ISE 7 In-Depth Tutorial

XILINX®

About This Tutorial

About the In-Depth Tutorial

This tutorial gives a description of the features and additions to Xilinx® ISE™ 7. The primary focus of this tutorial is to show the relationship among the design entry tools, Xilinx and third-party tools, and the design implementation tools.

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

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

Additional Resources

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

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://www.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://www.xilinx.com/xlnx/xil_ans_browser.jsp
Application Notes	Descriptions of device-specific design techniques and approaches http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp?category=Application+Notes
Forums	Discussion groups and chat rooms for Xilinx software users http://toolbox.xilinx.com/cgi-bin/forum
Data Sheets	Data Sheet, which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

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

Tutorial Contents

This guide covers the following topics.

- **Chapter 1**, "Overview of ISE and Synthesis Tools," introduces you to the ISE primary user interface, Project Navigator, and the synthesis tools available for your design.
- **Chapter 2**, "HDL-Based Design," guides you through a typical HDL-based design procedure using a design of a runner's stopwatch.
- **Chapter 3**, "Schematic-Based Design," explains many different facets of a schematic-based ISE design flow using a design of a runner's stopwatch. This chapter also shows how to use ISE accessories such as StateCAD, Project Navigator, CORE Generator™, and ISE Text Editor.
- **Chapter 4**, "Behavioral Simulation," explains how to use the ModelSim Simulator to simulate a design before design implementation 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 four tutorial flows. In this section, the four 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
- Device Configuration-only Flow

HDL Design Flow

The HDL Design flow is as follows:

- **Chapter 2**, "HDL-Based Design"
- **Chapter 4**, "Behavioral Simulation"
  Note that behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 5**, "Design Implementation"
- **Chapter 6**, "Timing Simulation"
  Note that timing simulation is optional; however, it is strongly recommended.
- **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 behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 5**, "Design Implementation"
- **Chapter 6**,"Timing Simulation"
  Note that timing simulation is optional; however, it is strongly recommended.
- **Chapter 7**, "iMPACT Tutorial"

Implementation-only Flow

The Implementation-only flow is as follows:

- **Chapter 5**, "Design Implementation"
- **Chapter 7**, "iMPACT Tutorial"

Device Configuration-only Flow

For this flow, go to **Chapter 7**, "iMPACT Tutorial."

Table of Contents

Chapter 2: HDL-Based Design

Chapter 3: Schematic-Based Design

Chapter 4: Behavioral Simulation

Chapter 5: Design Implementation

Chapter 1

Overview of ISE and Synthesis Tools

This chapter includes the following sections:

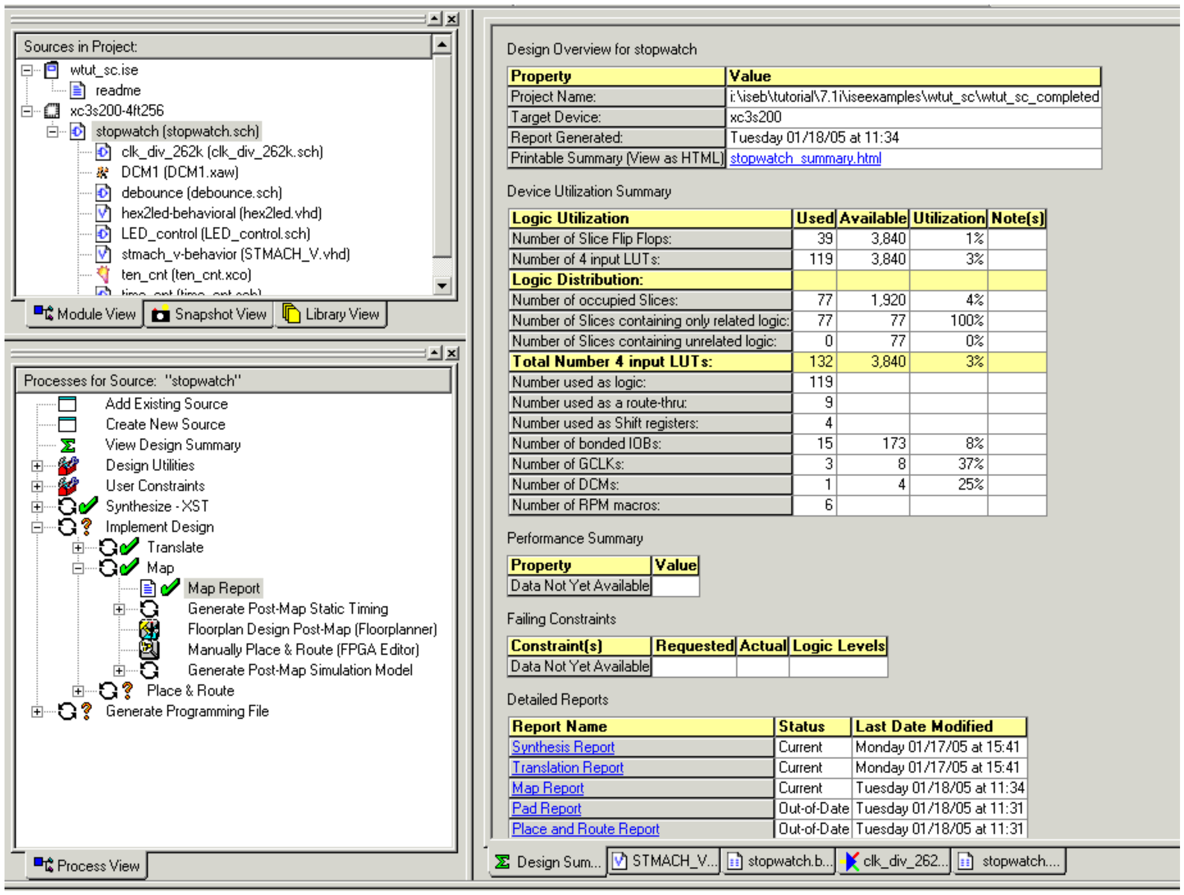
- "Overview of ISE"
- "Overview of Synthesis Tools"

Overview of ISE

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

Project Navigator Interface

The Project Navigator Interface is divided into four main subwindows, as seen in Figure 1-1. On the top left is the Sources in Project window which hierarchically displays the elements included in the project. Beneath the Sources in Project window is the Processes for Source window, which displays available processes. The third window at the bottom of the Project Navigator is the Transcript window which displays status messages, errors, and warnings and is updated during all project actions. 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 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.



Sources in Project:
- wtut_sc.ise
  - readme
- xc3s200-4ft256
  - stopwatch (stopwatch.sch)
    - clk_div_262k (clk_div_262k.sch)
    - DCM1 (DCM1.xaw)
    - debounce (debounce.sch)
    - hex2led-behavioral (hex2led.vhd)
    - LED_control (LED_control.sch)
    - stmach_v-behavior (STMACH_V.vhd)
    - ten_cnt (ten_cnt.xco)

Module View | Snapshot View | Library View

Processes for Source: "stopwatch"
- Add Existing Source
- Create New Source
- View Design Summary
- Design Utilities
- User Constraints
- Synthesize - XST
- Implement Design
  - Translate
  - Map
    - Map Report
    - Generate Post-Map Static Timing
    - Floorplan Design Post-Map (Floorplanner)
    - Manually Place & Route (FPGA Editor)
    - Generate Post-Map Simulation Model
  - Place & Route
- Generate Programming File

Process View

Design Overview for stopwatch

Property	Value
Project Name:	i:\iseb\tutorial\7.1i\iseexamples\wtut_sc\wtut_sc_completed
Target Device:	xc3s200
Report Generated:	Tuesday 01/18/05 at 11:34
Printable Summary (View as HTML)	stopwatch_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	39	3,840	1%	
Number of 4 input LUTs:	119	3,840	3%	
Logic Distribution:				
Number of occupied Slices:	77	1,920	4%	
Number of Slices containing only related logic:	77	77	100%	
Number of Slices containing unrelated logic:	0	77	0%	
Total Number 4 input LUTs:	132	3,840	3%	
Number used as logic:	119			
Number used as a route-thru:	9			
Number used as Shift registers:	4			
Number of bonded IOBs:	15	173	8%	
Number of GCLKs:	3	8	37%	
Number of DCMs:	1	4	25%	
Number of RPM macros:	6			

Performance Summary

Property	Value
Data Not Yet Available	

Failing Constraints

Constraint(s)	Requested	Actual	Logic Levels
Data Not Yet Available			

Detailed Reports

Report Name	Status	Last Date Modified
Synthesis Report	Current	Monday 01/17/05 at 15:41
Translation Report	Current	Monday 01/17/05 at 15:41
Map Report	Current	Tuesday 01/18/05 at 11:34
Pad Report	Out-of-Date	Tuesday 01/18/05 at 11:31
Place and Route Report	Out-of-Date	Tuesday 01/18/05 at 11:31

Design Sum... | STMACH_V... | stopwatch.b... | clk_div_262... | stopwatch....

Figure 1-1: **Project Navigator**

Sources in Project Window

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

Module View

The Module View tab displays the project name, any user documents, the specified part type and design flow / synthesis tool, and design source files. Each file in the Module View has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example). For a complete list of possible source types and their associated icons, see the 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.

Snapshot View

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

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

Library View

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

Processes for Source Window

This window contains one default tab called the Process View tab.

Process View

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

- **Add an Existing Source**

  Provides access to the Add Existing Sources dialog box, in which you can select a file in your project directory or in a remote directory to add to the project.

- **Create New Source**

  Provides access to the New Source wizard, in which you can create a new source file and add the new source file to your project.

- **View Design Summary**

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

- **Design Utilities**

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

- **User Constraints**

  Provides access to editing location and timing and constraints.

- **Synthesize**

  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 for Source window incorporates automake technology. This enables you 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 in the Console tab of the Transcript window, click the **+** next to Design Utilities to expand the process hierarchy, and double-click **View Command Line Log File**. See the Using Command Line section of Chapter 5, "Design Implementation" for further details.

Transcript Window

The Transcript window displays errors, warnings, and informational messages. Errors are signified by a red X next to the message, while warnings have a yellow exclamation mark (!). Warning and Error messages may also be viewed separate from other console text messages by selecting either the Warnings or Errors tab at the bottom of the console window.

Error Navigation to Source

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

Error Navigation to Solution Record

You can navigate from an error or warning message in the Transcript window to the relevant solution records on the www.xilinx.com/support website. To navigate to the solution record, select the error or warning message, right-click the mouse, and from the menu select **Goto Solution Record**. The default web browser opens and displays all solution records applicable to this message.

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 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 **Editor**. The default editor is the ISE Text Editor. ISE Text Editor enables you to edit source files. You can access the Language Templates, a catalog of ABEL, Verilog and VHDL language, and User Constraint File templates, and use these templates your own design.

ISE Simulator / Waveform Editor

ISE Simulator / Waveform Editor is a PC-based test bench and test fixture creation tool integrated in 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 Snapshots

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

Creating a Snapshot

To create a snapshot:

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

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

Restoring a Snapshot

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

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

Before the snapshot replaces the current project, the current project is stored in a snapshot so that your work is not lost.

