

ISE In-Depth Tutorial

9.1



Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design 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 the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

Copyright © 1995-2007 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

About This Tutorial

About the In-Depth Tutorial

This tutorial gives a description of the features and additions to Xilinx® ISE™ 9.1i. 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 the three tutorial flows available in this document. For information about the tutorial flows, see [“Tutorial Flows.”](#)

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, CORE Generator™, and ISE Text 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.
- **Chapter 7, “iMPACT Tutorial”** explains how to program a device with a newly created design using the IMPACT configuration tool.

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 although behavioral simulation is optional, it is strongly recommended in this tutorial flow.
- **Chapter 5, “Design Implementation”**
- **Chapter 6, “Timing Simulation”**
Note that although timing simulation is optional, it is strongly recommended in this tutorial flow.
- **Chapter 7, “iMPACT Tutorial”**

Schematic Design Flow

The Schematic Design flow is as follows:

- **Chapter 3, “Schematic-Based Design”**
- **Chapter 4, “Behavioral Simulation”**
Note that although behavioral simulation is optional, it is strongly recommended in this tutorial flow.
- **Chapter 5, “Design Implementation”**
- **Chapter 6, “Timing Simulation”**
Note that although timing simulation is optional, it is strongly recommended.
- **Chapter 7, “iMPACT Tutorial”**

Implementation-only Flow

The Implementation-only flow is as follows:

- **Chapter 5, “Design Implementation”**
- **Chapter 6, “Timing Simulation”**
Note that although timing simulation is optional, it is strongly recommended in this tutorial flow.
- **Chapter 7, “iMPACT Tutorial”**

Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/literature>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support>.

Table of Contents

Preface: About This Tutorial

About the In-Depth Tutorial	3
Tutorial Contents	3
Tutorial Flows	4
HDL Design Flow	4
Schematic Design Flow	4
Implementation-only Flow	4
Additional Resources	5

Chapter 1: Overview of ISE and Synthesis Tools

Overview of ISE	13
Project Navigator Interface	13
Sources Window	14
Sources Tab	14
Snapshots Tab	15
Libraries Tab	15
Processes Window	15
Processes Tab	15
Transcript Window	16
Error Navigation to Source	16
Error Navigation to Answer Record	16
Workspace	17
Design Summary	17
Text Editor	17
ISE Simulator / Waveform Editor	17
Schematic Editor	17
Using Revision Control Features	17
Using Snapshots	17
Creating a Snapshot	17
Restoring a Snapshot	17
Viewing a Snapshot	18
Using Project Archives	18
Creating an Archive	18
Restoring an Archive	18
Using Export/Import Source Control	18
Exporting a Project	18
Importing a Project	19
Overview of Synthesis Tools	19
Precision Synthesis	19
Process Properties	19
Synplify/Synplify Pro	20
Process Properties	20
Xilinx Synthesis Technology (XST)	20
Process Properties	20

Chapter 2: HDL-Based Design

Overview of HDL-Based Design	23
Getting Started	24
Required Software	24
Optional Software Requirements	24
VHDL or Verilog?	24
Installing the Tutorial Project Files	24
Starting the ISE Software	25
Creating a New Project	25
Creating a New Project: Using the New Project Wizard.....	25
Creating a New Project: Using a Tcl Script	27
Stopping the Tutorial.....	27
Design Description	28
Inputs	28
Outputs	28
Functional Blocks	28
Design Entry	29
Adding Source Files.....	30
Checking the Syntax	30
Correcting HDL Errors	31
Creating an HDL-Based Module	32
Using the New Source Wizard and ISE Text Editor	32
Using the Language Templates.....	34
Adding a Language Template to Your File.....	35
Creating a CORE Generator Module	36
Creating a CORE Generator Module	36
Instantiating the CORE Generator Module in the HDL Code.....	38
Creating a DCM Module.....	39
Using the Clocking Wizard.....	39
Instantiating the dcm1 Macro - VHDL Design	41
Instantiating the dcm1 Macro - Verilog	42
Synthesizing the Design	43
Synthesizing the Design using XST.....	44
Entering Constraints	44
Entering Synthesis Options.....	45
Synthesizing the Design	45
The RTL / Technology Viewer.....	45
Synthesizing the Design using Synplify/Synplify Pro	47
Examining Synthesis Results	47
Synthesizing the Design Using Precision Synthesis	48
Entering Synthesis Options through ISE.....	49
The RTL/Technology Viewer.....	49

Chapter 3: Schematic-Based Design

Overview of Schematic-Based Design	51
Getting Started	51
Required Software	51
Installing the Tutorial Project Files	52
Starting the ISE Software	52
Creating a New Project	52
Creating a New Project: Using New Project Wizard.....	53

Creating a New Project: Using a Tcl Script	54
Stopping the Tutorial	54
Design Description	54
Inputs	55
Outputs	56
Functional Blocks	56
Design Entry	57
Opening the Schematic File in the Xilinx Schematic Editor	57
Manipulating the Window View	58
Creating a Schematic-Based Macro	58
Defining the time_cnt Schematic	59
Adding I/O Markers	60
Adding Components to time_cnt	60
Correcting Mistakes	63
Drawing Wires	63
Adding Buses	63
Adding Bus Taps	64
Adding Net Names	65
Checking the Schematic	66
Saving the Schematic	67
Creating and Placing the time_cnt Symbol	67
Creating the time_cnt symbol	67
Placing the time_cnt Symbol	67
Creating a CORE Generator Module	68
Creating a CORE Generator Module	68
Creating a State Machine Module	70
Adding New States	71
Adding a Transition	72
Adding a State Action	73
Adding a State Machine Reset Condition	75
Creating the State Machine HDL output file	75
Creating the State Machine Symbol	76
Creating a DCM Module	76
Using the Clocking Wizard	76
Creating the dcm1 Symbol	78
Creating an HDL-Based Module	78
Using the New Source Wizard and ISE Text Editor	78
Using the Language Templates	81
Adding a Language Template to Your File	82
Creating the debounce Symbol	83
Placing the statmach, timer_preset, dcm1 and debounce Symbols	83
Changing Instance Names	84
Hierarchy Push/Pop	85
Specifying Device Inputs/Outputs	85
Adding Input Pins	85
Adding I/O Markers and Net Names	86
Assigning Pin Locations	87
Completing the Schematic	87

Chapter 4: Behavioral Simulation

Overview of Behavioral Simulation Flow	91
ModelSim Setup	91

ModelSim PE and SE	91
ModelSim Xilinx Edition	92
ISE Simulator Setup	92
Getting Started	92
Required Files	92
Design Files (VHDL, Verilog, or Schematic)	92
Test Bench File	92
Xilinx Simulation Libraries	92
Xilinx Simulation Libraries	92
Updating the Xilinx Simulation Libraries	93
Mapping Simulation Libraries in the Modelsim.ini File	93
Adding an HDL Test Bench	94
Adding Tutorial Test Bench File	94
VHDL Design	94
Verilog Design	95
Behavioral Simulation Using ModelSim	95
Locating the Simulation Processes	95
Specifying Simulation Properties	96
Performing Simulation	97
Adding Signals	97
Adding Dividers	99
Rerunning Simulation	99
Analyzing the Signals	100
Saving the Simulation	101
Behavioral Simulation Using ISE Simulator	101
Locating the Simulation Processes	101
Specifying Simulation Properties	102
Performing Simulation	103
Adding Signals	103
Rerunning Simulation	104
Analyzing the Signals	105
Creating a Test Bench Waveform Using the Waveform Editor	105
Creating a Test Bench Waveform Source	105
Applying Stimulus	107

Chapter 5: Design Implementation

Overview of Design Implementation	109
Getting Started	110
Continuing from Design Entry	110
Starting from Design Implementation	110
Specifying Options	111
Creating Partitions	113
Creating Timing Constraints	114
Translating the Design	115
Using the Constraints Editor	116
Using the Floorplan Editor	120
Mapping the Design	123
Using Timing Analysis to Evaluate Block Delays After Mapping	126
Estimating Timing Goals with the 50/50 Rule	126

Report Paths in Timing Constraints Option	126
Placing and Routing the Design	127
Using FPGA Editor to Verify the Place and Route	129
Evaluating Post-Layout Timing	131
Changing HDL with Partition	132
Creating Configuration Data	134
Creating a PROM File with iMPACT	135
Command Line Implementation	138

Chapter 6: Timing Simulation

Overview of Timing Simulation Flow	139
Getting Started	139
Required Software	139
Required Files	140
Specifying a Simulator	140
Timing Simulation Using ModelSim	140
Specifying Simulation Process Properties	141
Performing Simulation	143
Adding Signals	143
Adding Dividers	145
Rerunning Simulation	146
Analyzing the Signals	146
Saving the Simulation	147
Timing Simulation Using Xilinx ISE Simulator	148
Specifying Simulation Process Properties	148
Performing Simulation	149
Adding Signals	149
Viewing Full Signal Names	150
Rerunning Simulation	150
Analyzing the Signals	151

Chapter 7: iMPACT Tutorial

Device Support	153
Download Cable Support	154
Parallel Cable IV	154
Platform Cable USB	154
MultiPRO Cable	154
Configuration Mode Support	154
Getting Started	154
Generating the Configuration Files	154
Connecting the Cable	155
Starting the Software	155
Opening iMPACT from Project Navigator	155
Opening iMPACT stand-alone	155
Creating a iMPACT New Project File	156
Using Boundary Scan Configuration Mode	156
Specifying Boundary Scan Configuration Mode	156
Assigning Configuration Files	158

Saving the Project File	159
Editing Preferences	159
Performing Boundary Scan Operations	159
Troubleshooting Boundary Scan Configuration	162
Verifying Cable Connection	162
Verifying Chain Setup	163
Creating an SVF File	164
Setting up Boundary Scan Chain	165
JTAG chain setup for SVF generation	165
Manual JTAG chain setup for SVF generation	165
Writing to the SVF File	165
Stop Writing to the SVF	167
Playing back the SVF or XSVF file	167
Other Configuration Modes	167
Slave Serial Configuration Mode	167
SelectMAP Configuration Mode	168

Overview of ISE and Synthesis Tools

This chapter includes the following sections:

- “Overview of ISE”
- “Using Revision Control Features”
- “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 design entry and design implementation tools. You can also access the files and documents associated with your project.

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 window which hierarchically displays the elements included in the project. Beneath the Sources window is the Processes window, which displays available processes for the currently selected source. The third window at the bottom of the Project Navigator is the Transcript window which displays status messages, errors, and warnings and also contains interactive tabs for Tcl scripting and the Find in Files function. The fourth window to the right is a multi-document interface (MDI) window referred to as the *Workspace*. It enables you to view html reports, ASCII text files, schematics, and simulation waveforms. Each window may be resized, undocked from Project Navigator or moved to a new location within the main Project Navigator window. The default layout can always be restored by selecting **View > Restore Default Layout**. These windows are discussed in more detail in the following sections.

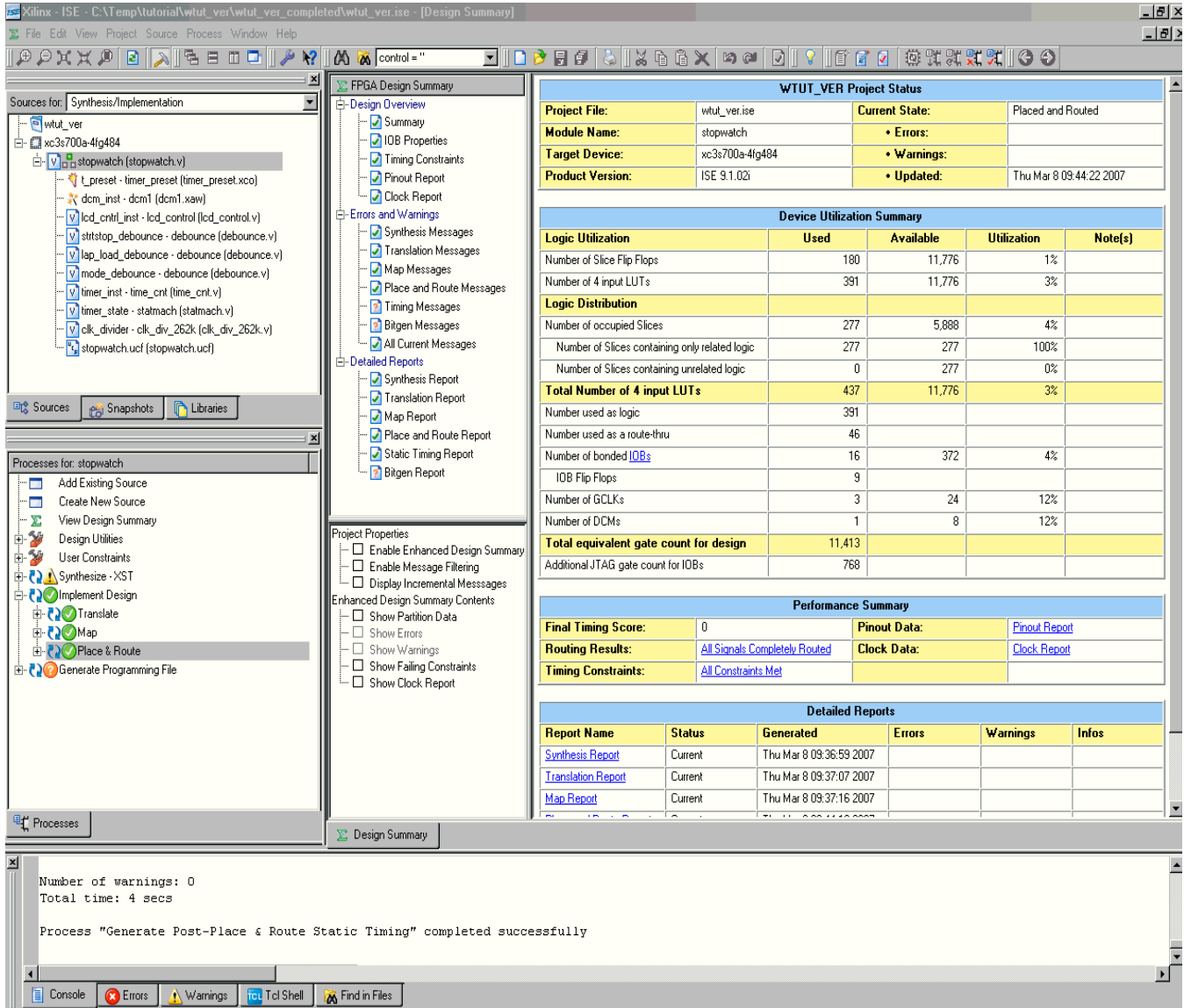


Figure 1-1: Project Navigator

Sources Window

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

Sources Tab

The Sources tab displays the project name, the specified device, and user documents and design source files associated with the selected Design View. The Design View ("Sources for") drop-down list at the top of the Sources tab allows you to view only those source files associated with the selected Design View, such as Synthesis/Implementation. In the "Number of" drop-down list, a Resources column and a Preserve Column are available for Designs that use Partitions. The use of partitions is covered in Chapter 5, "Design Implementation".

Each file in a Design View has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). For a complete list of possible source types and their associated icons, see the ISE™ Help. Select **Help > ISE Help Contents**, select the Index tab and search for 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.

Snapshots Tab

The Snapshots 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 the snapshot can be viewed in the Snapshots tab. You can view the reports, user documents, and source files for all snapshots. All information displayed in the Snapshots tab is read-only. Using snapshots provides an excellent version control system, enabling subteams to do simultaneous development on the same design.

Libraries Tab

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

Processes Window

This window contains one default tab called the Processes tab.

Processes Tab

The Processes tab is context sensitive and it changes based upon the source type selected in the Sources tab and the Top-Level Source in your project. From the Processes tab, you can run the functions necessary to define, run and view your design. The Processes tab provides access to the following functions:

- **Add an Existing Source**
- **Create New Source**
- **View Design Summary**
- **Design Utilities**
Provides access to symbol generation, instantiation templates, View command line Log File, and simulation library compilation.
- **User Constraints**
Provides access to editing location and timing constraints.
- **Synthesis**
Provides access to Check Syntax, Synthesis, View RTL or Technology Schematic, and synthesis reports. This varies depending on the synthesis tools you use.

- **Implement Design**
Provides access to implementation tools, design flow reports, and point tools.
- **Generate Programming File**
Provides access to the configuration tools and bitstream generation.

The Processes tab 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 Implement Design 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 used on the current project, expand Design Utilities and select **View Command Line Log File**. See the Command Line Implementation section of [Chapter 5, “Design Implementation”](#) for further details.

Transcript Window

The Transcript window contains five default tabs: Console, Errors, Warnings, Tcl Shell, Find in Files.

- **Console**
Displays errors, warnings, and information messages. Errors are signified by a red (X) next to the message, while warnings have a yellow exclamation mark (!).
- **Warnings**
Displays only warning messages. Other console messages are filtered out.
- **Errors**
Displays only error messages. Other console messages are filtered out.
- **Tcl Console**
Is a user interactive console. In addition to displaying errors, warnings and informational messages, the Tcl Console allows a user to enter Project Navigator specific Tcl commands. For more information on Tcl commands, see the ISE Help.
- **Find in Files**
Displays the results of the **Edit > Find in Files** function.

Error Navigation to Source

You can navigate from a synthesis error or warning message in the Transcript 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 select **Go to Source** from the right-click menu. The HDL source file opens and the cursor moves to the line with the error.

Error Navigation to Answer Record

You can navigate from an error or warning message in the Transcript window to the relevant Answer Records on the www.xilinx.com/support website. To navigate to the Answer Record(s), select the error or warning message, right-click the mouse, and select **Go to Answer Record** from the right-click menu. The default web browser opens and displays all Answer Records applicable to this message.

Workspace

Design Summary

The Design Summary lists high-level information about your project, including overview information, a device utilization summary, performance data gathered from the Place & Route (PAR) report, constraints information, and summary information from all reports with links to the individual reports.

Text Editor

Source files and other text documents can be opened in a user designated editor. The editor is determined by the setting found by selecting **Edit > Preferences**, expand ISE General and click **Editors**. The default editor is the ISE Text Editor. ISE Text Editor enables you to edit source files and user documents. You can access the Language Templates, which is a catalog of ABEL, Verilog, VHDL, Tcl and User Constraints File templates that you can use and modify in your own design.

ISE Simulator / Waveform Editor

ISE Simulator / Waveform Editor can be used to create and simulate test bench and test fixture within the Project Navigator framework. Waveform Editor can be used to graphically enter stimuli and the expected response, then generate a VHDL test bench or Verilog test fixture. For details, refer to [“Creating a Test Bench Waveform Using the Waveform Editor” in Chapter 4.](#)

Schematic Editor

The Schematic Editor is integrated in the Project Navigator framework. The Schematic Editor can be used to graphically create and view logical designs.

Using Revision Control Features

Using Snapshots

Snapshots enable you to maintain revision control over the design. A snapshot contains a copy of all of the files in the project directory. See also [“Snapshots Tab.”](#)

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.

The snapshot containing all of the files in the project directory along with project settings is displayed in the Snapshots tab.

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 Snapshots tab, select the snapshot from the drop-down list.
2. Select **Project > Make Snapshot Current**.

Before the snapshot replaces the current project, you are given the option to place the current project in a snapshot so that your work is not lost.

Note: Remote sources are copied with the snapshot and placed in a subdirectory named `remote_sources`. These files will need to be manually copied to the original location or added to the project again in a new directory

Viewing a Snapshot

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

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

Using Project Archives

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

Creating an Archive

To create an archive:

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

Note: The archive contains all of the files in the project directory along with project settings. Remote sources are included in the archive under a folder named `remote_sources`. For more information, see the ISE Help.

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.

Using Export/Import Source Control

You can use this feature to export to and retrieve project files and sources from a “staging area.” The Export/Import processes are typically used in conjunction with a 3rd party source control system but can be used independently as a valid means of backing up a design.

Exporting a Project

To export a project

1. Select **Project > Source Control > Export**

- Follow the Export Project Wizard to determine which files to export, where to export the files to, and optionally create a Tcl script that can be used to regenerate the ISE project.

Importing a Project

To import a project

- Select **Project > Source Control > Import**
- Select the Project File (.ise or Tcl import script) and directory location to import the project file and sources.

Overview of Synthesis Tools

You can synthesize your design using various synthesis tools. The following section lists the supported synthesis tools and includes some process properties information.

Precision Synthesis

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. With 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.

Process Properties

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

For more information, see the Precision online help.

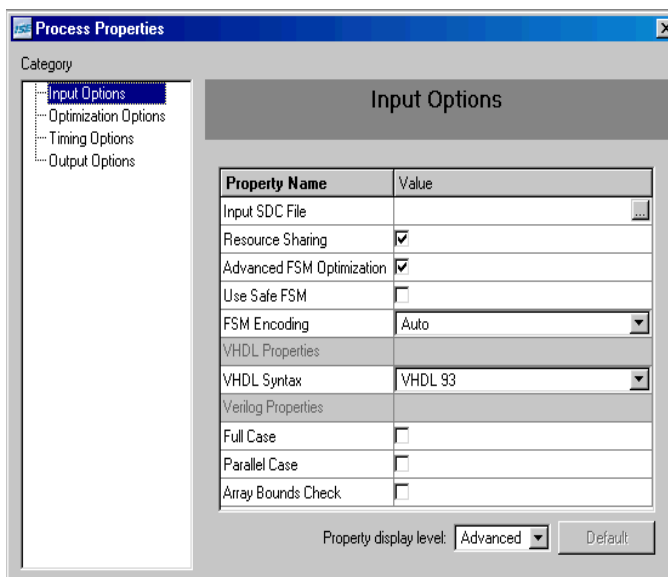


Figure 1-2: Precision Synthesis Process Properties

Synplify/Synplify Pro

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

Process Properties

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

For more information, see the Synplify/Synplify Pro online help.

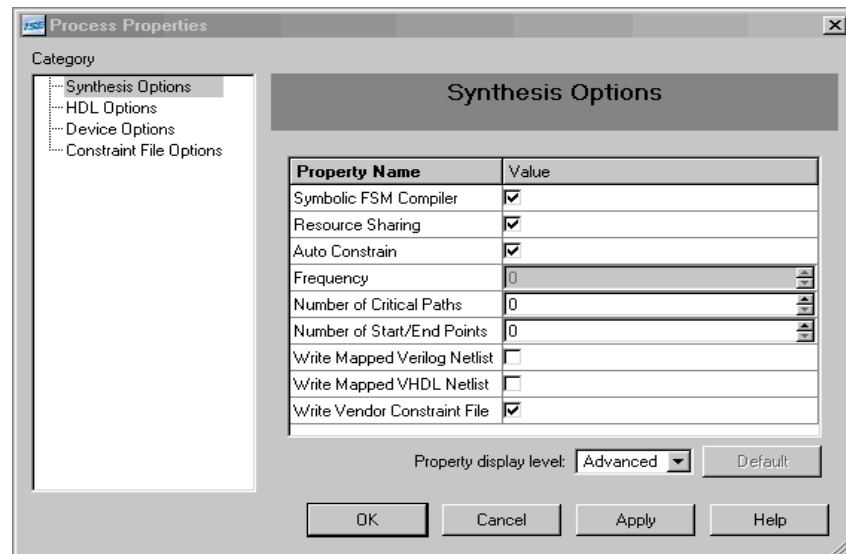


Figure 1-3: Synplify/Synplify Pro Synthesis Process Properties

Xilinx Synthesis Technology (XST)

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

Process Properties

Process properties enable you to control the synthesis results of XST. Two commonly used properties are Optimization Goal and Optimization Effort. With 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. From ISE, select **Help > Software Manuals**, or go on the web at http://www.xilinx.com/support/sw_manuals/xilinx8/.

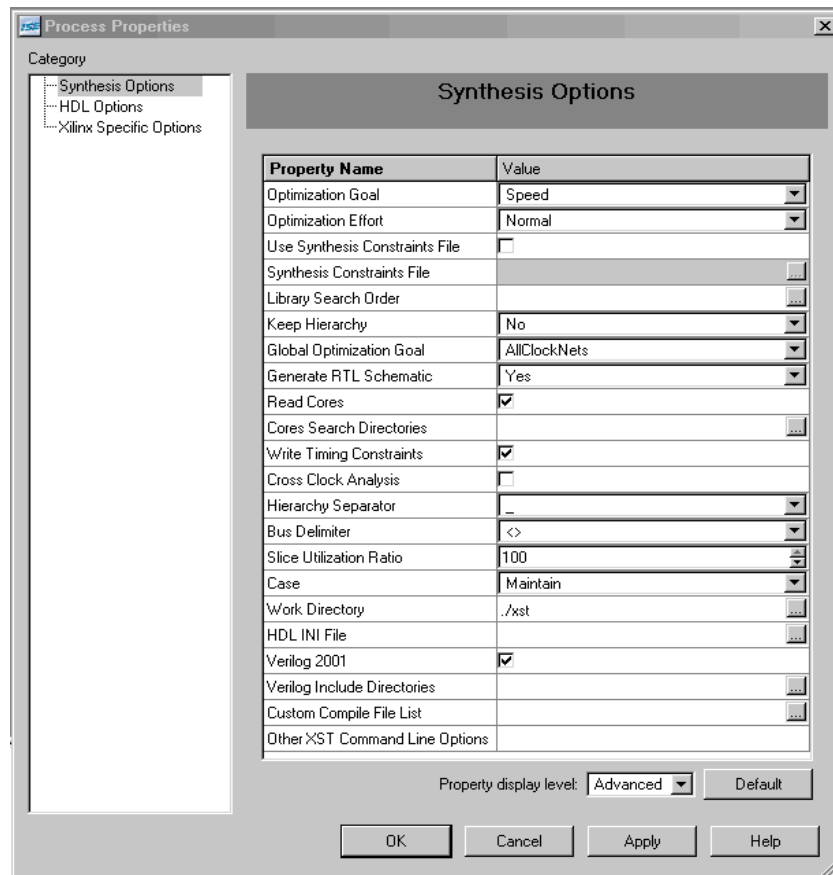


Figure 1-4: XST Synthesis Process Properties

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 Spartan™-3A 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 two cores. You can synthesize the design using Xilinx Synthesis Technology (XST), Synplify/Synplify Pro, or Precision.

This chapter is the first chapter in the [“HDL Design Flow.”](#) After the design is successfully defined, you will perform behavioral simulation ([Chapter 4, “Behavioral Simulation”](#)), run implementation with the Xilinx Implementation Tools ([Chapter 5, “Design Implementation”](#)), perform timing simulation ([Chapter 6, “Timing Simulation”](#)), and configure and download to the Spartan-3A demo board ([Chapter 7, “iMPACT Tutorial”](#)).

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™ 9.1i
- Spartan-3A libraries and device files

Note: For detailed software installation instructions, refer to the *ISE Release Notes and Installation Guide*.

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 your installation path for `c:\xilinx` in the procedures that follow.

Optional Software Requirements

The following third-party synthesis tools are incorporated into this tutorial, and may be used in place of the Xilinx Synthesis Tool (XST):

- Synplcity Synplify/Synplify PRO 8.8.02 (or above)
- Mentor Precision Synthesis 2006a2.7 (or above)

The following third-party simulation tool is optional for this tutorial, and may be used in place of the ISE Simulator:

- ModelSim

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 for that language. XST can synthesize a mixed-language design. However, this tutorial does not go over the mixed language feature.

Installing the Tutorial Project Files

The Stopwatch tutorial projects can be downloaded from <http://www.xilinx.com/support/techsup/tutorials/tutorials9.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 into the `c:\xilinx\ISEexamples` directory, replacing any existing files in that directory.

When you unzip the tutorial project files into `c:\xilinx\ISEexamples`, 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 newly-created directory.

The following table lists the locations of tutorial source files.

Table 2-1: Tutorial Directories

Directory	Description
<code>wtut_vhd</code>	Incomplete VHDL Source Files
<code>wtut_ver</code>	Incomplete Verilog Source Files
<code>wtut_vhd\wtut_vhd_completed</code>	Completed VHDL Source Files
<code>wtut_ver\wtut_ver_completed</code>	Completed Verilog Source Files

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

The completed directories contain the finished HDL source files.

This tutorial assumes that the files are unzipped under `c:\xilinx\ISEexamples`, but you can unzip the source files into any directory with read-write permissions. If you unzip the files into a different location, substitute your project path for `c:\xilinx\ISEexamples` in the procedures that follow.

Starting the ISE Software

To start ISE:

Double-click the **ISE Project Navigator** icon on your desktop or select **Start > All Programs > Xilinx ISE 9.1i > Project Navigator**.



Figure 2-1: Project Navigator Desktop Icon

Creating a New Project

Note: Two methods are provided for creating a new project: Using the New Project Wizard and Using a Tcl Script. Use either method provided below.

Creating a New Project: Using the New Project Wizard

1. From Project Navigator, select **File > New Project**.
The New Project Wizard appears.

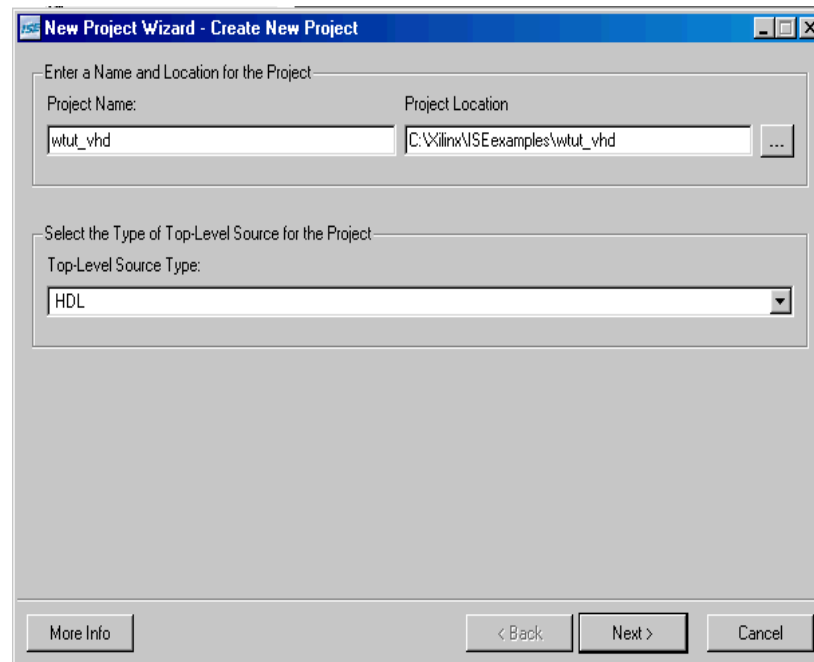


Figure 2-2: New Project Wizard - Create New Project

2. In the Project Location field, browse to `c:\xilinx\ISEexamples` or to the directory in which you installed the project.
3. Type `wtut_vhd` or `wtut_ver` in the Project Name field.
4. Verify that HDL is selected as the Top-Level Source Type and click **Next**.

The New Project Wizard - Device Properties window appears.

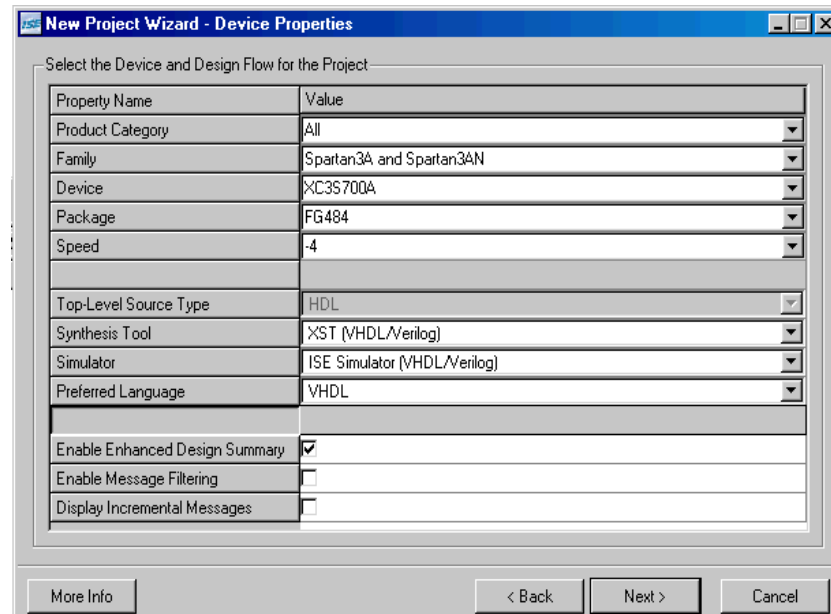


Figure 2-3: New Project Wizard - Device Properties

5. Select the following values in the New Project Wizard - Device Properties window:
 - ◆ Product Category: **All**
 - ◆ Family: **Spartan3A and Spartan3AN**
 - ◆ Device: **XC3S700A**
 - ◆ Package: **FG484**
 - ◆ Speed: **-4**
 - ◆ Synthesis Tool: **XST (VHDL/Verilog)**
 - ◆ Simulator: **ISE Simulator (VHDL/Verilog)**
 - ◆ Preferred Language: **VHDL** or **Verilog** depending on preference. This will determine the default language for all processes that generate HDL files.
6. Click **Next**, then **Next**, and then click **Add Source** in the New Project Wizard - Add Existing Sources window.
7. Browse to `c:\xilinx\ISEexamples\wtut_vhd` or `c:\xilinx\ISEexamples\wtut_ver`.
8. Select the following files (`.vhd` files for VHDL design entry or `.v` files for Verilog design entry) and click **Open**.
 - ◆ `clk_div_262k`
 - ◆ `lcd_control`
 - ◆ `statmach`
 - ◆ `stopwatch`
9. Click **Next**, then **Finish** to complete the New Project Wizard.
10. In the Adding Source Files dialog box, verify that all added HDL files are associated with Synthesis/Imp + Simulation, then click **OK**.

Creating a New Project: Using a Tcl Script

1. With Project Navigator open, select the Tcl Shell tab.
2. Change the current working directory to the directory where the tutorial source files were unzipped by typing `cd c:/xilinx/ISEexamples/wtut_vhd` (VHDL design entry) or `cd c:/xilinx/ISEexamples/wtut_ver` (Verilog design entry).
3. Run the project creation Tcl script by typing `source create_wtut_vhd.tcl` (VHDL design entry) or `source create_wtut_ver.tcl` (Verilog design entry).
4. (VHDL only) Once the project is created and opened, right-click on the device line and select **Properties...** Change the Preferred Language: to **VHDL**. This will determine the default language for all processes that generate HDL files.

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 IP modules.

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

In the runner's stopwatch design, there are five external inputs and four external output buses. The system clock is an externally generated signal. The following list summarizes the input and output signals of the design.

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**
Puts the stopwatch in clocking mode and resets the time to 0:00:00.
- **clk**
Externally generated system clock.
- **mode**
Toggles between clocking and timer modes. This input is only functional while the clock or timer is not counting.
- **lap_load**
This is a dual function signal. In clocking mode it displays the current clock value in the 'Lap' display area. In timer mode it loads the pre assigned values from the ROM to the timer display when the timer is not counting.

Outputs

The following are outputs signals for the design.

- **lcd_e, lcd_rs, lcd_rw**
These outputs are the control signals for the LCLD display of the Spartan-3A demo board used to display the stopwatch times.
- **sf_d[7:0]**
Provides the data values for the LCD display.

Functional Blocks

The completed design consists of the following functional blocks.

- **clk_div_262k**

Macro which divides a clock frequency by 262,14. Converts 26.2144 MHz clock into 100 Hz 50% duty cycle clock.

- **dcm1**
Clocking Wizard macro with internal feedback, frequency controlled output, and duty-cycle correction. The CLKFX_OUT output converts the 50 MHz clock of the Spartan-3A demo board to 26.2144 MHz.
- **debounce**
Schematic module implementing a simplistic debounce circuit for the strtstop, mode, and lap_load input signals.
- **lcd_control**
Module controlling the initialization of and output to the LCD display.
- **statmach**
State Machine module defined and implemented in State Diagram Editor. Controls the state of the stopwatch.
- **timer_preset**
CORE Generator™ 64X20 ROM. This macro contains 64 preset times from 0:00:00 to 9:59:99 which can be loaded into the timer.
- **time_cnt**
Up/down counter module which counts between 0:00:00 to 9:59:99 decimal. This macro has five 4-bit outputs, which represent the digits of the stopwatch time.

Design Entry

For this hierarchical design, you will examine HDL files, correct syntax errors, create an HDL macro, and add a CORE Generator and a Clocking module. 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.isc` or `wtut_ver.isc` project open in Project Navigator, the Sources tab displays all of the source files currently added to the project, with the associated entity or module names (see [Figure 2-4](#)). In the current project, `time_cnt` 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.

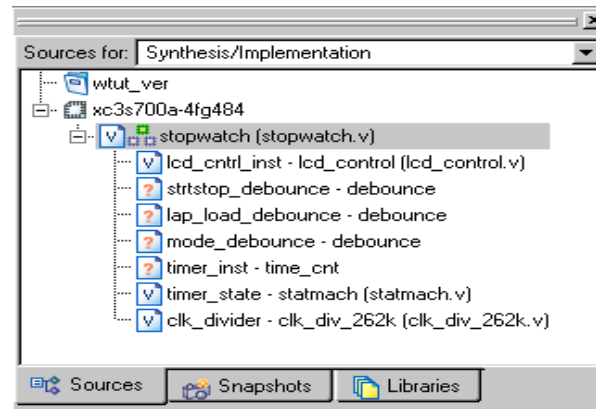


Figure 2-4: Sources Tab Showing Completed Design

Adding Source Files

HDL files must be added to the project before they can be synthesized. Three HDL files have already been added to this project. An additional file must be added.

1. Select **Project > Add Source**.
2. Select `time_cnt.vhd` or `time_cnt.v` from the project directory and click **Open**.
3. In the Adding Source Files dialog box, verify that `time_cnt` is associated with Synthesis/Imp + Simulation and click **OK**.

Note: Alternatively, the `time_cnt.vhd` file could be added to the project by entering the following command in the Tcl Console tab.

```
xfile add time_cnt.vhd
```

The red question-mark (?) for `time_cnt` should change to show the VHD file icon.



Figure 2-5: time_cnt.vhd File in Sources Tab

Checking the Syntax

To check the syntax of source files:

1. Select `stopwatch.vhd` or `stopwatch.v` in the Sources tab.
When you select the HDL file, the Processes tab displays all processes available for this file.
2. In the Processes tab, click the + next to Synthesize to expand the process hierarchy.
3. Double-click **Check Syntax** in the Synthesize hierarchy.

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

Correcting HDL Errors

The `time_cnt` module contains a syntax error that must be corrected. The red “x” beside the Check Syntax process indicates an error was found during the analysis. In the Console tab of the Transcript window, Project Navigator reports errors with a red (X) and warnings with a yellow (!).

To display the error in the source file:

1. Click the file name in the error message in the Console or Errors tab. The source code comes up in the main display tab.
2. Correct any errors in the HDL source file. The comments next to the error explain this simple fix.
3. Select **File > Save** to save the file.
4. Re-analyze the file by selecting the HDL file and double-clicking **Check Syntax**.

Creating an HDL-Based Module

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

You will author a new HDL module. This macro will be used to debounce the strtstop, mode and lap_load inputs.

Using the New Source Wizard and ISE Text Editor

In this section, you create a file using the New Source wizard, specifying the name and ports of the component. The resulting HDL file is then modified in the ISE Text Editor.

To create the source file:

1. Select **Project > New Source**.
The dialog box New Source Wizard 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 **debounce**.
4. Click **Next**.
5. Enter two input ports named sig_in and clk and an output port named sig_out for the debounce component in this way:
 - a. In the first three Port Name fields type **sig_in**, **clk** and **sig_out**.
 - b. Set the Direction field to **input** for **sig_in** and **clk** and to **output** for **sig_out**.
 - c. Leave the Bus designation boxes unchecked.

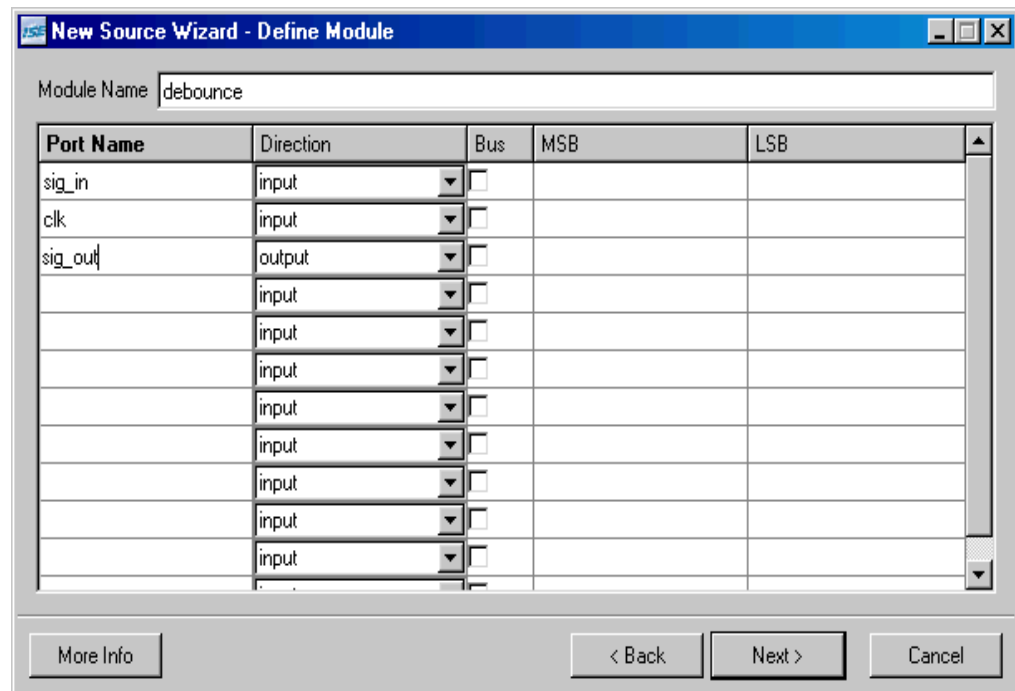


Figure 2-6: New Source Wizard for Verilog

6. Click **Next** to complete the Wizard session.
A description of the module displays.
7. Click **Finish** to open the empty HDL file in the ISE Text Editor.

