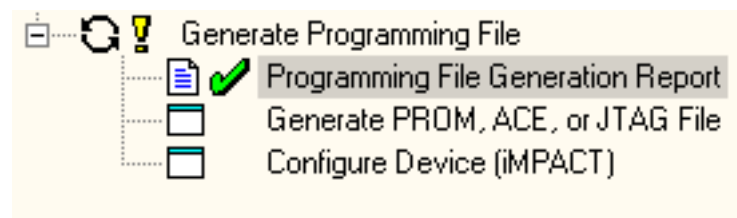


Figure 5-23: Programming File Generation Report

## Creating a PROM File with iMPACT

To program a single device using iMPACT, all you need is a *design.bit* file. To program several devices in a daisy chain configuration, or to program your devices using a PROM, you must use iMPACT to create a PROM file. iMPACT accepts any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations.

To start iMPACT:

1. Double-click **Generate PROM, ACE, JTAG File**.
2. iMPACT opens with a wizard to help you create the PROM File.

Use the wizard 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.

To create a PROM file with iMPACT:

1. Open iMPACT.
2. In the Operation Mode Selection dialog box, select **Prepare Configuration Files**.
3. Click **Next**.

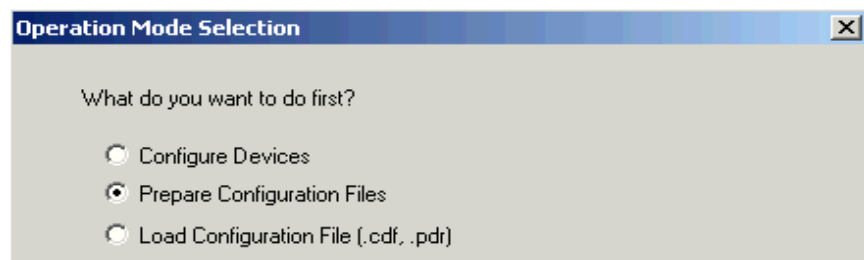


Figure 5-24: Operation Mode Selection dialog box

4. In the Prepare PROM Files dialog box, under “I want to create a:”, select **PROM File**.
5. Click **Next**.

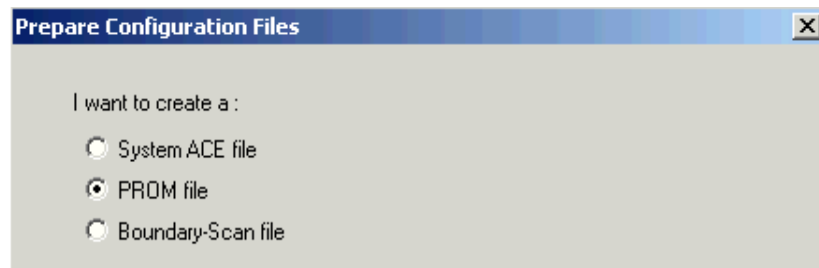


Figure 5-25: Prepare Configuration Files Dialog

6. In the “I want to target a:” dialog box:
  - a. Under PROM File Format, select **MCS**.
  - b. Under PROM File Name, type ‘stopwatch1’.
7. Click **Next**.

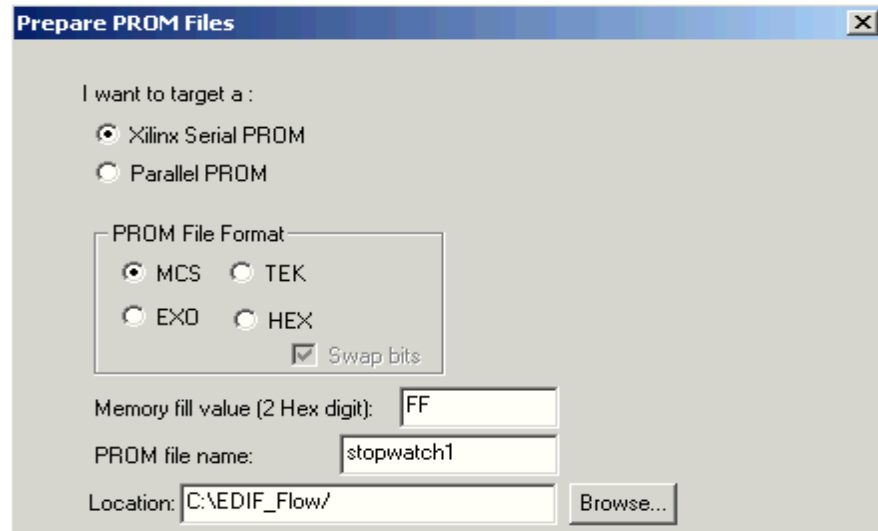


Figure 5-26: Prepare PROM Files Dialog

8. In the Specify Xilinx Serial PROM Device dialog box, check the box associated with **Auto Select PROM**.
9. Click **Next**.
10. 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.