Viewing a Snapshot

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

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

Using Project Archives

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

Creating an Archive

To create an archive:

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

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

Restoring an Archive

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

Overview of Synthesis Tools

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

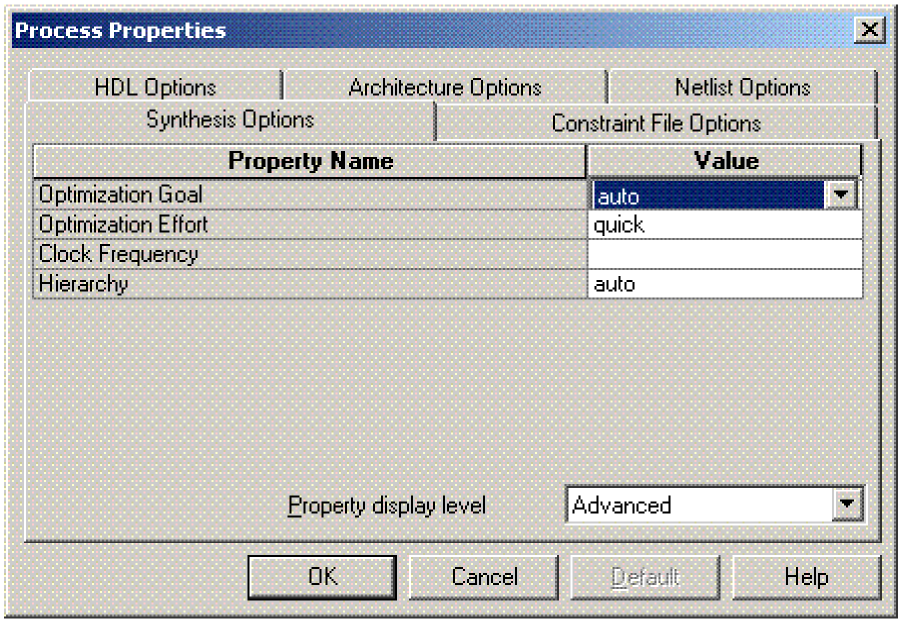
LeonardoSpectrum

This synthesis tool is not part of the ISE package and is not available unless purchased separately. Two commonly used properties are Optimization Goal and Optimization Effort. 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 LeonardoSpectrum. Most of the commonly used synthesis options available for the LeonardoSpectrum stand-alone version are available for LeonardoSpectrum synthesis through ISE.

For more information, see the LeonardoSpectrum online help.



Property Name	Value
Optimization Goal	auto
Optimization Effort	quick
Clock Frequency	
Hierarchy	auto

Property display level: Advanced

Figure 1-2: **LeonardoSpectrum Synthesis Process Properties**

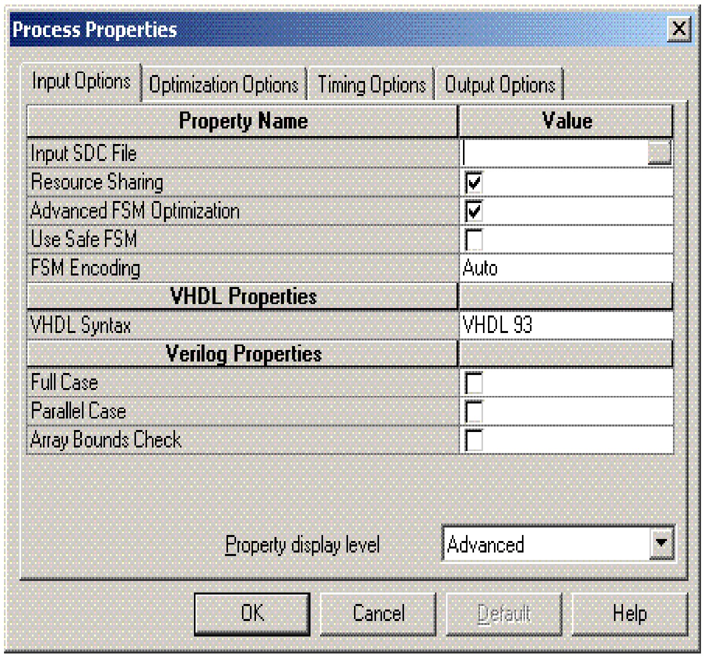
Precision Synthesis

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.


Process Properties

Input Options | Optimization Options | Timing Options | Output Options

Property Name	Value
Input SDC File	
Resource Sharing	✓
Advanced FSM Optimization	✓
Use Safe FSM	☐
FSM Encoding	Auto
VHDL Properties	
VHDL Syntax	VHDL 93
Verilog Properties	
Full Case	☐
Parallel Case	☐
Array Bounds Check	☐

Property display level: Advanced

Figure 1-3: **Precision Synthesis Process Properties**

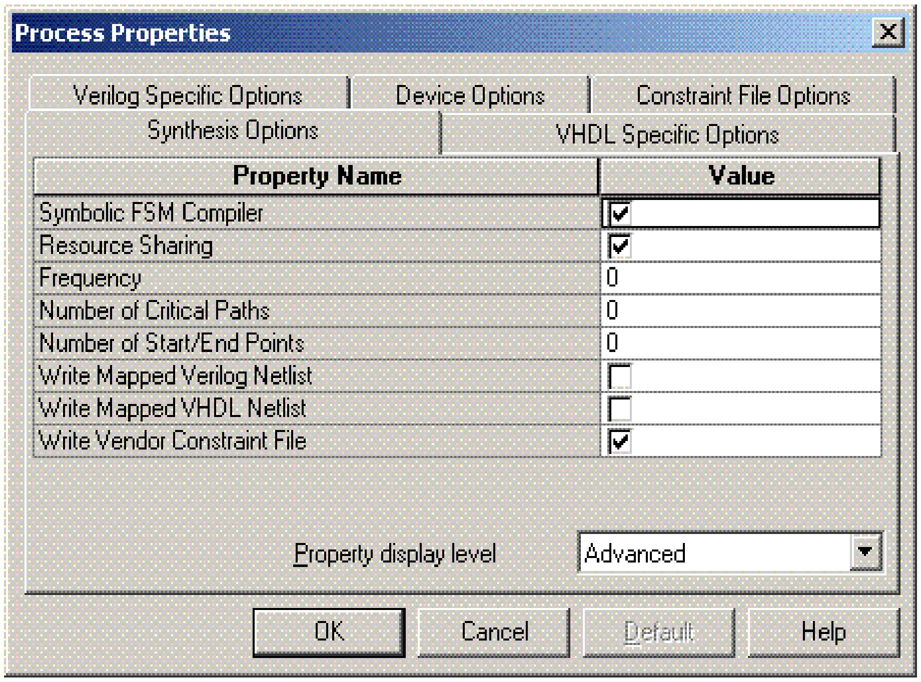
Synplify/Synplify Pro

This synthesis tool is not part of the ISE package and is not available unless purchased separately. This synthesis tool is 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.

More detailed information about the specific synthesis options is available in the Synplify/Synplify Pro online help.



Property Name	Value
Symbolic FSM Compiler	✓
Resource Sharing	✓
Frequency	0
Number of Critical Paths	0
Number of Start/End Points	0
Write Mapped Verilog Netlist	☐
Write Mapped VHDL Netlist	☐
Write Vendor Constraint File	✓

Property display level: Advanced

Figure 1-4: **Synplify/Synplify Pro Synthesis Process Properties**

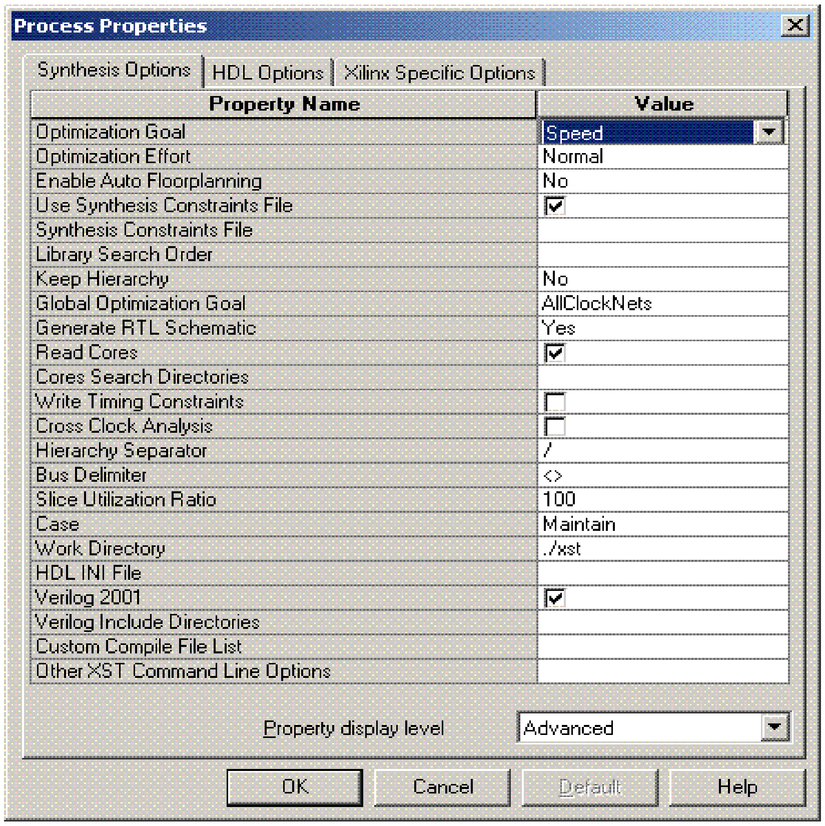
Xilinx Synthesis Technology (XST)

This synthesis tool is part of the ISE package and is available for both an HDL- or 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. Through these properties you can control the synthesis results for area or speed, and the amount of time the synthesizer runs.

More detailed information is available in the *XST User Guide*, available in the collection of software manuals available with ISE by selecting **Help** → **Online Documentation**, or on the web at http://www.xilinx.com/support/sw_manuals/xilinx7/.



Property Name	Value
Optimization Goal	Speed
Optimization Effort	Normal
Enable Auto Floorplanning	No
Use Synthesis Constraints File	✓
Synthesis Constraints File	
Library Search Order	
Keep Hierarchy	No
Global Optimization Goal	AllClockNets
Generate RTL Schematic	Yes
Read Cores	✓
Cores Search Directories	
Write Timing Constraints	☐
Cross Clock Analysis	☐
Hierarchy Separator	/
Bus Delimiter	<>
Slice Utilization Ratio	100
Case	Maintain
Work Directory	./xst
HDL INI File	
Verilog 2001	✓
Verilog Include Directories	
Custom Compile File List	
Other XST Command Line Options	

Property display level: Advanced

Figure 1-5: **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™-3 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, LeonardoSpectrum or Precision.

This chapter is the first 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-3 demo board (Chapter 7, "iMPACT Tutorial.")