The VHDL file is displayed in [Figure 2-7](#). The Verilog HDL file is displayed in [Figure 2-8](#).

```

12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  -----
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity debounce is
31      Port ( sig_in : in  STD_LOGIC;
32            clk     : in  STD_LOGIC;
33            sig_out : out STD_LOGIC);
34  end debounce;
35
36  architecture Behavioral of debounce is
37
38  begin
39
40

```

Figure 2-7: VHDL File in ISE Text Editor

```

1  -----
2  //////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    14:12:53 03/15/2007
7  // Design Name:
8  // Module Name:   debounce
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////
21 module debounce(sig_in, clk, sig_out);
22     input sig_in;
23     input clk;
24     output sig_out;
25
26
27 endmodule
28

```

Figure 2-8: Verilog File in ISE Text Editor

In the ISE Text Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are displayed in blue, comments in green, and values are black. The file is color-coded to enhance readability and help you recognize 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 Debounce Circuit template for this exercise.

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

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

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

Each HDL language in the Language Templates is divided into five sections: Common Constructs, Device Primitive Instantiation, Simulation Constructs, Synthesis Constructs and User Templates. To expand the view of any of these sections, click the **+** next to the section. Click any of the listed templates to view the template contents in the right pane.

2. Under either the VHDL or Verilog hierarchy, expand the Synthesis Constructs hierarchy, expand the Coding Examples hierarchy, expand the Misc hierarchy, and select the template called One-shot, Debounce Circuit. Use the appropriate template for the language you are using.

When the template is selected in the hierarchy the contents display in the right pane.

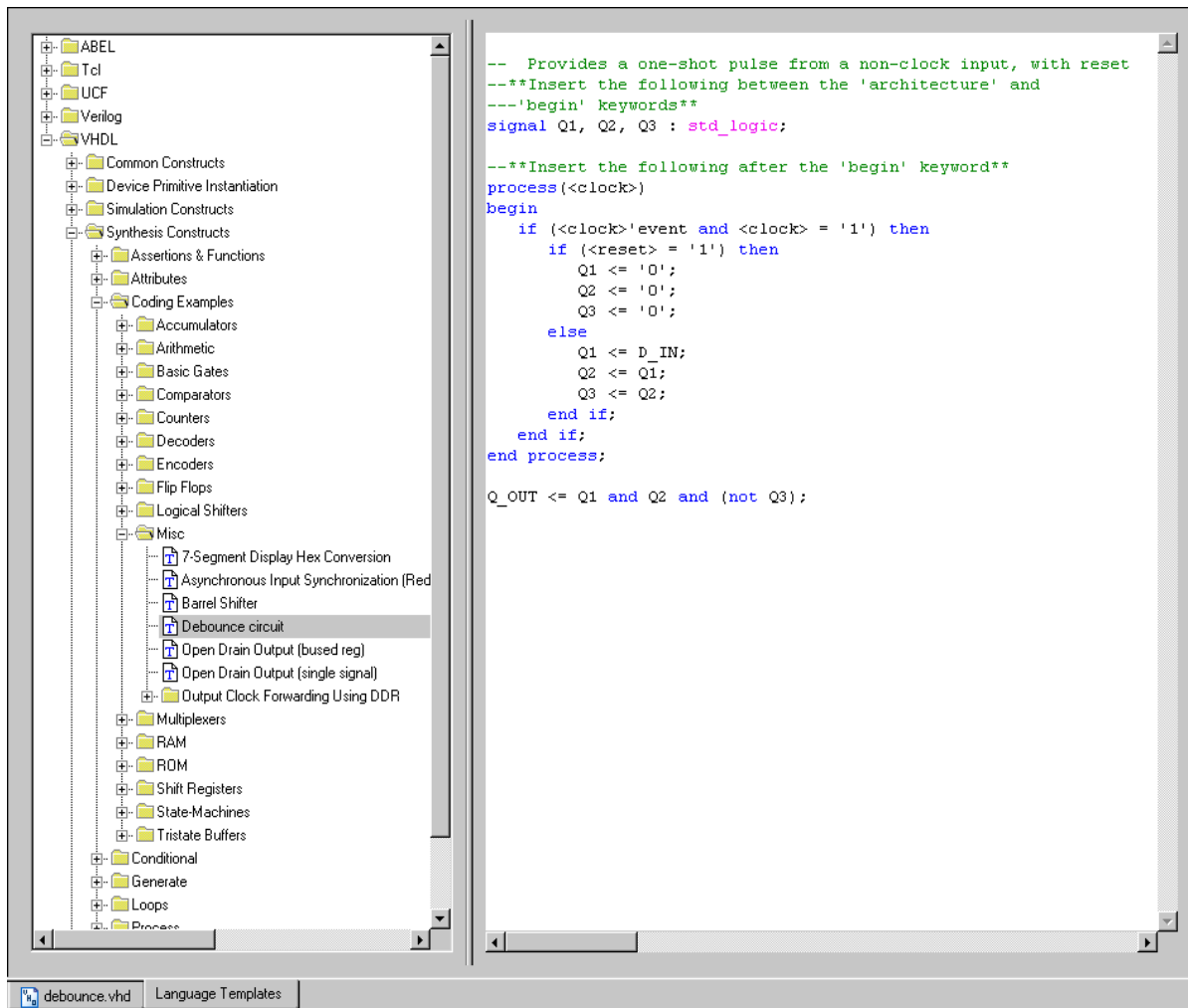


Figure 2-9: Language Templates

Adding a Language Template to Your File

You will now use the drag and drop method for adding templates to your HDL file. Refer to “Working with Language Templates” in the ISE Help for additional usability options.

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

1. Select **Window > Tile Vertically** to show both the HDL file and the Language Templates window.
2. Click and drag the **Debounce Circuit** name from the Language Template topology into the `debounce.vhd` file under the architecture `begin` statement, or into the `debounce.v` file under the module and pin declarations.
3. Close the Language Templates window.
4. (Verilog only) Complete the Verilog module by doing the following:
 - a. Remove the reset logic (not used in this design) by deleting the three lines beginning with `if` and ending with `else`.
 - b. Change `<reg_name>` to `q` in all six locations.

- c. Change `<clock>` to `clk`; `<input>` to `sig_in`; and `<output>` to `sig_out`.
5. (VHDL only) Complete the VHDL module by doing the following:
 - a. Move the line beginning with the word **signal** so that it is between the **architecture** and **begin** keywords.
 - b. Remove the reset logic (not used in this design) by deleting the five lines beginning with `if (<reset>...` and ending with `else` and delete one of the `end if;` lines.
 - c. Change `<clock>` to `clk`; `D_IN` to `sig_in`; and `Q_OUT` to `sig_out`.
You now have complete and functional HDL code.
6. Save the file by selecting **File > Save**.
7. Select `debounce` in the Sources tab.
8. In the Processes tab, double-click **Check Syntax**.
9. Close the ISE Text Editor.

Creating a CORE Generator Module

CORE Generator is a graphical interactive design tool that enables you to create high-level modules such as memory elements, math functions and communications and IO interface cores. 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, SRL16s, and distributed and block RAM.

In this section, you will create a CORE Generator module called `timer_preset`. The module will be used to store a set of 64 values to load into the timer.

Creating a CORE Generator Module

To create a CORE Generator module:

1. In Project Navigator, select **Project > New Source**.
2. Select **IP (Coregen & Architecture Wizard)**.
3. Type `preset_timer` in the File name field.
4. Click **Next**.
5. Double-click **Memories & Storage Elements**, then double-click **RAMs & ROMs**.
6. Select **Distributed Memory Generator**, then click **Next** and click **Finish** to open the Distributed Memory Generator customization GUI. This customization GUI enables you to customize the memory to the design specifications.
7. Fill in the Distributed Memory Generator customization GUI with the following settings:
 - ◆ Component Name: **timer_preset** - Defines the name of the module.
 - ◆ Depth: **64** - Defines the number of values to be stored
 - ◆ Data Width: **20** - Defines the width of the output bus.
 - ◆ Memory Type: **ROM**
 - ◆ Click **Next**.
 - ◆ Leave Input and output options as Non Registered; Click **Next**.

- ◆ Coefficients File: Double click on the **Browse** button and select definition1_times.coe.

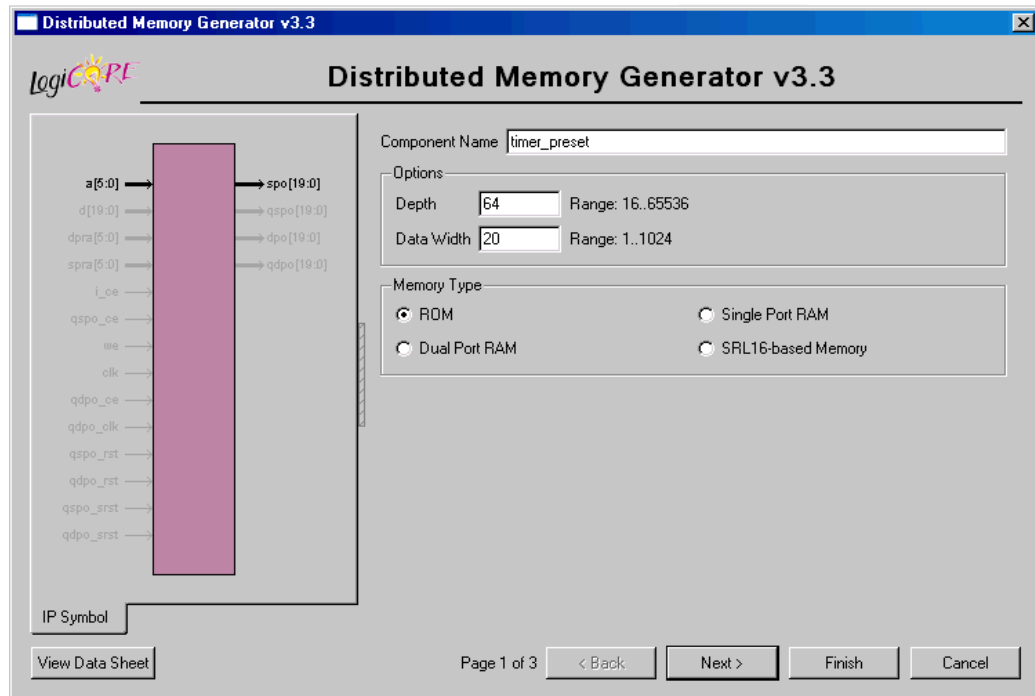


Figure 2-10: CORE Generator - Distributed Memory Generator Customization GUI

8. Check that *only* the following pins are used (used pins are highlighted on the symbol on the left side of the customization GUI):
 - ◆ **a[5:0]**
 - ◆ **spo[19:0]**
9. Click **Finish**.

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

Note: A number of files are added to the project directory. Some of these files are:

- ◆ **timer_preset.vho** or **timer_preset.veo**
These are the instantiation templates used to incorporate the CORE Generator module into your source HDL.
- ◆ **timer_preset.vhd** or **timer_preset.v**
These are HDL wrapper files for the core and are used only for simulation.
- ◆ **timer_preset.edn**
This file is the netlist that is used during the Translate phase of implementation.
- ◆ **timer_preset.xco**
This file stores the configuration information for the timer_preset module and is used as a project source.
- ◆ **timer_preset.mif**
This file provides the initialization values of the ROM for simulation.

Instantiating the CORE Generator Module in the HDL Code

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

VHDL Flow

To instantiate the CORE Generator module using a VHDL flow:

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

```
-- Insert CORE Generator ROM component declaration here
```

3. Select **Edit > Insert File**, then select `timer_preset.vho` and click **Open**.

The VHDL template file for the CORE Generator instantiation is inserted.

```
121 ----- Begin Cut here for COMPONENT Declaration ----- COMP_TAG
122 component timer_preset
123     port (
124         a: IN std_logic_VECTOR(5 downto 0);
125         spo: OUT std_logic_VECTOR(19 downto 0));
126 end component;
127
128 -- Synplicity black box declaration
129 attribute syn_black_box : boolean;
130 attribute syn_black_box of timer_preset: component is true;
131
132 -- COMP_TAG_END ----- End COMPONENT Declaration -----
```

Figure 2-11: VHDL Component Declaration for CORE Generator Module

4. Highlight the inserted code from

```
-- Begin Cut here for INSTANTIATION Template ----
to
```

```
--INST_TAG_END ----- END INSTANTIATION Template -----
```

5. Select **Edit > Cut**.

6. Place the cursor after the line that states:

```
--Insert CORE Generator ROM Instantiation here
```

7. Select **Edit > Paste** to place the core instantiation.

8. Change the instance name from `your_instance_name` to `t_preset`.

9. Edit this instantiated code to connect the signals in the Stopwatch design to the ports of the CORE Generator module as shown in [Figure 2-12](#).

```
169 ----- Insert CORE Generator ROM instantiation here
170 ----- Begin Cut here for INSTANTIATION Template ----- INST_TAG
171 t_preset : timer_preset
172     port map (
173         a => address,
174         spo => preset_time);
175 -- INST_TAG_END ----- End INSTANTIATION Template -----
```

Figure 2-12: VHDL Component Instantiation of CORE Generator Module

10. The inserted code of `timer_preset.vho` contains several lines of commented text for instruction and legal documentation. Delete these commented lines if desired.
11. Save the design using **File > Save**, and close the ISE Text Editor.

Verilog Flow

To instantiate the CORE Generator module using a Verilog flow:

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

```
//Place the Coregen module instantiation for timer_preset here
```
3. Select **Edit > Insert File**, and select `timer_preset.vco`.
4. The inserted code of `timer_preset.vco` contains several lines of commented text for instruction and legal documentation. Delete these commented lines if desired.
5. Change the instance name from **YourInstanceName** to **t_preset**.
6. Edit this code to connect the signals in the Stopwatch design to the ports of the CORE Generator module as shown in [Figure 2-13](#).

```

36 // Place the Coregen module instantiation for timer_preset here
37 //----- Begin Cut here for INSTANTIATION Template ---// INST_TAG
38 timer_preset t_preset (
39     .a(address), // Bus [5 : 0]
40     .spo(preset_time)); // Bus [19 : 0]
41
42 // INST_TAG_END ----- End INSTANTIATION Template -----

```

Figure 2-13: Verilog Component Instantiation of the CORE Generator Module

7. Save the design using **File > Save** and close `stopwatch.v` in the ISE Text Editor.

Creating a DCM Module

The Clocking Wizard, a 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 the Clocking Wizard

To create the `dcm1` module:

1. In Project Navigator, select **Project > New Source**.
2. In the New Source dialog box, select **IP (CoreGen & Architecture Wizard)** source and type **dcm1** for the file name.
3. Click **Next**.
4. In the Select IP dialog box, select **FPGA Features and Design > Clocking > Spartan-3E, Spartan-3A > Single DCM SP**.

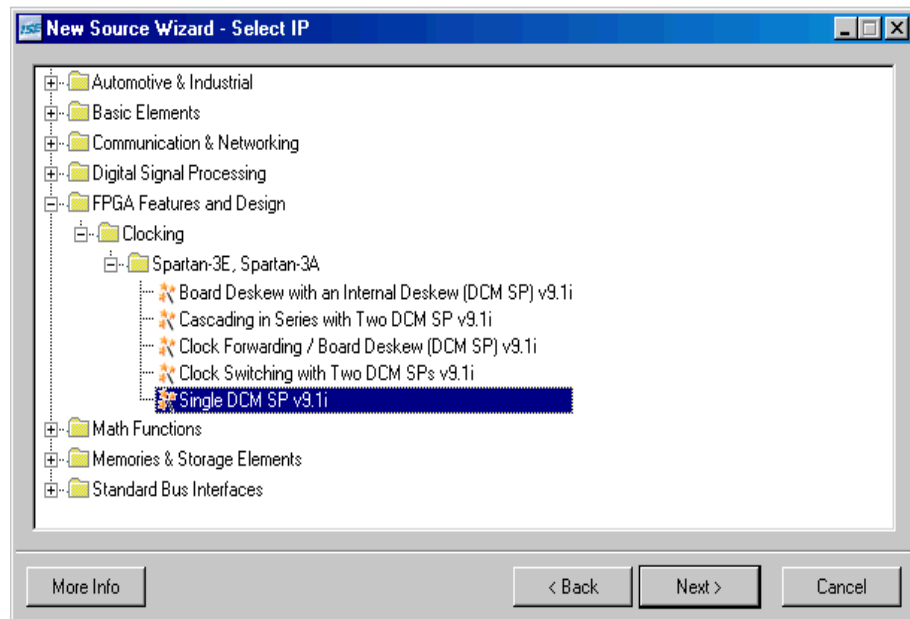


Figure 2-14: Selecting Single DCM IP Type

5. Click **Next**, then **Finish**. The Clocking Wizard is launched.
6. Verify that **RST**, **CLK0** and **LOCKED** ports are selected.
7. Select **CLKFX** port.
8. Type **50** and select **MHz** for the Input Clock Frequency.
9. Verify the following settings:
 - ◆ Phase Shift: **NONE**
 - ◆ CLKIN Source: **External, Single**
 - ◆ Feedback Source: **Internal**
 - ◆ Feedback Value: **1X**
 - ◆ Use Duty Cycle Correction: Selected
10. Click the **Advanced** button.
11. Select **Wait for DCM lock before DONE Signal goes high**.
12. Click **OK**.
13. Click **Next**, and then click **Next** again.
14. Select **Use output frequency** and type **26.2144** in the box and select **MHz**.

$$(26.2144\text{MHz}) / 2^{18} = 100\text{Hz}$$

15. Click **Next**, and then click **Finish**.

The dcm1 .xaw file is added to the list of project source files in the Sources tab.

Instantiating the dcm1 Macro - VHDL Design

Next, you will 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 tab, select `dcm1.xaw`.
2. In the Processes tab, right-click **View HDL Instantiation Template** and select **Properties**.
3. Choose **VHDL** for the HDL Instantiation Template Target Language value and click **OK**.
4. In the Processes tab, double-click **View HDL Instantiation Template**.
5. Highlight the component declaration template in the newly opened HDL Instantiation Template (`dcm1.vhi`), shown below.

```

4  -- Notes:
5  -- 1) This instantiation template has been au
6  -- std_logic and std_logic_vector for the por
7  -- 2) To use this template to instantiate thi
8
9  COMPONENT dcm1
10 PORT(
11     CLKIN_IN : IN std_logic;
12     RST_IN   : IN std_logic;
13     CLKFX_OUT : OUT std_logic;
14     CLKIN_IBUFG_OUT : OUT std_logic;
15     CLKO_OUT : OUT std_logic;
16     LOCKED_OUT : OUT std_logic
17 );
18 END COMPONENT;
```

Figure 2-15: VHDL DCM Component Declaration

6. Select **Edit > Copy**.
7. Place the cursor in the `stopwatch.vhd` file in a section labeled `-- Insert dcm1 component declaration here.`
8. Select **Edit > Paste** to paste the component declaration.
9. Highlight the instantiation template in the newly opened HDL Instantiation Template, shown below.

```

19
20     Inst_dcm1: dcm1 PORT MAP(
21         CLKIN_IN => ,
22         RST_IN   => ,
23         CLKFX_OUT => ,
24         CLKIN_IBUFG_OUT => ,
25         CLKO_OUT => ,
26         LOCKED_OUT =>
27     );
28
```

Figure 2-16: VHDL DCM Component Instantiation

10. Select **Edit > Copy**.
11. Place the cursor in the `stopwatch.vhd` file below the line labeled `-- Insert dcm1 instantiation here"`.

12. Select **Edit > Paste** to paste the instantiation template.
13. Make the necessary changes as shown in the figure below.

```

141 ----- Insert dcm1 instantiation here
142     Inst_dcm1: dcm1 PORT MAP(
143         CLKIN_IN => clk,
144         RST_IN => reset,
145         CLKFX_OUT => clk_26214k,
146         CLKIN_IBUFG_OUT => open,
147         CLKO_OUT => open,
148         LOCKED_OUT => locked
149     );

```

Figure 2-17: VHDL Instantiation for dcm1

14. Slect **File > Save** to save the stopwatch.vhd file.

Instantiating the dcm1 Macro - Verilog

To instantiate the dcm1 macro for your Verilog design:

1. In Project Navigator, in the Sources tab, select dcm1.xaw.
2. In the Processes tab, double-click **View HDL Instantiation Template**.
3. From the newly opened HDL Instantiation Template (dcm1.tfi), copy the instantiation template, shown below.

```

4 // Instantiate the module
5 dcm1 instance_name (
6     .CLKIN_IN(CLKIN_IN),
7     .RST_IN(RST_IN),
8     .CLKFX_OUT(CLKFX_OUT),
9     .CLKIN_IBUFG_OUT(CLKIN_IBUFG_OUT),
10    .CLKO_OUT(CLKO_OUT),
11    .LOCKED_OUT(LOCKED_OUT)
12 );

```

Figure 2-18: dcm1 Macro and Instantiation Templates

4. Paste the instantiation template into the section in stopwatch.v labeled //Insert dcm1 instantiation here.
5. Make the necessary changes as shown in the figure below.

```

45 //Insert dcm1 instantiation here
46 dcm1 instance_name (
47     .CLKIN_IN(clk),
48     .RST_IN(reset),
49     .CLKFX_OUT(clk_26214k),
50     .CLKIN_IBUFG_OUT(),
51     .CLKO_OUT(),
52     .LOCKED_OUT(locked)
53 );

```

Figure 2-19: Verilog Instantiation for dcm1

6. Slect **File > Save** to save the stopwatch.v file.

Synthesizing the Design

So far you have been using XST (the Xilinx synthesis tool) for syntax checking. Next, you will synthesize the design using either XST, Synplify/Synplify Pro or Precision. The synthesis tool uses the design's HDL code and generates a supported netlist type (EDIF or NGC) for the Xilinx implementation tools. The synthesis tool performs three general steps (although all synthesis tools further break down these general steps) to create the netlist:

- **Analyze / Check Syntax**
Checks the syntax of the source code.
- **Compile**
Translates and optimizes the HDL code into a set of components that the synthesis tool can recognize.
- **Map**
Translates the components from the compile stage into the target technology's primitive components.

The synthesis tool can be changed at any time during the design flow. To change the synthesis tool:

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

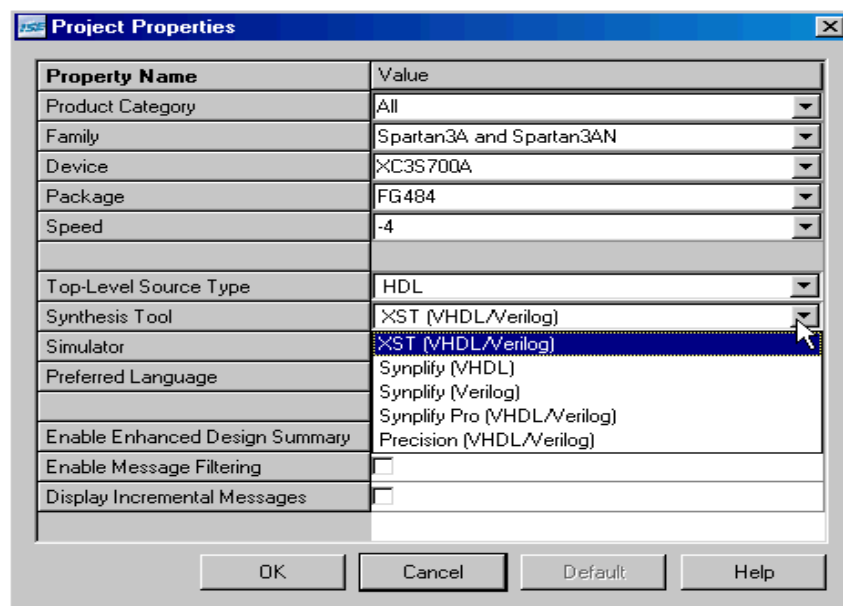


Figure 2-20: Specifying Synthesis Tool

Note: If you do not see your synthesis tool among the options in the list, you may not have the software installed or may not have it configured in ISE. The Synthesis tools are configured in the Preferences dialog box (**Edit > Preferences**, expand ISE General, then click **Integrated Tools**).

Changing the design flow results in the deletion of implementation data. You have not yet created any implementation data in this tutorial. For projects that contain implementation data, Xilinx recommends that you take a snapshot of the project before changing the

synthesis tool to preserve this data. For more information about taking a snapshot, see “Creating a Snapshot” in Chapter 1.

A summary of available synthesis tools is available in “Overview of Synthesis Tools” in Chapter 1

Read the section for your synthesis tool:

- “Synthesizing the Design using XST”
- “Synthesizing the Design using Synplify/Synplify Pro”
- “Synthesizing the Design Using Precision Synthesis”

Synthesizing the Design using XST

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

Processes available for synthesis using XST are as follows:

- **View Synthesis Report**
Gives a synthesis mapping and timing summary as well as optimizations that took place.
- **View RTL Schematic**
Generates a schematic view of your RTL netlist.
- **View Technology Schematic**
Generates a schematic view of your Technology netlist.
- **Check Syntax**
Verifies that the HDL code is entered properly.
- **Generate Post-Synthesis Simulation Model**
Creates HDL simulation models based on the synthesis netlist.

Entering Constraints

XST supports a User Constraint File (UCF) style syntax to define synthesis and timing constraints. This format is called the Xilinx Constraint File (XCF), and the file has an .xcf file extension. XST uses the .xcf extension to determine if the file is a constraints file.

To create a new Xilinx Constraint File:

1. Select **Project > Add Source**.
2. In the Add Existing Sources dialog box, change the Files of type: to ‘All Files (*.*)’ and then select and add **stopwatch.xcf**.
3. Notice that stopwatch.xcf is added as a User document.

Note: In place of the three steps above, you may add the .xcf file through the the Tcl Console, using the following command and then selecting **View > Refresh**.

```
xfile add stopwatch.xcf
```

4. Double-click `stopwatch.xcf` to open the file in the ISE Text Editor.
5. The following constraints should exist in the `stopwatch.xcf` file:

```
NET "CLK" TNM_NET = "CLK_GROUP";
TIMESPEC "TS_CLK" = PERIOD "CLK_GROUP" 20 ns;
BEGIN MODEL stopwatch
    NET "sf_d<7>" LOC = "Y15";
    NET "sf_d<6>" LOC = "AB16";
    NET "sf_d<5>" LOC = "Y16";
    NET "sf_d<4>" LOC = "AA12";
    NET "sf_d<3>" LOC = "AB12";
    NET "sf_d<2>" LOC = "AB17";
    NET "sf_d<1>" LOC = "AB18";
    NET "sf_d<0>" LOC = "Y13";
END;
```

6. Close `stopwatch.xcf`.

Note: For more constraint options in the implementation tools, see “Using the Constraints Editor” and “Using the Floorplan Editor” in Chapter 5, “Design Implementation.”

Entering Synthesis Options

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

To enter synthesis options:

1. Select `stopwatch.vhd` (or `stopwatch.v`) in the Sources tab.
2. In the Processes tab, right-click the **Synthesize** process and select **Properties**.
3. Under the Synthesis Options tab, click in the **Synthesis Constraints File** property field and enter `stopwatch.xcf`.
4. Check the **Write Timing Constraints** box.
5. Click **OK**.

Synthesizing the Design

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

1. Select `stopwatch.vhd` (or `stopwatch.v`).
2. Double-click the **Synthesize** process in the Processes tab.

The RTL / Technology Viewer

XST can generate a schematic representation of the HDL code that you have entered. A schematic view of the code helps you analyze your design by displaying a graphical connection between the various components that XST has inferred. There are two forms of the schematic representation:

- **RTL View** - Pre-optimization of the HDL code.
- **Technology View** - Post-synthesis view of the HDL design mapped to the target technology.

To view a schematic representation of your HDL code:

1. In the Processes tab, click the **+** next to Synthesize to expand the process hierarchy.
2. Double-click **View RTL Schematic** or **View Technology Schematic**.

The RTL Viewer displays the top level schematic symbol for the design. Double-click on the symbol to push into the schematic and view the various design elements and connectivity. Right-click the schematic to view the various operations that can be performed in the schematic viewer.

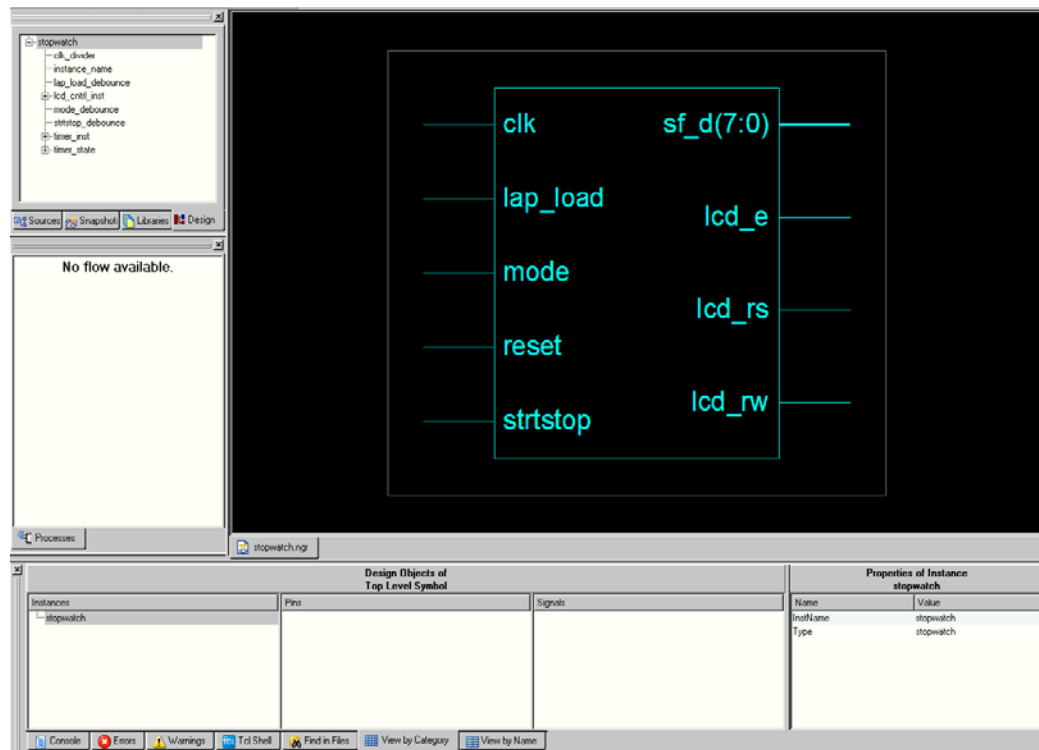


Figure 2-21: RTL Viewer

You have completed XST synthesis. An NGC file now exists for the Stopwatch design.

To continue with the HDL flow:

- Go to [Chapter 4, “Behavioral Simulation,”](#) to perform a pre-synthesis simulation of this design.
OR
- Proceed to [Chapter 5, “Design Implementation,”](#) to place and route the design.

Note: For more information about XST constraints, options, reports, or running XST from the command line, see the *XST User Guide*. This guide is available in the collection of software manuals and is accessible from ISE by selecting **Help > Software Manuals**, or from the web at http://www.xilinx.com/support/sw_manuals/xilinx8/.

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. In the Processes tab, right-click the **Synthesize** process and select **Properties**.
3. Check the **Write Vendor Constraint File** box.
4. Click **OK** to accept these values.
5. 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 tab, and selecting **Process > Run**.

Processes available in Synplify and Synplify Pro synthesis include:

- **View Synthesis Report**
Lists the synthesis optimizations that were performed on the design and gives a brief timing and mapping report.
- **View RTL Schematic**
Accessible from the Launch Tools hierarchy, this process displays Synplify or Synplify Pro with a schematic view of your HDL code
- **View Technology Schematic**
Accessible from the Launch Tools hierarchy, this process displays Synplify or Synplify Pro with a schematic view of your HDL code mapped to the primitives associated with the target technology.

Examining Synthesis Results

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

- "Compiler Report"
- "Mapper Report"
- "Timing Report"
- "Resource Utilization"

Compiler Report

The compiler report lists each HDL file that was compiled, names which file is the top level, and displays the syntax checking result for each file that was compiled. The report also lists FSM extractions, inferred memory, warnings on latches, unused ports, and removal of redundant logic.

Note: Black boxes (modules not read into a design environment) are always noted as unbound in the Synplify reports. As long as the underlying netlist (`.ngo`, `.ngc` or `.edn`) for a black box exists in the project directory, the implementation tools merge the netlist into the design during the Translate phase.

Mapper Report

The mapper report lists the constraint files used, the target technology, and attributes set in the design. The report lists the mapping results of flattened instances, extracted counters, optimized flip-flops, clock and buffered nets that were created, and how FSMs were coded.

Timing Report

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

Performance Summary

Worst slack in design: -1.581

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack
stopwatch clk_divider.clk_100_inferred_clock	305.3 MHz	259.5 MHz	3.276	3.854	-0.578
stopwatch dcm1_inst.CLKFX_BUF_derived_clock	111.6 MHz	94.9 MHz	8.959	10.540	-1.581

Figure 2-22: Synplify’s Estimated Timing Data

Resource Utilization

This section of the report lists all of the resources that Synplify uses for the given target technology.

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

To continue with the HDL flow:

- Go to [Chapter 4, “Behavioral Simulation,”](#) to perform a pre-synthesis simulation of this design.
OR
- Proceed to [Chapter 5, “Design Implementation,”](#) to place and route the design.

Synthesizing the Design Using Precision Synthesis

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 Precision Synthesis include:

- **Check Syntax**
Checks the syntax of the HDL code.
- **View Log File**

Lists the synthesis optimizations that were performed on the design and gives a brief timing and mapping report.

- **View RTL Schematic**

Accessible from the Launch Tools hierarchy, this process displays Precision with a schematic-like view of your HDL code

- **View Technology Schematic**

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

Entering Synthesis Options through ISE

Synthesis options enable you to modify the behavior of the synthesis tool to optimize according to the needs of the design. For the tutorial, the default property settings will be used.

1. Select `stopwatch.vhd` (or `stopwatch.v`) in the Sources tab.
2. Double-click the **Synthesize** process in the Processes tab.

The RTL/Technology Viewer

Precision Synthesis can generate a schematic representation of the HDL code that you have entered. A schematic view of the code helps you analyze your design by seeing a graphical connection between the various components that Precision has inferred. To launch the design in the RTL viewer, double-click the **View RTL Schematic** process. The following figure displays the design in an RTL view.

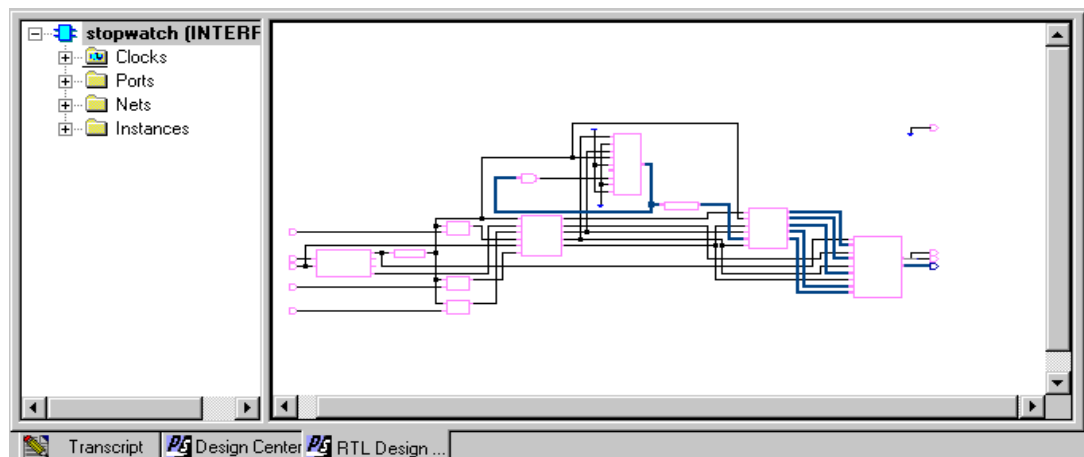


Figure 2-23: Stopwatch Design in Precision Synthesis RTL Viewer

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

To continue with the HDL flow:

- Go to [Chapter 4, “Behavioral Simulation,”](#) to perform a pre-synthesis simulation of this design.
OR
- Proceed to [Chapter 5, “Design Implementation,”](#) to place and route 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 the 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 stopwatch design targets a Spartan™-3A device; however, all of the principles and flows taught are applicable to any Xilinx® device family, unless otherwise noted.

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 HDL macros. After the design is successfully entered in the Schematic Editor, you will perform behavioral simulation ([Chapter 4, “Behavioral Simulation”](#)), run implementation with the Xilinx Implementation Tools ([Chapter 5, “Design Implementation”](#)), perform timing simulation ([Chapter 6, “Timing Simulation”](#)), and configure and download to the Spartan-3 demo board (see [Chapter 7, “iMPACT Tutorial.”](#)).

Getting Started

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

Required Software

You must have Xilinx ISE 9.1i installed to follow this tutorial. For this design you must install the Spartan-3A libraries and device files.

A schematic design flow is supported on Windows, Solaris, and Linux platforms.

This tutorial assumes that the software is installed in the default location, at `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 9.1i Installation Guide and Release Notes*.

Installing the Tutorial Project Files

The tutorial project files can be downloaded to your local machine from <http://www.xilinx.com/support/techsup/tutorials/tutorials9.htm>.

Download the Watch Schematic Design Files (`wtut_sch.zip`). The download contains two directories:

- `wtut_sc\`
(Contains source files for schematic tutorial. The schematic tutorial project will be created in this directory).
- `wtut_sc\wtut_sc_completed\`
(Contains the completed design files for the schematic-based tutorial design, including schematic, HDL, and State Machine files. Do not overwrite files under this directory.)

Unzip the tutorial design files in any directory with read-write permissions. The schematic tutorial files are copied into the directories when you unzip the files. This tutorial assumes that the files are unarchived under `c:\xilinx\ISEexamples`. If you restore the files to a different location, substitute `c:\xilinx\ISEexamples` with the project path.

Starting the ISE Software

To launch the ISE software package:

1. Double-click the **ISE Project Navigator** icon on your desktop, or select **Start > All Programs > Xilinx ISE 9.1i > Project Navigator**.



Figure 3-1: Project Navigator Desktop Icon

Creating a New Project

Two methods are provided for creating a new project: Using the New Project Wizard, and Using a Tcl Script.

Creating a New Project: Using New Project Wizard

1. From Project Navigator, select **File > New Project**. The New Project Wizard appears.

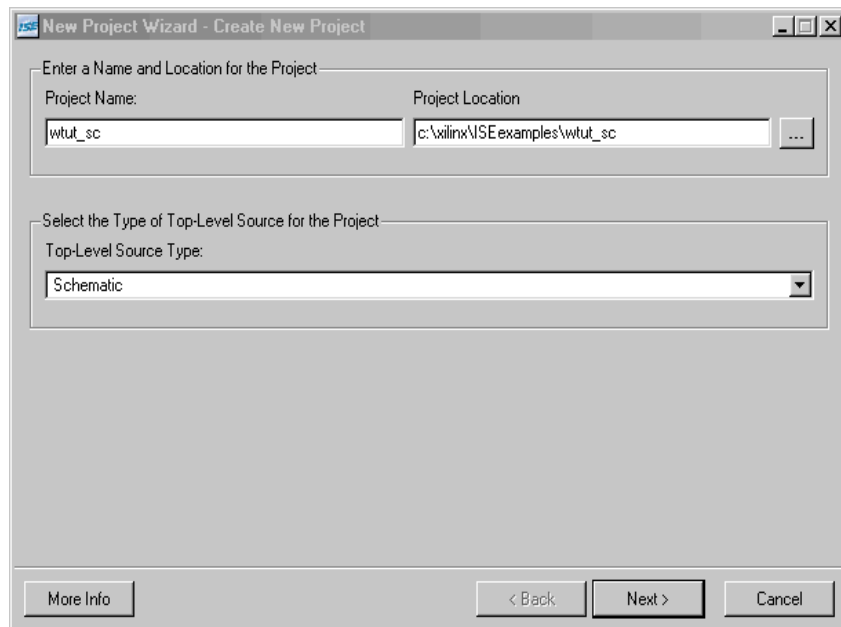


Figure 3-2: New Project Wizard - Create New Project

2. Type `wtut_sc` as the Project Name. Notice that `wtut_sc` is appended to the Project Location value.
3. Browse to `c:\xilinx\ISEexamples` or enter the directory in the Project Location field.
4. Select **Schematic** as the Top-Level Source Type, and then click **Next**.

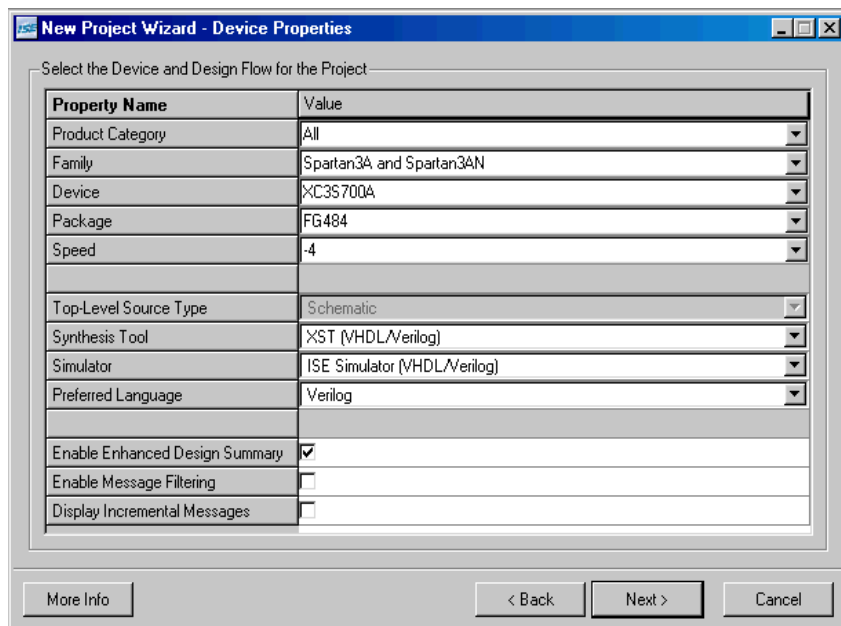


Figure 3-3: New Project Wizard - Device Properties

5. Select the following values in the New Project Wizard - Device Properties window:
 - ◆ Product Category: **All**
 - ◆ Family: **Spartan3A and Spartan3AN**
 - ◆ Device: **XC3S700A**
 - ◆ Package: **FG484**
 - ◆ Speed: **-4**
 - ◆ Synthesis Tool: **XST (VHDL/Verilog)**
 - ◆ Simulator: **ISE Simulator (VHDL/Verilog)**
 - ◆ Preferred Language: **VHDL** or **Verilog** depending on preference. This will determine the default language for all processes that generate HDL files.
6. Click **Next** twice, and then click **Add Source** in the New Project Wizard - Add Existing Sources window.
7. Browse to `c:\xilinx\ISEexamples\wtut_sc`.
8. Select the following files and click **Open**.
 - ◆ `cd4rled.sch`
 - ◆ `ch4rled.sch`
 - ◆ `clk_div_262k.vhd`
 - ◆ `lcd_control.vhd`
 - ◆ `stopwatch.sch`
9. Click **Next**, then **Finish** to complete the New Project Wizard.
10. Verify that all added schematic files are associated with Synthesis/Imp + Simulation, and that the `.dia` file is associated with Synthesis/Implementation Only.
11. Click **OK**.