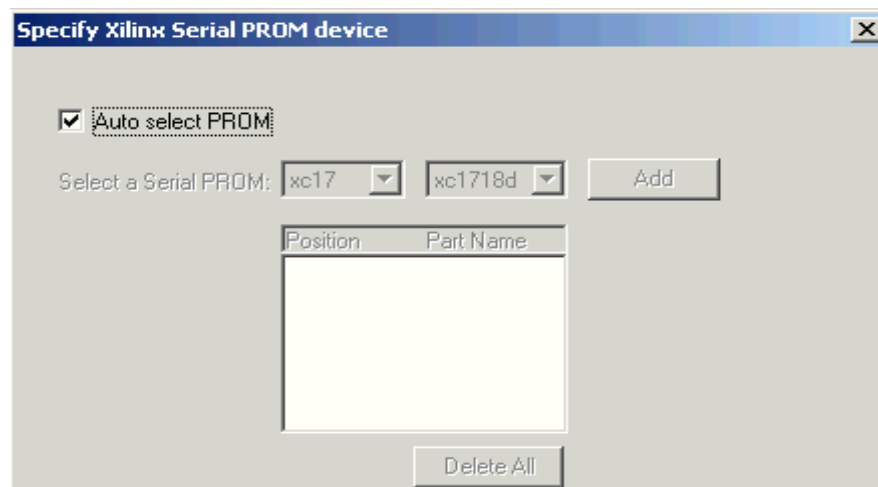


Figure 5-27: Specify Xilinx Serial PROM Device Dialog Box

11. In the File Generation Summary dialog box, click **Next**.

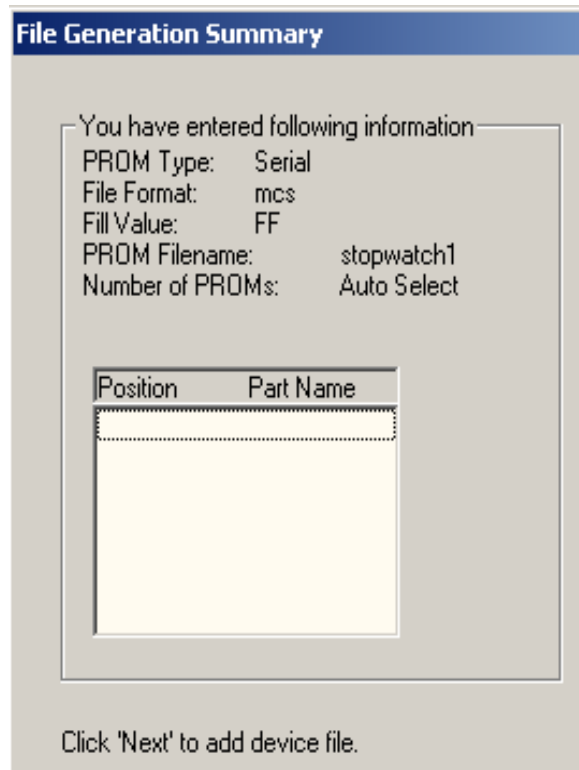


Figure 5-28: File Generation Summary Dialog Box

12. In the Add Device File dialog box:
  - a. Click **Add Device**.
  - a. Select the *stopwatch.bit* file.

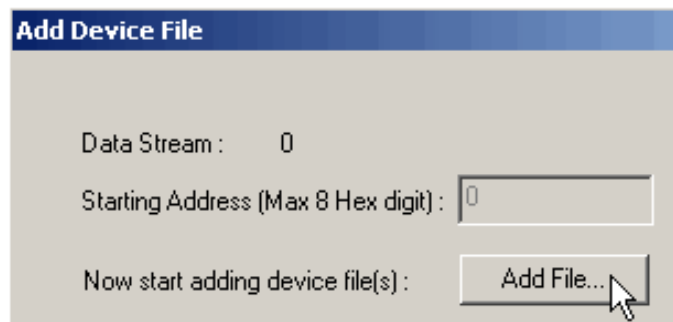


Figure 5-29: Add Device File Dialog Box

13. Select **No** when you are asked if you would like to add another design file to the datastream.
14. Select **Finish**.  
iMPACT displays the PROM associated with your bit file.
15. When asked to generate a file now, select **Yes**. This creates the PROM file.