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

Getting Started

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

Required Software

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

- Xilinx Series ISE 7.x
- Spartan-3 libraries and device files

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

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

Optional Software Requirements

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

- Synplify/Synplify PRO 8.0 (or above)
- LeonardoSpectrum 2004.b (or above)
- Precision Synthesis 2004.c (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 accordingly. Starting with the 6.1i release, XST can now synthesize a mixed-language design. However, this tutorial does not go over the mixed language feature.

Installing the Tutorial Project Files

The Stopwatch tutorial projects can be downloaded from http://www.xilinx.com/support/techsup/tutorials/tutorials7.htm. Download either the VHDL or the Verilog design flow project files.

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

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

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

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

Directory	Description
wtut_vhd	Incomplete Watch Tutorial - VHDL
wtut_ver	Incomplete Watch Tutorial - Verilog
wtut_vhd\wtut_vhd_completed	Solution for Watch - VHDL
wtut_ver\wtut_ver_completed	Solution for Watch - Verilog

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

The solution projects contain the design files for the completed tutorials, including HDL files. To conserve disk space, some intermediate files are not provided.

Unzip the tutorial projects in any directory with read-write permissions. The HDL tutorial files are copied into the directories when you unzip the project 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** → **Programs** → **Xilinx ISE** → **Project Navigator**.

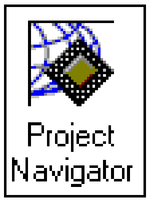


Figure 2-1: **Project Navigator Desktop Icon**

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

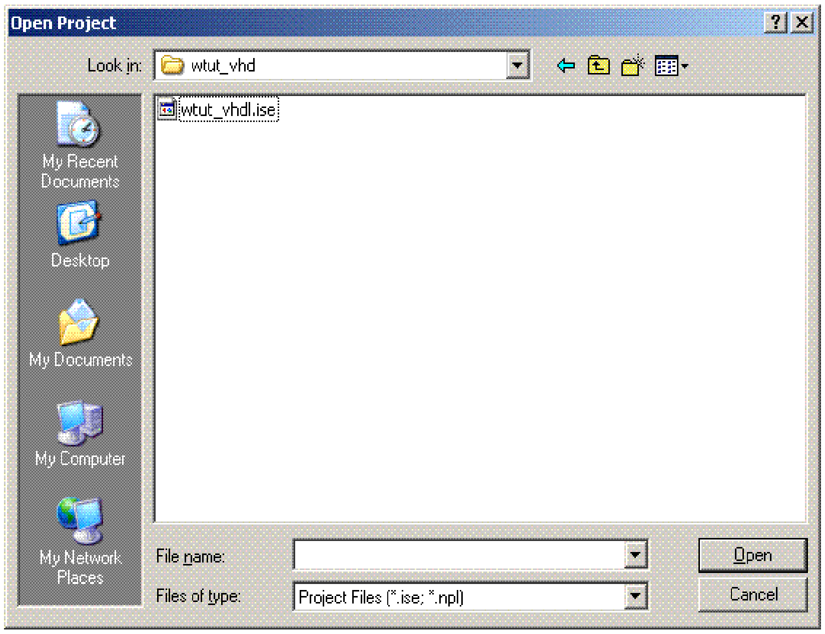


Figure 2-2: **Open Project Dialog Box**

3. Browse to *c:\xilinx\ISEexamples\wtut_vhd* or *c:\xilinx\ISEexamples\wtut_ver.*
4. Double-click *wtut_vhd.ise* (VHDL design entry) or *wtut_ver.ise* (Verilog design entry).

Stopping the Tutorial

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

Design Description

The design used in this tutorial is a hierarchical, HDL-based design, which means that the top-level design file is an HDL file that references several other lower-level macros. The lower-level macros are either HDL modules or 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, simulate it to verify the design's functionality.

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

Inputs

The following are input signals for the tutorial stopwatch design.

- **STRTSTOP**

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

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

  Externally generated system clock.

Outputs

The following are outputs signals for the design.

- **SEG_A, SEG_B, SEG_C, SEG_D, SEG_F, SEG_G, SEG_DP**

  These outputs drive the individual segments and the decimal point for all four digits of the stopwatch design. The digits of the stopwatch are displayed on 7-segment LED displays.
- **AN[3:0]**

  This is a one-hot vector signal which drives the anodes of the four 7-segment LED displays to determine which display will be lighted.

Functional Blocks

The completed design consists of the following functional blocks.

- **CLK_DIV_262k**

  Macro which divides a clock frequency by 262,144.
- **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-3 demo board to 26.2144 MHz.
- **DEBOUNCE**

  Schematic module implementing a simplistic debounce circuit for the STRTSTOP input signal.
- **HEX2LED**

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

  Module controlling the data multiplexing to the four 7-segment LED displays.
- **STATMACH**

  State Machine module defined and implemented in StateCAD.

- **TEN_CNT**

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

- **TIME_CNT**

  Module which counts from 0:0 to 9:59 decimal. This macro has three 4-bit outputs, which represent the minutes and seconds digits of the decimal point.

Design Entry

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

With the *wtut_vhd.ise* or *wtut_ver.ise* project open in Project Navigator, the Sources in Project window displays all of the source files currently added to the project, with the associated entity or module names (see Figure 2-3). In the current project, *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.

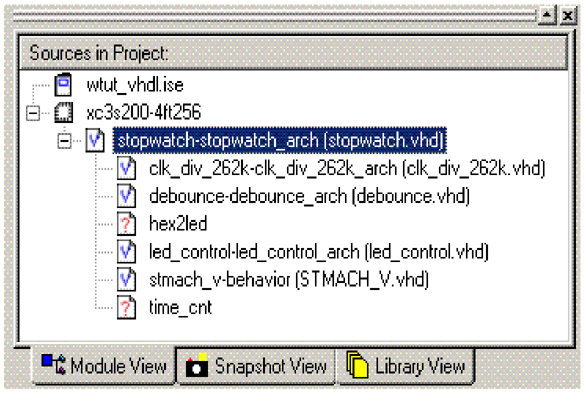

Sources in Project:
- wtut_vhdl.ise
- xc3s200-4ft256
  - stopwatch-stopwatch_arch (stopwatch.vhd)
    - clk_div_262k-clk_div_262k_arch (clk_div_262k.vhd)
    - debounce-debounce_arch (debounce.vhd)
    - hex2led
    - led_control-led_control_arch (led_control.vhd)
    - stmach_v-behavior (STMACH_V.vhd)
    - time_cnt

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

Adding Source Files

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

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

   Upon selecting the HDL file, the process window displays all processes available for this file.

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

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

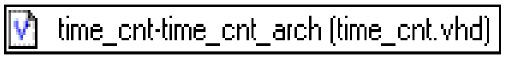

time_cnt-time_cnt_arch (time_cnt.vhd)

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

Checking the Syntax

To check the syntax of source files:

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

   Upon selecting the HDL file, the Processes for Source window displays all processes available for this file.
2. 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. Project Navigator reports errors in red and warnings in yellow in the Transcript window.

To display the error in the source file:

1. Double-click the error message in the Transcript window. The source code will come up in the main display window.
2. Correct any errors in the HDL source file. The comments next to the error explain this simple fix.
3. Select **File** → **Save** to save the file.
4. Re-analyze the file by selecting the HDL file and double-clicking **Check Syntax**.

Creating an HDL-Based Module

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

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

Using the New Source Wizard and ISE Text Editor

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:

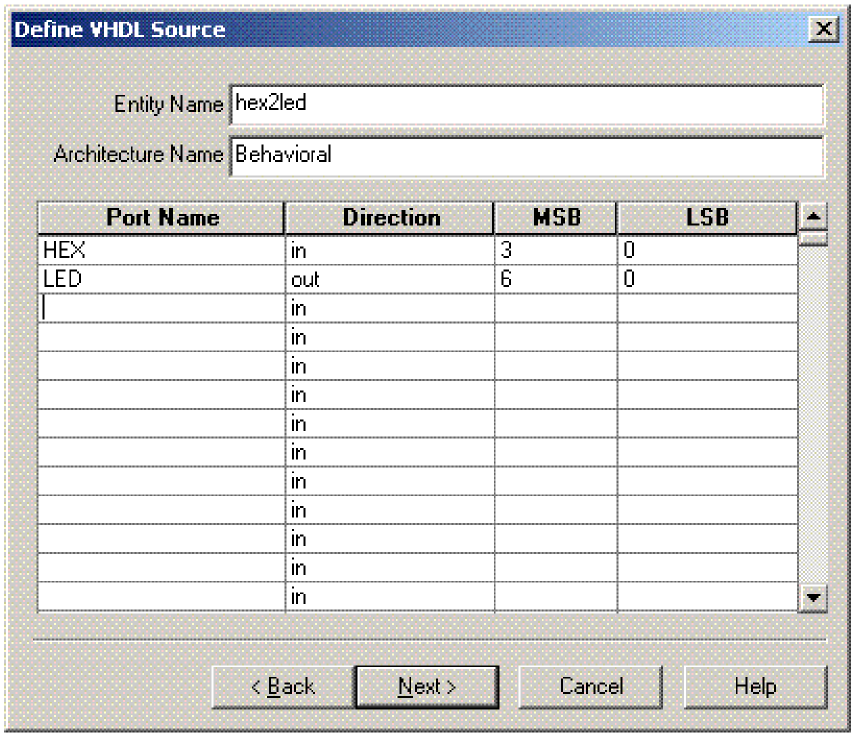
1. Select **Project** → **New Source**.

   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 **hex2led**.
4. Click **Next**.

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

1. Click in the Port Name field and type **HEX**.
2. Click in the Direction field and set the direction to **in**.
3. In the MSB field enter **3**, and in the LSB field enter **0**. Refer to Figure 2-5.



Entity Name: hex2led

Architecture Name: Behavioral

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

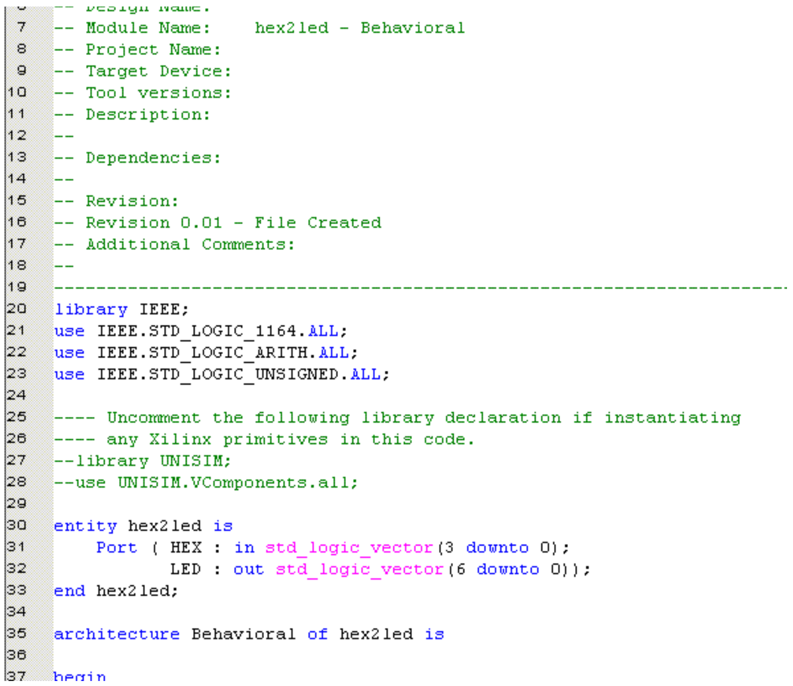
Figure 2-5: **New Source Wizard for VHDL**

4. Repeat the previous steps for the **LED[6:0]** output bus. Be sure that the direction is set to **out**, MSB is set to **6** and LSB is set to **0**.
5. Click **Next** to complete the Wizard session.