Creating a New Project: Using a Tcl Script

With Project Navigator open, select the Tcl Shell tab.

1. Change the current working directory to the directory where the tutorial source files were unzipped. (e.g. Type `cd c:/xilinx/ISEexamples/wtut_sc`).
2. Type `source create_wtut_sc.tcl`.

Stopping the Tutorial

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

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 schematic-based modules, a CORE Generator module, a state machine module, an Architecture Wizard module, and HDL modules.

The runner's stopwatch design begins as an unfinished design. Throughout the tutorial, you will complete the design by creating some of the modules and by completing others from existing files. A schematic of the completed stopwatch design is shown in the

following figure. Through the course of this chapter, you will create these modules, instantiate them, and then connect them.

After the design is complete, you will simulate the design to verify its functionality. For more information about simulating your design, see [Chapter 4, “Behavioral Simulation.”](#)

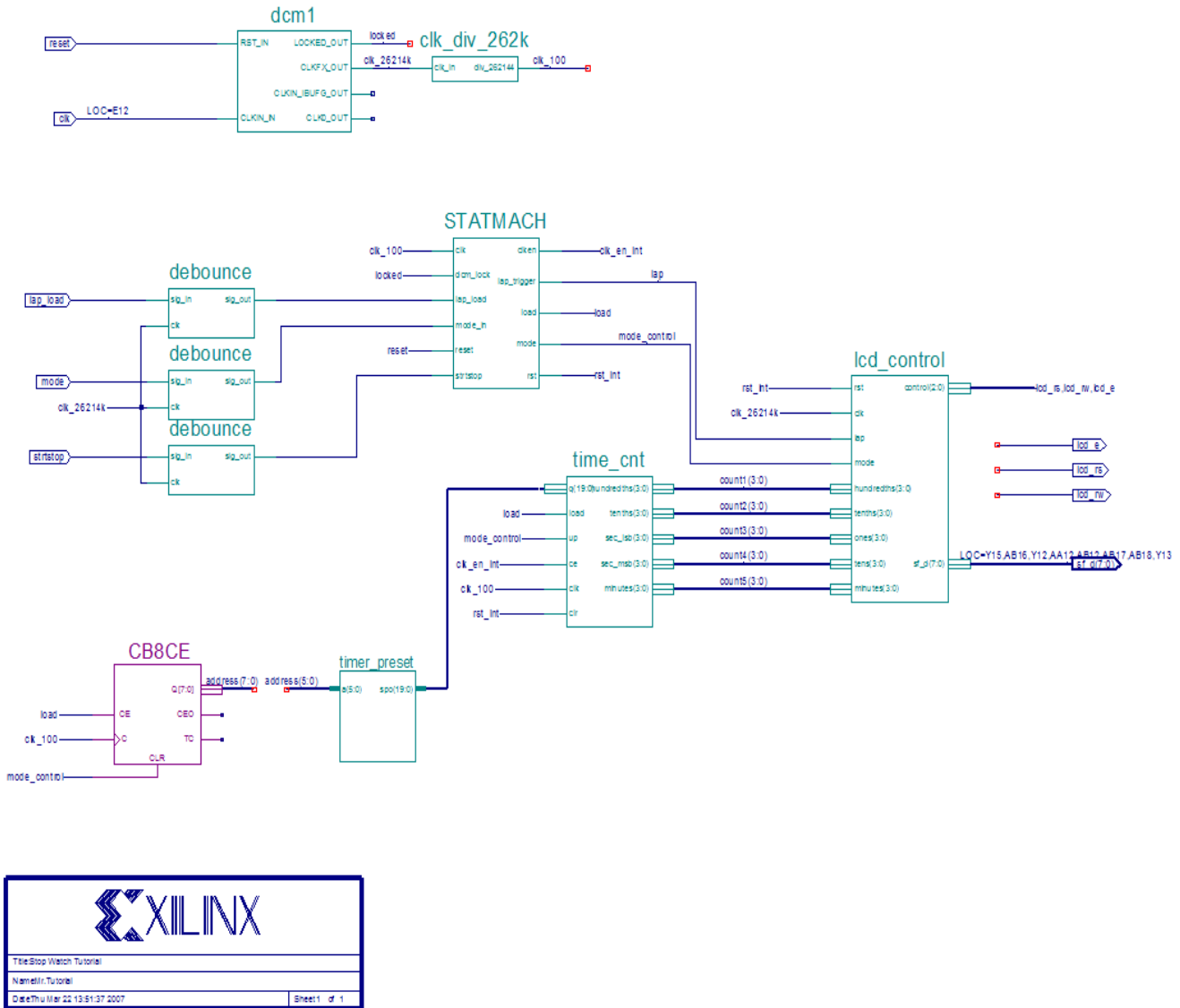


Figure 3-4: Completed Watch Schematic

There are five external inputs and four external outputs in the completed design. The following sections summarize the inputs and outputs, and their respective functions.

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**
Puts the stopwatch in clocking mode and resets the time to 0:00:00.
- **clk**
Externally generated system clock.
- **mode**
Toggles between clocking and timer modes. This input is only functional while the clock or timer is not counting.
- **lap_load**
This is a dual function signal. In clocking mode it displays the current clock value in the 'Lap' display area. In timer mode it will load the pre-assigned values from the ROM to the timer display when the timer is not counting.

Outputs

The following are outputs signals for the design.

- **lcd_e, lcd_rs, lcd_rw**
These outputs are the control signals for the LCD display of the Spartan-3A demo board used to display the stopwatch times.
- **sf_d[7:0]**
Provides the data values for the LCD display.

Functional Blocks

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

The completed design consists of the following functional blocks.

- **clk_div_262k**
Macro which divides a clock frequency by 262,14. Converts 26.2144 MHz clock into 100 Hz 50 %duty cycle clock.
- **dcm1**
Clocking Wizard macro with internal feedback, frequency controlled output, and duty-cycle correction. The CLKFX_OUT output converts the 50 MHz clock of the Spartan-3A demo board to 26.2144 MHz.
- **debounce**
Module implementing a simplistic debounce circuit for the strtstop, mode, and lap_load input signals.
- **lcd_control**
Module controlling the initialization of and output to the LCD display.
- **statmach**

State Machine module defined and implemented in State Diagram Editor. Controls the state of the stopwatch.

- **timer_preset**

CORE Generator™ 64X20 ROM. This macro contains 64 preset times from 0:00:00 to 9:59:99 which can be loaded into the timer.

- **time_cnt**

Up/down counter module which counts between 0:00:00 to 9:59:99 decimal. This macro has five 4-bit outputs, which represent the digits of the stopwatch time.

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 you will connect the macros together to create the completed stopwatch design. All procedures used in the tutorial can be used later for your own designs.

Opening the Schematic File in the Xilinx Schematic Editor

The stopwatch schematic available in the `wtut_sc` project is incomplete. In this tutorial, you will update the schematic in the Schematic Editor. After you have created the project in ISE, you can now open the `stopwatch.sch` file for editing. To open the schematic file, double-click `stopwatch.sch` in the Sources window.

The stopwatch schematic diagram opens in the Project Navigator Workspace. You will see the unfinished design with elements in the lower right corner as shown in the figure below.

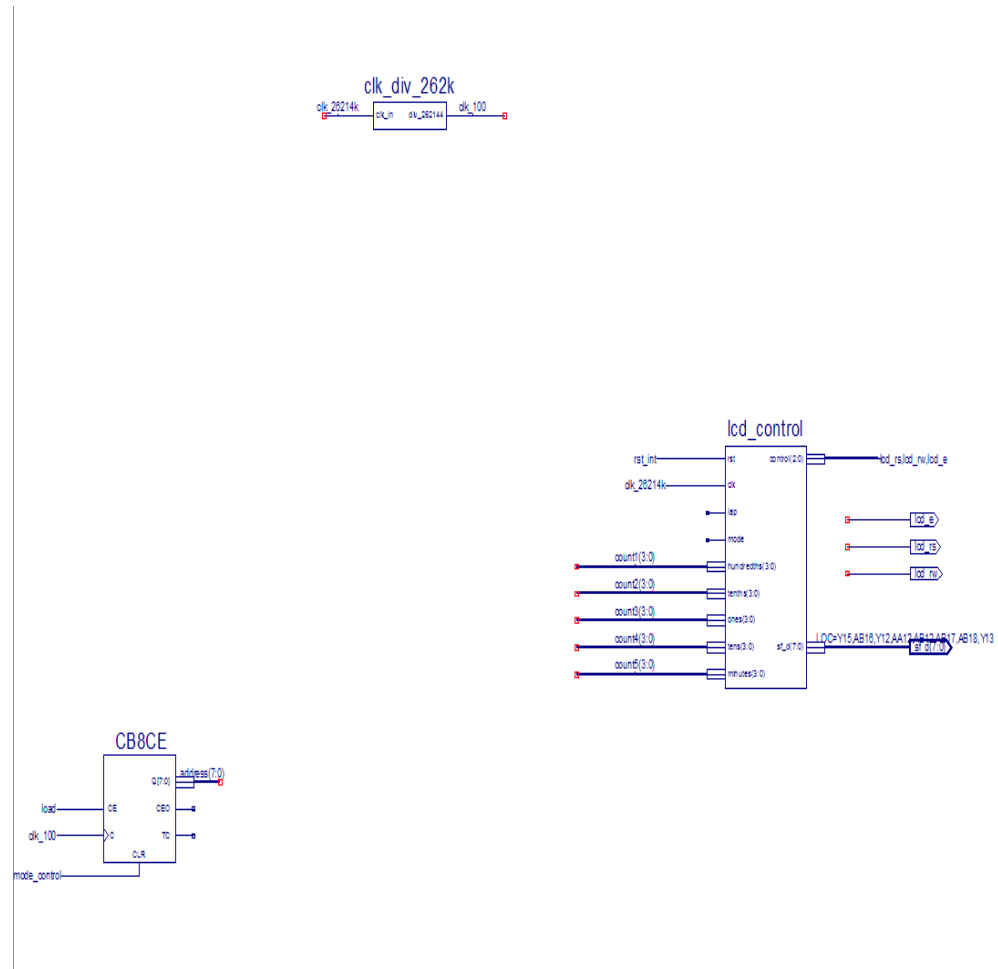


Figure 3-5: Incomplete Stopwatch Schematic

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.

The schematic window can be undocked from the Project Navigator framework by selecting **Window > Float** while the schematic is selected in the workspace.

After being undocked, the schematic window can be redocked by selecting **Window > Dock**.

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. The corresponding symbol or schematic file can then be generated automatically.

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

The macro you will create is called `time_cnt`. This macro is a binary counter with five, 4-bit outputs, representing the digits of the stopwatch.

To create a schematic-based macro:

1. In Project Navigator, select **Project > New Source**. The New Source dialog box opens:

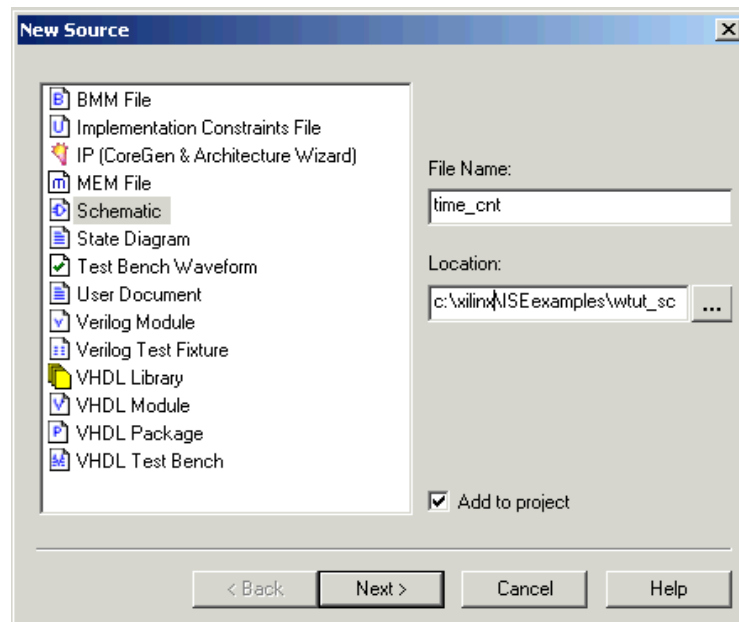


Figure 3-6: New Source Dialog Box

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

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

A new schematic called **time_cnt** is created, added to the project, and opened for editing.

5. Change the size of the schematic sheet by doing the following.
 - ◆ Right-click on the schematic page and select **Object Properties**.
 - ◆ Click on the down arrow next to the sheet size value and select **D = 34 x 22**.
 - ◆ Click **OK** and then click **Yes** to acknowledge that changing the sheet size cannot be undone with the **Edit > Undo** option.

Defining the `time_cnt` Schematic

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

Adding I/O Markers

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

To add the I/O markers:

1. Select **Tools > Create I/O Markers**.
The Create I/O Markers dialog box opens.
2. In the Inputs box, enter **q(19:0),load,up,ce,clk,clr**.
3. In the Outputs box, enter **hundredths(3:0),tenths(3:0),sec_lsb(3:0),sec_msb(3:0),minutes(3:0)**.

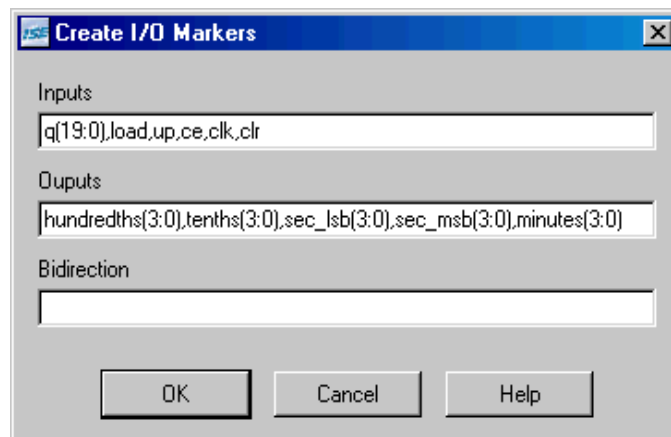


Figure 3-7: Creating I/O Markers

4. Click **OK**. The eleven I/O markers are added to the schematic sheet.

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

Adding Components to `time_cnt`

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

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

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



Figure 3-8: Add Symbol Icon

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

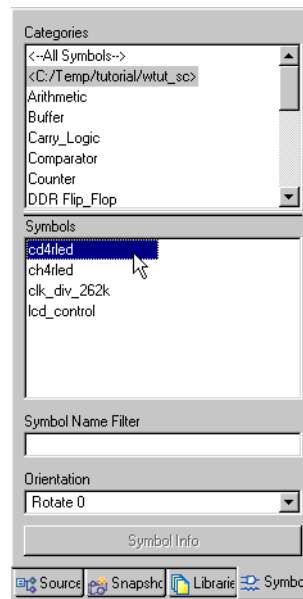


Figure 3-9: Symbol Browser

The first component you will place is a `cd4rled`, a 4-bit, loadable, bi-directional, BCD counter with clock enable and synchronous clear.

2. Select the `cd4rled` component, using one of two ways:
 - ◆ Highlight the project directory category from the Symbol Browser dialog box and select the component **cd4rled** from the symbols list.
 - or
 - ◆ Select **All Symbols** and type **cd4rled** 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 `cd4rled` symbol.
4. Move the symbol outline near the top and center of the sheet 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 component, and then selecting `Ctrl+R`.

5. Place three more cd4rled symbols on the schematic by moving the cursor with attached symbol outline to the desired location, and clicking the left mouse button. See Figure 3-10.

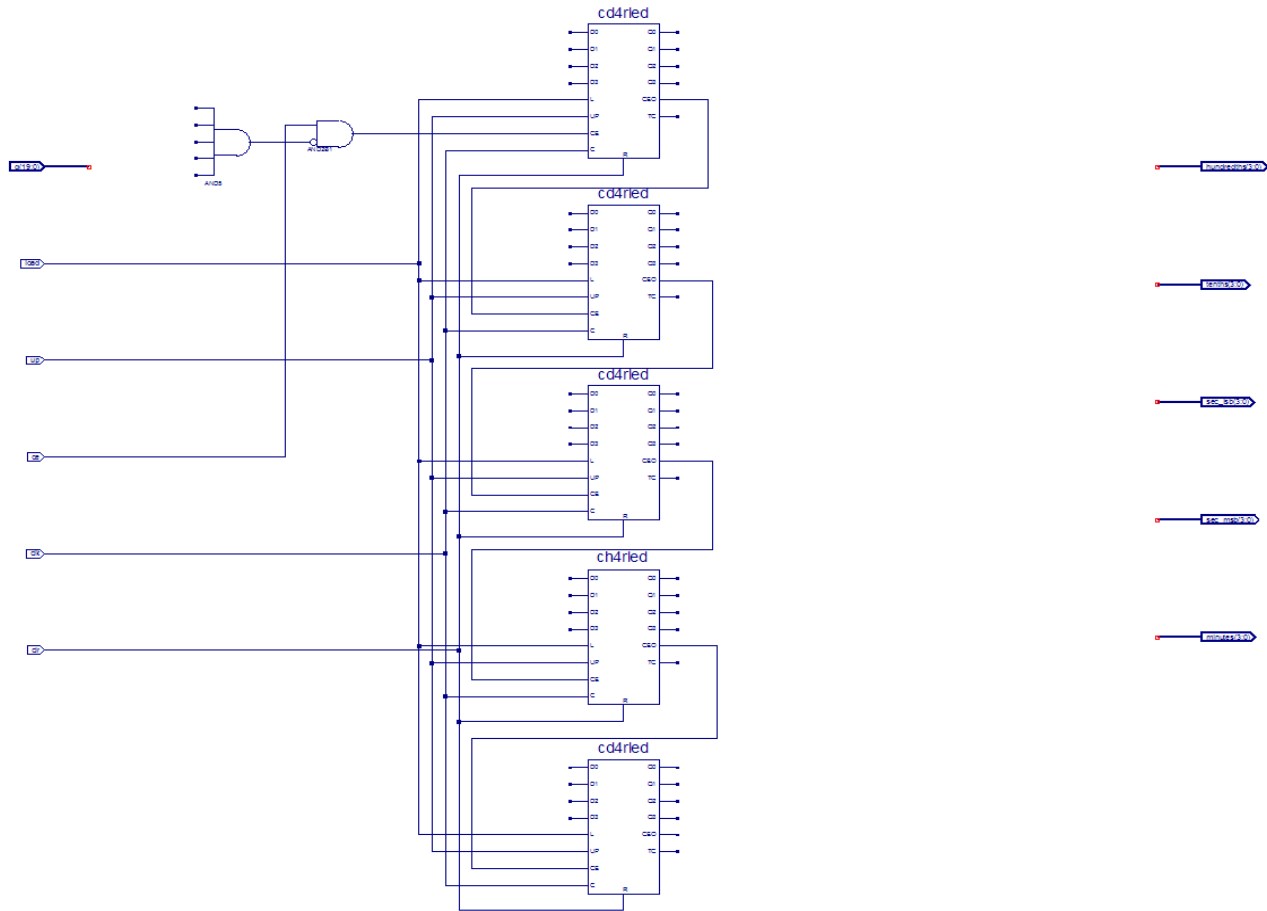


Figure 3-10: Partially Completed time_cnt Schematic

6. Follow the procedure outlined in steps 1 through 4 above to place the following components on the schematic sheet:
 - AND2b1
 - ch4rled
 - AND5

Refer to Figure 3-10 for placement locations.

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

For a detailed description of the functionality of Xilinx Library components, right-click on the component and select **Object Properties**. In the Object Properties window, select **Symbol Info**. Symbol information is also available in the Libraries Guides, accessible from the collection of software manuals on the web at http://www.xilinx.com/support/sw_manuals/xilinx9/.

Correcting Mistakes

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

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

Delete a placed component in one of two ways:

- Click the component and press the **Delete** key on your keyboard.
- or
- Right-click the component and select **Delete**.

Drawing Wires

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

Perform the following steps to draw a net between the AND2b1 and top cd4rled components on the *time_cnt* schematic.

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



Figure 3-11: Add Wire Icon

2. Click the output pin of the AND2b1 and then click the destination pin CE on the cd4rled component. The Schematic Editor draws a net between the two pins.
3. Draw the nets to connect the output of the AND5 component to the inverted input of the AND2b1 component
4. Connect the scalar input IO markers to each of the five counter blocks and connect the CEO pin of the first four counters to the CE pin of the next counter as shown in [Figure 3-10](#).

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.

Adding Buses

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

The next step is to create three buses for each of the five outputs of the `time_cnt` schematic. The results can be found in the completed schematic.

To add the buses `hundredths(3:0)`, `tenths(3:0)`, `sec_lsb(3:0)`, `sec_msb(3:0)` and `minutes(3:0)` to the schematic, perform the following steps:

1. Select all of the output IO markers by drawing a box around them and then drag the group so that `minutes(3:0)` is below the Q3 output of the bottom counter block.
2. Select **Add > Wire** or click the **Add Wire** icon in the Tools toolbar.
3. Click in the open space just above and to the right of the top `cd4rled` and then click again on the pin of the `hundredths(3:0)` I/O marker. The wire should automatically be drawn as a bus with the name matching that of the I/O marker.
4. To verify this, zoom in. The bus is represented visually by a thicker wire.

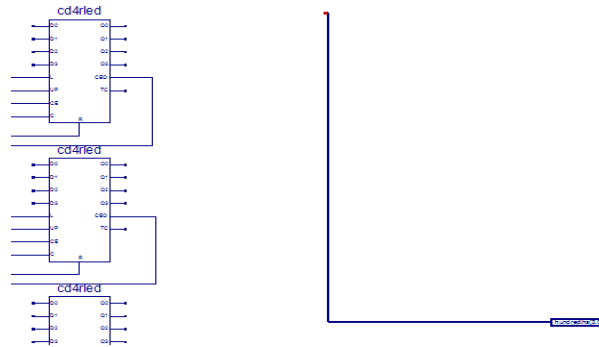


Figure 3-12: Adding a Bus

5. Repeat Steps 2 through 4 for the four remaining buses.
6. After adding the five buses, press **Esc** or right-click at the end of the bus to exit the Add Wire mode.

Adding Bus Taps

Next, add nets to attach the appropriate pins from the `cd4rled` and `ch4rled` 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 enables 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 3-13: Add Bus Tap Icon

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

2. From the Options tab to the left of the schematic, choose the **--< Right** orientation for the bus tap.
3. Click on the `hundredths(3:0)` bus with the center bar of the cursor.

The **Selected Bus Name** and the **Net Name** values of the options window are now populated.

Note: The indexes of the Net Name may be incremented or decremented by clicking the arrow buttons next to the Net Name box.

4. With hundredths(3) as the Net Name value, move the cursor so the tip of the attached tap touches the Q3 pin of the top cd4rled component.

Note: Four selection squares appear around the pin when the cursor is in the correct position.

5. Click once when the cursor is in the correct position.

A tap is connected to the hundredths(3:0) bus and a wire named hundreths(3) is drawn between the tap and the Q3 pin.

Click successively on pins Q2, Q1, and Q0 to create taps for the remaining bits of the hundredths(3:0) bus.

6. Repeat Steps 3 to 6 to tap off four bits from each of the five buses.

Note: It is the name of the wire that makes the electrical connection between the bus and the wire (e.g sec_msb(2) connects to the third bit of sec(3:0)). The bus tap figure is for visual purposes only. The following section shows additional electrical connections by name association.

7. Press **Esc** to exit the Add Net Name mode.
8. Compare your time_cnt schematic with [Figure 3-15](#) to ensure that all connections are made properly.

Adding Net Names

First, add a hanging wire to each of the five inputs of the AND5 component and to the TC pin of each of the counter blocks.

Next, add net names to the wires. To add the net names:

1. Select **Add > Net Name** or click the **Add Net Name** icon in the Tools toolbar.



Figure 3-14: Add Net Name Icon

2. Type `tc_out0` in the Name box and select **Increase the Name** in the Add Net Names Options dialog box.

The net name `tc_out0` is now attached to the cursor.

3. Click the net attached to the first input of the AND5 component.

The name is then attached to the net. The net name appears above the net if the name is placed on any point of the net other than an end point.

4. Click on the remaining input nets of the AND5 to add `tc_out1`, `tc_out2`, `tc_out3` and `tc_out4`.

The Schematic Editor increments the net Name as each name is placed on a net.

Alternatively, name the first net `tc_out4` and select **Decrease the name** in the Add Net Names Options dialog box, and nets are named from the bottom up.

5. Repeat step 2 and then click successively on the nets connected to the TC output to add `tc_out0`, `tc_out1`, `tc_out2`, `tc_out3`, and `tc_out4` to these nets.

Note: Each of the wires with identical names are now electrically connected. In this case, the nets do not need to be physically connected on the schematic to make the logical connection.

Finally, connect the input pins of the counters through net name association.

1. Select **Add > Wire** or click the **Add Wire** icon and add a hanging net to the four data pins of each of the five counters.
2. Select **Add > Net Name** or click the **Add Net Name** icon in the Tools toolbar.
3. Type **q(0)** in the Name box of the Add Net Name options dialog box.
4. Select **Increase the name** in the Add Net Name options dialog box.

The net name q(0) is now attached to the cursor.

5. Click successively on each of the nets connected to data inputs, starting from the top so that the net named q(0) is attached to the D0 pin of the top counter and the net named q(19) is attached to the D3 pin of the bottom counter. Refer to [Figure 3-15](#).

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

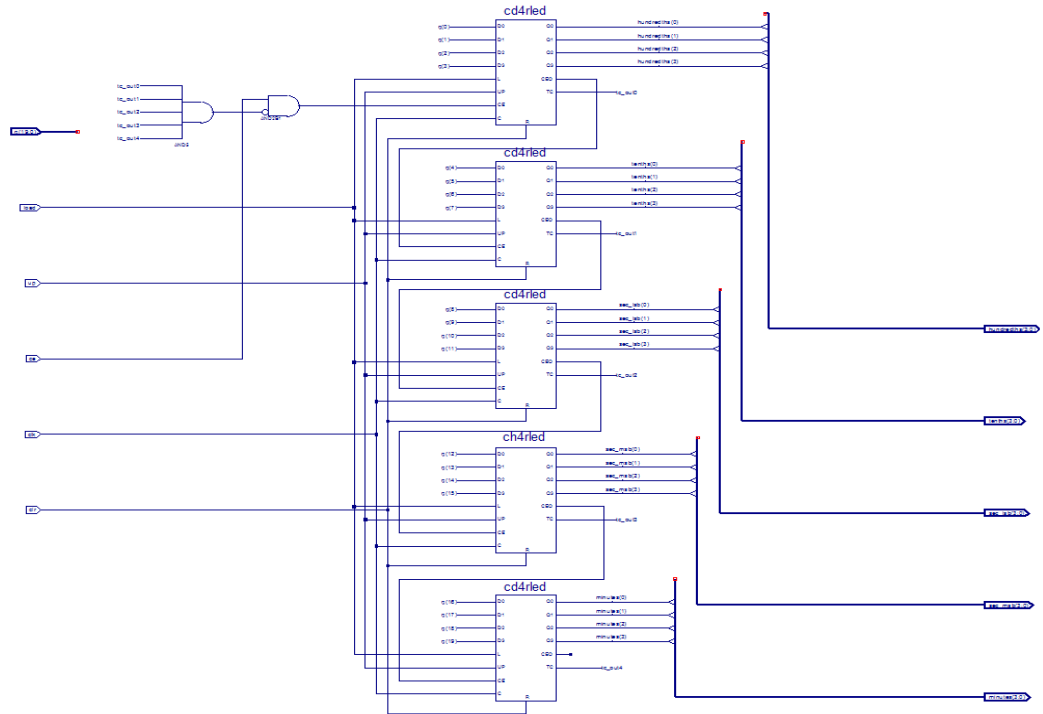


Figure 3-15: Completed time_cnt Schematic

Checking the Schematic

The time_cnt schematic is now complete.

Verify that the schematic does not contain logical errors by running a design rule check (DRC). To do this, select **Tools > Check Schematic**. The Console window should report that no errors or warnings are detected. If an error or warning is displayed, fix the reported problem before proceeding.

Saving the Schematic

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



Figure 3-16: **Save Icon**

2. Close the `time_cnt` schematic.

Creating and Placing the `time_cnt` Symbol

The next step is to create a “symbol” that represents the `time_cnt` macro. The symbol is an instantiation of the macro. After you create a symbol for `time_cnt`, you will add the symbol to a top-level schematic of the stopwatch design. In the top-level schematic, the symbol of the `time_cnt` macro will be connected to other components in a later section in this chapter.

Creating the `time_cnt` symbol

You can create a symbol using either a Project Navigator process or a Tools menu command.

To create a symbol that represents the `time_cnt` schematic using a Project Navigator process:

1. In the Sources window, select `time_cnt.sch`.
2. In the Processes window, click the **+** beside Design Utilities to expand the hierarchy.
3. Double-click **Create Schematic Symbol**.

To create a symbol that represents the `time_cnt` schematic using a Tools menu command:

1. With the `time_cnt` schematic sheet open, select **Tools > Symbol Wizard**.
2. In the Symbol Wizard, select **Using Schematic**, and then select `time_cnt` in the schematic value field.

Click **Next**, then **Next**, then **Next** again, and then **Finish** to use the wizard defaults.

3. View and then close the `time_cnt` symbol.

Placing the `time_cnt` Symbol

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

1. In the Sources window, double-click `stopwatch.sch` to open the schematic.
2. Select the **Add Symbol** icon.



Figure 3-17: **Add Symbol Icon**

3. In the Symbol Browser, select the local symbols library (c:\xilinx\ISEexamples\wtut_sc), and then select the newly created time_cnt symbol.
4. Place the time_cnt symbol in the schematic so that the output pins line up with the five buses driving inputs to the lcd_control component. This should be close to grid position [1612,1728]. Grid position is shown at the bottom right corner of the Project Navigator window, and is updated as the cursor is moved around the schematic.
Note: Do not worry about connecting nets to the input pins of the time_cnt symbol. You will do this after adding other components to the stopwatch schematic.
5. Save the changes and close stopwatch.sch.

Creating a CORE Generator Module

CORE Generator is a graphical interactive design tool that enables you to create high-level modules such as memory elements, math functions, communications, and IO interface cores. 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, SRL16s, and distributed and block RAM.

In this section, you will create a CORE Generator module called timer_preset. The module is used to store a set of 64 values to load into the timer.

Creating a CORE Generator Module

To create a CORE Generator module:

1. In Project Navigator, select **Project > New Source**.
2. Select **IP (Coregen & Architecture Wizard)**.
3. Type **timer_preset** in the File name field.
4. Click **Next**.
5. Double-click **Memories & Storage Elements**, then double-click **RAMs & ROMs**.
6. Select **Distributed Memory Generator**, then click **Next** and click **Finish** to open the Distributed Memory Generator customization GUI. This customization GUI enables you to customize the memory to the design specifications.
7. Fill in the Distributed Memory Generator customization GUI with the following settings:
 - ◆ Component Name: **timer_preset** - Defines the name of the module.
 - ◆ Depth: **64** - Defines the number of values to be stored
 - ◆ Data Width: **20** - Defines the width of the output bus.
 - ◆ Memory Type: **ROM**
 - ◆ Click **Next**.
 - ◆ Leave Input and Output options as Non Registered; Click **Next**.

- ◆ Coefficients File: Click the Browse button and select definition1_times.coe.

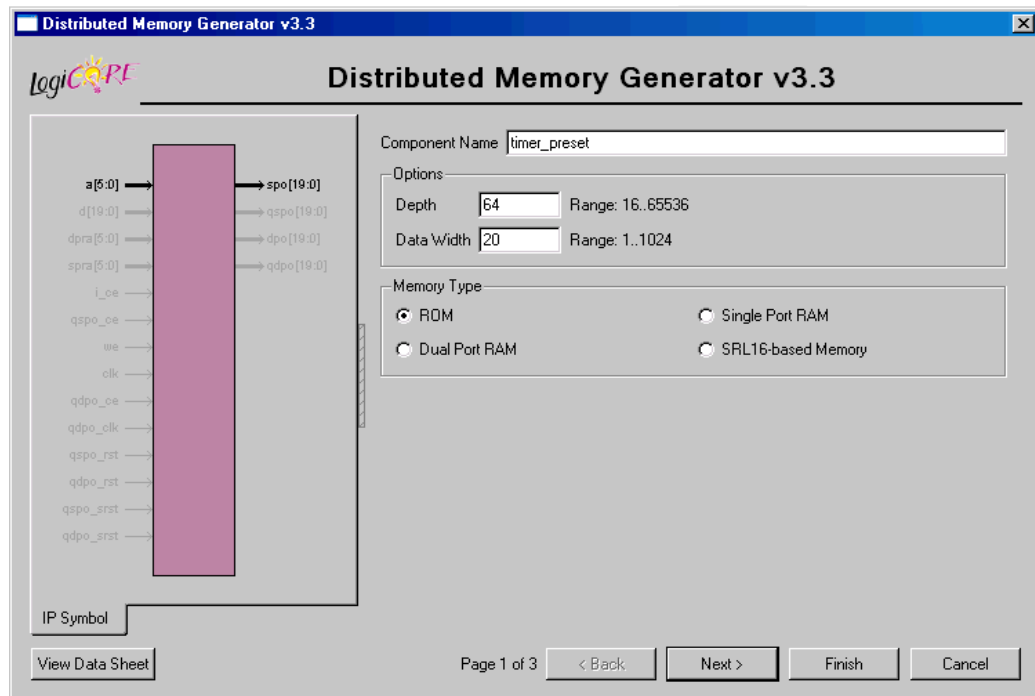


Figure 3-18: CORE Generator - Distributed Memory Generator Customization GUI

8. Check that *only* the following pins are used (used pins are highlighted on the symbol on the left side of the customization GUI):
 - ◆ **a[5:0]**
 - ◆ **spo[19:0]**
9. Click **Finish**.

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

Note: A number of files are added to the project directory. Some of these files are:

- ◆ **timer_preset.sym**
This file is a schematic symbol file.
- ◆ **timer_preset.vhd** or **timer_preset.v**
These are HDL wrapper files for the core and are used only for simulation.
- ◆ **timer_preset.ngc**
This file is the netlist that is used during the Translate phase of implementation.
- ◆ **timer_preset.xco**
This file stores the configuration information for the timer_preset module and is used as a project source.
- ◆ **timer_preset.mif**
This file provides the initialization values of the ROM for simulation.

Creating a State Machine Module

With the State Diagram Editor, you can graphically create finite state machines that include states, inputs/outputs, and state transition conditions. Transition conditions and state actions are typed into the diagram using language independent syntax. The State Diagram Editor then exports the diagram to either VHDL, Verilog or ABEL code. The resulting HDL file is finally synthesized to create a netlist and a macro symbol is created 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, synthesize the module into a macro and place it on the stopwatch schematic.

Note: The State Diagram Editor is only available for Windows operating systems. A completed VHDL and Verilog file for the State Machine diagram in this section has been provided for you in the `wtut_sc\wtut_sc_completed` directory. To skip this section, copy either `statmach.v` or `stamach.vhd` from the `c:\xilinx\ISEexamples\stut_sc\wtut_sc_completed` directory to the `c:\xilinx\ISEexamples\stut_sc` directory, then proceed to [“Creating the State Machine Symbol.”](#)

To open the partially complete diagram, first add the `statmach.dia` file to the project by selecting **Project > Add Source** and selecting `statmach.dia`. Then, double-click `statmach.dia` in the Sources tab. The State Diagram Editor is launched with the partially completed state machine diagram.

In the incomplete state machine diagram below:

- The circles represent the various states.
- The black expressions are the transition conditions, defining how you move between states.
- The output expressions for each state are found in the circles representing the states.
- The transition conditions and the state actions are written in language-independent syntax and are then exported to Verilog, VHDL, or ABEL.

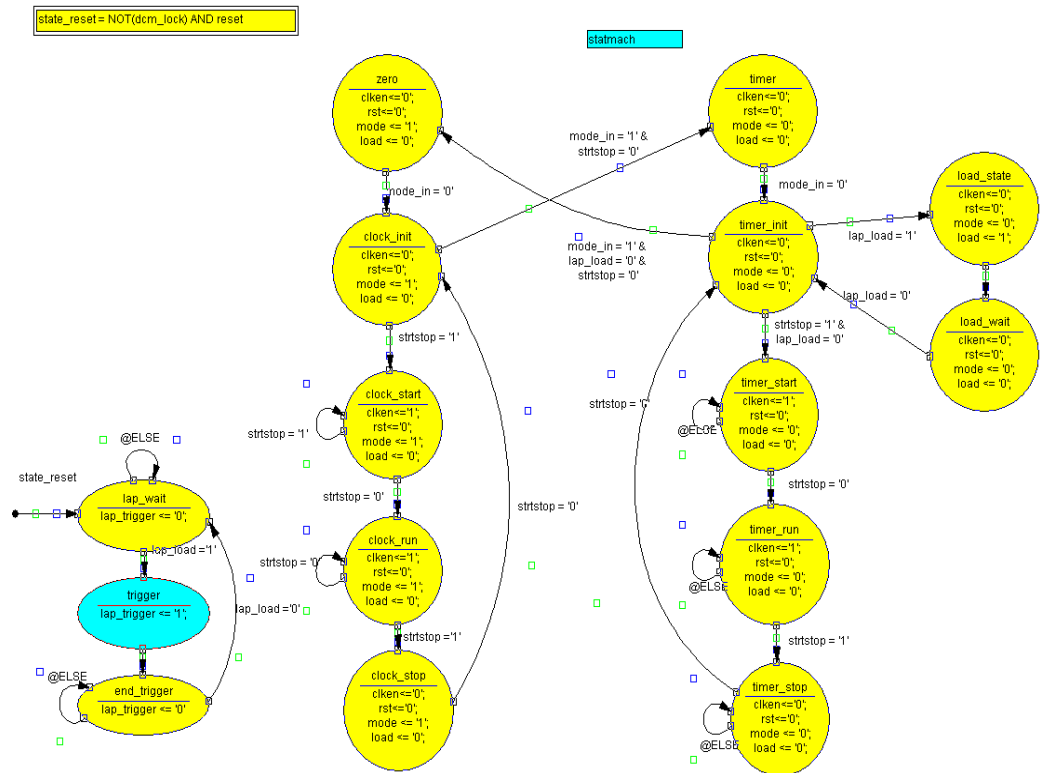


Figure 3-19: Incomplete State Machine Diagram

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

Adding New States

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

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



Figure 3-20: 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 3-21.
3. Click the mouse to place the state bubble.

The state is given the default name, STATE0.

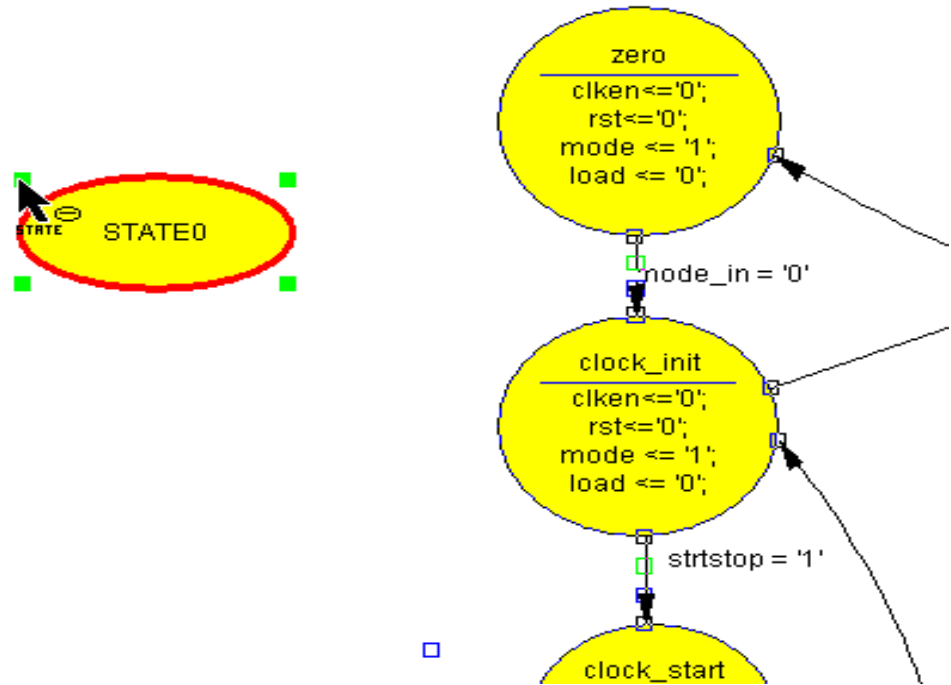


Figure 3-21: Adding the CLEAR State

4. 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.
5. Click **OK**.

To change the shape of the state bubble, click one of the four squares surrounding 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 add 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 Transition** icon in the vertical toolbar.



Figure 3-22: Add Transitions Icon

2. Click on the **clear** state to start the transition.
3. Click on the **zero** state to complete the transition arrow.

- To manipulate the arrow's shape, click and drag it in any directory.