16. Select **File** → **Save** to save the supporting files associated with iMPACT.
17. Select **File** → **Exit** to close iMPACT.

This completes this chapter of the tutorial. For more information on this design flow and implementation methodologies (especially some of the tools and programs that were not covered), see the *iMPACT User Guide*, available in the collection of software manuals on the web, at [http://support.xilinx.com/support/sw\\_manuals/xilinx5/](http://support.xilinx.com/support/sw_manuals/xilinx5/).



# Timing Simulation

---

This chapter includes the following sections.

- “Overview of Timing Simulation Flow”
- “Getting Started”
- “Starting ModelSim”
- “Adding Signals”
- “Saving the Simulation”

## 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, also known as back-annotated simulation, uses the detailed timing and design layout information that is available after place and route to create a VHDL or Verilog simulation netlist. 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 5.x, you must have ModelSim release 5.4 or above installed on your machine to follow the tutorial.

**Note:** ModelSim 5.6a XE is used for the examples in this tutorial.

### Required Files

The timing simulation flow requires the following files:

- **Design Files (VHDL or Verilog)**  
The design file is produced by the Xilinx software.

- **Stimulus File (VHDL or Verilog)**

This is also known as the testbench. You can use the same testbench for functional simulation as well as timing simulation. Also, you can create the testbench with Xilinx HDL Bencher—See [Chapter 4, “Behavioral Simulation”](#) for information on this flow.
- **ModelSim Script File (Optional)**

The script file (.do) automates the simulation to a large extent, and makes it easy to re-run the simulation. Alternatively, the commands are entered one-by-one into the simulator. Xilinx ISE creates the script file needed to run simulation in ModelSim.
- **Xilinx Simulation Libraries**

For timing simulation, the SIMPRIMS HDL simulation library must be used. Details about this library are provided in the following section.

## Xilinx Simulation Libraries

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 located in the SIMPRIM library. The recommended mapping name for the VHDL SIMPRIM library is SIMPRIM, and for the Verilog SIMPRIM library it is SIMPRIMS\_VER.

**Note:** If using ModelSim Xilinx Edition, there is *no need to compile the models*. MXE (ModelSim Xilinx Edition) comes with the models precompiled.

For detailed instructions on compiling these libraries, see Xilinx Answer Record # 2561, which can be accessed as follows:

1. Go to <http://support.xilinx.com>.
2. Enter **2561** in the search box, and check to see that the search engine is pointing to **Answer Records**.
3. Click **OK**.
4. Click the link to Answer Record # 2561.

## Starting ModelSim

Xilinx ISE is fully integrated with any version of the ModelSim Simulator. ISE provides work directory creation, source file compilation, simulation initialization, and simulation property control.

## Specifying Simulation Process Properties

To set the simulation process properties:

1. In the Sources in Project window, select *stopwatch\_tb.tbw*.
2. Click the + to expand the ModelSim Simulator hierarchy.
3. Select and right-click **Simulate Post-Place & Route VHDL (Verilog) Model**.
4. Select **Properties**.

The following properties are available and can be edited from the pop-up GUI (as shown in [Figure 6-1](#) and [Figure 6-2](#)):

## Simulation Properties

Options available in this tab are:

- **Custom Do File**

This option enables users to select a user-created .do file.

- **Use Automatic Do File**

If this option is unchecked, ModelSim will start but will not run the processes required to simulate the design. You must manually run the .do file from ModelSim, or enter the commands one by one to run simulation.

- **Simulation Run Time**

Specifies default time for which simulation is run. The “-all” setting tells ModelSim to run the simulation until it encounters a break specified in the testbench. Alternatively, a setting of “1000ns” can be entered so that the simulation only runs for 1000ns.

- **Simulation Resolution**

This is set to 1 ps by default. All Xilinx simulations should be run with the resolution set to 1 ps.