   A description of the module displays.

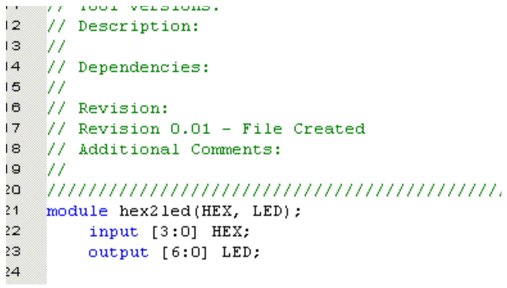
6. Click **Finish** to open the empty HDL file in ISE Text Editor.

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



```
-- Design Name:
-- Module Name:    hex2led - Behavioral
-- Project Name:
-- Target Device:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of hex2led is

begin
```

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



```
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////
module hex2led(HEX, LED);
    input [3:0] HEX;
    output [6:0] LED;
```

Figure 2-7: **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 recognition of typographical errors.

Using the Language Templates

The ISE Language Templates include HDL constructs and synthesis templates which represent commonly used logic components, such as counters, D flip-flops, multiplexers, and primitives. You will use the HEX2LED Converter template for this exercise. This template provides source code to convert a 4-bit value to 7-segment LED display format.

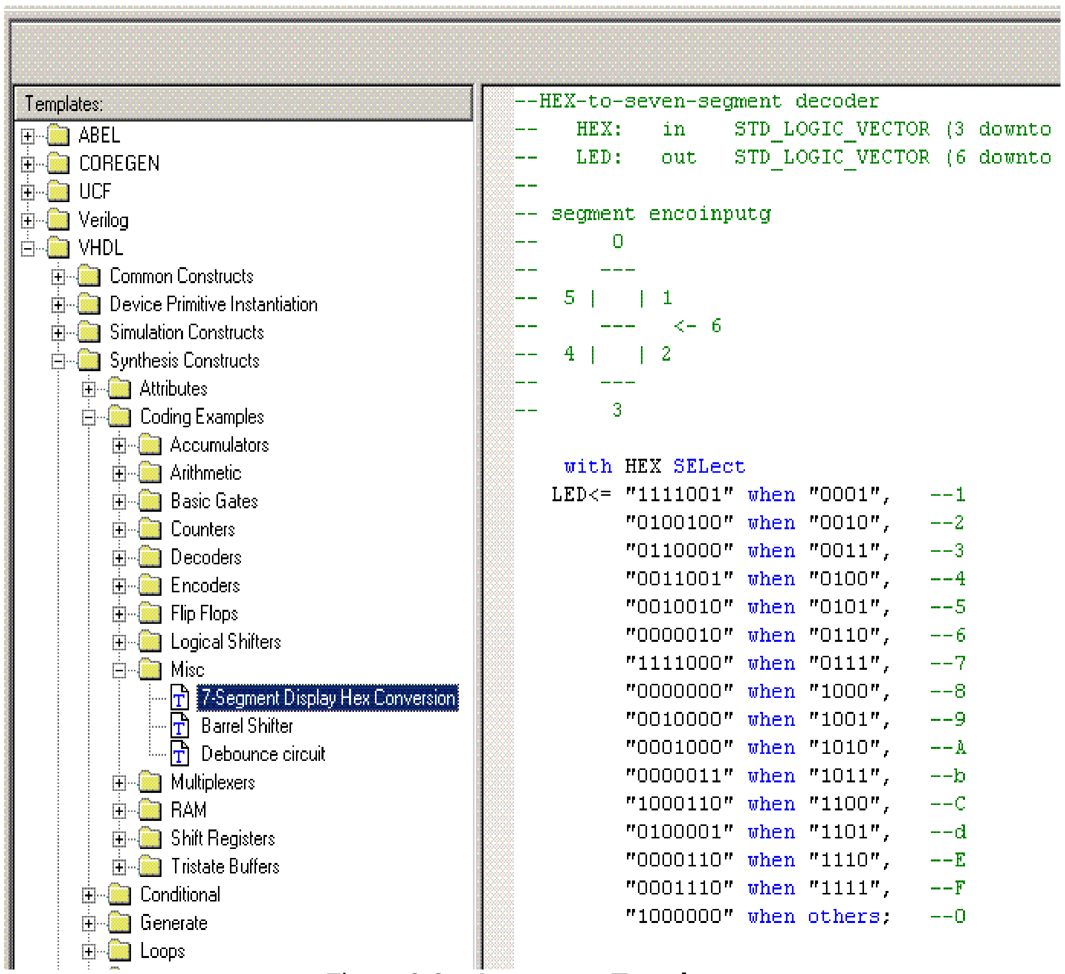
Note: You can add your own templates to the Language Templates for components or constructs that you use often.

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 topic. 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 **7-Segment Display Hex Conversion**. Use the appropriate template for the language you are using.
3. To preview the template, click the template in the hierarchy. The contents display in the right pane.



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

    with HEX SELect
   LED<= "1111001" when "0001",    --1
         "0100100" when "0010",    --2
         "0110000" when "0011",    --3
         "0011001" when "0100",    --4
         "0010010" when "0101",    --5
         "0000010" when "0110",    --6
         "1111000" when "0111",    --7
         "0000000" when "1000",    --8
         "0010000" when "1001",    --9
         "0001000" when "1010",    --A
         "0000011" when "1011",    --b
         "1000110" when "1100",    --C
         "0100001" when "1101",    --d
         "0000110" when "1110",    --E
         "0001110" when "1111",    --F
         "1000000" when others;    --0
```

Figure 2-8: **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. A copy and paste function is also available from the Language Templates Edit Menu and the right-click menu.

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

1. In the Language Templates, click and drag the 7-Segment Display Hex Conversion name into the *hex2led.vhd* file under the architecture statement, or into the *hex2led.v* file under the module declaration.
2. Close the Language Templates window.
3. (Verilog only) After the input and output statements and before the HEX2LED converter that you just added, add the following line of code to the HDL file to allow an assignment:

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

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

Creating a CORE Generator Module

CORE Generator is a graphical interactive design tool used to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and pre-optimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic for arithmetic functions and on-chip RAM for dual-port and synchronous RAM.

In this section, you will create a CORE Generator module called ten_cnt which is a 4-bit binary encoded counter. The 4-bit number is decoded to count the tenths digit of the stopwatch's time value.

Creating the CORE Generator Module

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

To create the module:

1. In Project Navigator, select **Project** → **New Source**.
2. Select **IP (CoreGen & Architecture Wizard)** as the source type.
3. Enter **ten_cnt** in the File Name field.
4. Click **Basic Elements** → **Counters** → **Binary Counters**.
5. Click **Next** and then **Finish**.

   The Binary Counter dialog box opens.

6. Fill in the Binary Counter dialog box with the following settings:
   - Component Name: **ten_cnt**

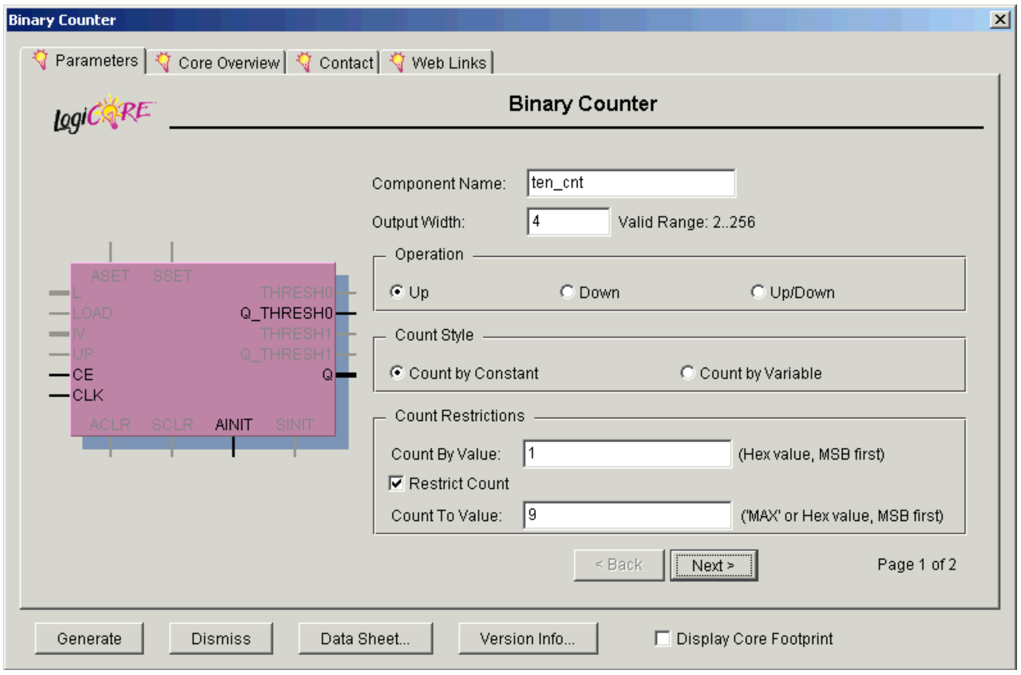
     Defines the name of the module.
   - Output Width: **4**

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

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

     Allows counting by a constant or a user supplied variable.
   - Count Restrictions: **Restrict Count**
   - Count Restrictions: Count To Value **9**

     This dictates the maximum count value.



Component Name: ten_cnt
Output Width: 4 Valid Range: 2..256
Operation: [✓ Up] Down Up/Down
Count Style: [✓ Count by Constant] Count by Variable
Count Restrictions
Count By Value: 1 (Hex value, MSB first)
[✓ Restrict Count]
Count To Value: 9 ('MAX' or Hex value, MSB first)
Page 1 of 2
☐ Display Core Footprint

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

7. Select the **Next** button.
8. Continue to fill in the Binary Counter dialog with the following settings:
   - Threshold Options: **Threshold 0 set to 9**

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

10. In the Register Options dialog box, enter the following settings:
    - Clock Enable: **Selected**
    - Asynchronous Settings: **Init with a value of 0**
    - Synchronous Settings: **None**
11. Click **OK**.
12. Check that *only* the following pins are used (used pins will be highlighted on the model symbol to the left side of the CORE Generator window):
    - **AINIT**
    - **CE**
    - **Q**
    - **Q_THRESH0**
    - **CLK**
13. Click **Generate**.

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

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

- *ten_cnt.sym*

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

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

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

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

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

  This file stores the CORE Generator configuration for the project.

Instantiating the CORE Generator Module in the HDL Code

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

VHDL Flow

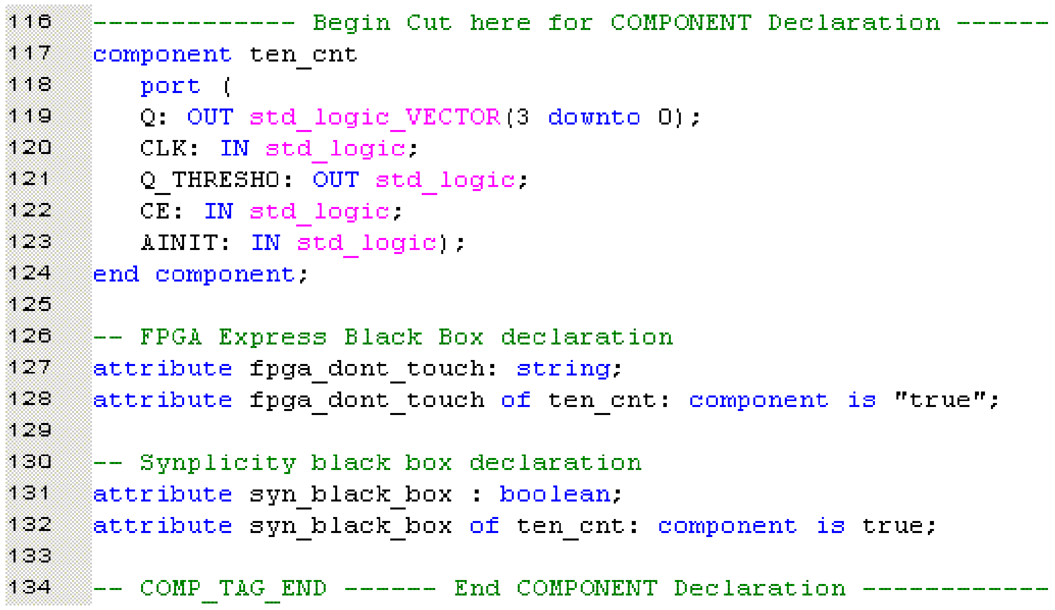
To instantiate the CORE Generator module using a VHDL flow:

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