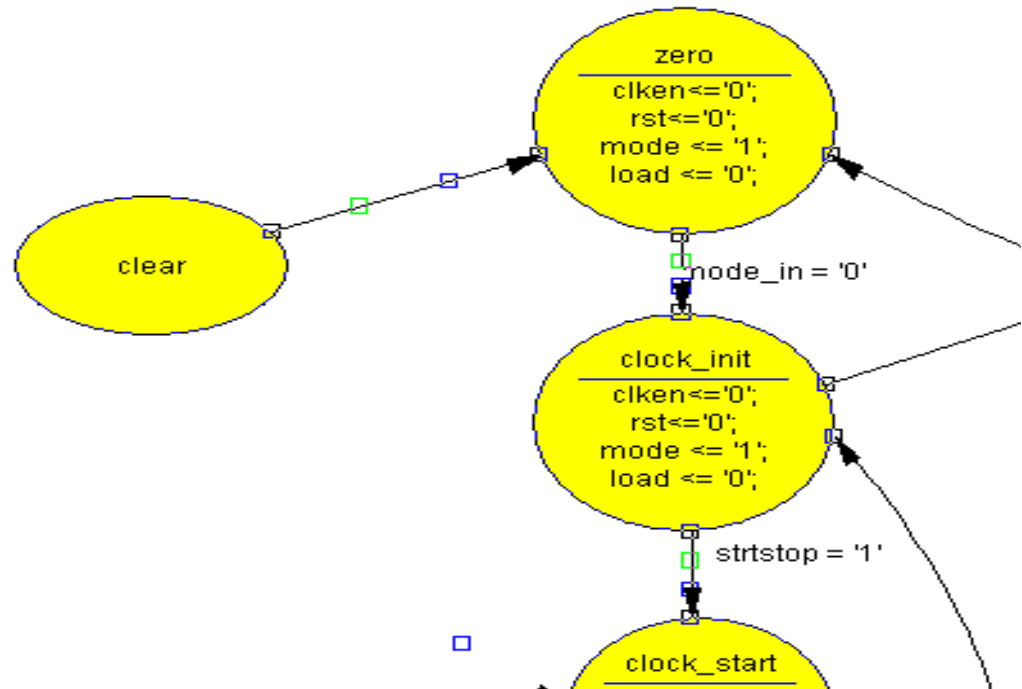


Figure 3-23: Adding State Transition

- Click the **Select Objects** (or **Pointer**) icon in the vertical toolbar to exit the Add Transition mode.



Figure 3-24: Select Objects Icon

Adding a State Action

A state action dictates how the outputs should behave in a given state. You will add a state action to the *clear* state to drive the RST output to 1.

To add a state action:

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

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

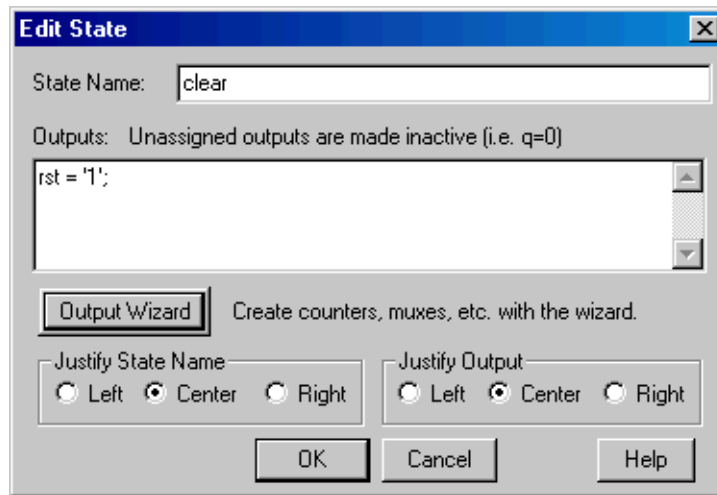


Figure 3-25: Edit State Dialog Box

2. Select the **Output Wizard** button.
3. In the Logic Wizard, enter the following values:
`DOUT = rst, CONSTANT = '1';`
4. Click **OK** to enter each individual value.
5. Click **OK** to exit the Edit State dialog box. The output is now added to the state.

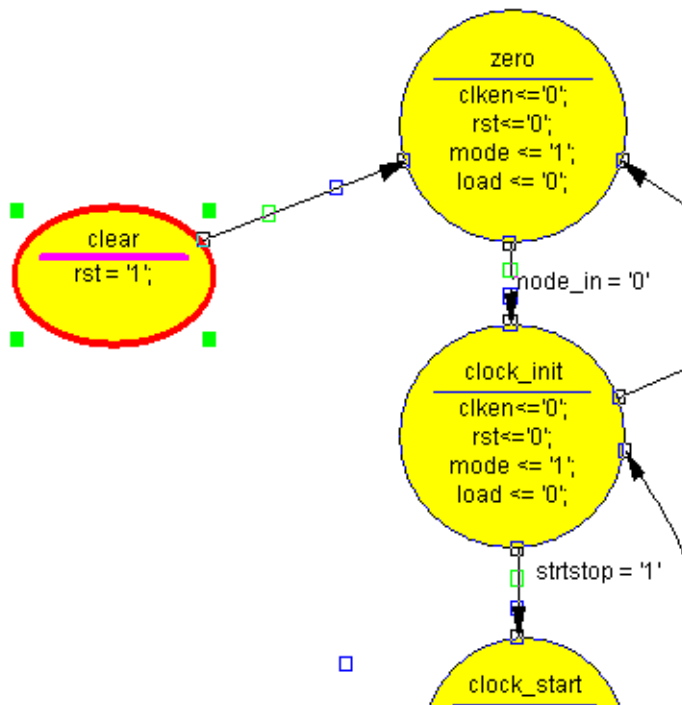


Figure 3-26: Adding State Outputs

Adding a State Machine Reset Condition

Using the State Machine Reset feature, 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 that sends the state machine to the *clear* state whenever either the reset signal is asserted high, or the DCM_lock signal is de-asserted low.

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



Figure 3-27: Add Reset Icon

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

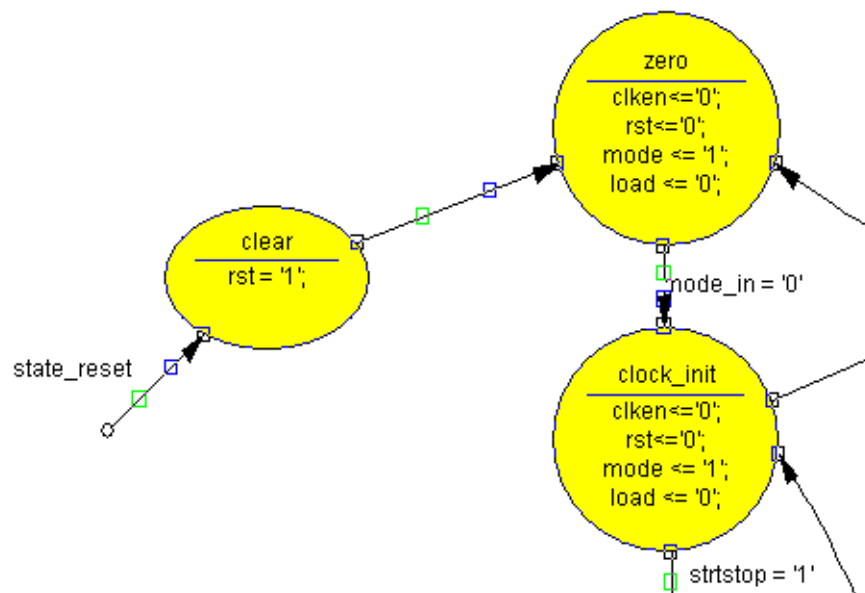


Figure 3-28: Adding a Reset Condition

Creating the State Machine HDL output file

1. Select **Options > Configuration**.
2. In the Language section, select either **Verilog** or **VHDL** as desired.
3. In the Language Vendor section select **Xilinx XST**.
4. Click **OK**.
5. To generate the HDL file, select the **Generate HDL** icon in the toolbar.
6. A Results window is displayed listing the compile status. Click **OK**.
7. A browser opens displaying the generated HDL file. Examine the code and then close the browser.

8. Save your changes by selecting **File > Save**.
9. Close the State Diagram Editor.

Creating the State Machine Symbol

In this section, you will create a macro symbol for the state machine that you can place on the stopwatch schematic. The macro symbol is added to the project library.

1. In Project Navigator, select **Project > Add Source**.
2. Select `statmach.v` or `statmach.vhd` (*this is the HDL file generated by State Diagram Editor*) and click **Open**.

The file `statmach.v` (`hd`) is added to the project in Project Navigator.

3. In the Sources tab, select `statmach.vhd` or `statmach.v`.
4. In the Processes tab, click the **+** beside Design Utilities to expand the hierarchy.
5. Double-click **Create Schematic Symbol**.

Creating a DCM Module

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

Using the Clocking Wizard

Create the `dcm1` module as follows:

1. Select **Project > New Source**.
2. In the New Source dialog box, select the **IP (Coregen & Architecture Wizard)** source type, and type the filename `dcm1`.
3. Click **Next**.
4. In the Select IP dialog box, select **FPGA Features and Design > Clocking > Spartan-3E, Spartan-3A > Single DCM SP**.

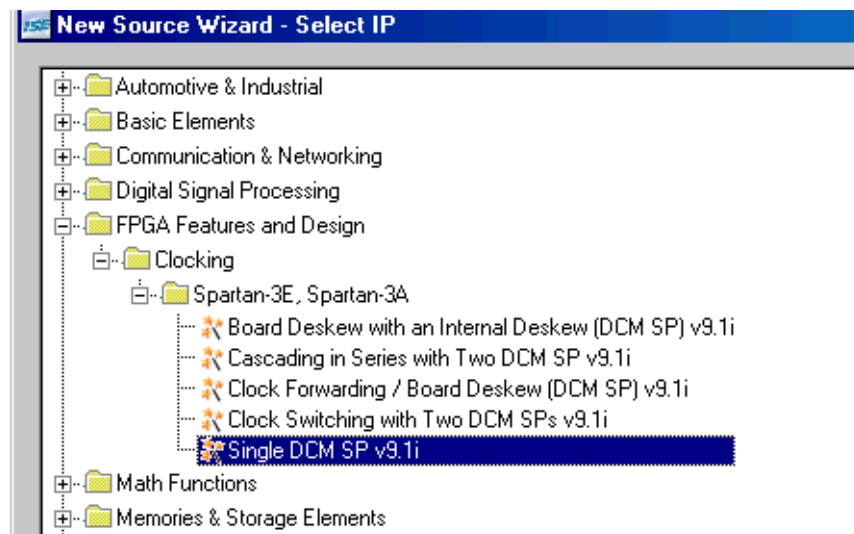


Figure 3-29: Selecting Single DCM Core Type

5. Click **Next**, then click **Finish**. The Clocking Wizard is launched.
6. Verify that **RST**, **CLK0** and **LOCKED** ports are selected.
7. Select **CLKFX** port.
8. Type **50** and select **MHz** for the Input Clock Frequency.
9. Verify the following settings:
 - ◆ Phase Shift: **NONE**
 - ◆ CLKIN Source: **External, Single**
 - ◆ Feedback Source: **Internal**
 - ◆ Feedback Value: **1X**
 - ◆ Use Duty Cycle Correction: Selected

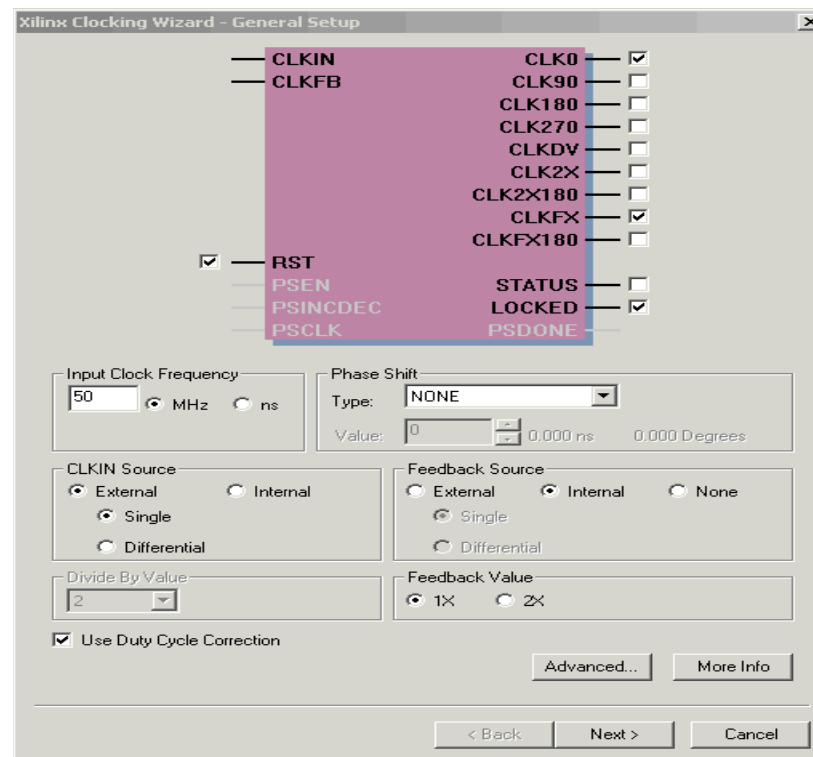


Figure 3-30: Xilinx Clocking Wizard - General Setup

10. Click the **Advanced** button.
11. Select the **Wait for DCM Lock before DONE Signal goes high** option.
12. Click **OK**.
13. Click **Next** twice to advance to the Clock Frequency Synthesizer page.
14. Select **Use output frequency** and type **26.2144** in the box and select **MHz**.

$$(26.2144\text{MHz}) / 2^{18} = 100\text{Hz}$$

15. Click **Next**, and then click **Finish**.

The dcm1 .xaw file is created and added to the list of project source files in the Sources tab.

Creating the dcm1 Symbol

Next, create a symbol representing the dcm1 macro. This symbol will be added to the top-level schematic (`stopwatch.sch`) later in the tutorial.

1. In Project Navigator, in the Sources tab, select `dcm1.xaw`.
2. In the Processes tab, double-click **Create Schematic Symbol**.

Creating an HDL-Based Module

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

You will author a new HDL module. This macro will be used to debounce the `strtstop`, `mode` and `lap_load` inputs.

Using the New Source Wizard and ISE Text Editor

In this section, you create a file using the New Source wizard, specifying the name and ports of the component. The resulting HDL file is then modified in the ISE Text Editor.

To create the source file:

1. Select **Project > New Source**.
A dialog box opens in which you specify the type of source you want to create.
2. Select **VHDL Module** or **Verilog Module**.
3. In the File Name field, type `debounce`.
4. Click **Next**.
5. Enter two input ports named `sig_in` and `clk` and an output port named `sig_out` for the `debounce` component as follows:
 - a. In the first three Port Name fields type `sig_in`, `clk` and `sig_out`.
 - b. Set the Direction field to **input** for `sig_in` and `clk` and to **output** for `sig_out`.
 - c. Leave the Bus designation boxes unchecked.

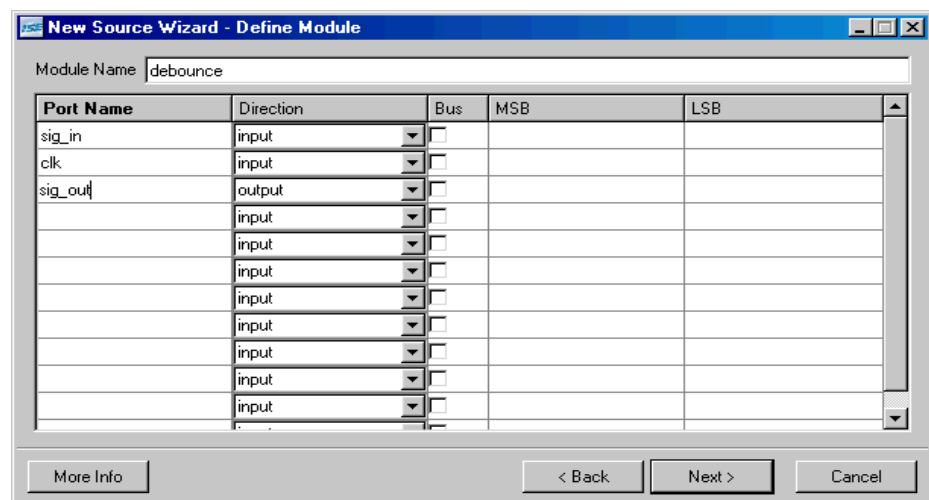


Figure 3-31: New Source Wizard for Verilog

6. Click **Next** to complete the Wizard session.
A description of the module displays.
7. Click **Finish** to open the empty HDL file in the ISE Text Editor.

The VHDL file is displayed in [Figure 3-32](#). The Verilog HDL file is displayed in [Figure 3-33](#).

```

12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  -----
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity debounce is
31      Port ( sig_in : in  STD_LOGIC;
32            clk     : in  STD_LOGIC;
33            sig_out : out STD_LOGIC);
34  end debounce;
35
36  architecture Behavioral of debounce is
37
38  begin
39
40

```

Figure 3-32: VHDL File in ISE Text Editor

```

1  -----
2  //////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    14:12:53 03/15/2007
7  // Design Name:
8  // Module Name:   debounce
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////
21 module debounce(sig_in, clk, sig_out);
22     input sig_in;
23     input clk;
24     output sig_out;
25
26
27 endmodule
28

```

Figure 3-33: Verilog File in ISE Text Editor

In the ISE Text Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are displayed in blue, comments in green, and values are black. The file is color-coded to enhance readability and help you recognize 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 Debounce Circuit template for this exercise.

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

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

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

Each HDL language in the Language Templates is divided into five sections: Common Constructs, Device Primitive Instantiation, Simulation Constructs, Synthesis Constructs and User Templates. To expand the view of any of these sections, click the **+** next to the section. Click any of the listed templates to view the template contents in the right pane.

2. Under either the VHDL or Verilog hierarchy, expand the Synthesis Constructs hierarchy, expand the Coding Examples hierarchy, expand the Misc hierarchy, and select the template called One-Shot, Debounce Circuit. Use the appropriate template for the language you are using.

When the template is selected in the hierarchy, the contents display in the right pane.

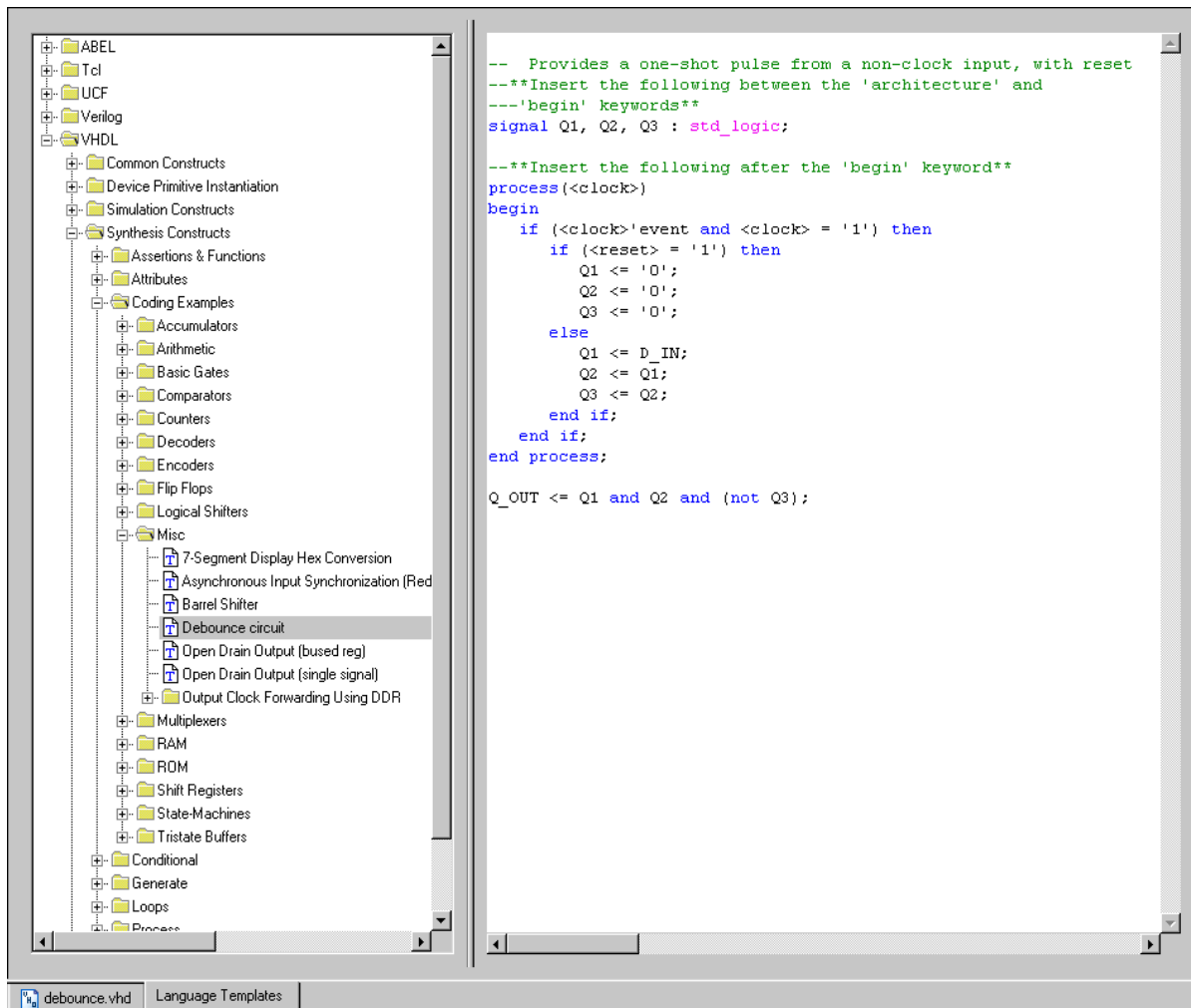


Figure 3-34: Language Templates

Adding a Language Template to Your File

You will now use the drag and drop method for adding templates to your HDL file. Refer to “Working with Language Templates” in the ISE Help for additional usability options.

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

1. Select **Window > Tile Vertically** to show both the HDL file and the Language Templates window.
2. Click and drag the **Debounce Circuit** name from the Language Template topology into the `debounce.vhd` file under the `architecture begin` statement, or into the `debounce.v` file under the `module` and pin declarations.
3. Close the Language Templates window.
4. (Verilog only) Complete the Verilog module by doing the following:
 - a. Remove the reset logic (not used in this design) by deleting the three lines beginning with `if` and ending with `else`.
 - b. Change `<reg_name>` to `q` in all six locations.

- c. Change `<clock>` to `clk`; `<input>` to `sig_in`; and `<output>` to `sig_out`.
5. (VHDL only) Complete the VHDL module by doing the following:
 - a. Move the line beginning with the word **signal** so that it is between the **architecture** and **begin** keywords.
 - b. Remove the reset logic (not used in this design) by deleting the five lines beginning with `if (<reset>...` and ending with `else` and delete one of the `end if;` lines.
 - c. Change `<clock>` to `clk`; `D_IN` to `sig_in`; and `Q_OUT` to `sig_out`.
 You now have complete and functional HDL code.
6. Save the file by selecting **File > Save**.
7. Select debounce in the Sources tab.
8. In the Processes tab, double-click **Check Syntax**.
9. Close the ISE Text Editor.

Creating the debounce Symbol

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

1. In the Sources tab, select `debounce.vhd` or `debounce.v`.
2. In the Processes tab, click the **+** beside Design Utilities to expand the hierarchy.
3. Double-click **Create Schematic Symbol**.

You are now ready to place the debounce symbol on the stopwatch schematic.

Placing the statmach, timer_preset, dcm1 and debounce Symbols

You can now place the `statmach`, `timer_preset`, `dcm1`, and `debounce` symbols on the stopwatch schematic (`stopwatch.sch`). In Project Navigator, double-click `stopwatch.sch`. The schematic file opens in the Workspace.

1. Select **Add > Symbol** or click the **Add Symbol** icon from the Tools toolbar.



Figure 3-35: Add Symbol Icon

This opens the Symbol Browser to the left of the Schematic Editor, which displays the libraries and their corresponding components.

2. View the list of available library components in the Symbol Browser.
3. Locate the project-specific macros by selecting the project directory name in the Categories window.
4. Select the appropriate symbol, and add it to the stopwatch schematic in the approximate location, as shown in [Figure 3-36](#).

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

5. Save the schematic.

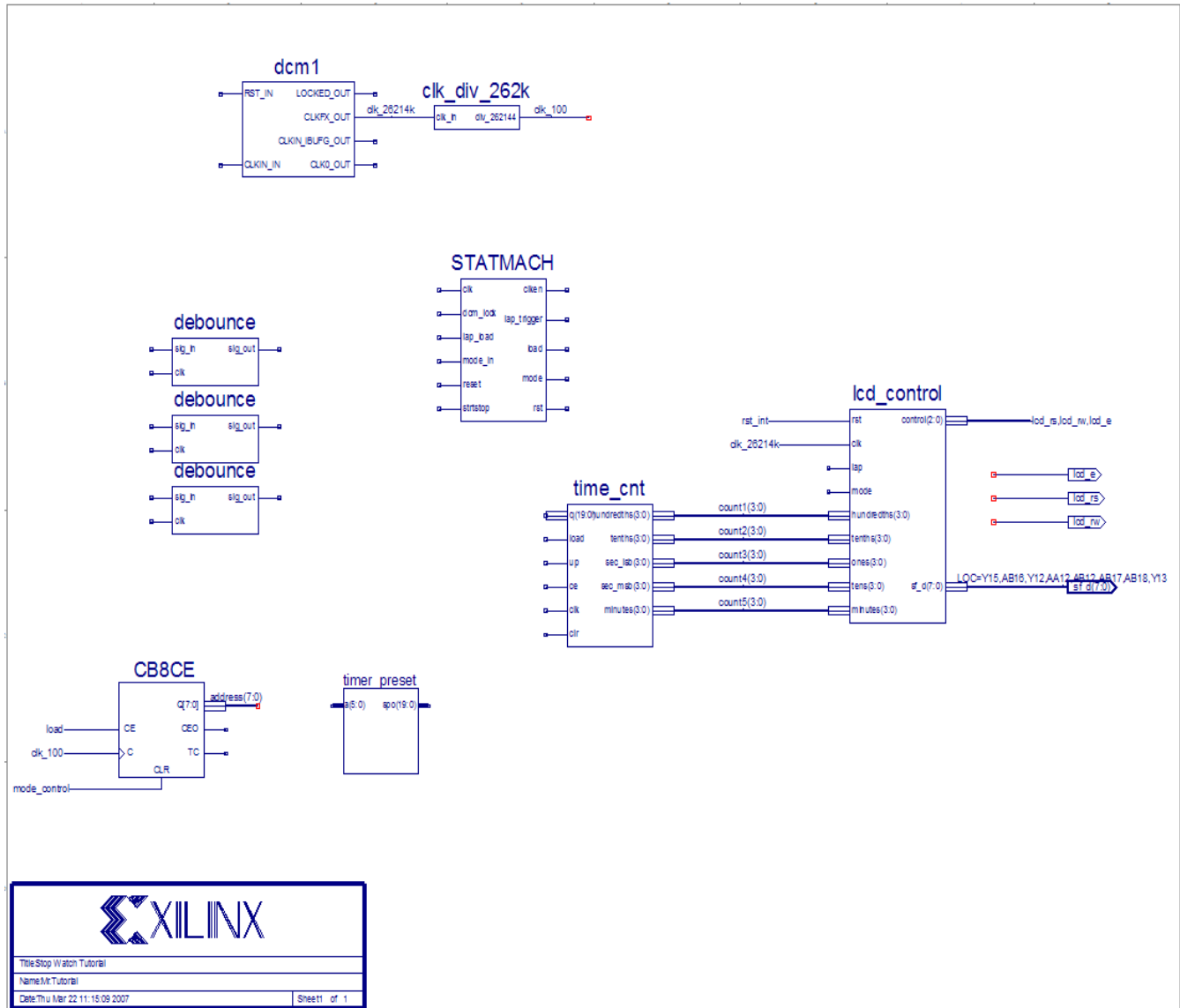


Figure 3-36: Placing Design Macros

Changing Instance Names

When a symbol is placed on a schematic sheet it is given a unique instance name beginning with the prefix XLXI_. To help make the hierarchy more readable in the Project Navigator Sources window, change the names of the added symbol instances as follows.

1. Right-click on the `dcm1` symbol instance and select **Object Properties** from the right-click menu
2. Change the value of the `InstName` field to `dcm_inst` and then click **OK**.

Repeat steps 1 and 2 to change the following symbol instance names.

- Name the `statmach` instance `timer_state`.

- Name the top debounce instance `lap_load_debounce`
- Name the middle debounce instance `mode_debounce`.
- Name the bottom debounce instance `strtstop_debounce`.
- Name the timer_preset instance `t_preset`.
- Name the time_cnt instance `timer_cnt`.

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 `time_cnt` macro, which is a schematic-based macro created earlier in this tutorial, and examine its components.

To push down into `time_cnt` from the top-level, stopwatch, schematic:

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



Figure 3-37: Hierarchy Push Icon

In the `time_cnt` schematic, you see five counter blocks. Push into any of the counter blocks by selecting the block and clicking on the **Hierarchy Push** icon. This process may be repeated until the schematic page contains only Xilinx primitive components. If a user pushes into a symbol that has an underlying HDL or IP core file, the appropriate text editor or customization GUI will open and be ready to edit the file.

2. After examining the macro, return to the top-level schematic by selecting **View > Pop to Calling Schematic**, or select the **Hierarchy Pop** icon when nothing in the schematic is selected. You can also right-click in an open space of the schematic and select **Pop to Calling Schematic**.



Figure 3-38: Hierarchy Pop Icon

Specifying Device Inputs/Outputs

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

Adding Input Pins

Next, add five input pins to the stopwatch schematic: `reset`, `clk`, `lap_load`, `mode` and `strtstop`.

To add these components:

- Draw a hanging wire to the two inputs of `dcm1` and to the `sig_in` pin of each debounce symbol

Refer to “[Drawing Wires](#)” for detailed instructions.

Adding I/O Markers and Net Names

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

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

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

1. Select **Add > Net Name**.
2. Type **reset** into the Name box.
The net name is now attached to the cursor.
3. Place the name on the leftmost end of the net, as illustrated in [Figure 3-39](#).
4. Repeat Steps 1 through 3 for the clk, lap_load, mode, and strtstop pins.
Once all of the nets have been labeled, add the I/O marker.
5. Select **Add > I/O Marker**.
6. Click and drag a box around the name of the five labeled nets, as illustrated in [Figure 3-39](#), to place an input port on each net.

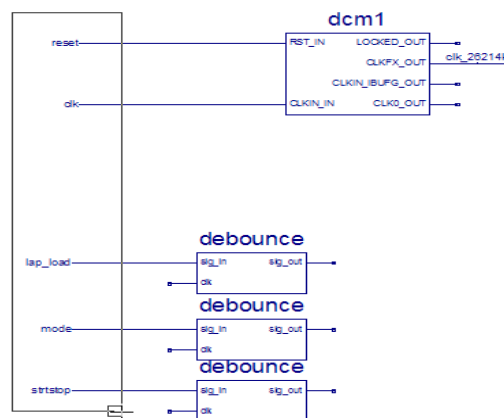


Figure 3-39: Adding I/O Markers to Labeled Nets

Assigning Pin Locations

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

For this tutorial, the inputs and outputs will be locked to specific pins in order to place and download the design to the Spartan-3A demo board. Because the tutorial stopwatch 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.

Assign a LOC parameter to the output nets on the stopwatch schematic, as follows:

1. Right-click on the clk net and select **Object Properties** from the right-click menu.
2. Click the **New** button under Net Attributes to add a new property.
3. Enter **LOC** for the Attribute Name and **E12** for the Attribute Value.
4. Click **OK** to return to the Object Properties dialog box.

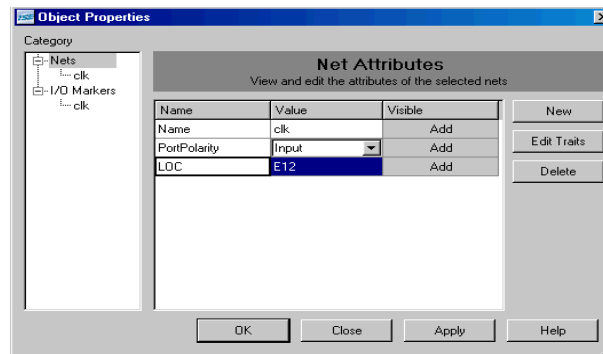


Figure 3-40: Assigning Pin Locations

5. To make the LOC attribute visible, select the **Add** button adjacent to the LOC attribute in the Attribute window.
6. In the Net Attribute Visibility window, select **Add** and then **OK**.

This will display the LOC attribute on the schematic above the clk net.

Click **OK** to close the Object properties window.

The above procedure constrains clk to pin E12. Notice that the LOC property has already been added to the sf_d(7:0) bus. The remaining pin location constraints will be added in “Using the Constraints Editor” and “Using the Floorplan Editor” of Chapter 5, “Design Implementation”.

Note: To turn off the Location constraint without deleting it, select the loc attribute, and click **Edit Traits**. Select **VHDL or Verilog** and deselect **Write this attribute**.

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. You may also want to use the completed schematic shown below to complete the schematic. Each of the actions referred

to in this section has been discussed in detail in earlier sections of the tutorial. Please see the earlier sections for detailed instructions.

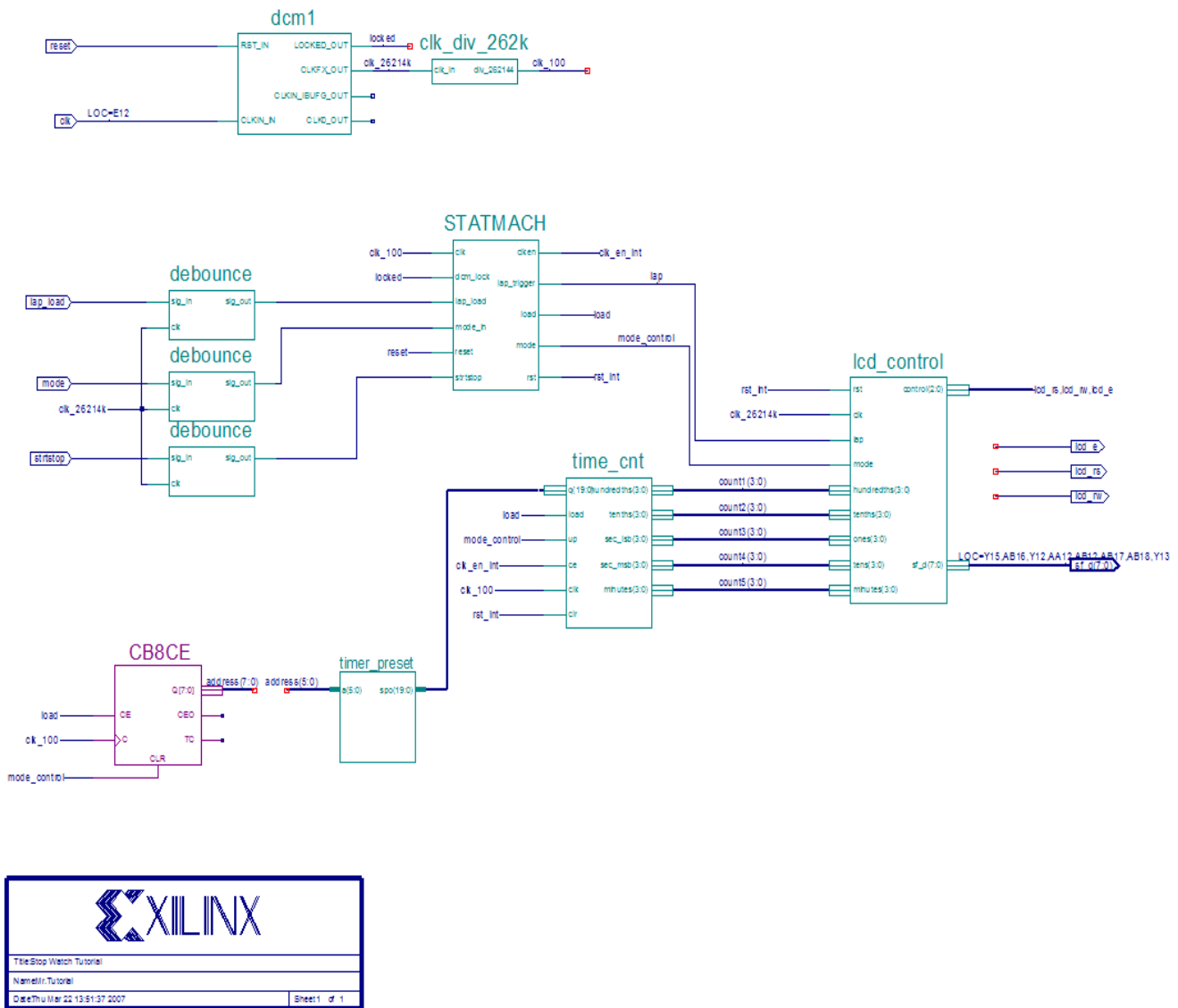


Figure 3-41: Completed Stopwatch Schematic

To complete the schematic diagram:

1. Draw a hanging wire to the LOCKED_OUT pin of DCM1 and name the wire **locked**. See “Drawing Wires” and “Adding Net Names.”
2. Draw a hanging wire to the clk input of both the time_cnt and statmach macros. (See “Drawing Wires.”)

3. Name both wires `clk_100`. (See [“Adding 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 `clk_100` and several other signals.
Draw a wire to connect the `clk` inputs of the three debounce macros and name the wire `clk_26214k`.
4. Draw wires between the `sig_out` pins of the debounce components and the `lap_load`, `mode_in` and `strtstop` pin of the `statmach` macro. Label the nets `ll_debounced`, `mode_debounced`, and `strtstop_debounced`. See [“Drawing Wires”](#) and [“Adding Net Names.”](#)
5. Add hanging wires to the `dcm_lock` pin and the `reset` pin of the `statmach` macro. Name them `locked` and `reset`, respectively.
6. Draw a hanging wire to the `clken` output of the `statmach` component and another hanging wire to the `ce` pin of the `time_cnt` component. Name both wires `clk_en_int`.
7. Draw hanging wires from the `rst` output pin of the `STMACH` macro and the `clr` pin of the `time_cnt` macro. See [“Drawing Wires.”](#) Label both wires `rst_int`.
8. Draw a wire from the bus output of the `timer_preset` to the `q(19:0)` input of the `time_cnt` macro. See [“Drawing Wires.”](#) Notice how the wire is automatically converted to a bus.
9. Draw a hanging bus on the input of the `timer_preset` macro and name the bus `address(5:0)`.
10. Draw wires from the `lap_trigger` and `mode` outputs of the `statmach` macro to the `lap` and `mode` inputs of the `lcd_control` macro. See [“Drawing Wires.”](#) Name the nets `lap` and `mode_control` respectively.
11. Draw hanging wires from the `load` output of the `statmach` macro and the `load` input of the `time_cnt` macro. See [“Drawing Wires.”](#) Name both wires `load`.
12. Draw a hanging wire to the `up` input `time_cnt` macro. See [“Drawing Wires.”](#) Name the wire `mode_control`.

The schematic is now complete.

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

You have now completed the schematic design.

To continue with the schematic flow, do the following:

- Go to [Chapter 4, “Behavioral Simulation,”](#) to perform a pre-synthesis simulation of this design.
- Proceed to [Chapter 5, “Design Implementation,”](#) to place and route the design.

Behavioral Simulation

This chapter contains the following sections.

- “Overview of Behavioral Simulation Flow”
- “ModelSim Setup”
- “ISE Simulator Setup”
- “Getting Started”
- “Adding an HDL Test Bench”
- “Behavioral Simulation Using ModelSim”
- “Behavioral Simulation Using ISE Simulator”

Overview of Behavioral Simulation Flow

Xilinx® ISE™ provides an integrated flow with the Mentor ModelSim simulator and the Xilinx ISE Simulator that allows simulations to be run from the Xilinx Project Navigator. The examples in this tutorial demonstrate how to use the integrated flow. Whether you use the ModelSim simulator or the ISE Simulator with this tutorial, you will achieve the same results.

For additional information about simulation, and for a list of other supported simulators, see Chapter 5 of the *Synthesis and Verification Guide*. This Guide is accessible from within ISE by selecting **Help > Software Manuals**, and from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/.

ModelSim Setup

In order to use this tutorial, you must install ModelSim on your computer. The following sections discuss requirements and setup for ModelSim PE, ModelSim SE, and ModelSim XE.

ModelSim PE and SE

ModelSim PE and ModelSim SE are full versions of ModelSim available for purchase directly from Mentor Graphics. In order to simulate with the ISE 9 libraries, use ModelSim 6.0 or later. Older versions may work but are not supported. For more information about purchasing ModelSim PE or SE version 6.0 or later, contact Mentor Graphics.

ModelSim Xilinx Edition

ModelSim Xilinx Edition III (MXE III) is the Xilinx version of ModelSim which is based on ModelSim PE. There are two versions: a free starter version, and a full version that can be purchased from Xilinx. To obtain the correct MXE III simulation libraries, go to the Download Center page. Be sure to select 9.1i as the ISE version.

http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=mx3_libs

ISE Simulator Setup

ISE Simulator is automatically installed and set up with the ISE 9.1i installer. The ISE Simulator is available on Windows 2000, Windows XP, and Red Hat Enterprise Linux 3 or 4 platforms only.

Getting Started

The following sections outline the requirements for performing behavioral simulation in this tutorial.

Required Files

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

Design Files (VHDL, Verilog, or Schematic)

This chapter assumes that you have completed the design entry tutorial in either [Chapter 2, “HDL-Based Design,”](#) or [Chapter 3, “Schematic-Based Design.”](#) After you have completed one of these chapters, your design includes the required design files, and is ready for simulation.

Test Bench File

In order to simulate the design, a test bench file is required to provide stimulus to the design. VHDL and Verilog test bench files are available with the tutorial files. Or, if you prefer, you can create your own test bench file. For instructions, see [“Creating a Test Bench Waveform Using the Waveform Editor.”](#)

Xilinx Simulation Libraries

Xilinx simulation libraries are required when any Xilinx primitive is instantiated in the design. The design in this tutorial requires the use of simulation libraries because it contains instantiations of a digital clock manager (DCM) and a CORE Generator™ component. For information on simulation libraries and how to compile them, see the next section, [“Xilinx Simulation Libraries.”](#)

Xilinx Simulation Libraries

To simulate designs that contain instantiated Xilinx primitives or CORE Generator components, you must use the Xilinx simulation libraries. These libraries contain models

for each component. These models reflect the functions of each component, and provide the simulator with the information required to perform simulation.