- **Simulation Mode**

For most devices, Maximum delay is the only available option.

**Note:** Not all the devices have Minimum and Typical delays specified, so there may not be a difference in the timing numbers, as they will be the same as maximum numbers.

- **VHDL Syntax**

Set this option to either 93 or 87, which tells the compiler to use either VHDL-87 or VHDL-93 syntax.

- **Use Explicit Declarations Only**

If this option is unchecked, the compiler will not resolve ambiguous overloads.

- **Design Unit Name**

This property enables you to specify the top-level model to be loaded in ModelSim. This property must be changed if the testbench, module, or entity is named something different than testbench.

- **Generate VCD File**

This will generate the VCD file from ModelSim so that it can be loaded into XPower.

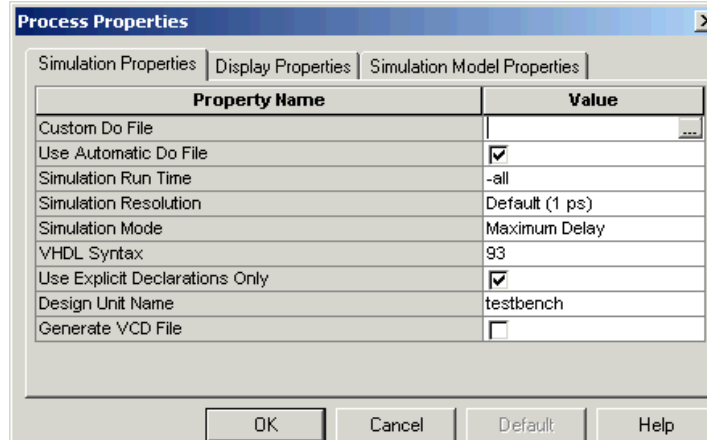


Figure 6-1: Simulation Properties

## Display Properties

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 Model Properties

Options available in this tab are as follows:

- **Correlate Simulation Data to Input Design**  
By selecting this property, you instruct the Xilinx post-place and route netlist generation tools to append the timing details to the input design (post NGDBUILD design). The advantage of using this option is that the user-defined signal names are preserved. The disadvantage is that the user design, rather than the physical (post-place and route) design, is used. Therefore, if there are any errors introduced by the place and route tools, they are not detected.
- **Bring Out Global Set/Reset Net as a Port**  
Use this option to create an external port in the simulation netlist that will enable you to control the power-on-reset from a port.
- **Global Set/Reset Port Name**  
Default is GSR.
- **Bring Out Global Tristate Net as a Port**
- **Global Tristate Port Name**  
Default is GTS.
- **Generate Test Fixture/Testbench File**  
Use this option to create a testbench.

The following options are available when the **Advanced** Process Setting is enabled in Project Navigator (accessed under **Edit** → **Preferences**, in the **Processes** tab).

- **Retain Hierarchy**
- **Change Device Speed To**  
This provides the option to vary the speed grade of the device for simulation purposes.
- **TOC Pulse Width**  
Use this option to set the duration of the Global Tristate on Configuration Pulse Width. The default is 0ns.
- **Global Disable of X-Generation for Simulation**  
Setting this option will disable the 'X' that gets propagated through the simulation when there is a timing violation.
- **ROC Pulse Width**  
Use this option to set the duration of the Global Reset on Configuration Pulse Width. The default is 100 ns.
- **TOC Pulse Width**  
Use this option to set the duration of the Global Tristate on Configuration Pulse Width. The default is 1 ns.