```
“-- Insert Coregen Counter Component Declaration”
```

3. Select **Edit** → **Insert File** and select *ten_cnt.vho*.

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



```
-------------- Begin Cut here for COMPONENT Declaration -------
component ten_cnt
   port (
   Q: OUT std_logic_VECTOR(3 downto 0);
   CLK: IN std_logic;
   Q_THRESH0: OUT std_logic;
   CE: IN std_logic;
   AINIT: IN std_logic);
end component;

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

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

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

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

4. Highlight the inserted code from

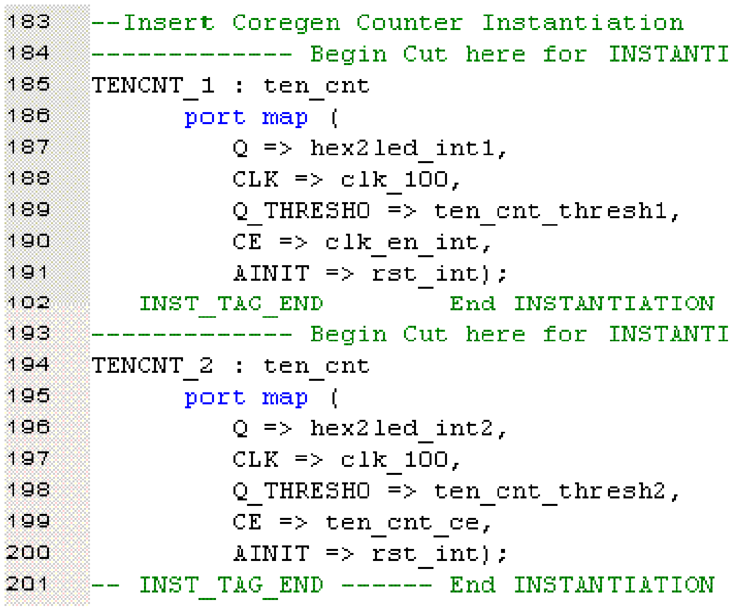
   “`-- Begin Cut here for INSTANTIATION Template`”

   to

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

   “`--Insert Coregen Counter Instantiation`”
7. Select **Edit** → **Paste** two times because two CORE Generator counters are needed.
8. Change the instance names from “`your_instance_name`” to “`TENCNT_1`” and “`TENCNT_2`”.

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



```
--Insert Coregen Counter Instantiation
------------- Begin Cut here for INSTANTI
TENCNT_1 : ten_cnt
      port map (
         Q => hex2led_int1,
         CLK => clk_100,
         Q_THRESH0 => ten_cnt_thresh1,
         CE => clk_en_int,
         AINIT => rst_int);
   INST_TAG_END          End INSTANTIATION
------------- Begin Cut here for INSTANTI
TENCNT_2 : ten_cnt
      port map (
         Q => hex2led_int2,
         CLK => clk_100,
         Q_THRESH0 => ten_cnt_thresh2,
         CE => ten_cnt_ce,
         AINIT => rst_int);
-- INST_TAG_END ------ End INSTANTIATION
```

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

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

Verilog Flow

To instantiate the CORE Generator module using a Verilog flow:

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

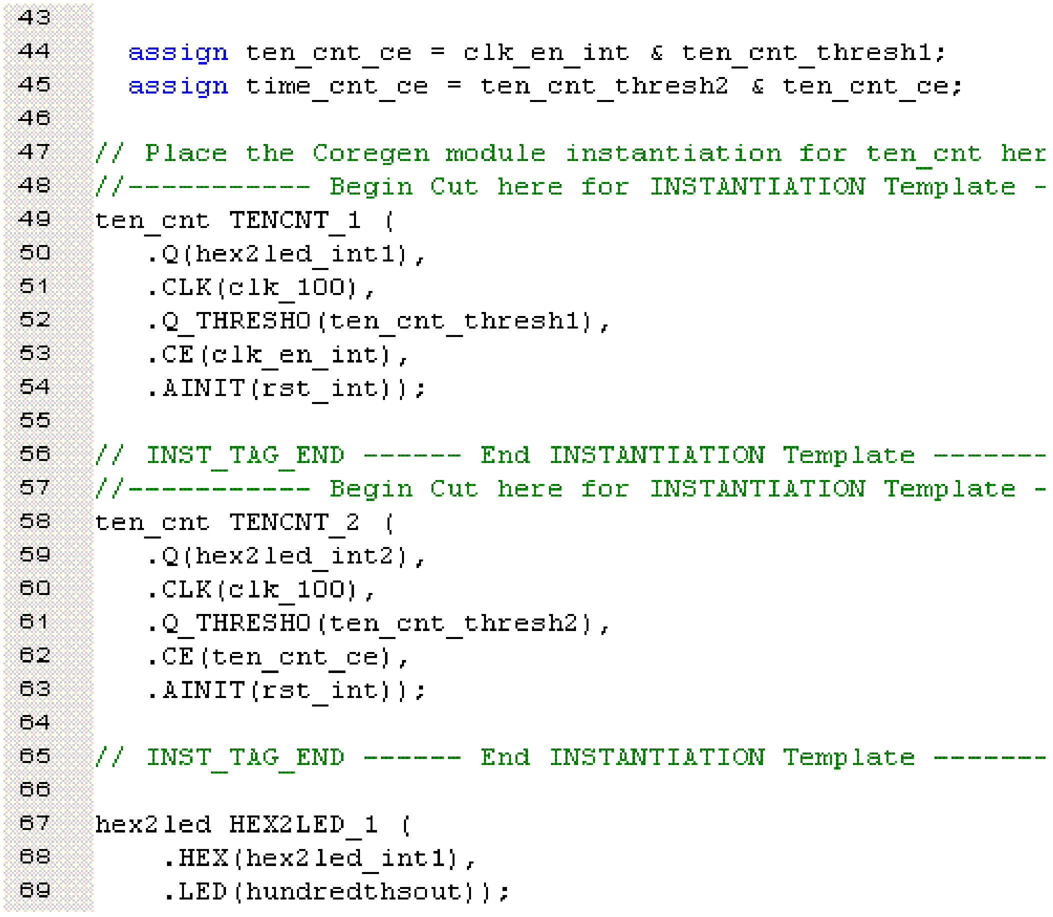
   `“//----------- Begin cut here for INSTANTIATION TEMPLATE ---//”`

   to

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

   `“// Place the Coregen module instantiation for ten_cnt here.”`
8. Select **Edit** → **Paste** two times because two CORE Generator counters are needed.
9. Change the instance names from “`YourInstanceName`” to “`TENCNT_1`” and “`TENCNT_2`”.

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



```

  assign ten_cnt_ce = clk_en_int & ten_cnt_thresh1;
  assign time_cnt_ce = ten_cnt_thresh2 & ten_cnt_ce;

// Place the Coregen module instantiation for ten_cnt her
//----------- Begin Cut here for INSTANTIATION Template -
ten_cnt TENCNT_1 (
   .Q(hex2led_int1),
   .CLK(clk_100),
   .Q_THRESH0(ten_cnt_thresh1),
   .CE(clk_en_int),
   .AINIT(rst_int));

// INST_TAG_END ------ End INSTANTIATION Template -------
//----------- Begin Cut here for INSTANTIATION Template -
ten_cnt TENCNT_2 (
   .Q(hex2led_int2),
   .CLK(clk_100),
   .Q_THRESH0(ten_cnt_thresh2),
   .CE(ten_cnt_ce),
   .AINIT(rst_int));

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

hex2led HEX2LED_1 (
    .HEX(hex2led_int1),
    .LED(hundredthsout));
```

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

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

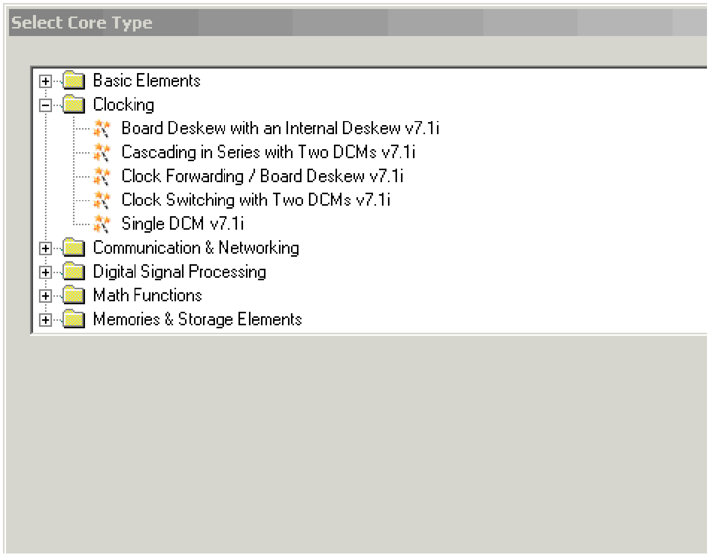
Creating a DCM Module

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

Using DCM Wizard

To create the DCM1 module:

1. In Project Navigator, select **Project** → **New Source**.
2. In the New Source dialog box, select **IP (CoreGen & Architecture Wizard)** source and type **DCM1** for the file name.
3. Click **Next**.
4. In the Select Core Type window, select **Clocking** → **Single DCM**.