For a detailed description of each library, see Chapter 5 of the *Synthesis and Verification Design Guide*. This Guide is accessible from within ISE by selecting **Help > Software Manuals**, and from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/.

Updating the Xilinx Simulation Libraries

The Xilinx simulation libraries contain models that are updated on a regular basis.

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

When the models are updated, you must recompile the libraries. The compiled Xilinx simulation libraries are then available during the simulation of any design.

ModelSim PE or SE

If you are using ModelSim PE or SE, you must compile the simulation libraries with the updated models. See Chapter 5 of the *Synthesis and Verification Design Guide*. This Guide is accessible from within ISE by selecting **Help > Software Manuals**, or from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/.

ModelSim Xilinx Edition III

If you are using ModelSim Xilinx Edition III (MXE III), the updated models are precompiled. They are available on the Xilinx support website. Download the latest precompiled models from the Download Center at http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=mx_e_libs.

Xilinx ISE Simulator

If you are using ISE Simulator, all the simulation libraries are precompiled. Setup is automatically updated with the latest version of the libraries.

Mapping Simulation Libraries in the Modelsim.ini File

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

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

Note: The `modelsim.ini` is not applicable to the ISE Simulator.

ModelSim searches for a `modelsim.ini` file in the following locations until one is found:

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

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

ModelSim PE or SE

If you are using ModelSim PE or SE, refer to the *Synthesis and Verification Design Guide* and use COMPXLIB to compile the libraries. While compiling the libraries, COMPXLIB also

updates the `modelsim.ini` file with the correct mapping. Open the `modelsim.ini` file and make sure that the library mappings are correct.

For future projects, you can copy the `modelsim.ini` file to the working directory and make changes that are specific to that project, or you can use the `MODELSIM` environment variable to point to the desired `modelsim.ini` file.

ModelSim Xilinx Edition III

If you are using ModelSim Xilinx Edition III (MXE III), open the `modelsim.ini` file in the directory where MXE III was installed. All of the Xilinx simulation libraries are already mapped to the proper location.

ISE Simulator

The `modelsim.ini` file is not applicable to the ISE Simulator.

Adding an HDL Test Bench

In order to add an HDL test bench to your design project, you can either add a test bench file provided with this tutorial, or create your own test bench file and add it to your project.

Adding Tutorial Test Bench File

This section demonstrates how to add pre-existing test bench file to the project. A VHDL test bench and Verilog test fixture are provided with this tutorial.

Note: To create your own test bench file in ISE, select **Project > New Source**, and select either **VHDL Test Bench** or **Verilog Text Fixture** in the New Source Wizard. An empty stimulus file is added to your project. You must define the test bench in a text editor.

VHDL Design

After downloading the file to your project directory, add the tutorial VHDL test bench to the project:

1. Select **Project > Add Source**.
2. Select the test bench file `stopwatch_tb.vhd`.
3. Click **Open**.
4. In the Choose Source Type dialog box, select **VHDL Test Bench File**.
5. Click **OK**.

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

Verilog Design

After downloading the file to your project directory, add the tutorial Verilog test fixture to the project:

1. Select **Project > Add Source**.
2. Select the file `stopwatch_tb.v`.
3. Click **Open**.
4. In the Choose Source Type dialog box, select **Verilog Test Fixture File**.
5. Click **OK**.

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

Behavioral Simulation Using ModelSim

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

To simulate with the ISE Simulator, skip to [“Behavioral Simulation Using ISE Simulator.”](#) Whether you choose to use the ModelSim simulator or the ISE Simulator for this tutorial, the end result is the same.

To select ModelSim as your project simulator:

1. In the Sources tab, right-click the device line (`xc3s700A-4fg484`).
2. Select **Properties**.
3. In the Simulator field of the Project Properties dialog box, select the Modelsim type and HDL language combination you are using.

Locating the Simulation Processes

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

1. In the Sources tab, select **Behavioral Simulation** in the Sources for field.
2. Select the test bench file (`stopwatch_tb`).
3. In the Processes tab, click the **+** beside **ModelSim Simulator** to expand the process hierarchy.

If the ModelSim Simulator processes do not appear, either ModelSim is not selected as the Simulator in the Project Properties dialog box, or Project Navigator cannot find `modelsim.exe`.

If ModelSim is installed but the processes are not available, the Project Navigator preferences may not be set correctly.

To set the ModelSim location:

1. Select **Edit > Preferences**.
2. Click the **+** next to ISE General to expand the ISE preferences
3. Click **Integrated Tools** in the left pane.

4. In the right pane, under Model Tech Simulator, browse to the location of the `modelsim.exe` file. For example,

```
c:\modeltech_xe\win32xoem\modelsim.exe
```

The following simulation processes are available:

- **Simulate Behavioral Model**

This process starts the design simulation.

- **Generate a self-checking HDL test bench**

This process enables you to generate a self-checking HDL test bench equivalent to a test bench waveform (TBW) file and add the test bench to your project. You can also use this process to update an existing self-checking test bench.

The test bench generated by this process contains output data and self-checking code that can be used to compare the data from later simulation runs.

Note: This process is available only if you have a test bench waveform file from the ISE Simulator's Test Bench Waveform Editor. If you double-click this process, ModelSim runs in the background to generate expected results and displays them in the ISE Simulator's Test Bench Waveform Editor. See ["Creating a Test Bench Waveform Using the Waveform Editor."](#)

Specifying Simulation Properties

You will perform a behavioral simulation on the stopwatch design after you have set some process properties for simulation.

ISE allows you to set several ModelSim Simulator properties in addition to the simulation netlist properties. To see the behavioral simulation properties, and to modify the properties for this tutorial:

1. In the Sources tab, select the test bench file (`stopwatch_tb`).
2. Click the **+** sign next to **ModelSim Simulator** to expand the hierarchy in the Processes tab.
3. Right-click **Simulate Behavioral Model**.
4. Select **Properties**.
5. In the Process Properties dialog box, ([Figure 4-1](#)) set the Property display level to **Advanced**. This global setting enables you to now see all available properties.

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

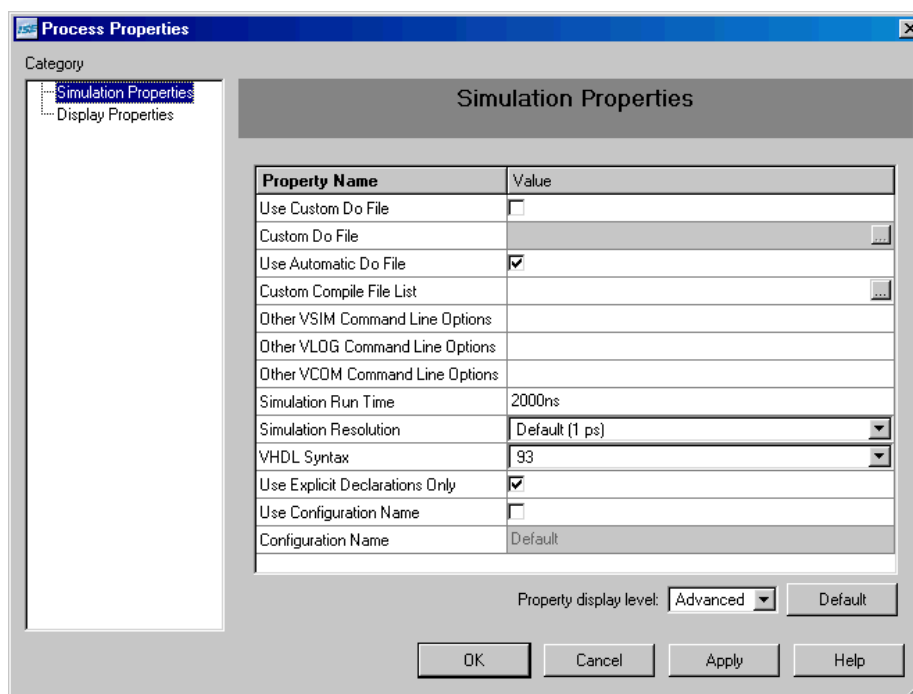


Figure 4-1: Behavioral Simulation Process Properties

- Click **OK**.

For a detailed description of each property available in the Process Properties dialog box, click **Help**.

Performing Simulation

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

The majority of this design runs at 100 Hz and would take a significant amount of time to simulate. The first outputs to transition after RESET is released are the SF_D and LCD_E control signals at around 33 mS. This is why the counter may seem like it is not working in a short simulation. For the purpose of this tutorial, only the DCM signals are monitored to verify that they work correctly.

Adding Signals

To view internal 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 on 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/Object window.
- Highlight signals in the Signal/Object window, and select **Add > Wave > Selected Signals**.

The following procedure explains how to add additional signals in the design hierarchy. In this tutorial, you will be adding the DCM signals to the waveform.

If you are using ModelSim version 6.0 or higher, all the windows are docked by default. To undock the windows, click the **Undock** icon.



Figure 4-2: Undock icon

To add additional signals in the design hierarchy:

1. In the Structure/Instance window, click the **+** next to **uut** to expand the hierarchy.

Figure 4-3 shows the Structure/Instance window for the Verilog flow. The graphics and the layout of the Structure/Instance window for a schematic or VHDL flow may be different.

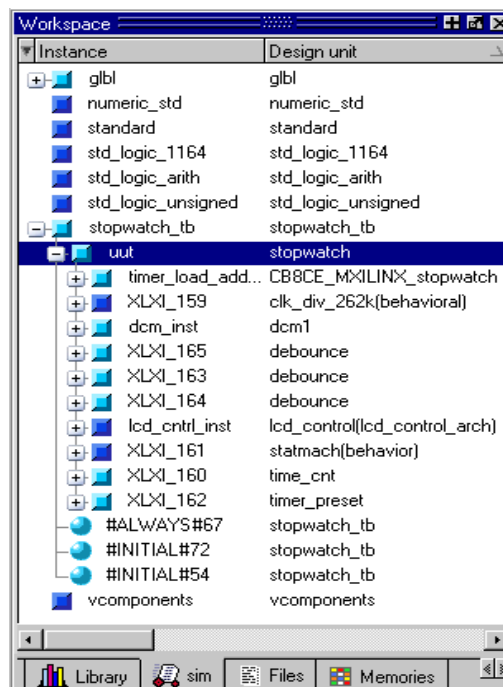


Figure 4-3: Structure/Instance Window - Verilog flow

2. Select **dcm1** in the Structure/Instance window. The signals listed in the Signal/Object window are updated.
3. Click and drag **CLKIN_IN** from the Signal/Object window to the Wave window.
4. In the Signal/Object window, select the following signals. To select multiple signals, hold down the Ctrl key.
 - ◆ RST_IN
 - ◆ CLKFX_OUT
 - ◆ CLK0_OUT
 - ◆ LOCKED_OUT

5. Right-click in the Signal/Object window.
6. Select **Add to Wave > Selected Signals**.

Adding Dividers

In ModelSim, you can add dividers in the Wave window to make it easier to differentiate the signals. To add a divider called **DCM Signals**:

1. Right click anywhere in the signal section of the Wave window. If necessary, undock the window and maximize the window for a larger view of the waveform.
2. Select **Insert Divider**.
3. Enter **DCM Signals** in the Divider Name box.
4. Click **OK**.
5. Click and drag the newly created divider to above the CLKIN_IN signal.

After adding the DCM Signals divider, the waveform will look like [Figure 4-4](#).

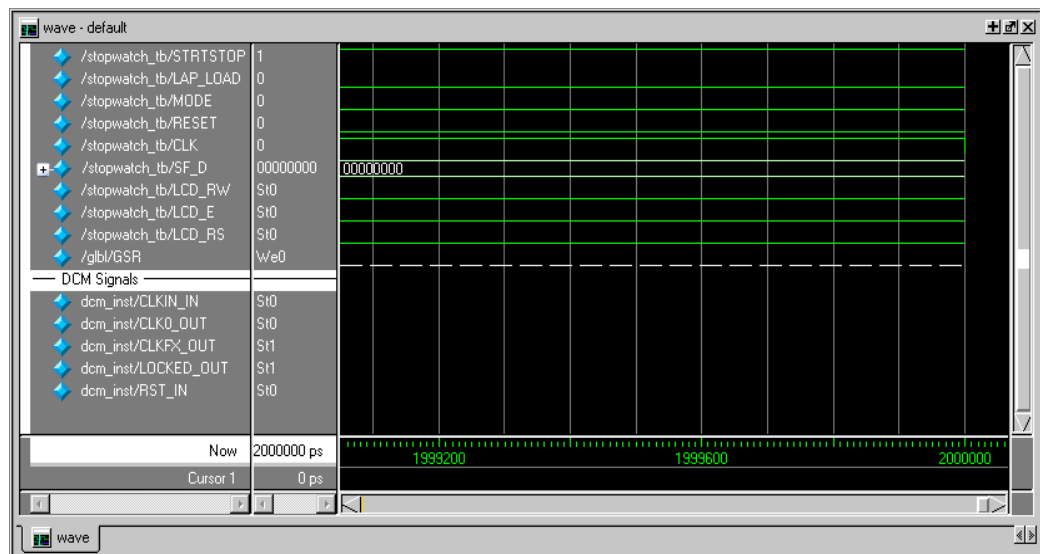


Figure 4-4: Waveform After Adding DCM Signals Divider

The waveforms have not been drawn for any of the newly added signals. This is because ModelSim did not record the data for these signals. By default, ModelSim records data only for the signals that have been added to the Wave window while the simulation is running. After new signals are added to the Wave window, you must rerun the simulation for the desired amount of time.

Rerunning Simulation

To rerun simulation in ModelSim:

1. Click the **Restart Simulation** icon.



Figure 4-5: Restart Simulation Icon

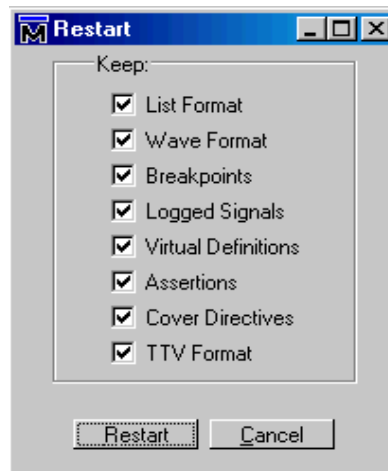


Figure 4-6: Restart Dialog Box

2. In the Restart dialog box, click **Restart**.
3. At the ModelSim command prompt, enter **run 2000 ns**.
4. Press **Enter**.

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

Figure 4-7: Entering the Run Command

The simulation runs for 2000 ns. The waveforms for the DCM are now visible in the Wave window.

Analyzing the Signals

The DCM signals can be analyzed to verify that they work as expected. The CLK0_OUT needs to be 50 MHz and the CLKFX_OUT should be ~26 MHz. The DCM outputs are valid only after the LOCKED_OUT signal is high; therefore, the DCM signals are analyzed only after the LOCKED_OUT signal has gone high.

ModelSim enables you to add cursors to measure the distance between signals. To measure the CLK0_OUT:

1. Select **Add > Wave > Cursor** twice to add two cursors.
2. Click and drag one cursor to the first rising edge transition on the CLK0_OUT signal after the LOCKED_OUT signal has gone high.
3. Click and drag the second cursor just to the right of the first.
4. Click the **Find Next Transition** icon twice to move the cursor to the next rising edge on the CLK0_OUT signal.



Figure 4-8: Find Next Transition Icon

5. Look at the bottom of the waveform for the distance between the two cursors.
The measurement should read 20000 ps. This converts to 50 MHz, which is the input frequency from the test bench, which in turn should be the DCM CLK0 output.
6. Measure CLKFX_OUT using the same steps as above. The measurement should read 38462 ps. This comes out to approximately 26 MHz.

Saving the Simulation

The ModelSim simulator enables you to save the signals list in the Wave window after new signals or stimuli are added, and after simulation is rerun. The saved signals list can easily be opened each time the simulation is started.

To save the signals list:

1. In the Wave window, select **File > Save as**.
2. In the Save Format dialog box, rename the file name from the default wave.do to dcm_signal.do.
3. Click **Save**.

After restarting the simulation, select **File > Load** in the Wave window to load this file.

Your behavioral simulation is complete. To implement the design, follow the steps in [Chapter 5, "Design Implementation."](#)

Behavioral Simulation Using ISE Simulator

Follow this section of the tutorial if you have skipped the previous section, "[Behavioral Simulation Using ModelSim.](#)"

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

To select ISE Simulator as your project simulator:

1. In the Sources tab, right-click the device line (xc3s700A-4fg484).
2. Select **Properties**.
3. In the Project Properties dialog box, select **ISE Simulator** in the Simulator field.

Locating the Simulation Processes

The simulation processes in ISE enable you to run simulation on the design using ISE Simulator. To locate the ISE Simulator processes:

1. In the Sources tab, select **Behavioral Simulation** in the Sources for field.
2. Select the test bench file (stopwatch_tb).
3. Click the + beside Xilinx ISE Simulator to expand the process hierarchy.

The following simulation processes are available:

- **Check Syntax**
This process checks for syntax errors in the test bench.
- **Simulate Behavioral Model**
This process starts the design simulation.
- **Generate a self-checking HDL test bench**
This process enables you to generate a self-checking HDL test bench equivalent to a test bench waveform (TBW) file and add the test bench to your project. You can also use this process to update an existing self-checking test bench.

The test bench generated by this process contains output data and self-checking code that can be used to compare the data from later simulation runs.

Note: This process is available only if you have a test bench waveform file from the ISE Simulator's Test Bench Waveform Editor. If you double-click this process, ModelSim runs in the background to generate expected results and displays them in the ISE Simulator's Test Bench Waveform Editor. See ["Creating a Test Bench Waveform Using the Waveform Editor."](#)

Specifying Simulation Properties

You will perform a behavioral simulation on the stopwatch design after you set some process properties for simulation.

ISE allows you to set several ISE Simulator properties in addition to the simulation netlist properties. To see the behavioral simulation properties, and to modify the properties for this tutorial:

1. In the Sources t, select the test beabnch file (`stopwatch_tb`).
2. Click the **+** sign next to ISE Simulator to expand the hierarchy in the Processes tab.
3. Right-click the **Simulate Behavioral Model** process.
4. Select **Properties**.
5. In the Process Properties dialog box ([Figure 4-9](#)), set the Property display level to **Advanced**. This global setting enables you to now see all available properties.
6. Change the Simulation Run Time to **2000 ns**.
7. Click **Apply** and click **OK**.

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

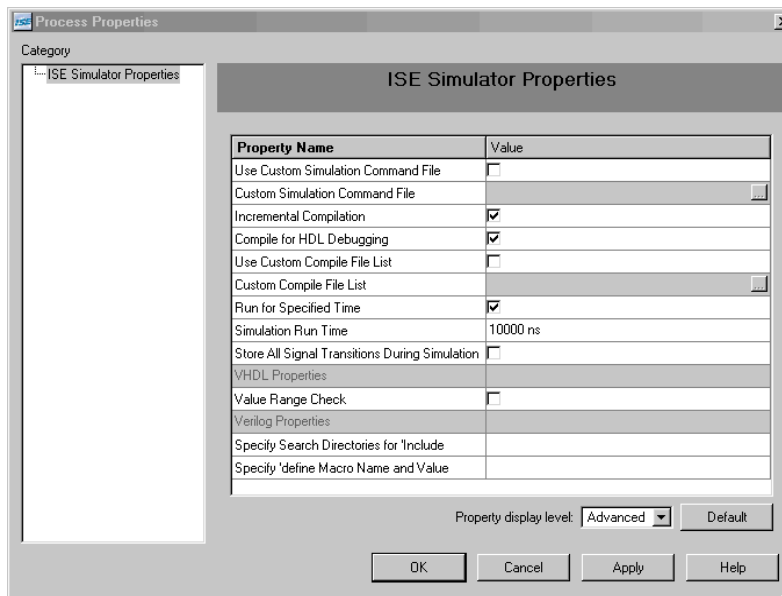


Figure 4-9: Behavioral Simulation Process Properties

Performing Simulation

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

The majority of this design runs at 100 Hz and would take a significant amount of time to simulate. The first outputs to transition after RESET is released are SF_D and LCD_E at around 33 mS. This is why the counter may seem like it is not working in a short simulation. For the purpose of this tutorial, only the DCM signals are monitored to verify that they work correctly.

Adding Signals

To view signals during the simulation, you must add them to the Waveform window. ISE automatically adds all the top-level ports to the Waveform window. Additional signals are displayed in the Sim Hierarchy window. The following procedure explains how to add additional signals in the design hierarchy. For the purpose of this tutorial, add the DCM signals to the waveform.

To add additional signals in the design hierarchy:

1. In the Sim Hierarchy window, click the **+** next to `stopwatch_tb` to expand the hierarchy.
2. Click the **+** next to **uut stopwatch** to expand the hierarchy.

Figure 4-10 shows the Sim hierarchy window for the Verilog flow. The graphics and the layout of the window for a schematic or VHDL flow may be different.

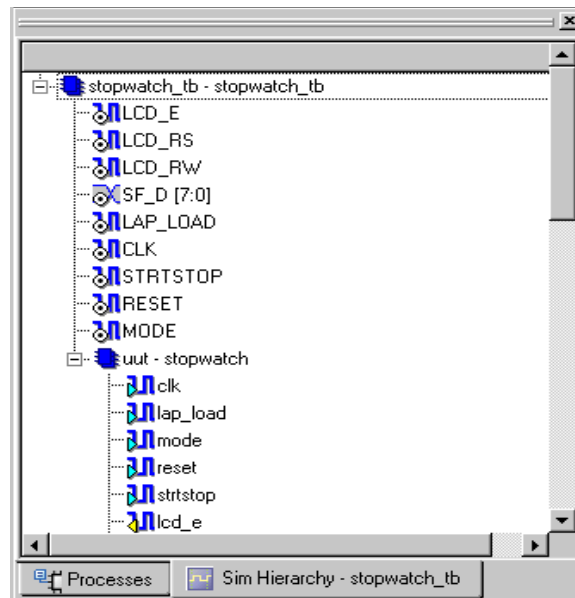


Figure 4-10: Sim Hierarchy Window - Verilog flow

3. Click the **+** next to dcm1 in the Sim Hierarchy window.
4. Click and drag **CLKIN_IN** from the Sim Hierarchy window to the Waveform window.
5. Select the following signals:

- ◆ RST_IN
- ◆ CLKFX_OUT
- ◆ CLK0_OUT
- ◆ LOCKED_OUT

To select multiple signals, hold down the **Ctrl** key.

6. Drag all the selected signals to the waveform. Alternatively, right click on a selected signal and select **Add To Waveform**.

Notice that the waveforms have not been drawn for the newly added signals. This is because ISE Simulator did not record the data for these signals. By default, ISE Simulator records data only for the signals that have been added to the waveform window while the simulation is running. Therefore, when new signals are added to the waveform window, you must rerun the simulation for the desired amount of time.

Rerunning Simulation

To rerun the simulation in ISE Simulation:

1. Click the **Restart Simulation** icon.



Figure 4-11: ISE Simulator Restart Simulation Icon

2. At the ISE Simulator command prompt in the Sim Console, enter **run 2000 ns** and press **Enter**.

The simulation runs for 2000 ns. The waveforms for the DCM are now visible in the Waveform window.

Analyzing the Signals

Now the DCM signals can be analyzed to verify that they work as expected. The CLK0_OUT should be 50 MHz and the CLKFX_OUT should be ~26 MHz. The DCM outputs are valid only after the LOCKED_OUT signal is high; therefore, the DCM signals are analyzed only after the LOCKED_OUT signal has gone high.

ISE Simulator can add markers to measure the distance between signals. To measure the CLK0_OUT:

1. If necessary, zoom in on the waveform.
2. Click the **Measure Marker** icon.



Figure 4-12: **Measure Marker Icon**

3. Place the marker on the first rising edge transition on the CLK0_OUT signal after the LOCKED_OUT signal has gone high.
4. Click and drag the other end of the marker to the next rising edge.
5. Look at the top of the waveform for the distance between the marker. The measurement should read 20.0 ns. This converts to 50 MHz, which is the input frequency from the test bench, which in turn is the DCM CLK0 output.
6. Measure CLKFX_OUT using the same steps as above. The measurement should read 38.5 ns. This equals approximately 26 MHz.

Your behavioral simulation is complete. To implement the design, follow the steps in [Chapter 5, "Design Implementation."](#)

Creating a Test Bench Waveform Using the Waveform Editor

This section demonstrates how to use the Waveform Editor. The Waveform Editor is a test bench creation tool in ISE. You can use the Waveform Editor to graphically enter stimuli, and to generate a VHDL test bench or Verilog test fixture. It is not necessary to follow this section if you have added the tutorial test bench to the project already.

Creating a Test Bench Waveform Source

In this tutorial, create the test bench waveform for a sub-module only. The Waveform Editor can be used to generate stimuli for top-level designs as well.

To create a test bench with the ISE Simulator Waveform Editor:

1. Select `time_cnt` in the Sources tab.
2. Select **Project > New Source**.
3. In the New Source Wizard, select **Test Bench Waveform** as the source type.
4. Type `time_cnt_tb`.

5. Click **Next**.

Note: In the Select dialog box, the `time_cnt` file is the default source file because it is selected in the Sources tab (step 1).

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

The Waveform Editor opens in ISE. The Initialize Timing dialog box displays, and enables you to specify the timing parameters used during simulation. The Clock Time High and Clock Time Low fields together define the clock period for which the design must operate. The Input Setup Time field defines when inputs must be valid. The Output Valid Delay field defines the time after active clock edge when the outputs must be valid.

8. In the Initialize Timing dialog box, fill in the fields as follows:
 - ◆ Clock Time High: **10**
 - ◆ Clock Time Low: **10**
 - ◆ Input Setup Time: **5**
 - ◆ Output Valid Delay: **5**
9. Select the **GSR (FPGA)** from the Global Signals section.
10. Change the Initial Length of Test Bench to **3000**.
11. Click **Finish**.

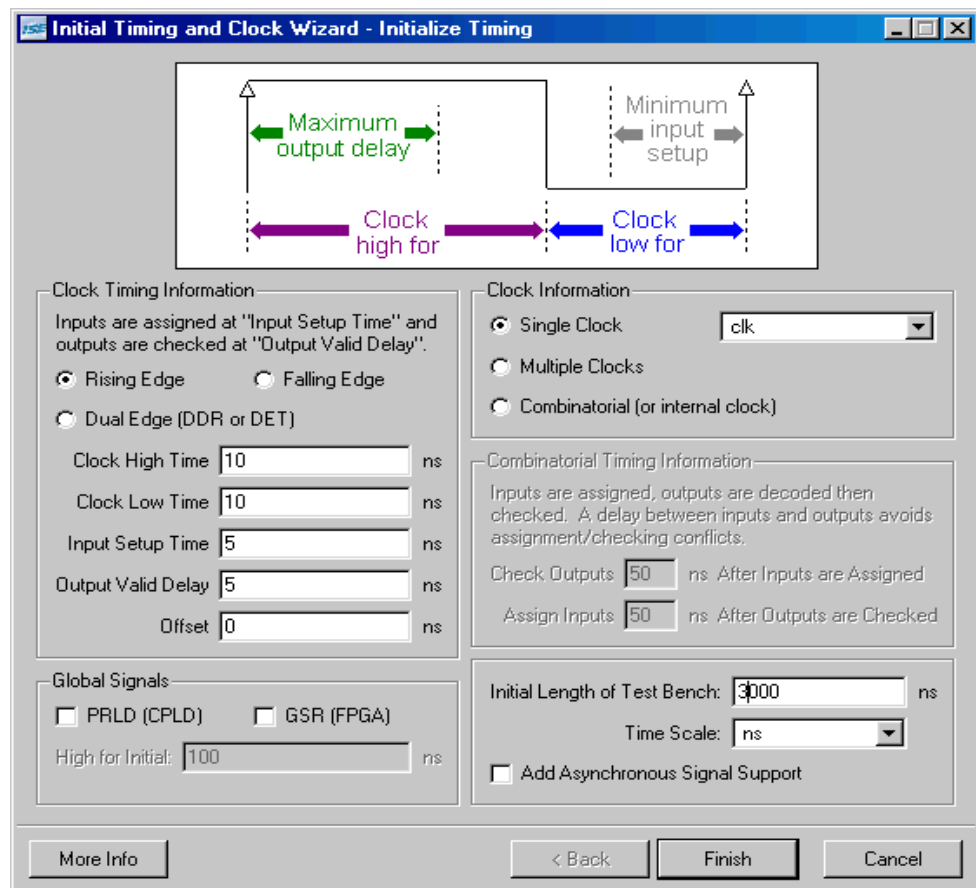


Figure 4-13: Waveform Editor - Initialize Timing Dialog Box

The ISE Simulator's Waveform Editor View opens in ISE.

Applying Stimulus

In the Waveform Editor, in the blue cell, you can apply a transition (high/low). The width of this cell is determined by the Input setup delay and the Output valid delay.

Enter the following input stimuli:

1. Click the CE cell at time 110 ns to set it high (CE is active high).
2. Click the CLR cell at time 150 ns to set it high.
3. Click the CLR cell at time 230 ns to set it low.

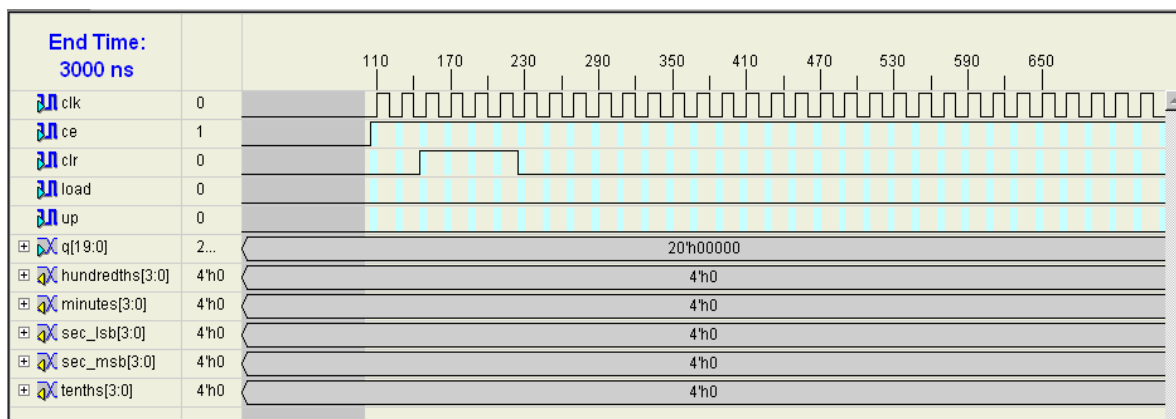


Figure 4-14: Applying Stimulus in the Waveform Editor View

4. Click the Save icon in the toolbar.

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

5. Select `time_cnt_tb.tbw` in the Sources tab.
6. Double-click **Generate Self-Checking Test Bench** in the Process tab.

A test bench containing output data and self checking code is generated and added to the project. The created test bench can be used to compare data from later simulation.

Design Implementation

This chapter contains the following sections.

- “Overview of Design Implementation”
- “Getting Started”
- “Specifying Options”
- “Creating Partitions”
- “Creating Timing Constraints”
- “Translating the Design”
- “Using the Constraints Editor”
- “Using the Floorplan Editor”
- “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”
- “Changing HDL with Partition”
- “Creating Configuration Data”
- “Creating a PROM File with iMPACT”
- “Command Line Implementation”

Overview of Design Implementation

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

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

This chapter demonstrates the ISE Implementation flow. The front-end design has already been compiled in an EDA interface tool. For details about compiling the design, see [Chapter 2, “HDL-Based Design”](#) or [Chapter 3, “Schematic-Based Design.”](#) In this chapter, you will be passing an input netlist (EDN, NGC) from the front-end tool to the back-end Design Implementation tools, and you will be 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 emulates a runner's stopwatch with actual and lap times. There are five inputs to the system: CLK, RESET, LAP_LOAD, MODE, and SRTSTP. This system generates a traditional stopwatch with lap times and a traditional timer on a LCD display.

Continuing from Design Entry

If you have followed the tutorial using either the HDL Design flow or the Schematic Design flow, you have created a project, design entry source files, and an EDIF netlist file.

If you do not have a `stopwatch.ucf` file in your project, create one as follows:

1. In the Sources tab, select the top-level source file **stopwatch**.
2. Select **Project > New Source**.
3. Select **Implementation Constraints File**.
4. Type `stopwatch.ucf` as the file name.
5. Click **Next**.
6. Select **stopwatch** from the list.
7. Click **Next**.
8. Click **Finish**.

With a UCF in the project, you are now ready to begin this chapter. Skip to the “[Specifying Options](#)” section.

Starting from Design Implementation

If you are beginning the tutorial from this chapter, you will need to download the pre-synthesized design files provided on the Xilinx® web-site, create a project in ISE and then add the downloaded source files to the project.

1. Create an empty working directory named Watch.
2. Go to <http://www.xilinx.com/support/techsup/tutorials/tutorial9.htm>, and copy the pre-synthesized tutorial files (see [Table 5-1](#)) to 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>timer_preset.ngc</i>	Timer netlist file (NGC)
<i>stopwatch.ucf</i>	User Constraints File

3. Open ISE.
 - a. On a workstation, enter `ise &`.
 - b. On a PC, select **Start > Programs > Xilinx ISE 9.1i > Project Navigator**.
4. Create a new project and add the EDIF netlist as follows:
 - a. Select **File > New Project**.
 - b. Type **EDIF_Flow** for the Project Name.
 - c. Select **EDIF** for the top_level SourceType.
 - d. Click **Next**.
 - e. Select **stopwatch.edn** for the Input Design file.
 - f. Select **stopwatch.ucf** for the Constraints file.
 - g. Click **Next**.
 - h. Select the following:
 - **Spartan3a** for the Device Family
 - **xc3s700a** for the Device
 - **-4** for the Speed Grade, **fg484** for the Package
 - i. Click **Next**.
 - j. Click **Finish**.
 - k. Copy the `timer_preset.ngc` file into the EDIF_Flow directory.

In the Sources tab, 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 properties for design implementation. The implementation properties control how the software maps, places, routes, and optimizes a design.

To set the implementation property options for this tutorial:

1. In the Sources tab, select the **stopwatch** top-level file.
2. In the Processes tab, right-click the **Implement Design** process.
3. Select **Properties** from the right-click menu.

The Process Properties dialog box provides access to the Translate, Map, Place and Route, Simulation, and Timing Report properties. You will notice a series of categories, each contains properties for a different aspect of design implementation.
4. Ensure that you have set the Property display level to **Advanced**. This setting is in the lower, right-hand corner of the dialog box.

This global setting enables you to see all available properties.
5. Click the **Post-Map Static Timing Report Properties** category.
6. Change Report Type to **Verbose Report**.

This report will be generated after the Map process is completed.
7. Click the **Post-Place & Route Static Timing Report Properties** category.
8. Change Report Type to **Verbose Report**.

This report will be generated after Place and Route is completed.

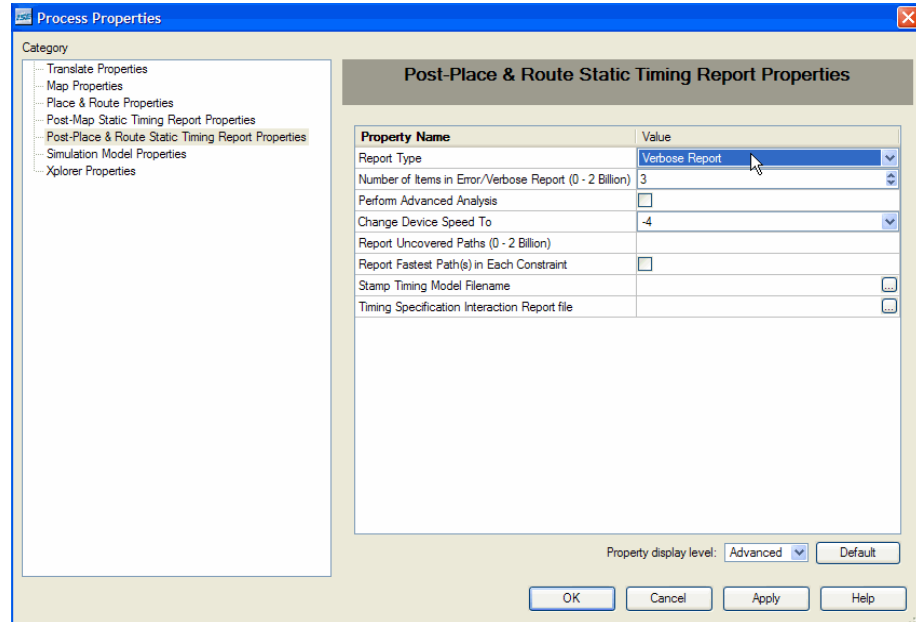


Figure 5-1: Post-Place & Route Static Timing Report Properties

9. Click the **Place & Route Properties** category.
10. Change the Place & Route Effort Level (Overall) to **High**.

This option increases the overall effort level of Place and Route during implementation.

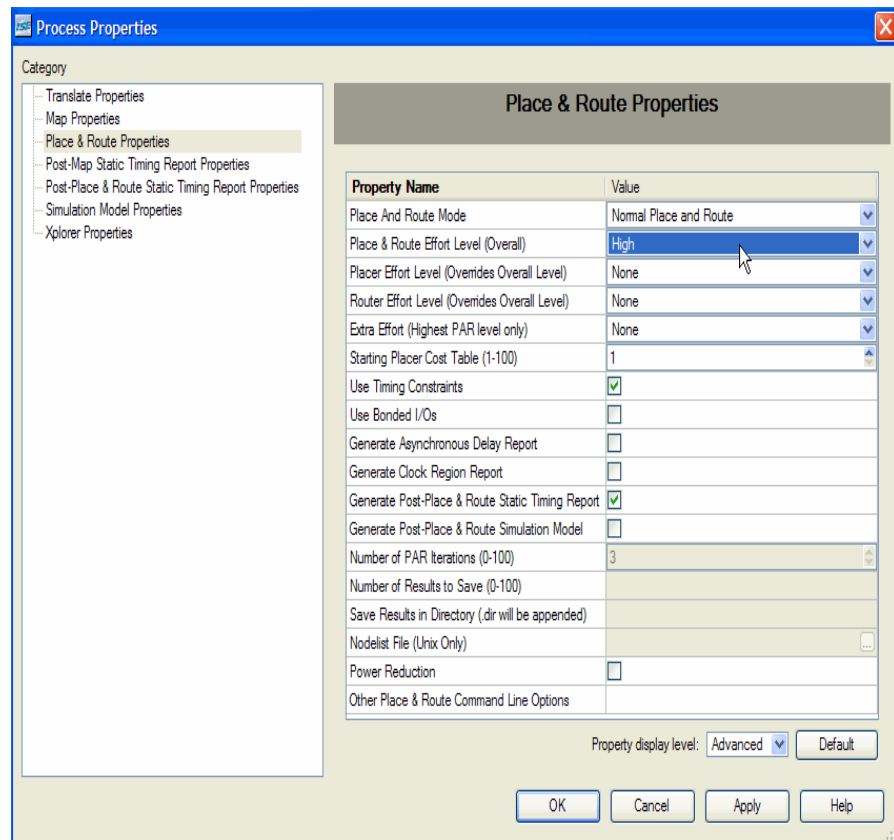


Figure 5-2: Place & Route Properties

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

Creating Partitions

A partition is created for an instance in your logical design to indicate that the implementation for that instance should be reused when possible. Partitions can be nested hierarchically and be defined on any HDL module instance in the design. In Verilog, the partition is set on the module instance and for VHDL, the partition is set on the entity architecture. A module with multiple instances can have multiple partitions—a partition on each instance. The top level of the HDL design has a default partition.

Partitions do not need ranges for implementation re-use, and logic can be at the top level partition. Partitions automatically detect input source changes, including HDL changes and certain constraint changes. Partitions also detect command line changes, such as effort levels on implementation tools, and only the effected partitions are re-implemented.

If you are doing the EDIF flow, you will not get the option to create Partitions in Project Navigator. The creation of Partitions was done through the synthesis tool. Please proceed to the next section.

To enable partitions for this design:

1. In the Sources tab, select **lcd_cntrl_inst** module and right-click on it.
2. Select **New Partition** from the menu.
3. Repeat steps 1 and 2 for the **timer_state** module.
4. Repeat steps 1 and 2 for the **timer_inst** module in VHDL and Verilog designs only.

Note: In ISE 9.1i there is a known issue that will not allow MAP to complete a second iteration if a Partition is placed on a schematic module.

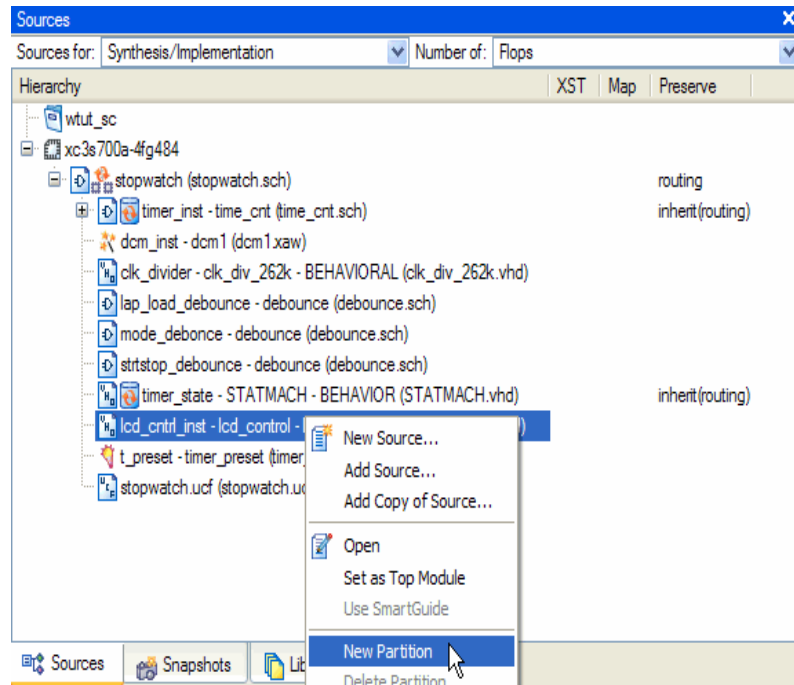


Figure 5-3: New Partitions on LCD_CNTRL_INST module

The Preserve status is set to *inherit*, which gets the status from the top level partition. The top level partition is set to *routing* by default. The preserve status can be changed to *Routing*, *Placement*, or *Synthesis*. To do this, right click on a partitioned source file and select **Partition Properties**.