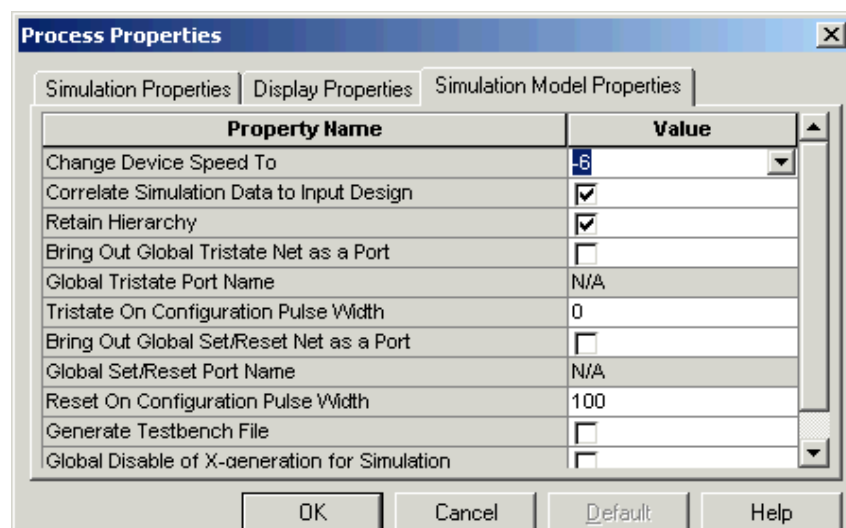


Figure 6-2: Simulation Model Properties

## Performing Simulation

Once you have set the process properties, ModelSim can be invoked. To start the timing simulation, double-click **Simulate Post-Place and Route VHDL Model** or **Simulate Post-Place and Route Verilog Model**.

ModelSim creates the working directory, compiles the source files, loads the design, and runs simulation for the time specified.

## Adding Signals

To view signals during simulation, you must first add them to the Wave window. Project Navigator automatically adds all of 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.
- Select **Add** → **Wave** → **Selected Signals** from the Signal Window.

The following procedure explains how to add additional signals in the design hierarchy. For this tutorial, add the `smallcnt` output flip-flops.

1. In ModelSim in the Structure window, click the + next to `uut:stopwatch(structure)`.
  - ◆ `uut` is the instance in the testbench.
  - ◆ `stopwatch` is the component/entity name.
  - ◆ `structure` is the architecture name.

**Note:** Figure 6-3 shows the Structure Window for the Verilog flow. The VHDL and Schematic flows may produce different results.

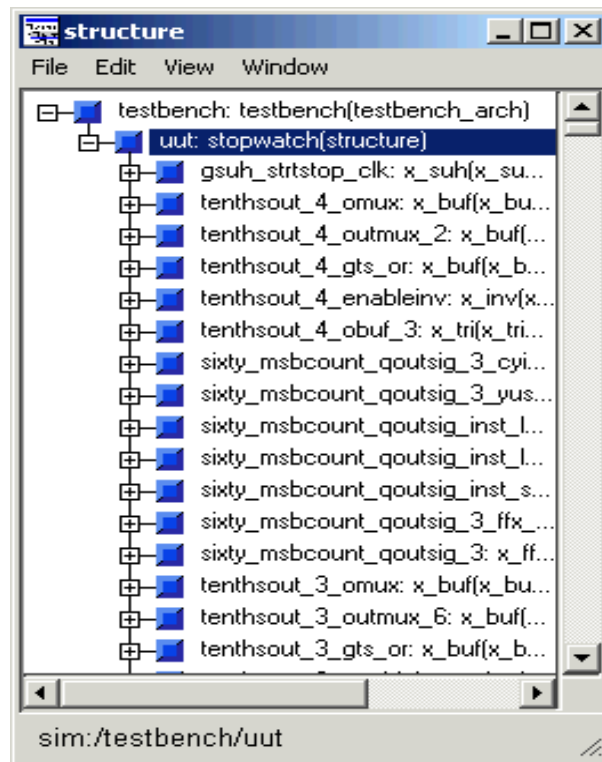


Figure 6-3: Structure Window - Verilog flow example

2. Select `sixty_lsbcount_QOUT<0>` in the signal window.
3. Click and drag `sixty_lsbcount_QOUT<0>` from the Signal window to the Wave window.
4. Select `sixty_lsbcount_QOUT<1>` 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 these signals.

When new signals are added to the waveform window, the simulation needs to be re-run.

To restart and re-run the simulation in ModelSim:

1. Click the Restart Simulation icon.



Figure 6-4: Restart Simulation Icon

The Restart dialog box opens.

2. Click **Restart**.
3. Click the Run-all or Run icon to re-run the simulation.



Figure 6-5: Run-All Icon

## Saving the Simulation

The ModelSim Simulator gives you the ability to save the signals list in the Wave window. The signals in this list are then opened and referenced each time the simulation is started.

To save signals to the Wave window:

1. In the Wave window, select **File** → **Save Format**.
2. In the Save Format dialog box, change *wave.do* to *sec\_signal.do*.



Figure 6-6: Save Format Dialog Box

3. Click **Save** to close the dialog box and close the ModelSim simulator.