Select Core Type

- Basic Elements
- Clocking
  - Board Deskew with an Internal Deskew v7.1i
  - Cascading in Series with Two DCMs v7.1i
  - Clock Forwarding / Board Deskew v7.1i
  - Clock Switching with Two DCMs v7.1i
  - Single DCM v7.1i
- Communication & Networking
- Digital Signal Processing
- Math Functions
- Memories & Storage Elements

Figure 2-13: **Selecting Single DCM Core Type**

5. Click **Next,** then **Finish.** The Clocking Wizard is launched.
6. Verify that RST, CLK0 and LOCKED are selected.
7. Select **CLKFX**.
8. Type **50** and select **MHz** for the Input Clock Frequency.
9. Verify the following settings:
   - CLKIN Source: **External, Single**
   - Feedback Source: **Internal**
   - Feedback Value: **1X**
   - Phase Shift: **None**
   - Duty Cycle Correction: **selected**
10. Select the **Advanced** button.
11. Select the **Wait for DCM lock before DONE Signal goes high** option.
12. Click **OK**.

    An informational message about the LCK_cycle and the STARTUP_WAIT BitGen option appears.
13. Click **OK**, click **Next**, click **Next**.
14. In the Clock Frequency Synthesizer window, type **26.144 MHz** in the Use output frequency box.

$$(26.2144 Mhz)/2^{18} = 100 Hz$$

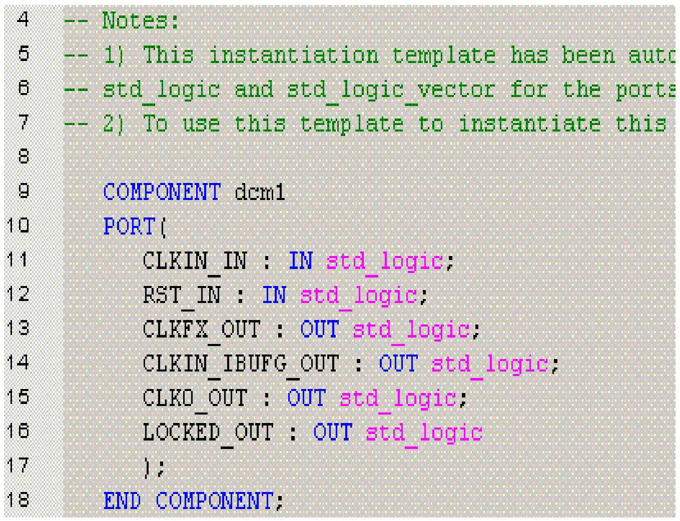
15. Click **Next** and then **Finish**.

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

Instantiating the DCM1 Macro - VHDL Design

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

1. In Project Navigator, in the Sources in the Project window, select *dcm1.xaw*.
2. In the Processes for Source window, click the **+** next to Design Utilities to expand the process hierarchy.
3. Double-click **View HDL Instantiation Template**.
4. Select the component declaration template in the newly opened HDL Instantiation Template.
5. Select **Edit** → **Copy**.

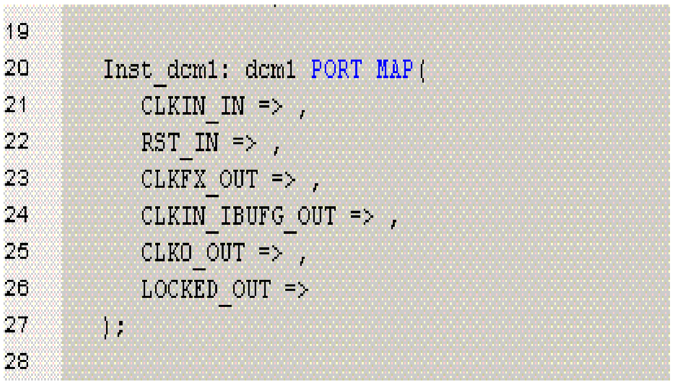


```
-- Notes:
-- 1) This instantiation template has been autc
-- std_logic and std_logic_vector for the ports
-- 2) To use this template to instantiate this

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

Figure 2-14: **VHDL DCM Component Declaration**

6. Place the cursor in the *stopwatch.vhd* file in a section labeled

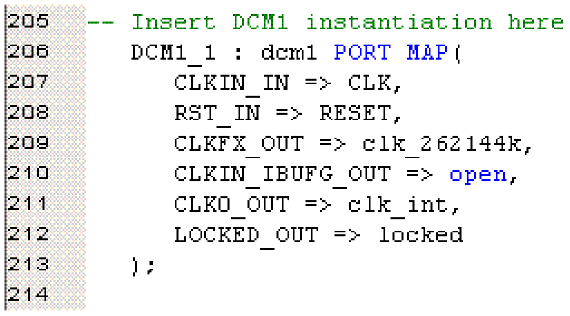
   `-- Insert DCM1 component declaration here.`
7. Select **Edit** → **Paste** to paste the component declaration.
8. Select the instantiation template in the newly opened HDL Instantiation Template.
9. Select **Edit** → **Copy**.



```

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

```

Figure 2-15: **VHDL DCM Component Instantiation**

10. Place the cursor in the *stopwatch.vhd* file in a section labeled
`-- Insert DCM1 instantiation here.`
11. Select **Edit** → **Paste** to paste the instantiation template.
12. Make the necessary changes as shown in the figure below.



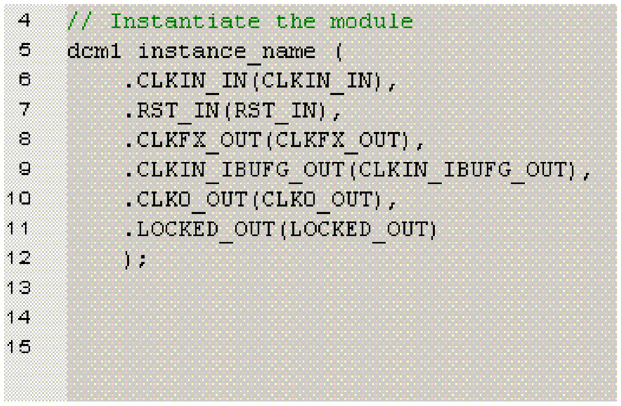
```
-- Insert DCM1 instantiation here
   DCM1_1 : dcm1 PORT MAP(
      CLKIN_IN => CLK,
      RST_IN => RESET,
      CLKFX_OUT => clk_262144k,
      CLKIN_IBUFG_OUT => open,
      CLK0_OUT => clk_int,
      LOCKED_OUT => locked
   );
```

Figure 2-16: **VHDL Instantiation for dcm1**

Instantiating the DCM1 Macro - Verilog

To instantiate the DCM1 macro for your Verilog design:

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

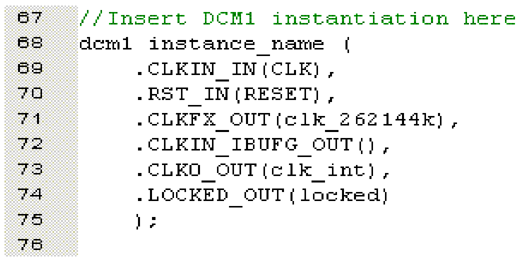


```
// Instantiate the module
dcm1 instance_name (
    .CLKIN_IN(CLKIN_IN),
    .RST_IN(RST_IN),
    .CLKFX_OUT(CLKFX_OUT),
    .CLKIN_IBUFG_OUT(CLKIN_IBUFG_OUT),
    .CLK0_OUT(CLK0_OUT),
    .LOCKED_OUT(LOCKED_OUT)
    );
```

Figure 2-17: **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.



```
//Insert DCM1 instantiation here
dcm1 instance_name (
     .CLKIN_IN(CLK),
     .RST_IN(RESET),
     .CLKFX_OUT(clk_262144k),
     .CLKIN_IBUFG_OUT(),
     .CLK0_OUT(clk_int),
     .LOCKED_OUT(locked)
     );
```

Figure 2-18: **Verilog Instantiation for dcm1**

Synthesizing the Design

So far you have been using XST for syntax checking. Next, you will synthesize the design using either XST, Synplify / Synplify Pro, LeonardoSpectrum 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**

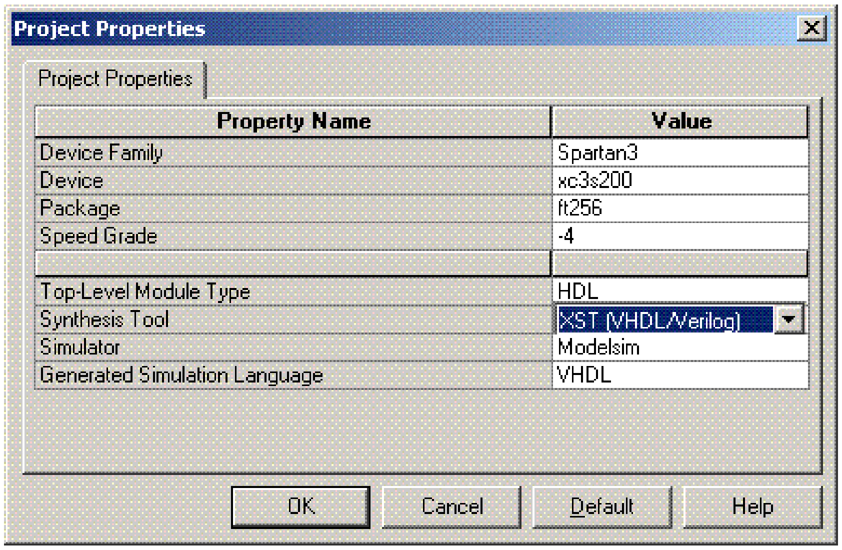
  Translates and optimizes the HDL code into a set of components that the synthesis tool can recognize.

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



Property Name	Value
Device Family	Spartan3
Device	xc3s200
Package	ft256
Speed Grade	-4
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim
Generated Simulation Language	VHDL

Figure 2-19: **Specifying Synthesis Tool**

Note: If you do not see the 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, 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."

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

Read the section for your synthesis tool:

- "Synthesizing the Design using XST"
- "Synthesizing the Design using Synplify / Synplify Pro"
- "Synthesizing the Design using LeonardoSpectrum"
- "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** → **New Source**.
2. In the New Source dialog box, select **User Document** as the source type, and enter the file name **stopwatch.xcf**.
3. Select **Next**, and **Finish**.

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