Creating Timing Constraints

The User Constraints File (UCF) provides a mechanism for constraining a logical design without returning to the design entry tools. However, without the design entry tools, you must understand the exact syntax needed to define constraints. The Constraints Editor and Floorplan Editor are graphical tools that enable you to enter timing and pin location constraints.

To launch the Constraints Editor:

1. In the Sources tab, select **Stopwatch**
2. In the Processes tab, expand the **User Constraints** hierarchy.

3. Double-click **Create Timing Constraints**.

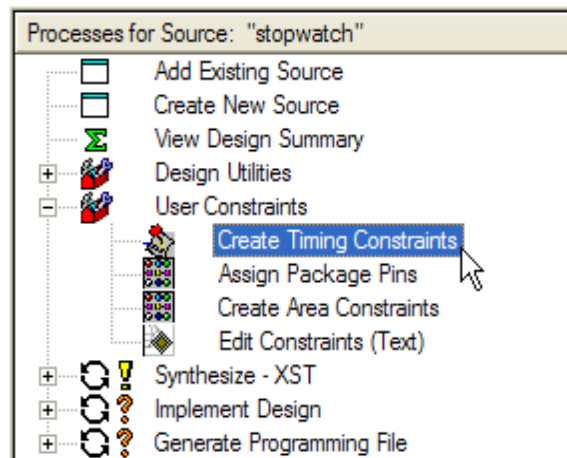


Figure 5-4: **Create Timing Constraints Process**

This automatically runs the Translate step, which is discussed in the following section. Then the Constraints Editor opens.

Translating the Design

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

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

Using the Constraints Editor

When you run the Create Timing Constraints process, Translate is run and ISE launches 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` and `stopwatch.ucf` files are automatically read into the Constraints Editor.

In the following section, a PERIOD, Global OFFSET IN, Global OFFSET OUT, and TIMEGRP OFFSET IN constraint will be written in the UCF and used during implementation. The Global branch of the Timing Constraints tab 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. Note that many of the internal names will vary depending on the design flow and synthesis tool used.

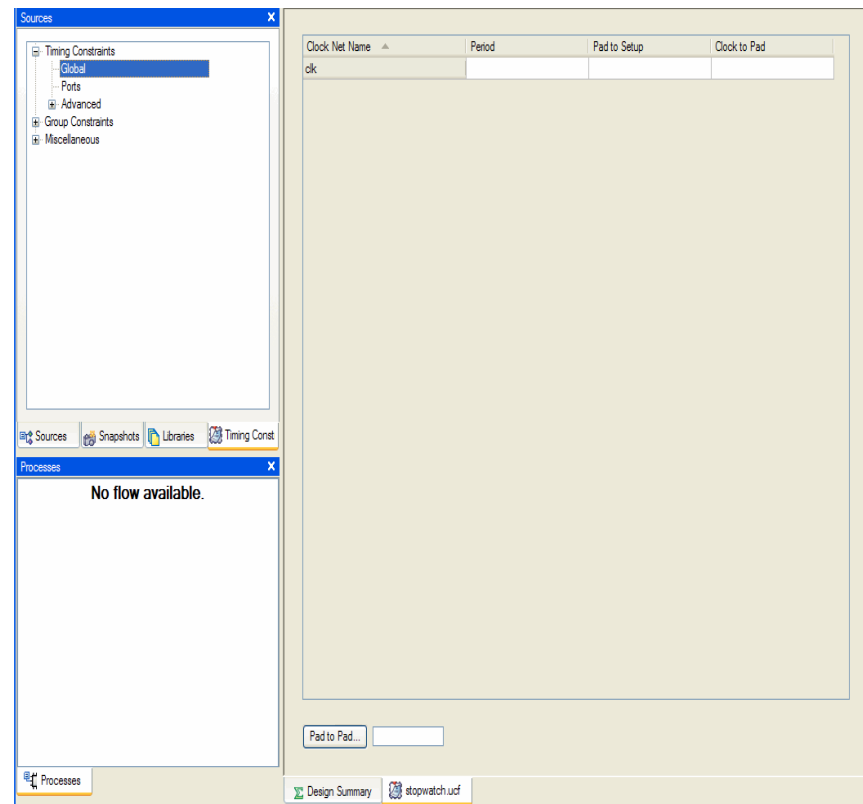


Figure 5-5: Constraints Editor in Project Navigator - Global Branch

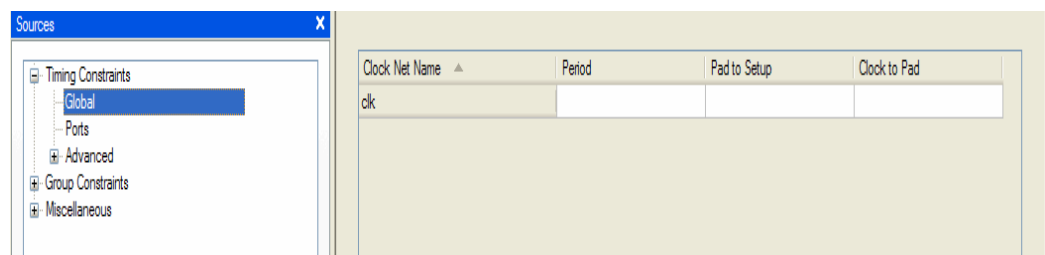


Figure 5-6: Constraints Editor - Global Branch

In the Constraints Editor, edit the constraints as follows:

1. Double-click the Period cell on the row associated with the clock net CLK. The Clock Period dialog box opens.
2. For the Clock Signal Definition, verify that **Specify Time** is selected. This enables you to define an explicit period for the clock.
3. Enter a value of **7.0** in the Time field.

- Verify that **ns** is selected from the Units drop-down list.

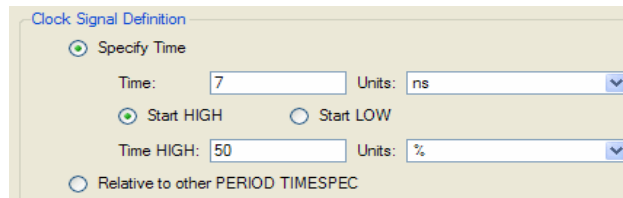


Figure 5-7: PERIOD Constraint Values

- For the Input Jitter section, enter a value of **60** in the Time field.
- Verify that **ps** is selected from the Units drop-down list.

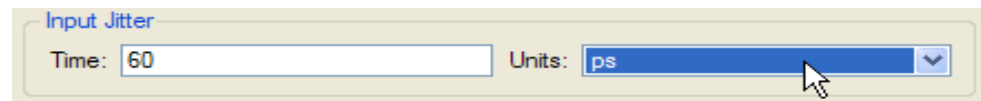


Figure 5-8: INPUT JITTER Constraint Value

- Click **OK**.
The period cell is updated with the global clock period constraint that you just defined (with a default 50% duty cycle).
- Single click the Pad to Setup cell for clock signal CLK and type 6.
This creates a Global OFFSET IN constraints for the CLK signal.
- Single click the Clock to Pad cell for clock signal CLK and type 38.
This creates a Global OFFSET OUT constraint for the CLK signal

Clock Net Name ▲	Period	Pad to Setup	Clock to Pad
clk	7 ns. HIGH 50% Input_Jitter 60	6 ns.	38 ns.

Figure 5-9: Completed Global constraints in Constraints Editor

- Select the **Ports** branch from the Timing Constraints tab of the Sources panel.
A listing of the current ports is displayed on the left.
- Select **SF_D<0>** in the Port Name Column.
- Hold down the **Shift** key and select **SF_D<7>**.
This selects the elements for creating a grouped offset.

- In the Group Name field, type **display_grp**, and click **Create Group** to create the group.

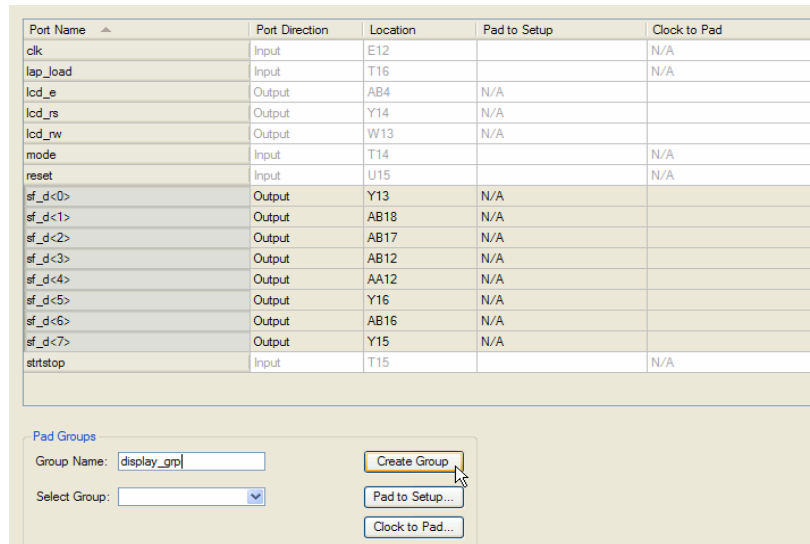


Figure 5-10: Creating a Grouped OFFSET

- In the Select Group drop-down list, select the group you just created.
- Click **Click to Pad**.

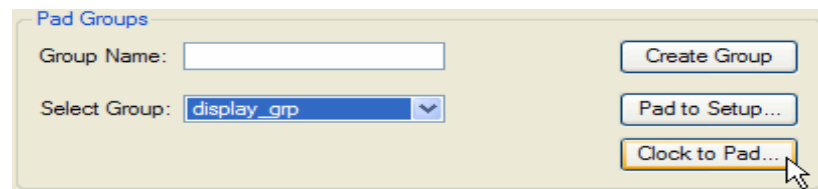


Figure 5-11: Setting Constraints for the Grouped OFFSET

The Clock to Pad dialog box opens.

- Enter **32** for the Timing Requirement.

17. Select **CLK** in the Relative to Clock Pad Net field.

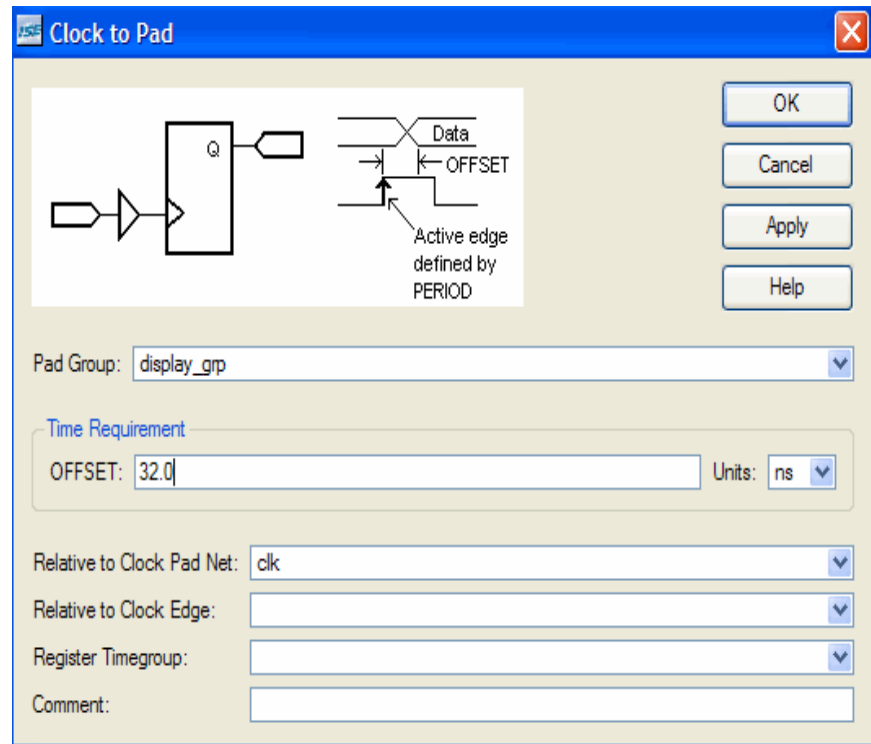


Figure 5-12: Clock to Pad Dialog Box

18. Click **OK**
19. Select **File > Save**.
The changes are now saved in the `stopwatch.ucf` file in your current working directory.
20. Close the Constraints Editor by selecting **File > Close**.

Using the Floorplan Editor

Use the Floorplan Editor to add and edit the pin locations and area group constraints defined in the NGD file. Floorplan Editor 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. Floorplan Editor also places Global Logic at the Slice level with Block Ram, Digital Clock Managers (DCMs), Gigabit Transceivers (GTs), and BUFs.

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

1. Select the stopwatch module in the Sources window.
2. click the **+** next to Implement Design to expand the process hierarchy in the Processes window.
3. Click the **+** next to Translate.
4. Double-click **Assign Package Pins Post-Translate**, located under the Translate process.

Note: For an EDIF project, double-click **Assign Package Pins**, located under the User Constraints process.

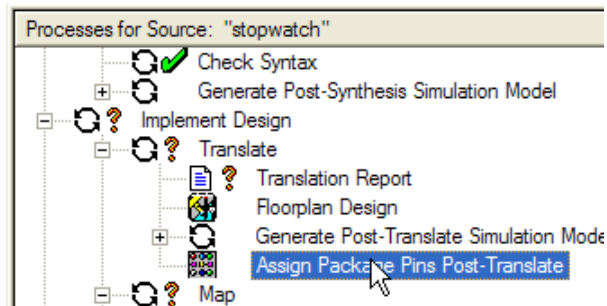


Figure 5-13: Edit Package Pin Placement

This process launches Floorplan Editor.

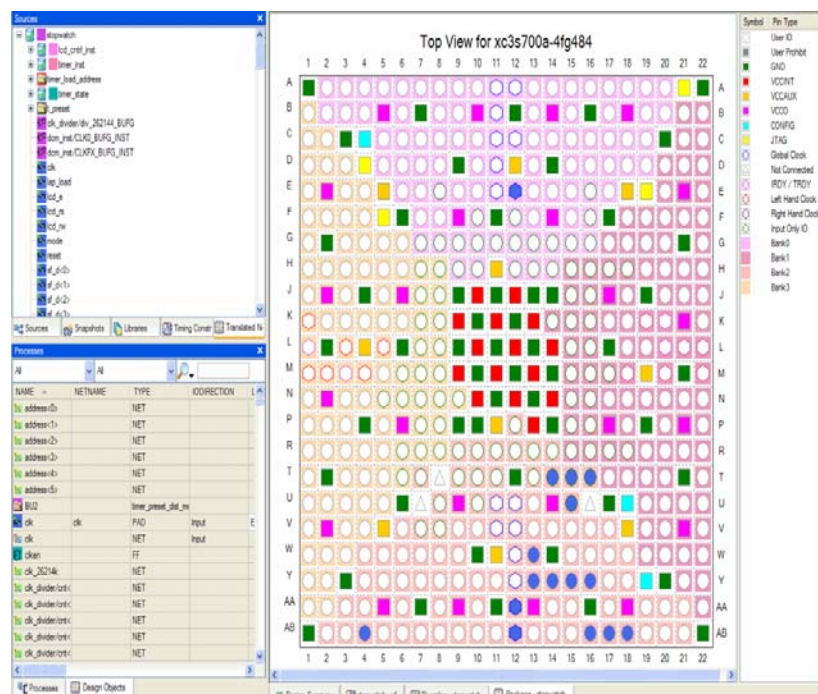


Figure 5-14: Pin-Out Area Constraints Editor (PACE)

- Select the **Package** tab in the work space.
This view shows the graphical representation of the device package.
- Select the Design Object tab in Processes panel.
This window displays all the objects in the design.
- In the Design Object tab, change the center filter from **ALL** to **IOs**, then scroll down to the nets that begin with "LCD_".
- In the LOC column, enter pin locations using the following values:
 - ◆ LCD_E → AB4
 - ◆ LCD_RS → Y14
 - ◆ LCD_RW → W13

NAME	NETNAME	TYPE	IO DIRECTION	LOC	BANK	IO STANDARD	VREF	VCCO	DRIVE	TERMINATION	SLEW	SUSPEND	IO DELAY	DIFF TYPE	DIFF PAIR
clk	clk	PAD	Input			LVC MOS33	N/A	N/A							
lap_load	lap_load	PAD	Input			LV TTL	N/A	N/A		PULLDOWN					
lcd_e	lcd_e	PAD	Output	AB4	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
lcd_rs	lcd_rs	PAD	Output	Y14	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
lcd_rw	lcd_rw	PAD	Output	W13	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
mode	mode	PAD	Input			LV TTL	N/A	N/A		PULLDOWN					
reset	reset	PAD	Input			LV TTL	N/A	N/A		PULLDOWN					
sf_d<0>	sf_d<0>	PAD	Output	Y13	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<1>	sf_d<1>	PAD	Output	AB18	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<2>	sf_d<2>	PAD	Output	AB17	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<3>	sf_d<3>	PAD	Output	AB12	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<4>	sf_d<4>	PAD	Output	AA12	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<5>	sf_d<5>	PAD	Output	Y16	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<6>	sf_d<6>	PAD	Output	AB16	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
sf_d<7>	sf_d<7>	PAD	Output	Y15	BANK2	LV TTL	N/A	3.30	4		QUIETIO				
strtstop	strtstop	PAD	Input			LV TTL	N/A	N/A		PULLDOWN					

Figure 5-15: Pin Locations

To place IOs in the Package Pin view using the drag and drop method:

9. From the Design Object tab, click and drag the following signals to the associated location in the Package Pin window:
 - ◆ LAP_LOAD → T16
 - ◆ RESET → U15
 - ◆ MODE → T14
 - ◆ STRTSTOP → T15

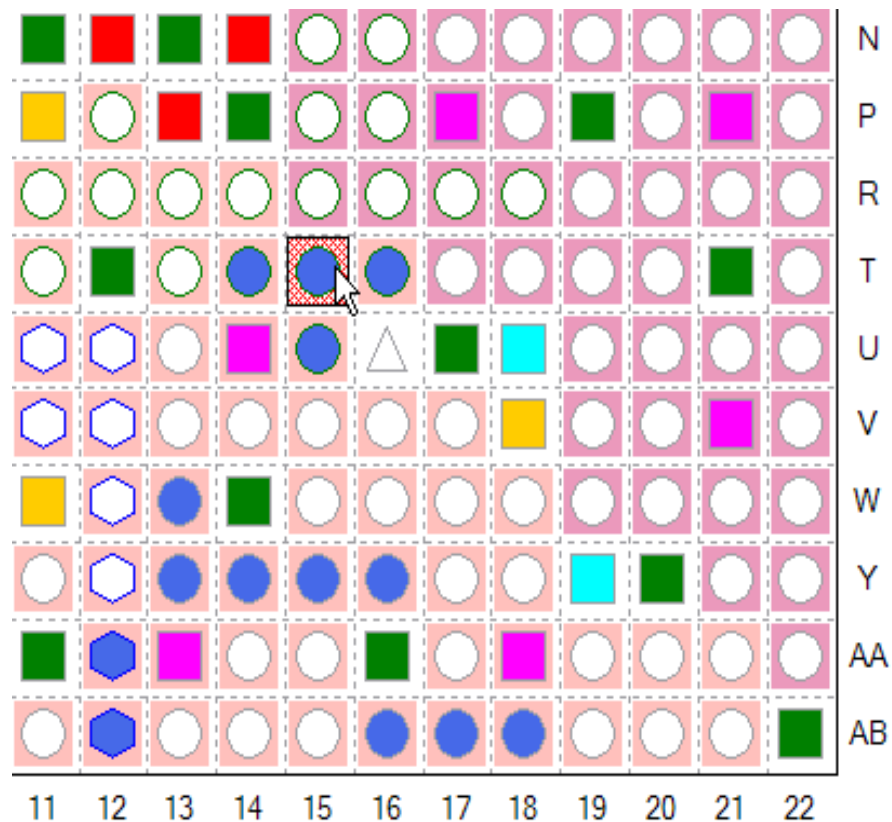


Figure 5-16: Drag and Drop IOs in the Package Pins View

10. Once the pins are locked down, select **File > Save**. The changes are saved in the stopwatch.ucf file in your current working directory.
11. Close Floorplan Editor by selecting **File > Close**.

Mapping the Design

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

1. Select the stopwatch module in the Sources window.
2. In the Processes tab, right-click on **Map** and select **Run**.
Note: This can also be accomplished by double-clicking Map.

- Expand the Implement Design hierarchy to see the progress through implementation.

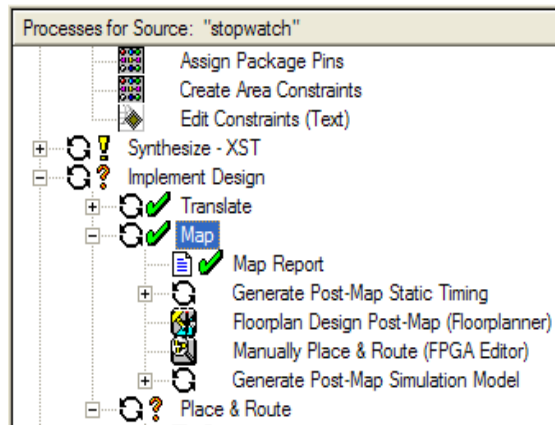


Figure 5-17: Mapping the Design

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

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

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

Table 5-2: Reports Generated by Map

Translation Report	Includes warning and error messages from the translation process.
Map Report	Includes information on how the target device resources are allocated, references to trimmed logic, and device utilization.
All NGDBUILD and MAP Reports	For detailed information on the Map reports, refer to the <i>Development System Reference Guide</i> . This Guide is available with the collection of software manuals and is accessible from ISE by selecting Help > Software Manuals , or from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/ .

To view a report:

1. Expand the Translate or Map hierarchy.
2. Double-click a report, such as Translation Report or Map Report.

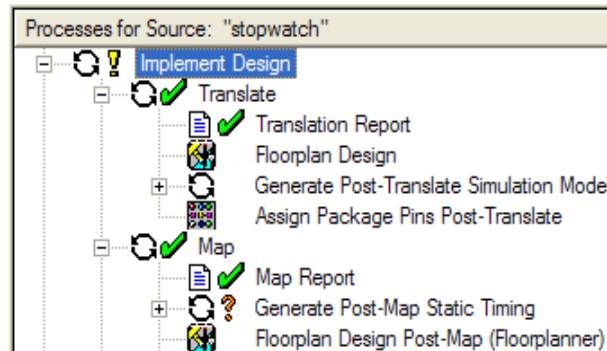
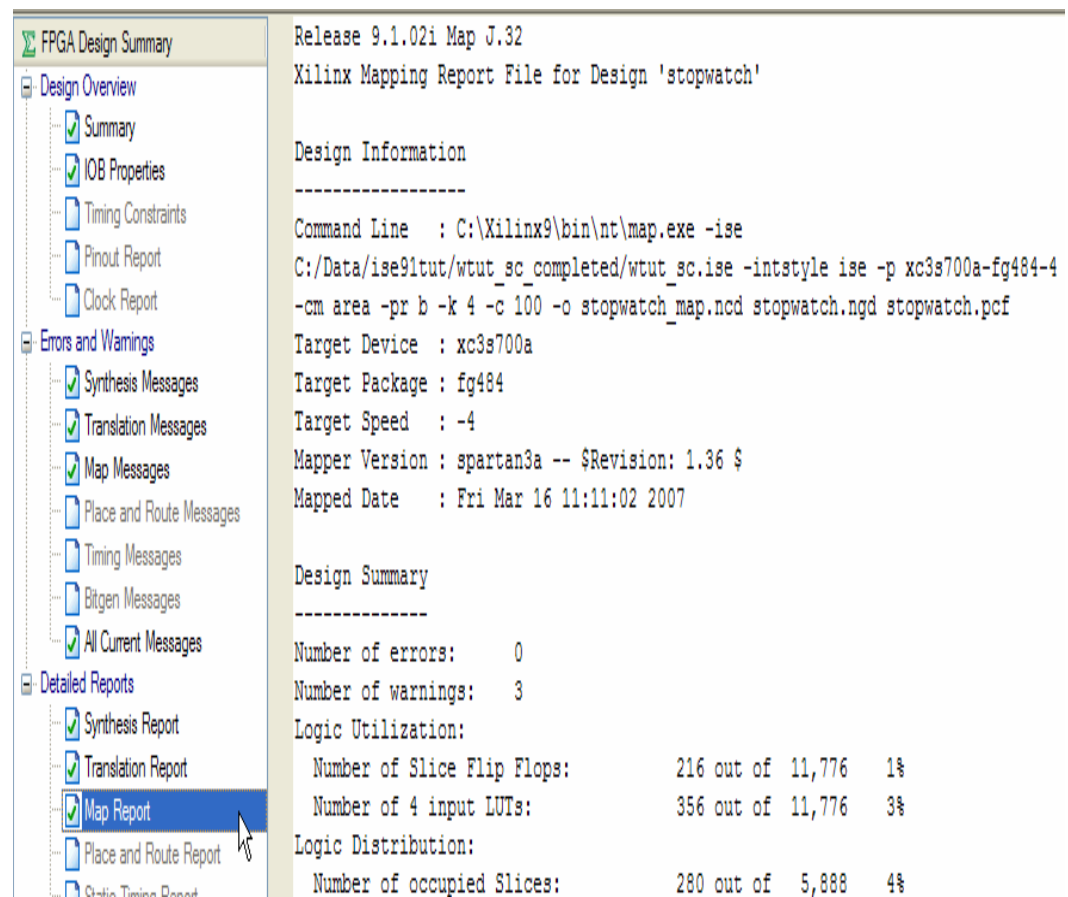


Figure 5-18: Translation Report and Map Report Processes

3. Review the report. Look for Warnings, Errors, and Information (INFO).

The Design Summary also contains these reports.



```

Release 9.1.02i Map J.32
Xilinx Mapping Report File for Design 'stopwatch'

Design Information
-----
Command Line   : C:\Xilinx9\bin\nt\map.exe -ise
                  C:/Data/ise91tut/wtut_sc_completed/wtut_sc.ise -intstyle ise -p xc3s700a-fg484-4
                  -cm area -pr b -k 4 -c 100 -o stopwatch_map.ncd stopwatch.ngd stopwatch.pcf
Target Device  : xc3s700a
Target Package : fg484
Target Speed   : -4
Mapper Version : spartan3a -- $Revision: 1.36 $
Mapped Date    : Fri Mar 16 11:11:02 2007

Design Summary
-----
Number of errors:      0
Number of warnings:   3

Logic Utilization:
  Number of Slice Flip Flops:      216 out of 11,776  1%
  Number of 4 input LUTs:         356 out of 11,776  3%

Logic Distribution:
  Number of occupied Slices:      280 out of 5,888  4%
  
```

Figure 5-19: Implementation Reports shown in Design Summary

Using Timing Analysis to Evaluate Block Delays After Mapping

After the design is mapped, evaluate the Logic Level details in the Post-Map Static Timing Report to evaluate the logical paths in the design. Evaluation verifies that block delays are reasonable given the design specifications. Because the design is not yet placed and routed, actual routing delay information is not available. The timing report describes the logical block delays and estimated routing delays. The net delays provided are based on an optimal distance between blocks (also referred to as *unplaced 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 10 ns of block delay should meet a 20 ns timing constraint after it is placed and routed.

If your design is extremely dense, the Post-Map Static Timing Report provides a summary analysis of your timing constraints based on block delays and estimates of route delays. This analysis can help to determine if your timing constraints are going to be met. This report is produced after Map and prior to Place and Route (PAR).

Report Paths in Timing Constraints Option

Use the Post-Map Static Timing Report to determine timing violations that may occur prior to running PAR. Since you defined timing constraints for the stopwatch design, the timing report will display the path for each of the timing constraints.

To view the Post-Map Static Timing Report and review the PERIOD Constraints that were entered earlier:

1. In the Processes tab, click the **+** next to Map to expand the process hierarchy.
2. Double-click **Generate Post-Map Static Timing**.
3. To open the Post-Map Static Timing Report, double-click **Analyze Post-Map Static Timing Report**.

Timing Analyzer automatically launches and displays the report.

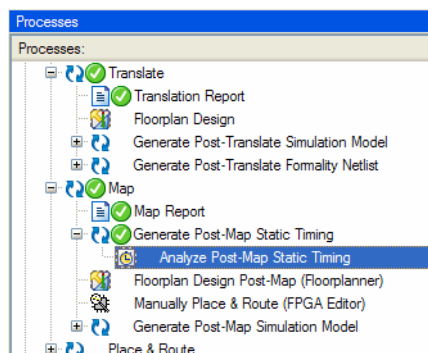


Figure 5-20: Post-Map Static Timing Report Process

- Select the TS_dcm_inst_CLKX_BUF timing constraint under the Timing tab.

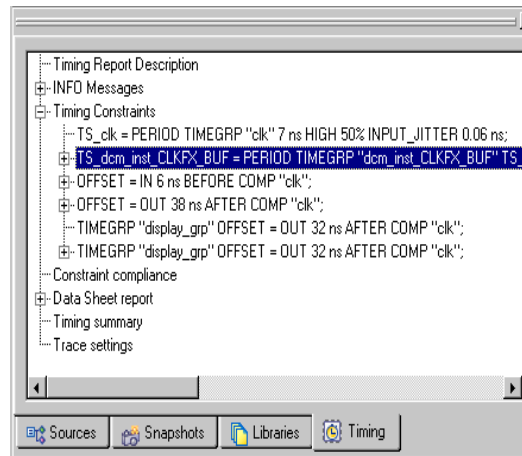


Figure 5-21: Selecting Post-Map Static Timing constraint

The work space shows the report for the selected constraint. At the top of this report, you will find the selected period constraint and the minimum period obtained by the tools after mapping. By default, only three paths per timing constraint will be shown. Selecting one of the three paths allows you to see a breakdown of the path which contains the component and routing delays.

Notice that the report displays the percentage of logic versus the percentage of routing at the end of each path (e.g. 88.0% logic, 12.0% route). The unplaced floors listed are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

- After viewing the report, close the Timing Analyzer by selecting **File > Close**.

Note: Even if you do not generate a 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**). The Post-Map Static Timing Report will list any pre-PAR timing violations.

Placing and Routing the Design

After the mapped design is evaluated, the design can be placed and routed.

One of two place-and-route algorithms is performed during the Place & Route (PAR) process:

- Timing Driven PAR**
 PAR is run with the timing constraints specified in the input netlist and/or in the constraints file.
- Non-Timing Driven PAR**
 PAR is run, ignoring all timing constraints.

Since you defined timing constraints earlier in this chapter, the Place & Route (PAR) process performs timing driven placement and routing.

- To run PAR, in the Processes tab, double-click **Place & Route**.

To review the reports that are generated after the Place & Route process is completed:

2. Click the **+** next to Place & Route to expand the process hierarchy.
3. Double-click **Place & Route Report**.

Note: You can also display and examine the **Pad Report** and **Asynchronous Delay Report**.

Table 5-3: Reports Generated by PAR

Report	Description
Place & Route Report	Provides a device utilization and delay summary. Use this report to verify that the design successfully routed and that all timing constraints were met.
Pad Report	Contains a report of the location of the device pins. Use this report to verify that pins locked down were placed in the correct location.
Asynchronous Delay Report	Lists all nets in the design and the delays of all loads on the net.
All PAR Reports	For detailed information on the PAR reports, refer to the <i>Development System Reference Guide</i> . This Guide is available with the collection of software manuals and is accessible from ISE by selecting Help > Online Documentation , or from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/ .

The Design Summary also displays the Place and Route Reports.

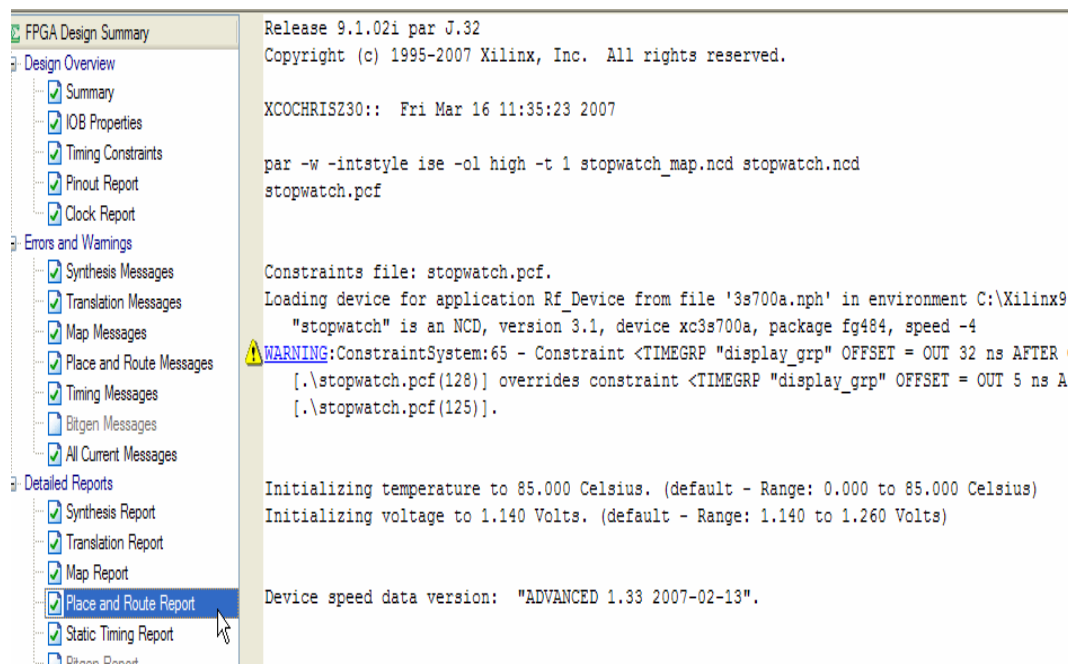


Figure 5-22: Design Summary of the Place & Route Report

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) and 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 of the FPGA:

1. Click the **+** next to Place & Route to expand the process hierarchy, and double-click **View/Edit Routed Design (FPGA Editor)**.

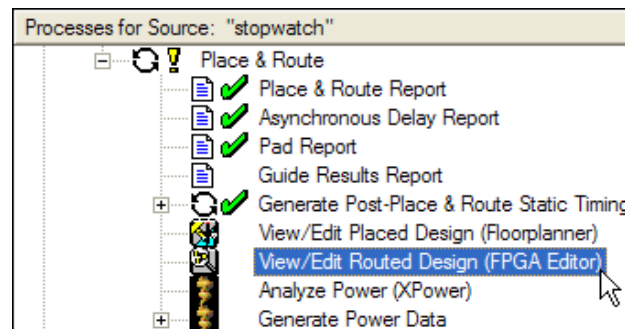


Figure 5-23: **View/Edit Routed Design (FPGA Editor) Process**

- In FPGA Editor, change the List Window from All Components to **All Nets**. This enables you to view all of the possible nets in the design.

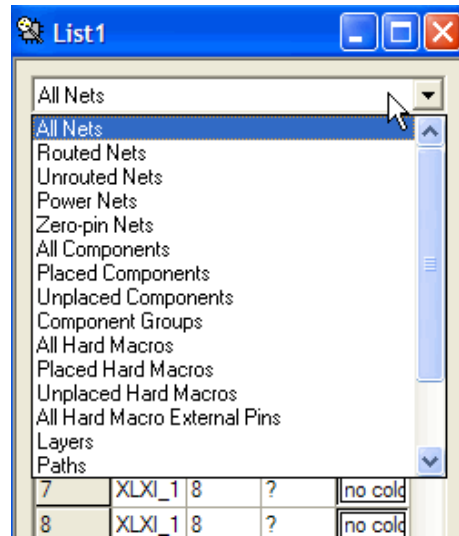


Figure 5-24: List Window in FPGA Editor

- Select the **clk_262144K** (Clock) net to see the fanout of the clock net.

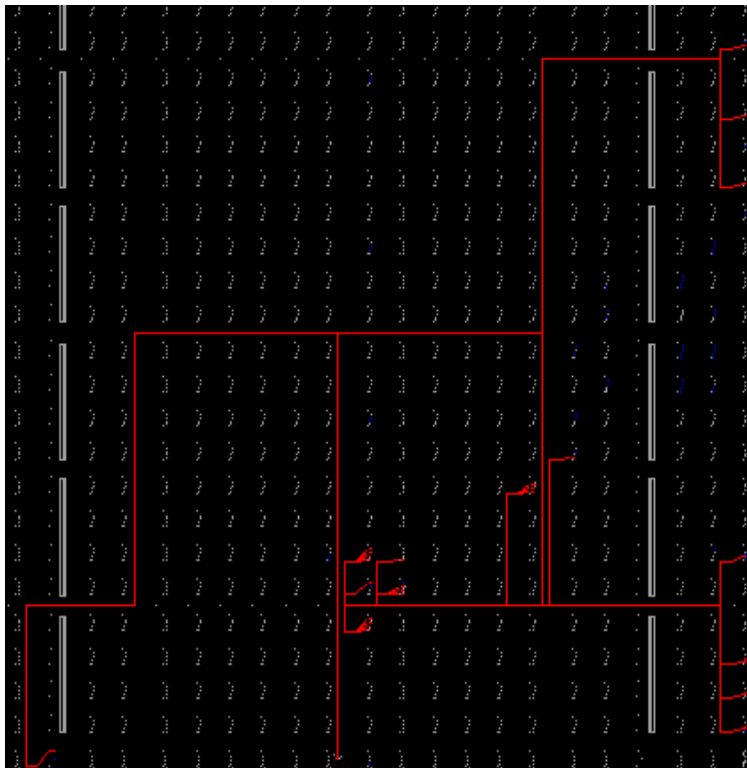


Figure 5-25: Clock Net

- 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. To display this report:

1. Expand the Generate Post-Place & Route Static Timing hierarchy.
2. Double-click **Analyze Post-Place & Route Static Timing Report** to open the report in Timing Analyzer.

Or you can select the Timing Constraint in the Design Summary and then a timing constraint hyperlink to launch Timing Analyzer.

The following is a summary of the Post-Place & Route Static Timing Report for the stopwatch design:

- ◆ The minimum period value increased due to the actual routing delays.
The Post-Map timing report showed logic delays contributed to 80% to 90% of the minimum period attained. The post-layout report indicates that the logical delay value now equals between 30% and 40% of the period. The total unplaced floors estimate changed as well.
 - ◆ The post-layout result does not necessarily follow the 50/50 rule previously described because the worst case path primarily includes component delays.
 - ◆ For some hard to meet timing constraints, the worst case path is mainly made up of logic delay. Since total routing delay makes up only a small percentage of the total path delay spread out across two or three nets, expecting the timing of these paths 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. Select the Timing tab in the Sources panel and select the Timing Objects tab in the Processes panel.
 4. With the TS_DCM_INST_CLKFX_BUF constraint highlighted in the Timing Analyzer tab, select the first hyperlink beginning with Maximum Data Path. This launches the Floorplan Implemented window and highlights the corresponding data path.
 5. Select **View > Overlays > Toggle Simplified and Actual Views**.
This changes the routing view for the highlighted data path from a simplified view to the actual routing

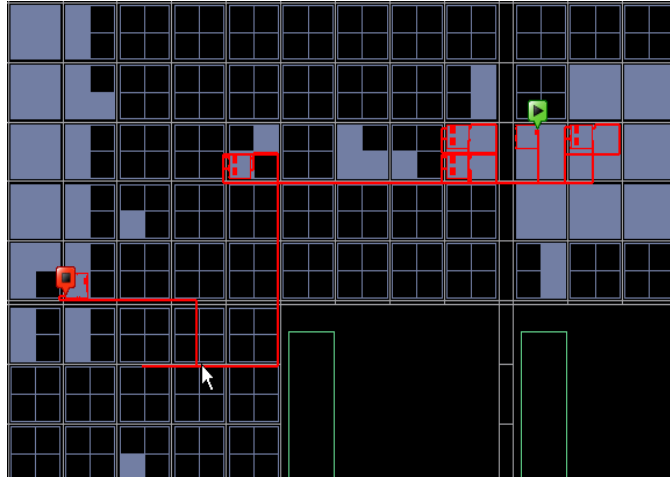


Figure 5-26: Floorplan Implemented - Data Path with Simplified Routing

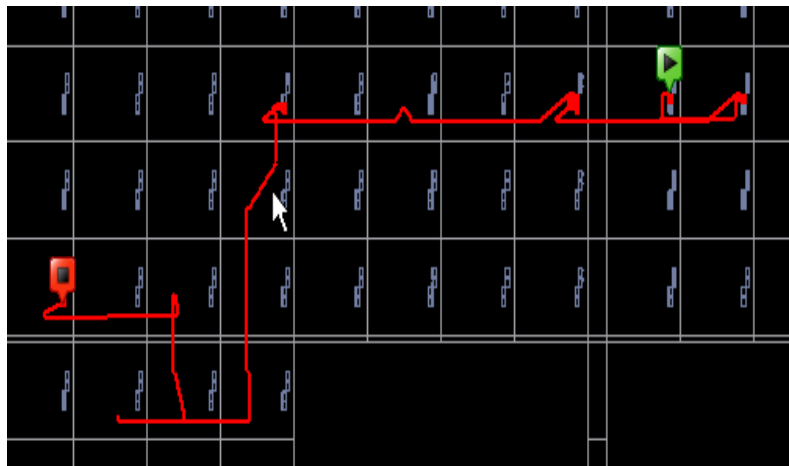


Figure 5-27: Floorplan Implemented - Data Path with Actual Routing

6. Hold the cursor over different nets or objects in the outlined path to observe timing delay and other information about that portion of the path.
7. Select **File > Close** to exit the Floorplan Implemented view.
8. Select **File > Close** again to close the Post-Place & Route Static Timing report.