```
NET “CLK” TNM_NET = “CLK_GROUP”;
TIMESPEC “TS_CLK”=PERIOD “CLK” 20 ns;
BEGIN MODEL stopwatch
    NET "CLK" TNM_NET = "CLK";
    NET "CLK"  LOC = "T9";
    NET "AN<0>" LOC = "d14";
    NET "AN<1>" LOC = "g14";
    NET "AN<2>" LOC = "f14";
    NET "AN<3>" LOC = "e13";
    NET "RESET" LOC = "L13";
    NET "SEG_A" LOC = "e14";
    NET "SEG_B" LOC = "g13";
    NET "SEG_C" LOC = "n15";
    NET "SEG_D" LOC = "p15";
    NET "SEG_E" LOC = "r16";
    NET "SEG_F" LOC = "f13";
    NET "SEG_G" LOC = "n16";
    NET "SEG_DP" LOC = "P16";
    NET "STRTSTOP"  LOC = "M13";
END;
```

5. Select **File** → **Save**

Note: For more constraint options in the implementation tools, see “Using the Constraints Editor” and “Using the Pin-out Area Constraints Editor (PACE)” in Chapter 5, “Design Implementation.”

Entering Synthesis Options

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

To enter synthesis options:

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

Synthesizing the Design

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

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

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

The RTL / Technology Viewer

XST can generate a schematic representation of the HDL code that you have entered. A schematic view of the code is helpful for analyzing your design to see a graphical connection between the various components that XST has inferred. 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 for Source window, click **+** next to Synthesize to expand the process hierarchy.
2. Double-click **View RTL Schematic** or **View Technology Schematic**.

The RTL Viewer displays the schematic. Right-click on the schematic to view various options for the schematic viewer.

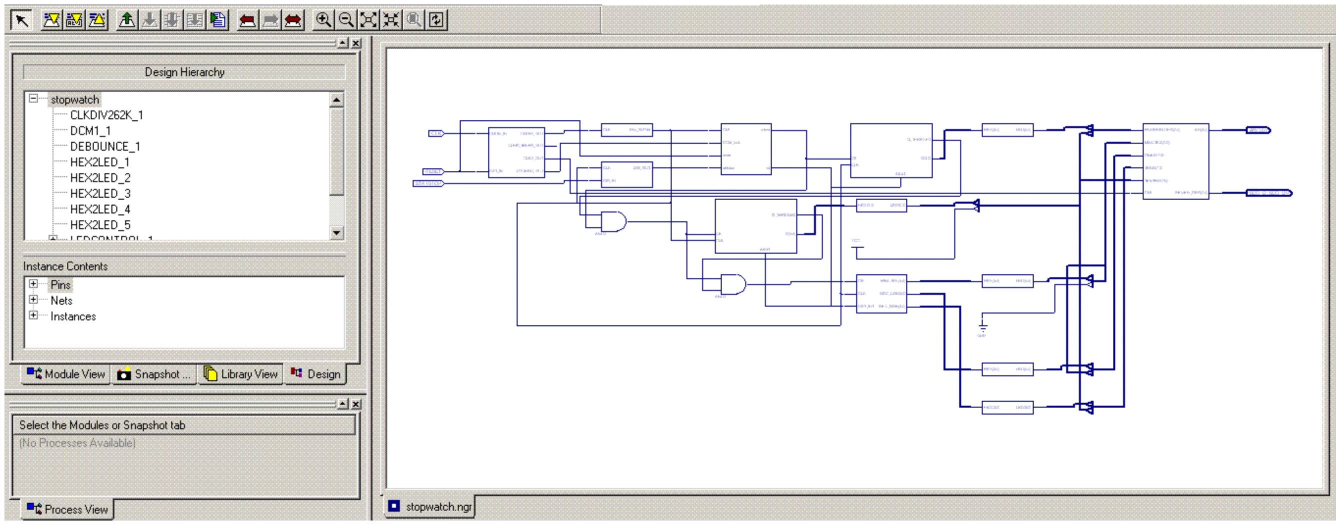


Figure 2-20: **RTL Viewer**

You have completed XST synthesis. An NGC file now exists for the Stopwatch design. To continue with the HDL Flow, go toChapter 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 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/xilinx7/.

Synthesizing the Design using Synplify/Synplify Pro

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

To synthesize the design, set the global synthesis options:

1. Select *stopwatch.vhd* (or *stopwatch.v*).
2. Right-click **Synthesize** in the Processes for Source window.
3. From the menu, select **Properties**.
4. Set the Default Frequency to **50** (units are in MHz), and check the **Write Vendor Constraint File** box.
5. Click **OK** to accept these values.
6. Select *stopwatch.vhd* (or *stopwatch.v*) and double-click the **Synthesize** process to run synthesis.

   Note: This step can also be done by selecting *stopwatch.vhd* (or *stopwatch.v*), clicking **Synthesize** in the Processes for Source window, 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 of each file that was compiled. The report also lists FSM extractions, inferred memory, warnings on latches, unused ports and removal of redundant logic.

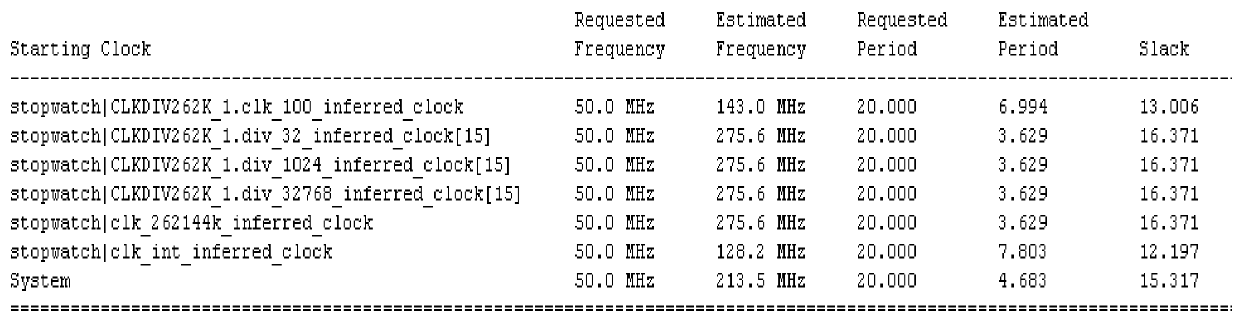
Note: Black boxes (modules not read into a design environment) are always noted as unbound in the Synplify reports. As long as the underlying netlist (.ngo, .ngc or .edn) for a black box exists in the project directory, the implementation tools merge the netlist into the design during the Translate phase.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

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.