Changing HDL with Partition

Now that the design has been implemented, the next step is to update the design with an HDL change. The change will be in the LCD_CNTRL_INST module, so only that partition will need to be re-synthesized and re-implemented. The Synthesis, MAP, and PAR reports will show which partitions are updated and which ones are preserved.

If you are doing the EDIF flow, you will not be able to do this section. Please proceed to the next section.

1. Open the LCD_CNTRL_INST module in the text editor by double-clicking it in the Sources tab.
2. Make the following change according to your HDL language:
 - ◆ If you are using Verilog: On line 377 and 564, change the code from `sf_d_temp = 8'b00111010; // [colon]` to `sf_d_temp = 8'b00101110; // [period]`
 - ◆ If you are using VHDL: On line 326 and 514, change the code from `sf_d_temp <= "00111010"; -- [colon]` to `sf_d_temp <= "00101110"; -- [period]`
3. To save the changes to LCD_CNTRL_INST, select **File > Save**.
4. In the Sources tab, select the top-level source file **stopwatch** and in the Processes tab, right-click on **Place & Route**.
5. Select **Run** from the menu.

Notice that the implementation is faster with partitions, since the implementation tools only need to re-implement the LCD_CNTRL_INST module. The rest of the design is re-used.

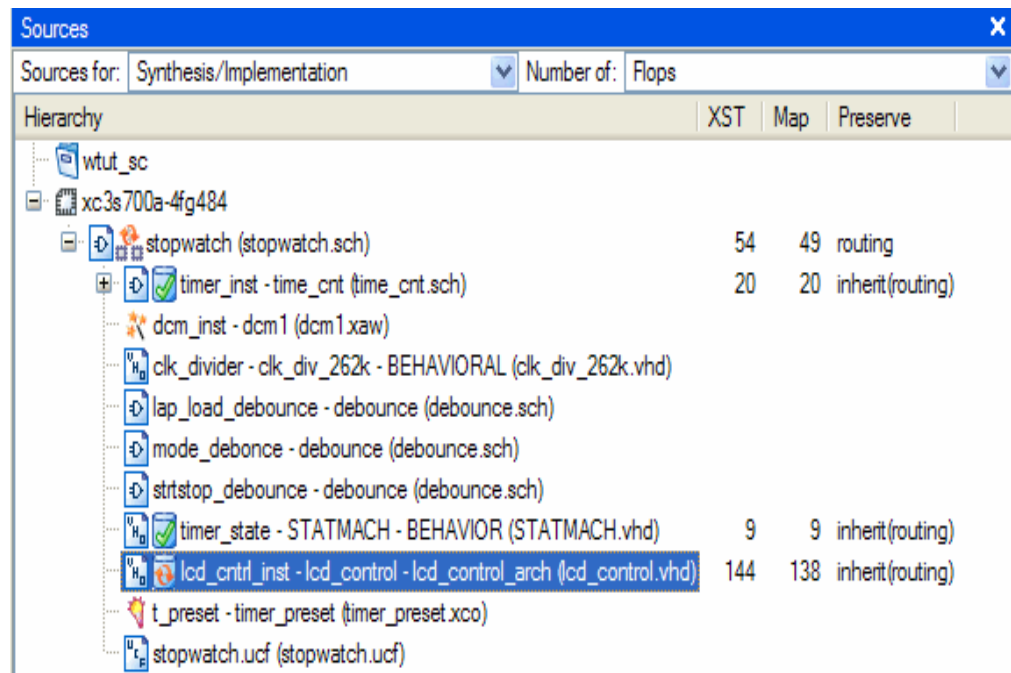


Figure 5-28: LCD_CNTRL_INST Partition is Out of Date

6. In the Design Summary view, view the following reports:
 - ◆ Select the **Synthesis Report**, then select **Partitions Report**. Notice which partitions were preserved and which partitions were re-used.
 - ◆ Select the **MAP Report**, then select **Guide Report**. Notice which partitions were preserved and which partitions were re-used.
 - ◆ Select the **Place and Route Report**, then select **Partition Status**. Notice which partitions were preserved and which partitions were re-used.

Creating Configuration Data

After analyzing the design through timing constraints in Timing Analyzer, you need to create configuration data. A configuration bitstream is created for downloading to a target device or for formatting into a PROM programming file.

In this tutorial, you will create configuration data for a Xilinx Serial PROM. To create a bitstream for the target device, set the properties and run configuration as follows:

1. Right-click the **Generate Programming File** process.
2. Select **Properties**. The Process Properties dialog box opens.

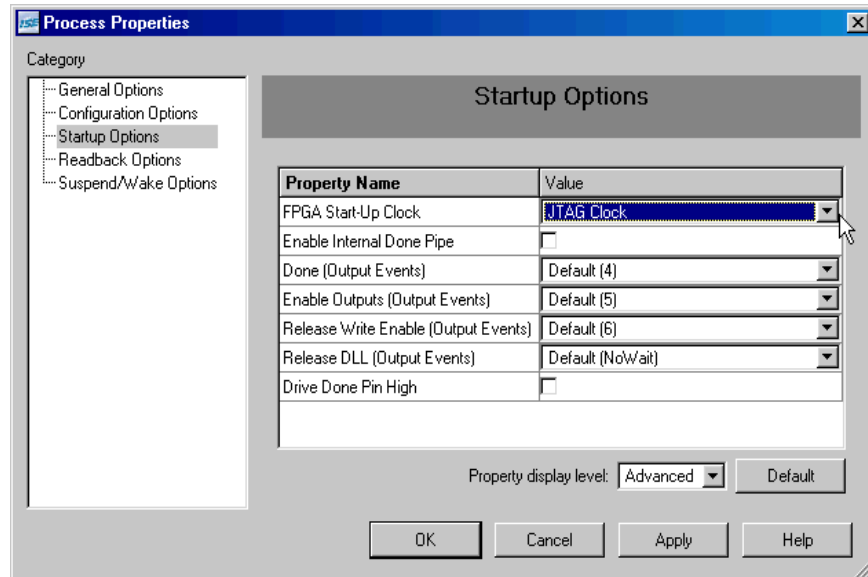


Figure 5-29: Process Properties Startup Options

3. Click the **Startup Options** category.
4. Change the FGPA Start-Up Clock property from CCLK to **JTAG Clock**.

Note: You can use CCLK if you are configuring Select Map or Serial Slave.

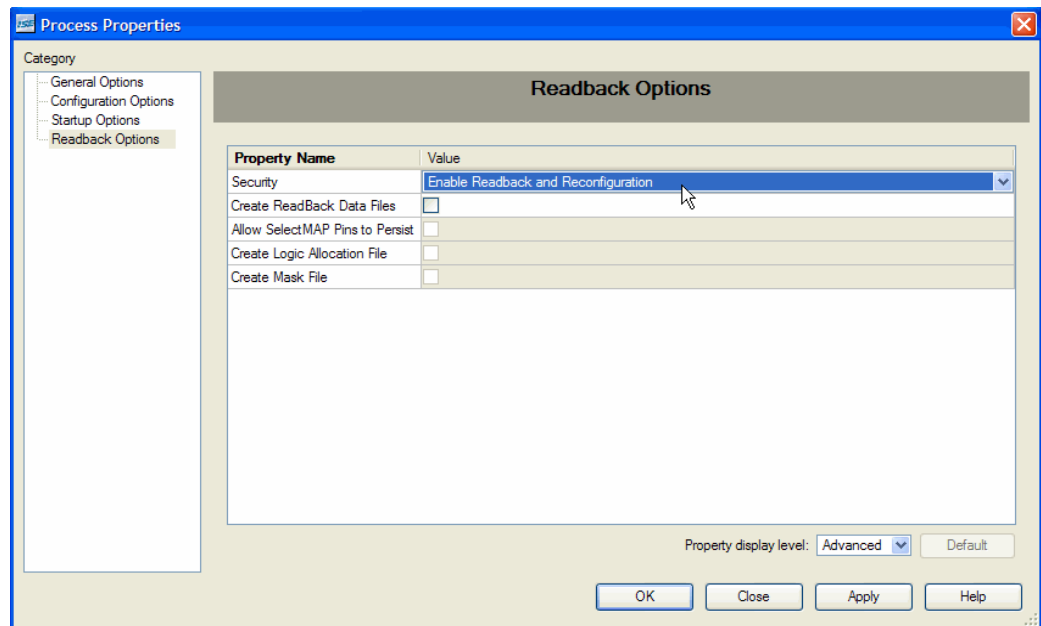


Figure 5-30: Process Properties Readback Options

5. Click the **Readback Options** category.
6. Change the Security property to **Enable Readback and Reconfiguration**.
7. Click **OK** to apply the new properties.
8. In the Processes tab, double-click **Generate Programming File** to create a bitstream of this design.

The BitGen program creates the bitstream file (in this tutorial, the `stopwatch.bit` file), which contains the actual configuration data.

9. Click the **+** next to Generate Programming File to expand the process hierarchy.
10. To review the Programming File Generation Report, double-click **Programming File Generation Report**. Verify that the specified options were used when creating the configuration data.

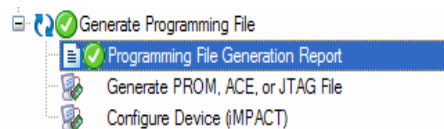


Figure 5-31: Programming File Generation Report Process

Creating a PROM File with iMPACT

To program a single device using iMPACT, all you need is a bitstream 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.

In iMPACT, a wizard enables you to do the following:

- Create a PROM file.
- Add additional bitstreams to the daisy chain.
- Create additional daisy chains.
- Remove the current bitstream and start over, or immediately save the current PROM file configuration.

For this tutorial, create a PROM file in iMPACT as follows:

1. In the Processes tab, double-click **Generate PROM, ACE, JTAG File**, located under the Generated Programming File process hierarchy.
2. In the Welcome to iMPACT dialog box, select **Prepare a PROM File**.
3. Click **Next**.

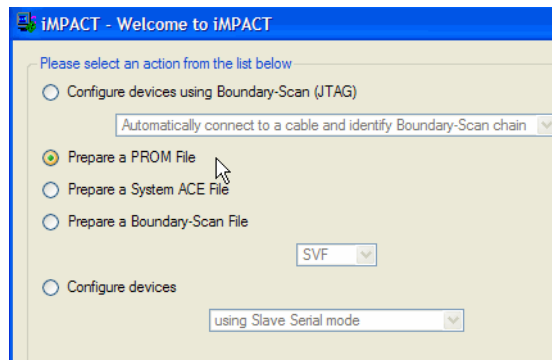


Figure 5-32: Welcome to iMPACT Dialog Box

4. In the Prepare PROM Files dialog box, set the following values:
 - ◆ Under “I want to target a:”, select **Xilinx PROM**.
 - ◆ Under PROM File Format, select **MCS**.
 - ◆ For PROM File Name, type **stopwatch1**.

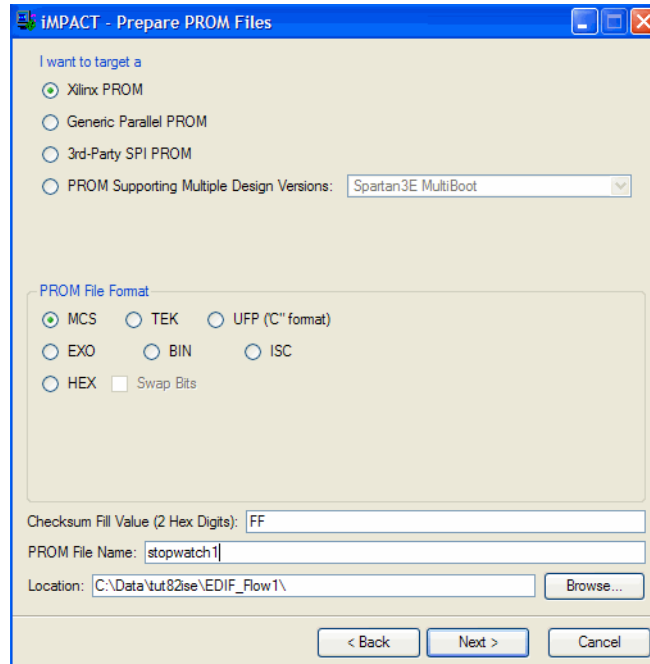


Figure 5-33: Prepare PROM Files Dialog Box

5. Click **Next**.
6. In the Specify Xilinx Serial PROM Device dialog box, select **Auto Select PROM**.
7. Click **Next**.

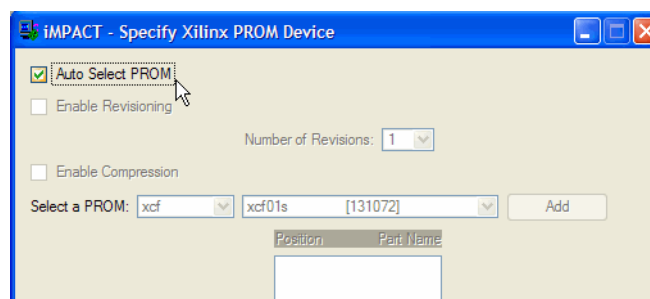


Figure 5-34: Specify Xilinx PROM Device Dialog Box

8. In the File Generation Summary dialog box, click **Finish**.
9. In the Add Device File dialog box, click **OK** and then select the `stopwatch.bit` file.

Note: You will receive a warning that the startup clock is being changed from `jtag` to `CCLK`.
10. Click **No** when you are asked if you would like to add another design file to the datastream.

11. Select **Operations > Generate File**.

iMPACT displays the PROM associated with your bit file.

12. To close iMPACT, select **File > Close**.

This completes the Design Implementation chapter of the tutorial. For more information on this design flow and implementation methodologies, see the ISE Help, available from the ISE application by selecting **Help > Help Topics**.

With the resulting `stopwatch.bit`, `stopwatch1.mcs` and a MSK file generated along with the BIT file, you are ready for programming your device using iMPACT. For more information on programming a device, see the iMPACT Help, available from the iMPACT application by selecting **Help > Help Topics**.

Command Line Implementation

ISE allows a user to easily view and extract the command line arguments for the various steps of the implementation process. This allows a user to verify the options being used or to create a command batch file to replicate the design flow.

At any stage of the design flow you can look at the command line arguments for completed processes by double-clicking **View Command Line Log File** from the Design Entry Utilities process hierarchy in the Processes tab. This process opens a file named `<source_name>.cmd_log` in read-only mode.

To create an editable batch file, select **File > Save As** and enter the desired file name.

Sections of the Command Line Log File may also be copied from `<source_name>.cmd_log` using either the copy-and-paste method or the drag-and-drop method into a text file.

For a complete listing of command line options for most Xilinx executables, refer to the *Development System Reference Guide*. Command line options are organized according to implementation tools. This Guide is available with the collection of software manuals and is accessible from ISE by selecting **Help > Software Manuals**, or from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/. Command line options may also be obtained by typing the executable name followed by the `-h` option at a command prompt.

Another useful tool for automating design implementation is XFLOW. XFLOW is a Xilinx command line tool that automates the Xilinx implementation and simulation flows. XFLOW reads a design file as input, flow and option files. For more information on XFLOW, refer to the "XFLOW" section in the *Development System Reference Guide*.

Timing Simulation

This chapter includes the following sections.

- [“Overview of Timing Simulation Flow”](#)
- [“Getting Started”](#)
- [“Timing Simulation Using ModelSim”](#)
- [“Timing Simulation Using Xilinx ISE Simulator”](#)

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 highly recommended part of the HDL design flow for Xilinx® devices. Timing simulation uses the detailed timing and design layout information that is available after place and route. This enables simulation of the design, which closely matches the actual device operation. Performing a timing simulation in addition to a static timing analysis will help to uncover issues that cannot be found in a static timing analysis alone. To verify the design, the design should be analyzed both statically and dynamically.

In this chapter, you will perform a timing simulation using either the ModelSim simulator or the Xilinx ISE Simulator.

Getting Started

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

Required Software

To simulate with ModelSim, you must have Xilinx ISE™ 9.1i and ModelSim simulator installed. Refer to [Chapter 4, “Behavioral Simulation”](#) for information on installing and setting up ModelSim. Simulating with the Xilinx ISE simulator requires that the ISE 9.1i software is installed

Required Files

The timing simulation flow requires the following files:

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

This chapter assumes that you have completed [Chapter 5, “Design Implementation,”](#) and thus, have a placed and routed design. The NetGen tool will be used in this chapter to create a simulation netlist from the placed and routed design which will be used to represent the design during the Timing Simulation.

- **Test Bench File (VHDL or Verilog)**

In order to simulate the design, a test bench is needed to provide stimulus to the design. You should use the same test bench that was used to perform the behavioral simulation. Please refer to the [“Adding an HDL Test Bench” in Chapter 4](#) if you do not already have a test bench in your project.

- **Xilinx Simulation Libraries**

For timing simulation, the SIMPRIM library is needed to simulate the design.

To perform timing simulation of Xilinx designs in any HDL simulator, the SIMPRIM library must be set up correctly. The timing simulation netlist created by Xilinx is composed entirely of instantiated primitives, which are modeled in the SIMPRIM library.

If you completed Chapter 4, “Behavioral Simulation”, the SIMPRIM library should already be compiled. For more information on compiling and setting up Xilinx simulation libraries, see to [“Xilinx Simulation Libraries” in Chapter 4](#).

Specifying a Simulator

To select either the desired simulator to simulate the stopwatch design, complete the following:

1. In the Sources tab, right-click the device line (xc3s700A-4fg484) and select **Properties**.
2. In the Project Properties dialog box click the down arrow in the Simulator value field to display a list of simulators.

Note: ModelSim simulators and the ISE Simulator are the only simulators that are integrated with Project Navigator. Selecting a different simulator (e.g. NC-Sim or VCS) will set the correct options for Netgen to create a simulation netlist for that simulator but Project Navigator will not directly open the simulator. For additional information about simulation, and for a list of other supported simulators, see Chapter 5 of the Synthesis and Verification Guide. This Guide is accessible from within ISE by selecting Help > Software Manuals, and from the web at http://www.xilinx.com/support/sw_manuals/xilinx9/.

3. Select **ISE Simulator (VHDL/Verilog)** or **Modelsim** with the appropriate version and language in the Simulator value field.

Timing Simulation Using ModelSim

Xilinx ISE provides an integrated flow with the Mentor ModelSim simulator. ISE enables you to create work directories, compile source files, initialize simulation, and control simulation properties for ModelSim.

Note: To simulate with the ISE Simulator, skip to [“Timing Simulation Using Xilinx ISE Simulator”](#). Whether you choose to use the ModelSim simulator or the ISE Simulator for this tutorial, the end result is the same.

Specifying Simulation Process Properties

To set the simulation process properties:

1. In the Sources tab, select **Post-Route Simulation** in the Sources for field.
2. Select the test bench file (`stopwatch_tb`).
3. In the Processes tab, click the **+** next to ModelSim Simulator to expand the process hierarchy.

Note: If the ModelSim Simulator processes do not appear, it means that either ModelSim is not selected as the Simulator in the Project Properties dialog box, or Project Navigator cannot find `modelsim.exe`.

If ModelSim is installed but the processes are not available, the Project Navigator preferences may not be set correctly. To set the ModelSim location, select **Edit > Preferences**, click the **+** next to ISE General to expand the ISE preferences, and click **Integrated Tools** in the left pane. In the right pane, under Model Tech Simulator, browse to the location of `modelsim.exe` file. For example,

```
c:\modeltech_xe\win32xoem\modelsim.exe.
```

4. Right-click **Simulate Post-Place & Route Model**.
5. Select **Properties**.

The Process Properties dialog box displays.

6. Select the **Simulation Model Properties** category.

The properties should appear as shown in [Figure 6-1](#). These properties set the options that NetGen uses when generating the simulation netlist. For a description of each property, click the **Help** button.

7. Ensure that you have set the Property display level to **Advanced**.

This global setting enables you to see all available properties.

For this tutorial, the default Simulation Model Properties are used.

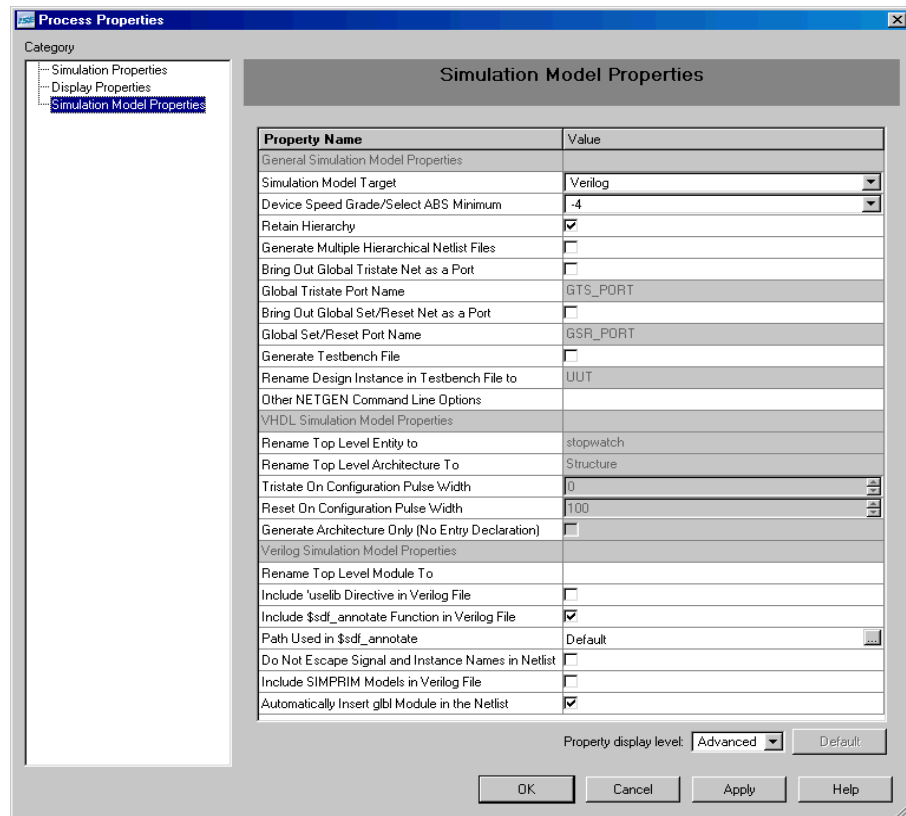


Figure 6-1: Simulation Model Properties

8. Select the **Display Properties** category.

This tab gives you control over the 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 Guide*.

9. Select the **Simulation Properties** category.

The properties should appear as shown in [Figure 6-2](#). These properties set the options that ModelSim uses to run the timing simulation. For a description of each property, click the **Help** button.

10. In the Simulation Properties tab, set the Simulation Run Time property to **2000 ns**.

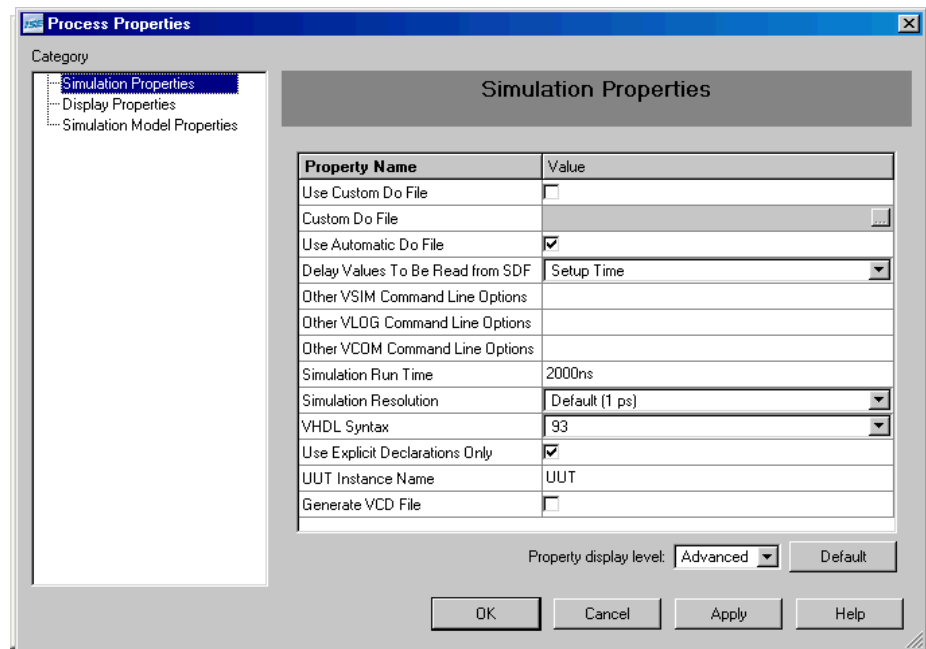


Figure 6-2: Simulation Properties

11. Click **OK** to close the Process Properties dialog box.

Performing Simulation

To start the timing simulation, double-click **Simulate Post-Place and Route Model** in the Processes tab.

ISE will run NetGen to create the timing simulation model. ISE will then call ModelSim and create the working directory, compile the source files, load the design, and run the simulation for the time specified.

Note: The majority of this design runs at 100 Hz and would take a significant amount of time to simulate. This is why the counter will seem like it is not working in a short simulation. For the purpose of this tutorial, only the DCM signals will be monitored to verify that they work correctly.

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 on 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/Object window.
- Highlight signals in the Signal/Object window and then select **Add > Wave > Selected Signals**.

The following procedure explains how to add additional signals in the design hierarchy. In this tutorial, you will be adding the DCM signals to the waveform.

Note: If you are using ModelSim version 6.0 or higher, all the windows are docked by default. All windows can be undocked by clicking the **Undock** icon.



Figure 6-3: Undock icon

1. In the Structure/Instance window, click the **+** next to uut to expand the hierarchy.

Figure 6-4 shows the Structure/Instance window for the Schematic flow. The graphics and the layout of the Structure/Instance window for a schematic or VHDL flow may appear different.

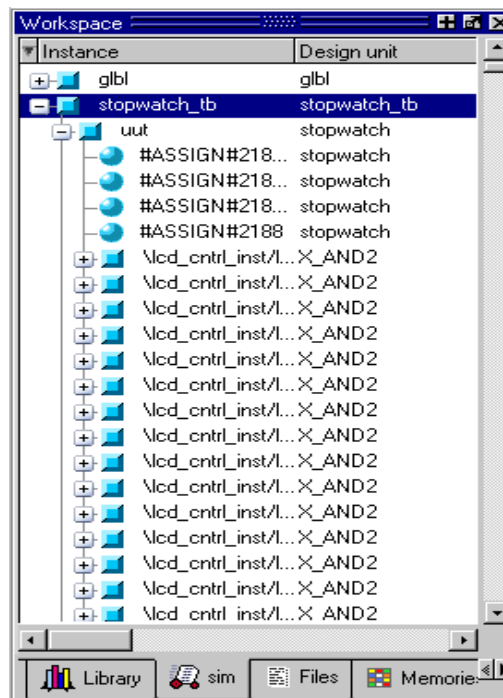


Figure 6-4: Structure/Instance Window - Schematic Flow

2. Click the Structure/Instance window and select **Edit > Find**.
3. Type in **X_DCM** in the search box and select **Entity/Module** in the Field section.
4. Once ModelSim locates X_DCM, select **X_DCM** and click on the signals/objects window. All the signal names for the DCM will be listed.
5. Select the Signal/Object window and select **Edit > Find**.
6. Type **CLKIN** in the search box and select the **Exact** checkbox.
7. Click and drag CLKIN from the Signal/Object window to the Wave window.

8. Click and drag the following signals from the Signal/Object window to the Wave window:
 - ◆ RST
 - ◆ CLKFX
 - ◆ CLK0
 - ◆ LOCKED

Note: Multiple signals can be selected by holding down the **Ctrl** key. In place of using the drag and drop method select **Add to Wave > Selected Signals**.

Adding Dividers

Modelsim has the capability to add dividers in the Wave window to make it easier to differentiate the signals. To add a divider called DCM Signals:

1. Click anywhere in the Wave window.
2. If necessary, undock the window and then maximize the window for a larger view of the waveform.
3. Select **Insert > Divider**.
4. Enter **DCM Signals** in the Divider Name box.
5. Click and drag the newly created divider to above the CLKIN signal.

Note: Stretch the first column in the waveform to see the signals clearly. The hierarchy in the signal name can also be turned off by selecting **Tools > Options > Wave Preferences**. In the Display Signal Path box, enter **2** and click **OK**.

The waveform should look as shown in [Figure 6-5](#).

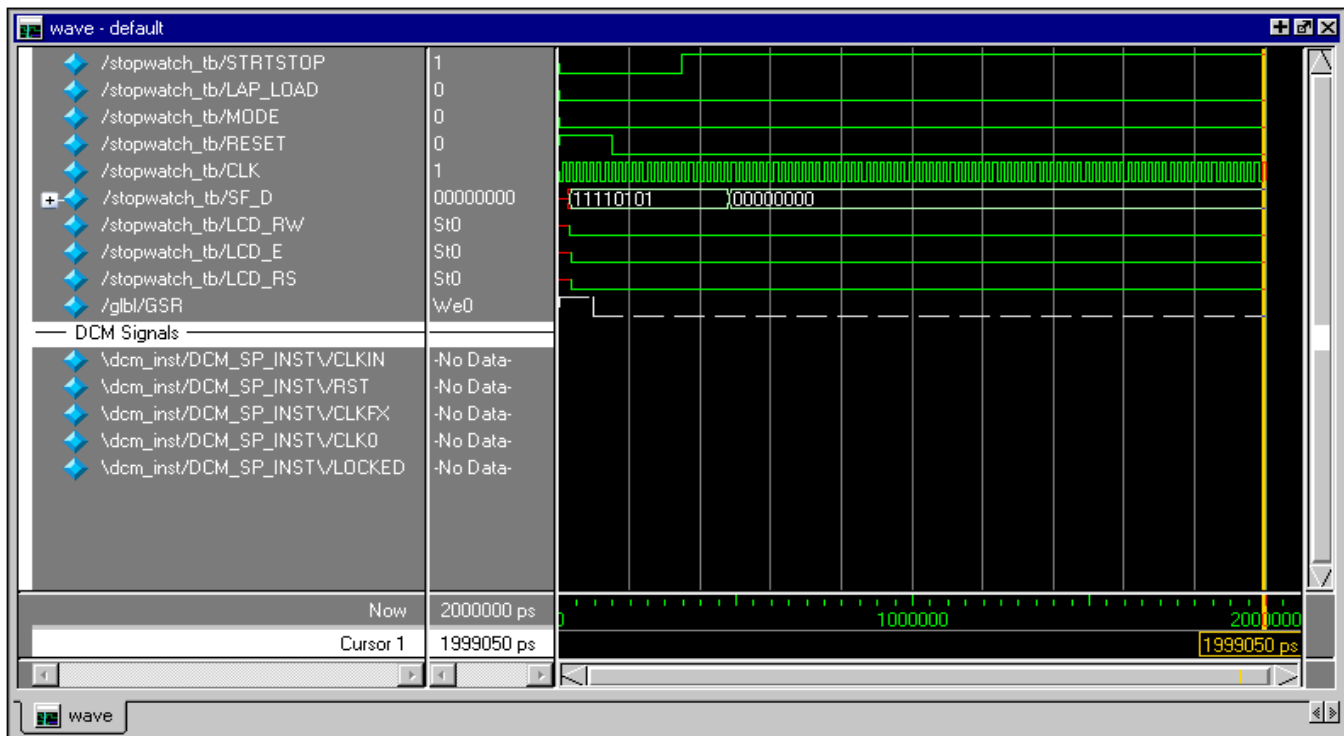


Figure 6-5: The Resulting Waveform

Notice that the waveforms have not been drawn for the newly added signals. This is because ModelSim did not record the data for these signals. By default, ModelSim will only record data for the signals that have been added to the Wave window while the simulation is running. Therefore, after new signals are added to the Wave window, you need to rerun the simulation for the desired amount of time.

Rerunning Simulation

To restart and re-run the simulation:

1. Click the **Restart Simulation** icon.



Figure 6-6: Restart Simulation Icon

The Restart dialog box opens.

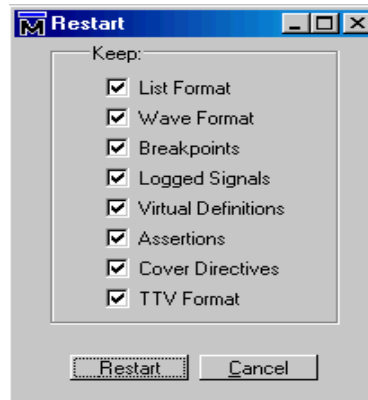


Figure 6-7: Restart Dialog Box

2. Click **Restart**.
3. At the ModelSim command prompt, enter `run 2000 ns` and hit the **Enter** key.

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

Figure 6-8: Entering the Run Command

The simulation will run for 2000 ns. The waveforms for the DCM should now be visible in the Wave window.

Analyzing the Signals

Now the DCM signals can be analyzed to verify that it works as expected. The CLK0 needs to be 50 Mhz and the CLKFX should be ~26 Mhz. The DCM signals should only be analyzed after the LOCKED signal has gone high. Until the LOCKED signal is high the DCM outputs are not valid.

Modelsim has the capability to add cursors to carefully measure the distance between signals.

To measure the CLK0:

1. Select **Add > Cursor** twice to place two cursors on the wave view.
2. Click and drag the first cursor to the rising edge transition on the CLK0 signal after the LOCKED signal has gone high.
3. Click and drag the second cursor to a position just right of the first cursor on the CLK0 signal.
4. Click the **Find Next Transition** icon twice to move the cursor to the next rising edge on the CLK0 signal.



Figure 6-9: Find Next Transition Icon

Look at the bottom of the waveform to view the distance between the two cursors. The measurement should read 20000 ps. This converts to 50 Mhz, which is the input frequency from the test bench, which in turn should be the DCM CLK0 output.

Measure CLKFX using the same steps as above. The measurement should read 38462 ps. This equals approximately 26 Mhz.

Saving the Simulation

The ModelSim Simulator provides the capability of saving the signals list in the Wave window. Save the signals list after new signals or stimuli are added, and after simulation is rerun. 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, rename the filename from the default wave.do to dcm_signal_tim.do.

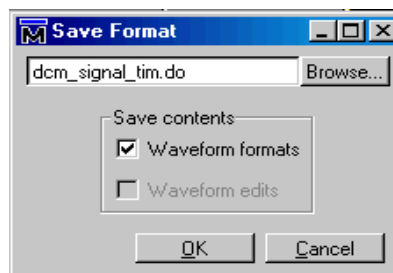


Figure 6-10: Save Format Dialog Box

3. Click **Save**.

After restarting the simulation, you can select **File > Load** in the Wave window to reload this file.

Your timing simulation is complete and you are ready to program your device by following [Chapter 7, "iMPACT Tutorial."](#)

Timing Simulation Using Xilinx ISE Simulator

Follow this section of the tutorial if you have skipped the previous section, “[Timing Simulation Using ModelSim](#).”

Specifying Simulation Process Properties

To set the simulation process properties:

1. In the Sources tab, select **Post-Route Simulation** in the Sources for field.
2. Select the test bench file (`stopwatch_tb`).
3. In the Processes tab, click the **+** next to Xilinx ISE Simulator to expand the process hierarchy.
4. Right-click **Simulate Post-Place & Route Model**.
5. Select **Properties**.

The Process Properties dialog box displays.

6. Select the **Simulation Model Properties** category.

These properties set the options that NetGen uses when generating the simulation netlist. For a description of each property, click the **Help** button.

7. Ensure that you have set the Property display level to **Advanced**.

This global setting enables you to now see all available properties.

For this tutorial, the default Simulation Model Properties are used.

8. Select the **ISE Simulator Properties** category.

The properties should appear as shown in [Figure 6-11](#). These properties set the options the simulator uses to run the timing simulation. For a description of each property, click the **Help** button.

9. In the Simulation Properties tab, set the Simulation Run Time property to **2000 ns**.

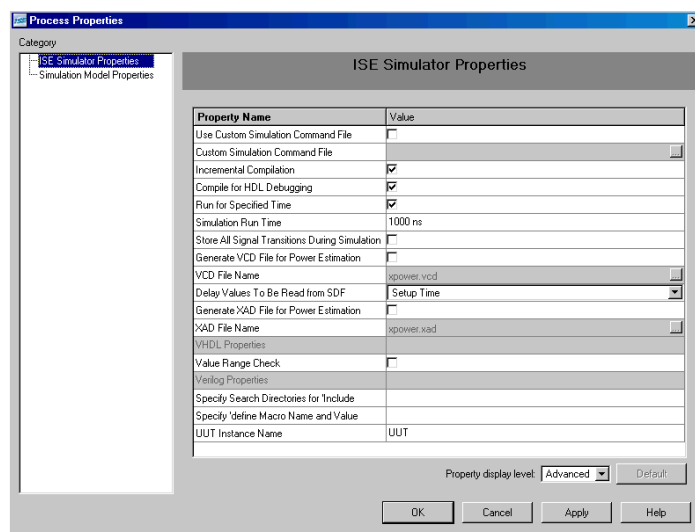


Figure 6-11: Simulation Properties

10. Click **OK** to close the Process Properties dialog box.

Performing Simulation

To start the timing simulation, double-click **Simulate Post-Place and Route Model** in the Processes tab.

When a simulation process is run, Project Navigator automatically runs NetGen to generate a timing simulation model from the placed and routed design. The ISE Simulator will then compile the source files, load the design, and run the simulation for the time specified.

Note: The majority of this design runs at 100 Hz and would take a significant amount of time to simulate. This is why the counter will seem like it is not working in a short simulation. For the purpose of this tutorial, only the DCM signals will be monitored to verify that they work correctly.

Adding Signals

To view signals during the simulation, you must add them to the waveform window. ISE automatically adds all the top-level ports to the waveform window. All available external (top-level ports) and internal signals are displayed in the Sim Hierarchy window.

The following procedure explains how to add additional signals in the design hierarchy. In this tutorial, you will be adding the DCM signals to the waveform.

1. In the Sim Hierarchy window, click the **+** next to uut to expand the hierarchy.
2. Right click in the Sim Hierarchy window and select **Find**.
3. Type the **locked** in the Find Signal dialog box and click on **OK**.
4. Select DCM_SP_INST/LOCKED signal and click on **OK**.

Figure 6-12 shows the Sim Hierarchy window for the VHDL flow. The signal names and layout in the Sim Hierarchy window for a schematic or VHDL flow may appear different.

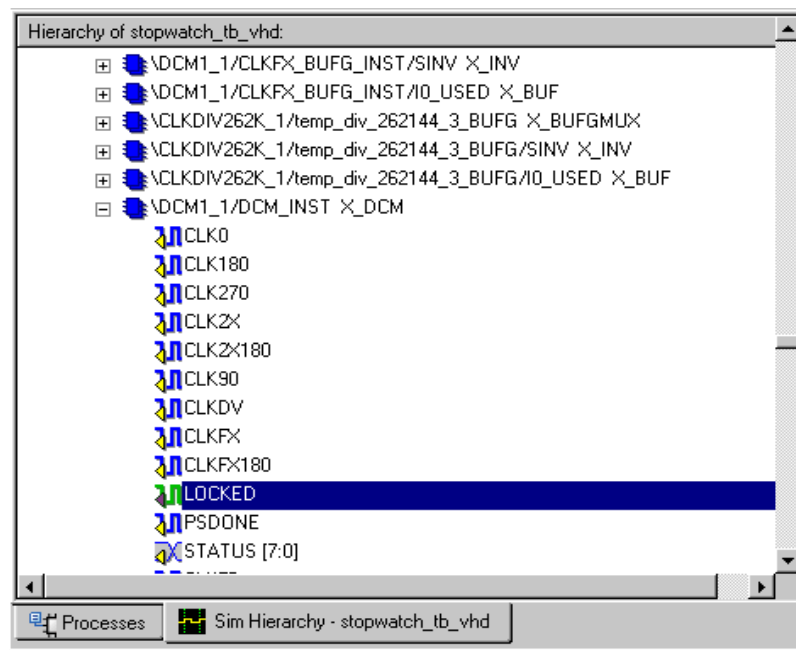


Figure 6-12: Sim Hierarchy Window - VHDL Flow

5. Click and drag LOCKED from the Sim Hierarchy window to the waveform window.

6. Click and drag the following X_DCM_SP signals from the SIM Hierarchy window to the waveform window:
 - ◆ RST
 - ◆ CLKFX
 - ◆ CLK0
 - ◆ CLKIN

Note: Multiple signals can be selected by holding down the **Ctrl** key.

Viewing Full Signal Names

A signal name may be viewed with either the complete hierarchical name or by the short name which omits hierarchy information. To change the signal name display;

1. Right click the desired signal in the waveform window.
2. Select Long Name or Short Name as desired.

Note: Stretch the first column in the waveform to see the signals clearly.

The waveform should appear as shown in [Figure 6-13](#).

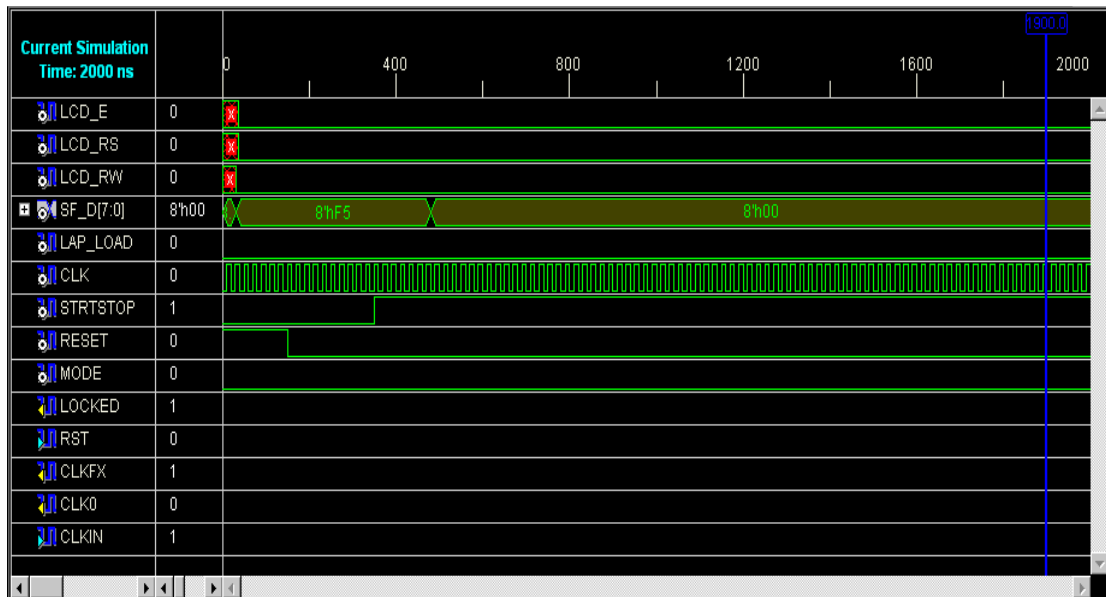


Figure 6-13: The Resulting Waveform

Notice that the waveforms have not been drawn for the newly added signals. This is because the ISE Simulator did not record the data for these signals. The ISE Simulator will only record data for the signals that have been added to the waveform window while the simulation is running. Therefore, after new signals are added to the waveform window, you need to rerun the simulation for the desired amount of time.

Rerunning Simulation

To restart and re-run the simulation:

1. Click the **Restart Simulation** icon.



Figure 6-14: Restart Simulation Icon

2. At the Sim Console command prompt, enter **run 2000 ns** and hit the **Enter** key.

```
% |run 2000 ns
```

Figure 6-15: Entering the Run Command

The simulation will run for 2000 ns. The waveforms for the DCM should now be visible in the Simulation window.

Analyzing the Signals

Now the DCM signals can be analyzed to verify that it does work as expected. The CLK0 needs to be 50 Mhz and the CLKFX should be ~26 Mhz. The DCM signals should only be analyzed after the LOCKED signal has gone high. Until the LOCKED signal is high the DCM outputs are not valid.

ISE Simulator has the capability to add cursors to carefully measure the distance between signals.

To measure the CLK0:

1. Right click the wave window and select **Add Measure**. The cursor changes to an up-arrow pointer.
2. Click a rising edge transition on the CLK0 signal after the LOCKED signal has gone high. Two vertical markers are placed on the waveform.
3. Click and drag the second marker to the next rising edge on the CLK0 signal.

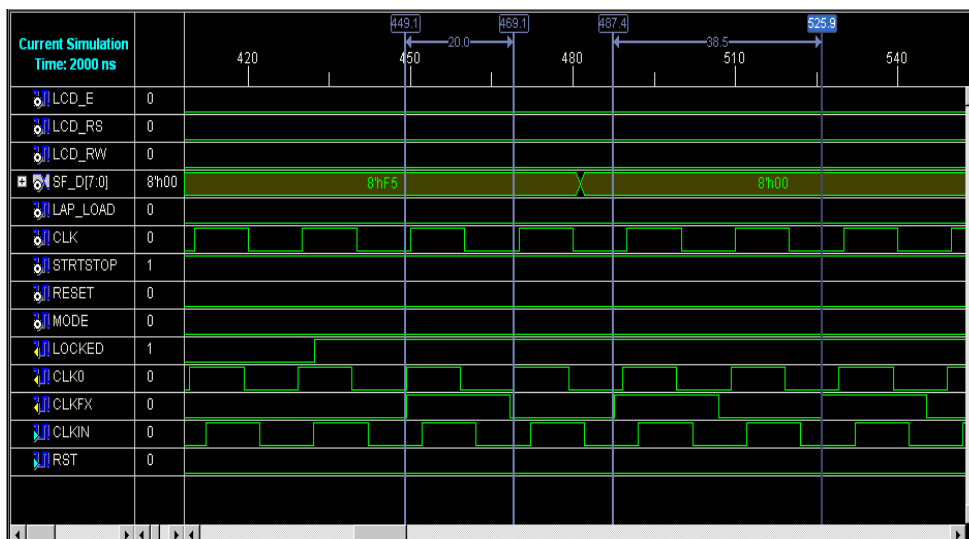


Figure 6-16: Adding Timing Markers

Note: You may need to Zoom In to place the markers precisely on the clock edge.

View the time value between the two markers to see the distance between the two clock edges. The measurement should read 20.0 ns. This converts to 50 Mhz, which is the input frequency from the test bench, which in turn should be the DCM CLK0 output.

Measure CLKFX using the same steps as above. The measurement should read 38.5 ns, this equals approximately 26 Mhz.

Your timing simulation is complete and you are ready to program your device by following [Chapter 7, "iMPACT Tutorial."](#)

iMPACT Tutorial

This chapter takes you on a tour of iMPACT, a file generation and device programming tool. iMPACT enables you to program through several parallel cables, including the Platform Cable USB. iMPACT can create bit files, System ACE files, PROM files, and SVF/XSVF files. The SVF/XSVF files can be played backed without having to recreate the chain.

This tutorial contains the following sections:

- “Device Support”
- “Download Cable Support”
- “Configuration Mode Support”
- “Getting Started”
- “Creating a iMPACT New Project File”
- “Using Boundary Scan Configuration Mode”
- “Troubleshooting Boundary Scan Configuration”
- “Creating an SVF File”
- “Other Configuration Modes”

Device Support

The following devices are supported.

- Virtex™/-E/-II/-II PRO/4/5
- Spartan™/-II/-IIE/XL/3/3E/3A
- XC4000™/E/L/EX/XL/XLA/XV
- CoolRunner™XPLA3/-II
- XC9500™/XL/XV
- XC18V00P
- XCF00S
- XCF00P

Download Cable Support

Parallel Cable IV

The Parallel Cable connects to the parallel port and can be used to facilitate Slave Serial and Boundary-Scan functionality. For more information, go to <http://www.xilinx.com/support>, select **Documentation > Data Sheets > Configuration Solutions > Configuration Hardware > Xilinx Parallel Cable IV**.

Platform Cable USB

The Platform Cable connects to the USB port and can be used to facilitate Slave Serial, and Boundary Scan functionality. For more information, go to <http://www.xilinx.com/support>, select **Documentation > Data Sheets > Configuration Solutions > Configuration Hardware > Platform Cable USB**.

MultiPRO Cable

The MultiPRO cable connects to the parallel port and can be used to facilitate Desktop Configuration Mode functionality. For more information, go to <http://www.xilinx.com/support>, select **Documentation > Data Sheets > Configuration Solutions > Configuration Hardware > MultiPRO Desktop Tool**.

Configuration Mode Support

Impact currently supports the following configuration modes:

- Boundary Scan—FPGAs, CPLDs, and PROMs(18V00,XCFS,XCFP)
- Slave Serial—FPGAs (Virtex™/-II/-II PRO/E/4/5 and Spartan™/-II/-IIE/3/3E/3A)
- SelectMAP—FPGAs (Virtex™/-II/-II PRO/E/4/5 and Spartan™/-II/-IIE/3/3E/3A)
- Desktop—FPGAs (Virtex™/-II/-II PRO/E/4/5 and Spartan™/-II/-IIE/3/3E/3A)

Getting Started

Generating the Configuration Files

In order to follow this chapter, you must have the following files for the stopwatch design:

- a BIT file—a binary file that contains proprietary header information as well as configuration data.
- a MCS file—an ASCII file that contains PROM configuration information.
- a MSK file—a binary file that contains the same configuration commands as a BIT file, but that has mask data in place of configuration data. This data is not used to configure the device, but is used for verification. If a mask bit is 0, the bit should be verified against the bit stream data. If a mask bit is 1, the bit should not be verified. This file generated along with the BIT file.

These files are generated in [Chapter 5, “Design Implementation.”](#)

- The Stopwatch tutorial projects can be downloaded from <http://www.xilinx.com/support/techsup/tutorials/tutorials8.htm>. Download the project files for either the VHDL, Verilog or Schematic design flow.

Connecting the Cable

Prior to launching iMPACT, connect the parallel side of the cable to your computer’s parallel port, and connect the cable to the Spartan-3 Starter Kit demo board. Be sure that the board is powered.

Starting the Software

This section describes how to start the iMPACT software from ISE™ and how to run it stand-alone.

Opening iMPACT from Project Navigator

To start iMPACT from Project Navigator, double-click **Configure Device (iMPACT)** in the Processes tab in the Processes window (see [Figure 7-1](#)).

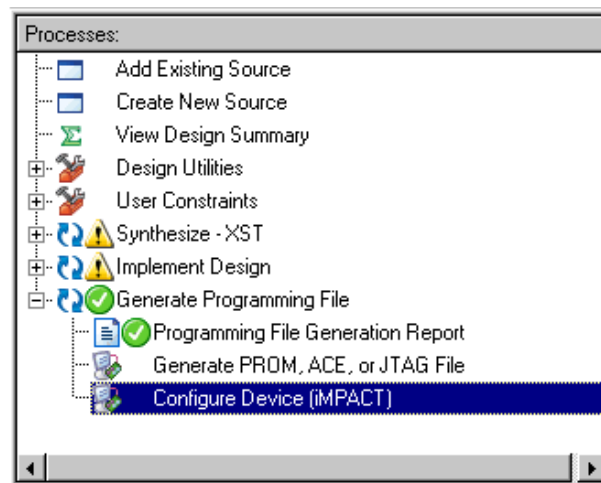


Figure 7-1: Opening iMPACT from ISE

Opening iMPACT stand-alone

To open iMPACT without going through an ISE project, use one of the following methods.

- PC — Click **Start > All Programs > Xilinx® ISE 9.1i > Accessories > iMPACT**.
- PC, UNIX, or Linux — Type **impact** at a command prompt.

Creating a iMPACT New Project File

When iMPACT is initially opened, the iMPACT Project dialog box displays. This dialog box enables you to load a recent project or to create a new project.

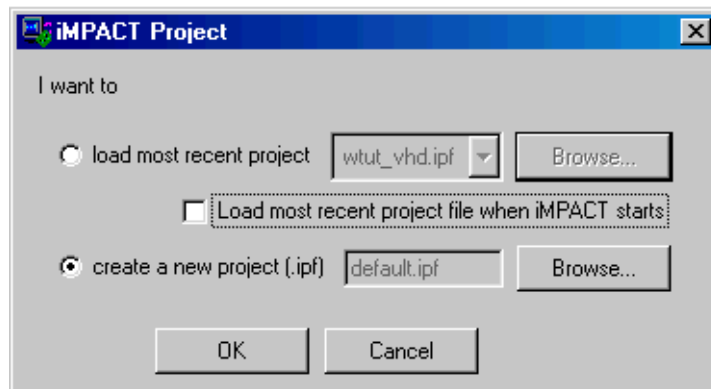


Figure 7-2: Creating an iMPACT Project

To create a new project for this tutorial:

1. In the iMPACT Project dialog box, select **create a new project (.ipf)**.
2. Click the **Browse** button.
3. Browse to the project directory and then enter stopwatch in the File Name field.
4. Click **Save**.
5. Click **OK**.

This creates a new project file in iMPACT. You are prompted to define the project, as described in the next section.

Using Boundary Scan Configuration Mode

For this tutorial, you will be using the Boundary Scan Configuration Mode. Boundary Scan Configuration Mode enables you to perform Boundary Scan Operations on any chain comprising JTAG compliant devices. The chain can consist of both Xilinx® and non-Xilinx devices; however, limited operations will be available for non-Xilinx devices. To perform operations, the cable must be connected and the JTAG pins, TDI, TCK, TMS, and TDO need to be connected from the cable to the board.

Specifying Boundary Scan Configuration Mode

After opening iMPACT, you are prompted to specify the configuration mode and which device you would like to program.

To select Boundary Scan Mode:

1. Select **Configure Devices using Boundary-Scan (JTAG)** and leave the selection box value of **Automatically connect to a cable and identify Boundary-Scan chain**.

Note: The selection box also gives you the option to Enter a Boundary Scan Chain, which enables you to then manually add devices to create chain. This option enables you to generate an SVF/XSVF programming file, and is discussed in a later section in this chapter. Automatically detecting and initializing the chain should be performed whenever possible.

2. Click **Finish**.

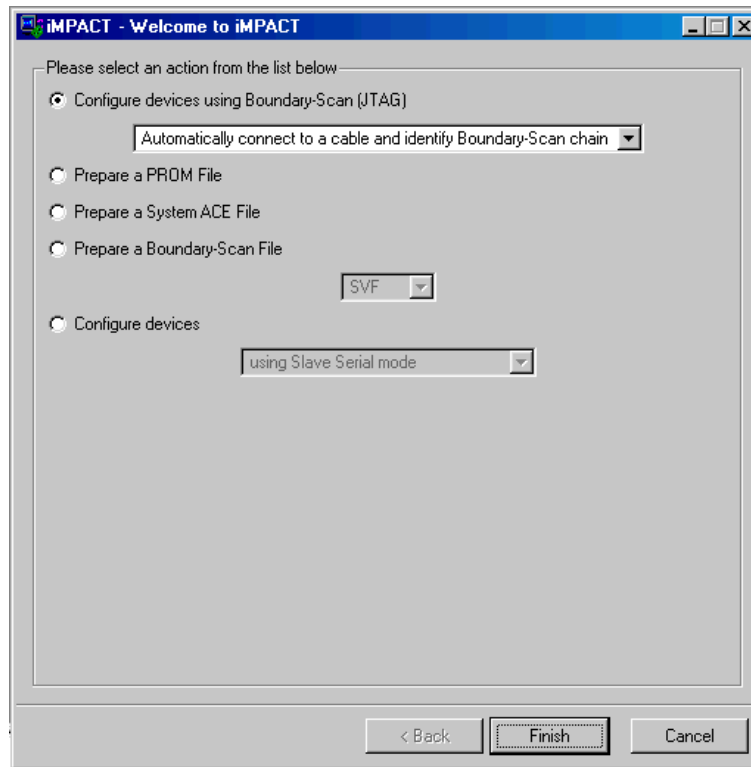


Figure 7-3: Selecting automatic boundary scan from Wizard

iMPACT will pass data through the devices and automatically identify the size and composition of the boundary scan chain. Any supported Xilinx device will be recognized and labeled in iMPACT. Any other device will be labeled as unknown. The software will then highlight each device in the chain and prompt you to assign a configuration file or BSDL file.

Note: If you were not prompted to select a configuration mode or automatic boundary scan mode, right-click in the iMPACT window and select **Initialize Chain**. The software will identify the chain if the connections to the board are working. Go to [“Troubleshooting Boundary Scan Configuration”](#) if you are having problems.

Assigning Configuration Files

After initializing a chain, the software prompts you for a configuration file (see Figure 7-4). The configuration file is used to program the device. There are several types of configuration files.

- A Bitstream file (*.bit, *.rbit, *.isc) is used to configure an FPGA.
- A JEDEC file (*.jed, *.isc) is used to configure a CPLD.
- A PROM file (*.mcs, .exo, .hex, or .tek) is used to configure a PROM.

When the software prompts you to select a configuration file for the first device (XC3S200):

1. Select the BIT file from your project working directory.
2. Click **Open**.

You should receive a warning stating that the startup clock has been changed to JtagClk.

3. Click **OK**.

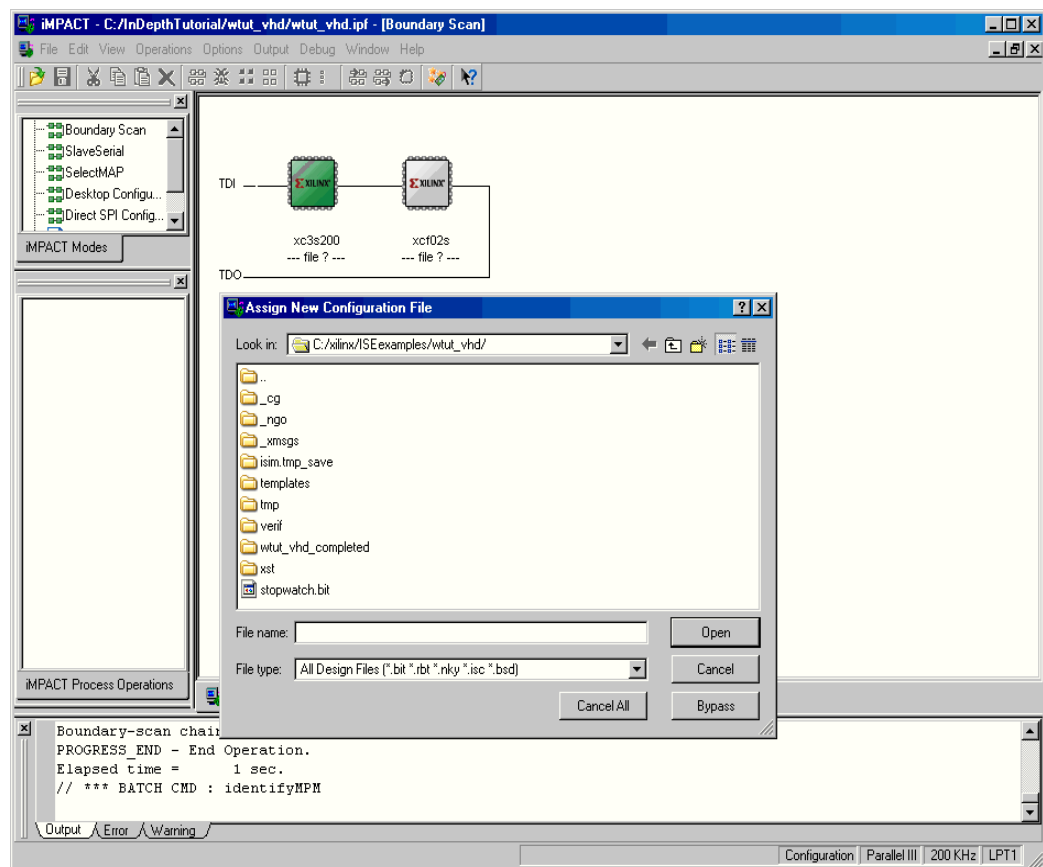


Figure 7-4: Selecting a Configuration File

Note: If a configuration file is not available, a Boundary Scan Description File (BSDL or BSD) file can be applied instead. The BSDL file provides the software with the necessary Boundary Scan information that allows a subset of the Boundary Scan Operations to be available for that device. To have ISE automatically select a BSDL file (for both Xilinx and non-Xilinx devices), select **Bypass** in the Assign New Configuration File dialog box.

4. When the software prompts you to select a configuration file for the second device (XCF02S):
5. Select the MCS file from your project working directory.
6. Click **Open**.

Saving the Project File

Once the chain has been fully described and configuration files are assigned, you should save your iMPACT Project File (IPF) for later use. To do this, select **File > Save Project As**. The Save As dialog box appears and you can browse and save your project file accordingly. To restore the chain after reopening iMPACT, select **File > Open Project** and browse to the IPF.

Note: Previous versions of ISE use Configuration Data Files (CDF). These files can still be opened and used in iMPACT. iMPACT Project Files can also be exported to a CDF.

Editing Preferences

To edit the preferences for the Boundary Scan Configuration, select **Edit > Preferences**. This selection opens the window shown in [Figure 7-5](#). Click **Help** for a description of the Preferences.

In this tutorial, keep the default values and click **OK**.

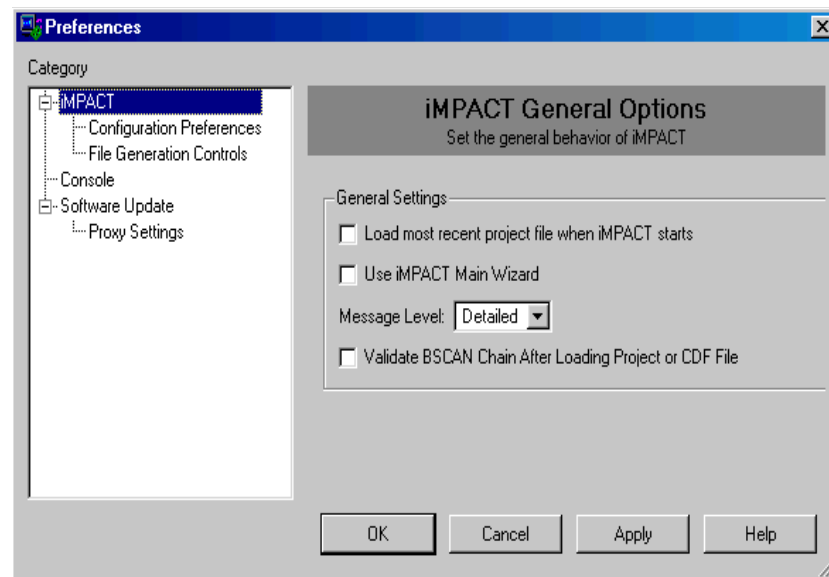


Figure 7-5: Edit Preferences

Performing Boundary Scan Operations

You can perform Boundary Scan operations on one device at a time. The available Boundary Scan operations vary based on the device and the configuration file that was applied to the device. To see a list of the available options, right-click on any device in the chain. This brings up a window with all of the available options.

When you select a device and perform an operation on that device, all other devices in the chain are automatically placed in BYPASS or HIGHZ, depending on your iMPACT Preferences setting. (For more information about Preferences, see “Editing Preferences.”)

To perform an operation, right-click on a device and select one of the options. In this section, you will retrieve the device ID and run the programming option to verify the first device.

1. Right-click on the XC3S200 device.
2. Select **Get Device ID** from the right-click menu.

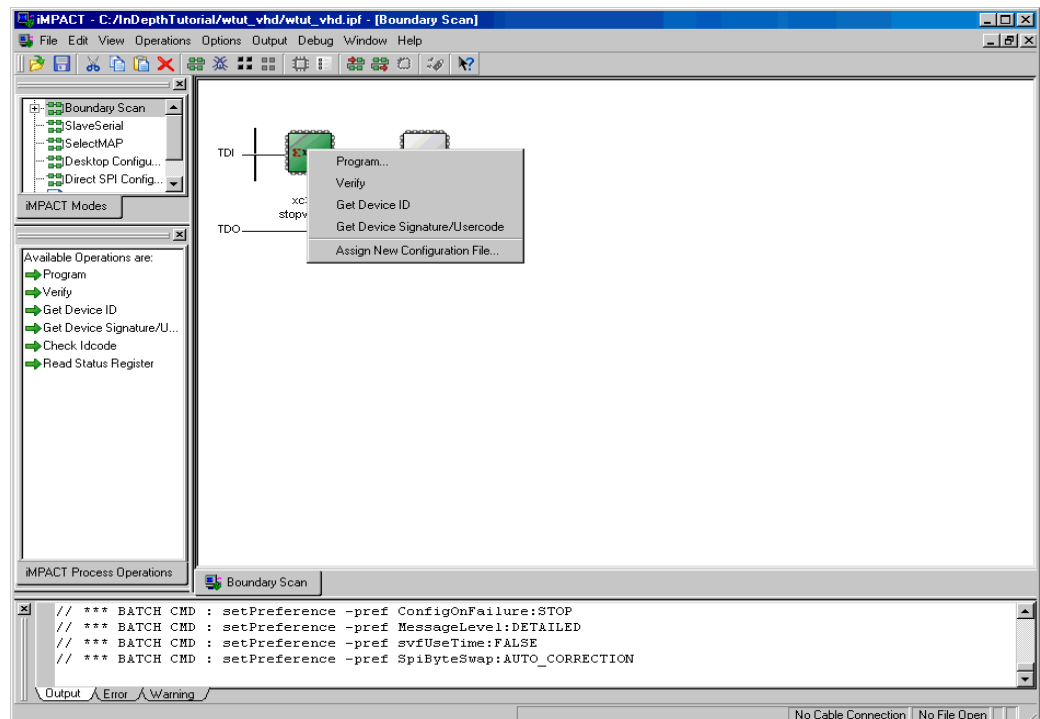


Figure 7-6: Available Boundary Scan Operations for an XC3S200 Device

The software accesses the IDCODE for this Spartan-3 device. The result is displayed in the log window (see Figure 7-7).

```
// *** BATCH CMD : ReadIdcode -p 1
Chain TCK freq = 0.
Validating chain...
Boundary-scan chain validated successfully.
'1': IDCODE is '000000010100000010100000010010011'
'1': IDCODE is '01414093' (in hex).
'1': : Manufacturer's ID =Xilinx xc3s200, Version : 0
```

Figure 7-7: Log Window Showing Result of Get Device ID

3. Right-click on the XC3S200 device
 4. Select **Program** from the right-click menu.
- The Program Options Dialog Box appears (see Figure 7-8).

5. Select the **Verify** option.
The Verify option enables the device to be readback and compared to the BIT file using the MSK file that was created earlier.
6. Click **OK** to begin programming.

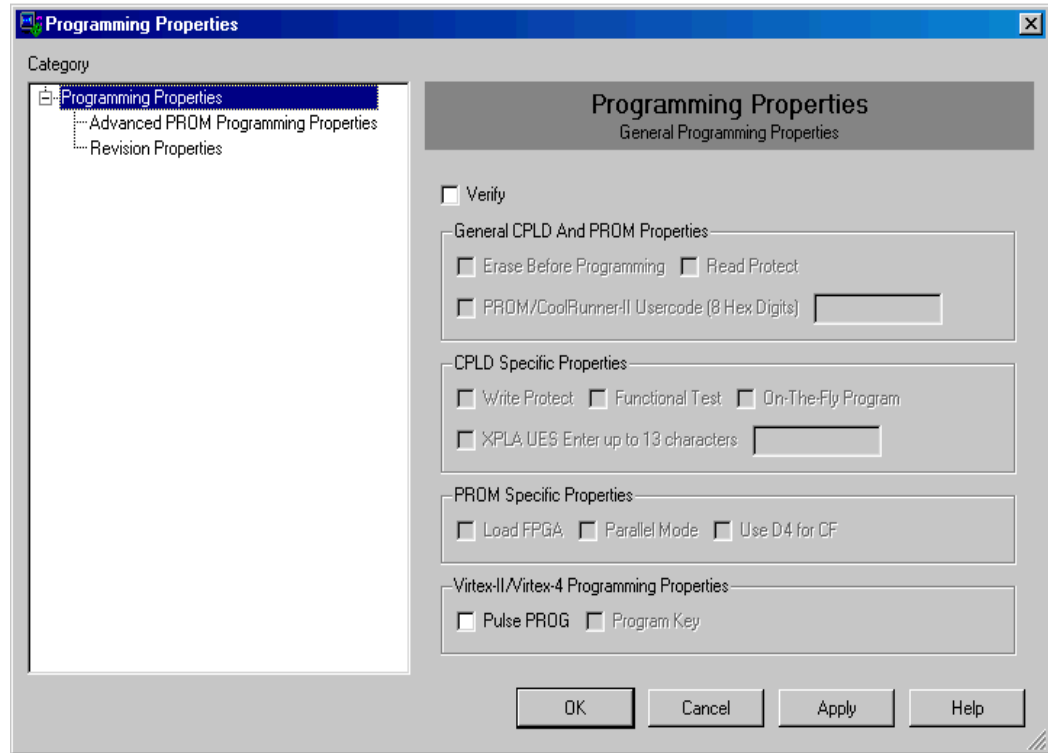


Figure 7-8: Program Options for XC3S200 Device

Note: The options available in the Program Options dialog box vary based on the device you have selected.

After clicking **OK**, the Program operation begins and an operation status window displays. At the same time, the log window reports all of the operations being performed.

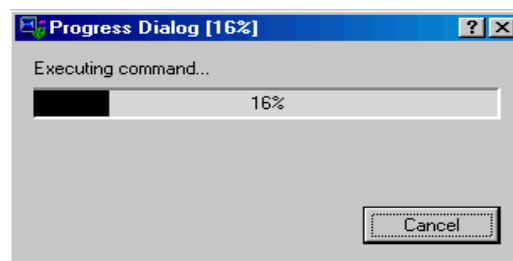


Figure 7-9: Operation Status

When the Program operation completes, a large blue message appears showing that programming was successful (see Figure 7-10). This message disappears after a couple of seconds.

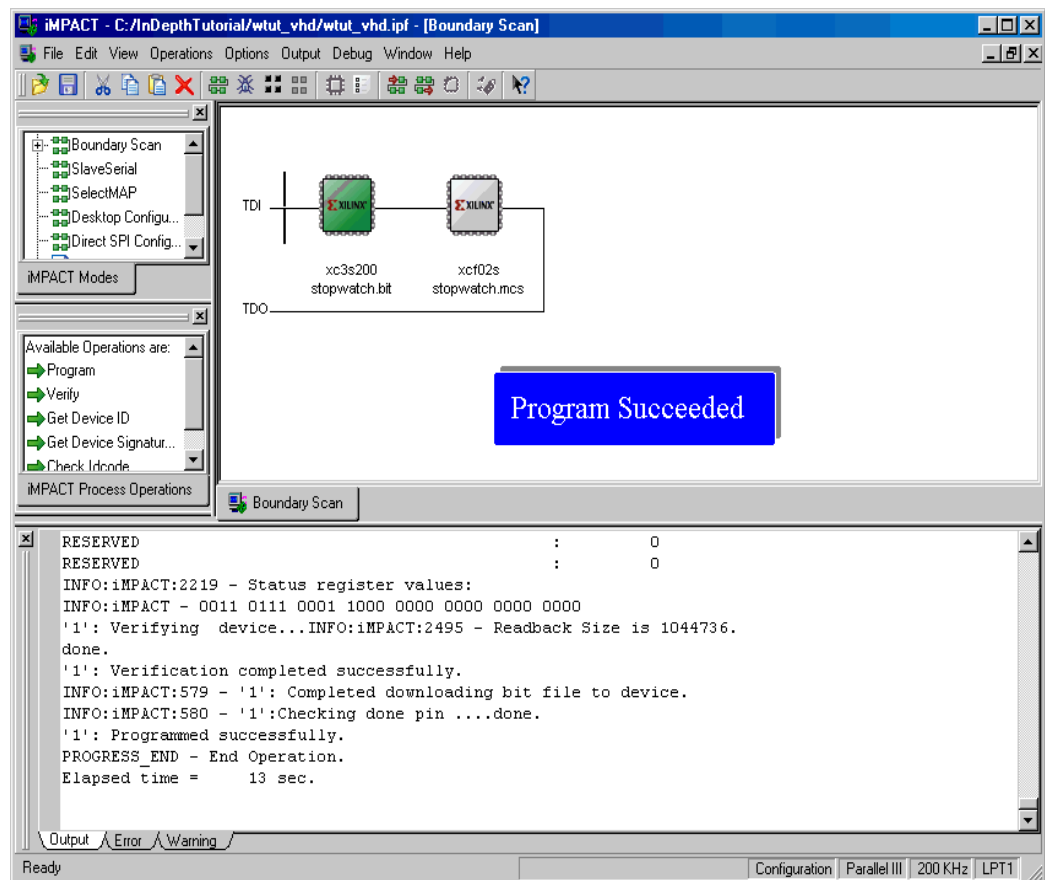


Figure 7-10: Programming Operation Completes

Your design has been programmed and has been verified. The board should now be working and should allow you to start, stop and reset the runner's stopwatch.

Troubleshooting Boundary Scan Configuration

Verifying Cable Connection

When an error occurs during a Boundary Scan operation, first verify that the cable connection is established and that the software autodetect function is working. If a connection is still not established after plugging the cable into the board and into your machine, right-click in a blank portion of the iMPACT window and select either **Cable Auto Connect** or **Cable Setup**. Cable Auto Connect will force the software to search every port for a connection. Cable Setup enables you to select the cable and the port to which the cable is connected.

When a connection is found, the bottom of the iMPACT window will display the type of cable connected, the port attached to the cable, and the cable speed (see [Figure 7-11](#)).



Figure 7-11: Cable Connection Successful

If a cable is connected to the system and the cable autodetection fails, refer to Xilinx Answer Record #15742. Go to <http://www.xilinx.com/support> and search for “15742”.

Verifying Chain Setup

When an error occurs during a Boundary Scan operation, verify that the chain is set up correctly and verify that the software can communicate with the devices. The easiest way to do this is to initialize the chain. To do so, right-click in the iMPACT window and select **Initialize Chain**. The software will identify the chain if the connections to the board are working.

If the chain cannot be initialized, it is likely that the hardware is not set up correctly or the cable is not properly connected. If the chain can be initialized, try performing simple operations. For instance, try getting the Device ID of every device in the chain. If this can be done, then the hardware is set up correctly and the cable is properly connected.

The debug chain can also be used to manually enter JTAG commands (see [Figure 7-12](#)). This can be used for testing commands and verifying that the chain is set up correctly. To use this feature, select **Debug > Start/Stop Debug Chain** in iMPACT.

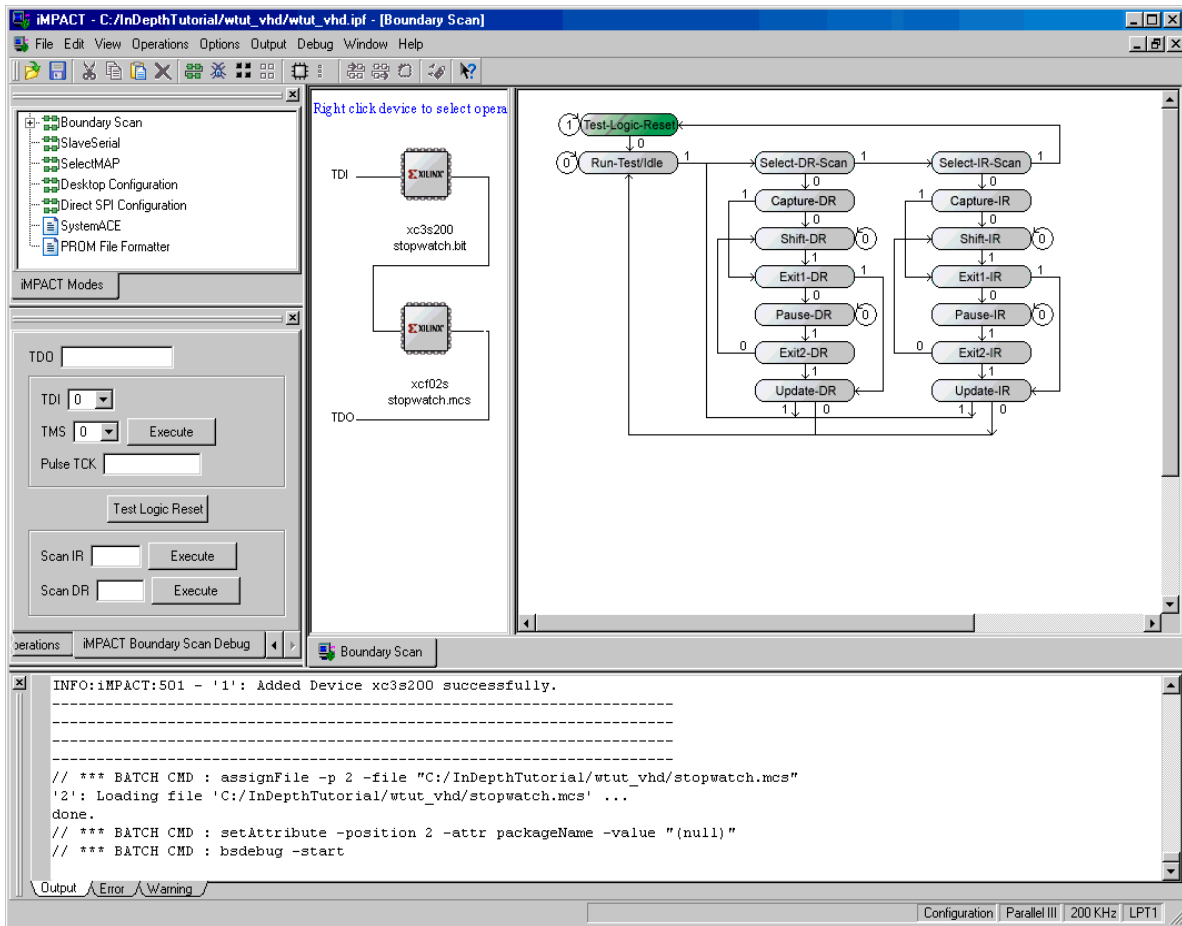


Figure 7-12: Debug Chain

For help using iMPACT Boundary-Scan Debug, use the iMPACT Help (accessible from **Help > Help Topics**), or file a Web case at <http://www.xilinx.com/support>.

Creating an SVF File

This section is optional and assumes that you have followed the “Using Boundary Scan Configuration Mode” section and have successfully programmed to a board. In this section, all of the configuration information is written to the SVF file.

iMPACT supports the creation of device programming files in three formats, SVF, XSVE, and STAPL. If you are using third-party programming solutions, you may need to set up your Boundary Scan chain manually and then create a device programming file. These programming files contain both programming instructions and configuration data, and they are used by ATE machines and embedded controllers to perform Boundary Scan operations. A cable normally does not need to be connected because no operations are being performed on devices.

Setting up Boundary Scan Chain

This section assumes that you are continuing from the previous sections of this chapter and already have the chain detected. If not, skip to “[Manual JTAG chain setup for SVF generation](#)” to define the chain manually.

JTAG chain setup for SVF generation

1. Select **Output > SVF File > Create SVF File** to indicate that you are creating a programming file.
2. Enter **getid** in the File Name field of the Create a New SVF File dialog box and click **Save**.
3. An informational message appears stating that all device operations will be directed to the .svf file. Click **OK**.

Manual JTAG chain setup for SVF generation

For this tutorial, you may skip this section if you completed the “[Using Boundary Scan Configuration Mode](#).” section.

The Boundary-Scan chain can be manually created or modified as well. To do this,

1. Ensure that you are in Boundary Scan Mode (click the **Boundary-Scan tab**).
You can now add one device at a time.
2. Right-click on an empty space in the iMPACT Boundary-Scan window and select **Add Xilinx Device** or **Add Non-Xilinx device**.
An Add Device dialog box appears allowing you to select a configuration file.
3. Select `stopwatch.bit` and then click **Open**.

The device is added where the large cursor is positioned. To add a device between existing devices, click on the line between them and then add the new device.

Repeat steps 2 and 3 to add the `stopwatch.mcs` file to the chain.

Note: The boundary scan chain that you manually create in the software must match the chain on the board, even if you intend to program only some of the devices. All devices must be represented in the iMPACT window.

Writing to the SVF File

The process of writing to an SVF file is identical to performing Boundary Scan operations with a cable. You simply right-click on a device and select an operation. Any number of operations can be written to an SVF file.

In this section, you will be writing the device ID to the programming file for the first device, and performing further instructions for the second device.

To write the device ID:

1. Right-click the first device (XC3S200).
2. Select **Get Device ID** from the right-click menu.

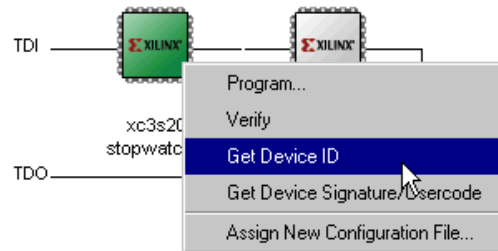


Figure 7-13: Selecting a Boundary Scan Operation

The instructions that are necessary to perform a Get Device ID operation are then written to the file.

3. To see the results, select **View > View SVF-STAPL File**. Figure 7-14 shows what the SVF file looks like after the Get Device ID operation is performed.

```
// Created using xilinx iMPACT Software [ISE Foundation sim - 7.1i]
TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET IDLE;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 8 TDI (ff) SMASK (ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 6 TDI (09) SMASK (3f) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f1414093) MASK (0xffffffff) ;
// Validating chain...
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 28 TDI (0fffeaaa) SMASK (0xffffffff) TDO (0aaa8101) MASK (0fffc307) ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 8 TDI (ff) SMASK (ff) ;
HDR 1 TDI (00) SMASK (01) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 6 TDI (09) SMASK (3f) ;
SDR 32 TDI (00000000) TDO (f1414093) ;
//Loading device with 'idcode' instruction.
SIR 6 TDI (09) ;
SDR 32 TDI (00000000) TDO (f1414093) ;
TIR 0 ;
HIR 8 TDI (ff) ;
HDR 1 TDI (00) ;
TDR 0 ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 14 TDI (3fff) SMASK (3fff) ;
SDR 2 TDI (00) SMASK (03) ;
```

Figure 7-14: SVF File that Gets a Device ID from the First Device in the Chain

To write further instructions to the SVF for the second device:

1. Right-click the second device (XCF02S).
2. Select **Program** from the right-click menu.

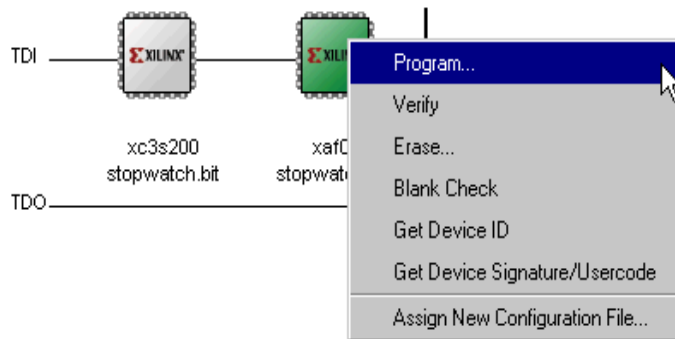


Figure 7-15: Available Boundary Scan Operations for a XCF00S Device

3. Click **OK** in the Programming Properties window.

The instructions and configuration data needed to program the second device are added to the SVF file.

Stop Writing to the SVF

After all the desired operations have been performed, you must add an instruction to close the file from further instructions. To stop writing to the programming file:

Select **Output > SVF File > Stop Writing to SVF File**.

To add other operations in the future, you can select **Output > SVF File > Append to SVF File**, select the SVF file and click **Save**.

Playing back the SVF or XSVF file

To play back the SVF file that you created to verify the instructions, you will

- Manually create a new chain.
- Assign the SVF file to the chain by right clicking and selecting **Add Xilinx Device** and selecting the SVF file in the search window..
- Right-click on the SVF file in the Boundary-Scan chain and select **Execute XSVF/SVF**.

Other Configuration Modes

Slave Serial Configuration Mode

Slave Serial Configuration mode allows you to program a single Xilinx device or a serial chain of Xilinx devices. To use the Slave Serial Configuration Mode, double-click Slave Serial in the iMPACT Modes window.

SelectMAP Configuration Mode

With iMPACT, SelectMAP Configuration mode allows you to program up to three Xilinx devices. The devices are programmed one at a time and are selected by the assertion of the correct CS pin. To use the SelectMAP Configuration Mode, double click SelectMAP in the iMPACT Modes window. Only the MultiPRO cable can be used for SelectMAP Configuration.

Note: These modes cannot be used with the Spartan-3 Starter Kit.