```
                                                 Requested     Estimated     Requested     Estimated
Starting Clock                                   Frequency     Frequency     Period        Period        Slack
--------------------------------------------------------------------------------------------------------------
stopwatch|CLKDIV262K_1.clk_100_inferred_clock    50.0 MHz      143.0 MHz     20.000        6.994         13.006
stopwatch|CLKDIV262K_1.div_32_inferred_clock[15] 50.0 MHz      275.6 MHz     20.000        3.629         16.371
stopwatch|CLKDIV262K_1.div_1024_inferred_clock[15] 50.0 MHz    275.6 MHz     20.000        3.629         16.371
stopwatch|CLKDIV262K_1.div_32768_inferred_clock[15] 50.0 MHz   275.6 MHz     20.000        3.629         16.371
stopwatch|clk_262144k_inferred_clock             50.0 MHz      275.6 MHz     20.000        3.629         16.371
stopwatch|clk_int_inferred_clock                 50.0 MHz      128.2 MHz     20.000        7.803         12.197
System                                           50.0 MHz      213.5 MHz     20.000        4.683         15.317
==============================================================================================================
```

Figure 2-21: **Synplify's Estimated Timing Data**

Resource Utilization

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

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

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

Synthesizing the Design using LeonardoSpectrum

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

Processes available in LeonardoSpectrum synthesis include:

- **Check Syntax**

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

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

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

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

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

- **View Technology Schematic**

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

- **View Critical Path Schematic**

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

Modifying Constraints

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

To modify constraints:

1. Click **+** next to Synthesize to expand the process hierarchy.
2. Double-click the **Modify Constraints** process.

   LeonardoSpectrum displays. For first time users, the LeonardoSpectrum tool launches in Quick Setup mode.
3. Click the **Advanced Flow** icon.

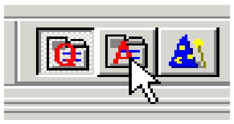


Figure 2-22: **LeonardoSpectrum Advanced Flow Icon**

4. Click the **Constraints** tab.

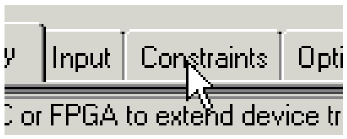


Figure 2-23: **LeonardoSpectrum Constraints Tab**

The constraints sub-tabs are as follows.

- **Global**

  Enables you to enter constraints that affect the entire design: PERIOD, OFFSETs and pad-to-pad type constraints. The constraints entered here modify the run script only. A constraints file is not generated.

- **Clock**

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

- **Input**

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

- **Output**

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

- **Signal**

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

- **Module**

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

- **Path**

  Enables you to create false and multicycle paths.

- **Report**

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

In the Constraints tab, enter the following constraints:

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

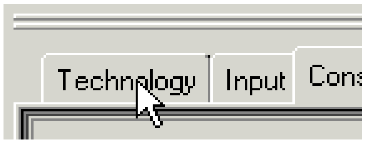


Figure 2-24: **LeonardoSpectrum Technology Tab**

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

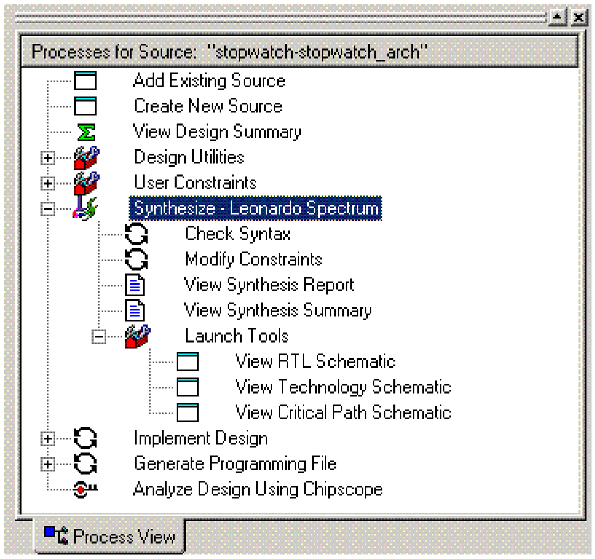
Note: For more constraint options in the implementation tools, see "Using the Constraints Editor" and "Using the Pin-out Area Constraints Editor (PACE)" in Chapter 5, "Design Implementation."

Entering Synthesis Options through ISE

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

For this tutorial, set the global synthesis options:

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


Processes for Source: "stopwatch-stopwatch_arch"

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

Figure 2-25: **LeonardoSpectrum Synthesis Processes**

The RTL/Technology Viewer

LeonardoSpectrum can generate a schematic representation of the HDL code that you have entered. A schematic view of the code is helpful for analyzing your design to see a graphical connection between the various components that LeonardoSpectrum has inferred. To launch the design in LeonardoSpectrum's RTL Viewer, double-click the **View RTL Schematic** process. The following figure displays the design in an RTL view.

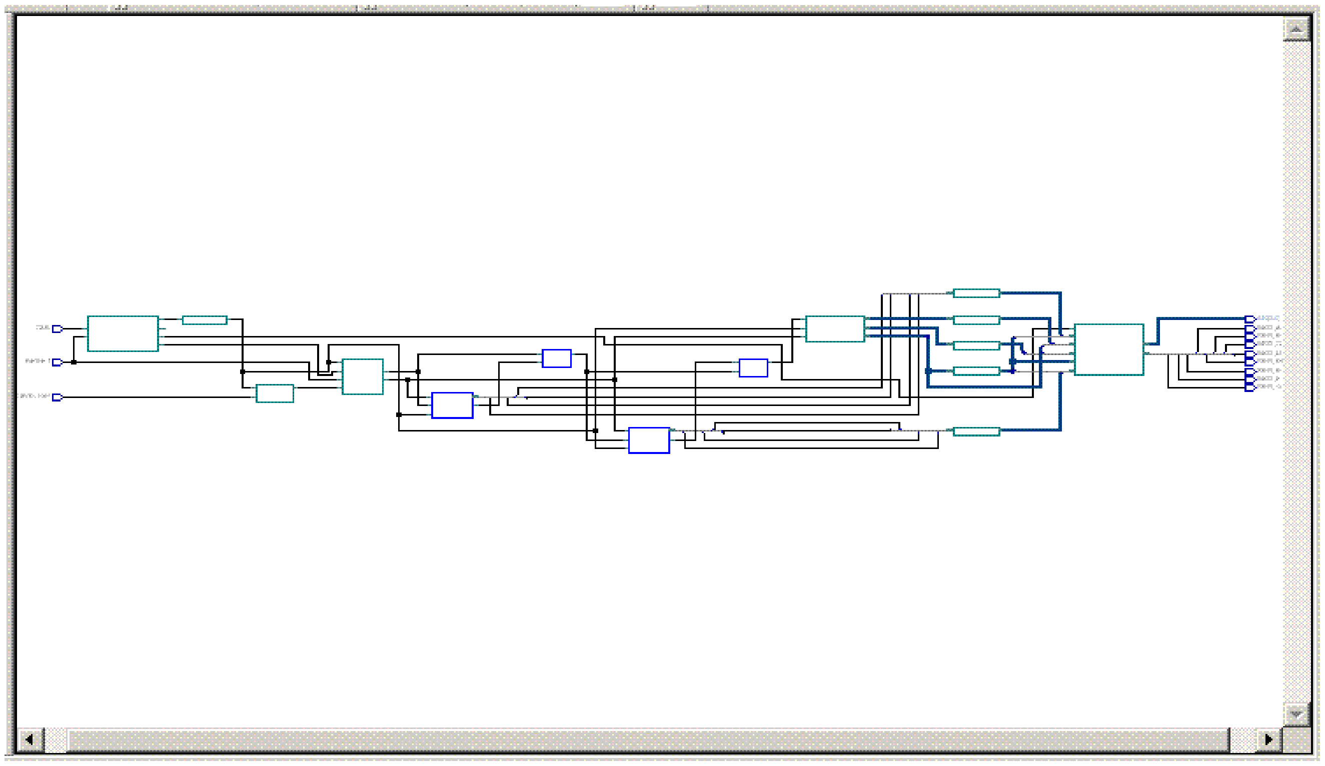


Figure 2-26: **Stopwatch Design in LeonardoSpectrum RTL Viewer**

LeonardoSpectrum also has the capability of generating a technology-specific view of the design after synthesis in the Technology Viewer. This schematic representation is useful for verifying the connections of the inferred elements.

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

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

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

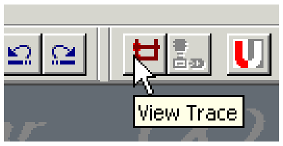


Figure 2-27: **LeonardoSpectrum View Trace Icon**

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

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

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

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 Area Report**

  Summarizes the resources that have been used for the target technology.

- **View Timing Report**

  Gives a detailed map and timing report with no information on the synthesis optimizations.

- **View RTL Schematic**

  Accessible from the Launch Tools hierarchy, this process displays 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.

- **Open Standalone Precision Project**

  Accessible from the Launch Tools hierarchy, this process enables you to open the Precision Synthesis project as it is currently setup from Project Navigator. This process enables you to use the debug features available from Precision Synthesis that are not available from within Project Navigator.

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.

Select *stopwatch.vhd* (or *stopwatch.v*) and double-click the **Synthesize** process in the Processes for Source window.

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 is helpful for analyzing your design to see 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.

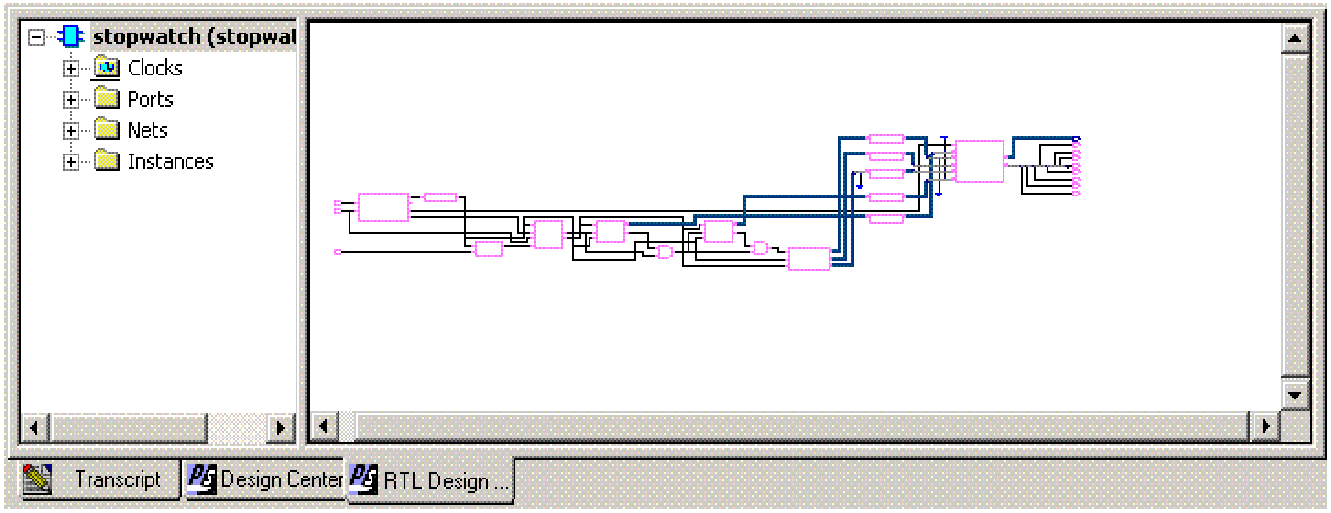


Figure 2-28: **Stopwatch Design in Precision Synthesis RTL Viewer**

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

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

Chapter 3

Schematic-Based Design

This chapter includes the following sections.

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

Overview of Schematic-based Design

This chapter guides you through a typical FPGA schematic-based design procedure using 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™-3 device; however, all of the principles and flows taught are applicable to any Xilinx® device family, unless otherwise noted.

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

This chapter is the first in the "Schematic Design Flow." In the first part of the tutorial, you will use the ISE design entry tools to complete the design. The design is composed of schematic elements, a state machine, a CORE Generator™ component, and 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 (Chapter 7, "iMPACT Tutorial.")

Getting Started

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

Required Software

You must have Xilinx ISE 7.x to perform this tutorial. For this design you must install the Spartan-3 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, *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 7.1i Installation Guide and Release Notes*.

Note: The free Webpack software tool available on the Xilinx Web site does not contain Core Generator software. Files and instructions are included to allow WebPack users to complete the tutorial without Core Generator.

Installing the Tutorial Project Files

The tutorial project files can be downloaded to your local machine from http://www.xilinx.com/support/techsup/tutorials/tutorials7.htm.

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

- `wtut_sc`
  (incomplete schematic tutorial)
- `wtut_sc\wtut_sc_completed`
  (complete schematic tutorial)

Unzip the tutorial projects in any directory with read-write permissions. The schematic tutorial files are copied into the directories when you unzip the project 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.

wtut_sc project

The *wtut_sc* project contains an incomplete copy of the tutorial design. You will create the remaining files when you perform the tutorial.

wtut_sc_completed solution project

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

Starting the ISE Software

To launch the ISE software package:

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

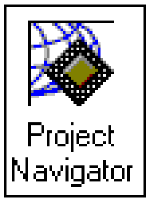
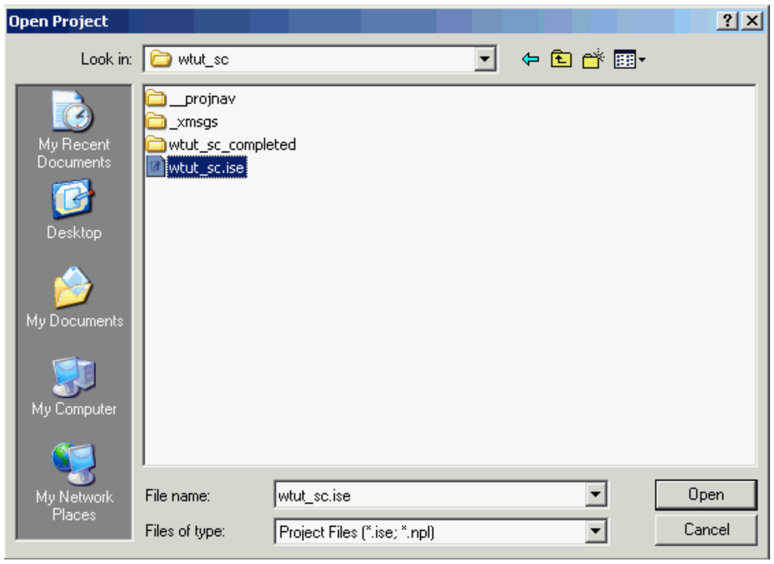


Figure 3-1: **Project Navigator Desktop Icon**

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



__projnav
_xmsgs
wtut_sc_completed
wtut_sc.ise

File name: wtut_sc.ise
Files of type: Project Files (*.ise; *.npl)

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

3. Browse to the directory *c:\xilinx\ISEexamples\wtut_sc.*
4. Double-click the project file, *wtut_sc.ise.*

Stopping the Tutorial

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

Design Description

The design used in this tutorial is a hierarchical, schematic-based design, which means that the top-level design file is a schematic sheet that refers to several other lower-level macros. The lower-level macros are a variety of different types of modules, including a 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 Watch design is shown in the following figure. Through the course of this chapter, you will create these modules, instantiate and connect them.

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

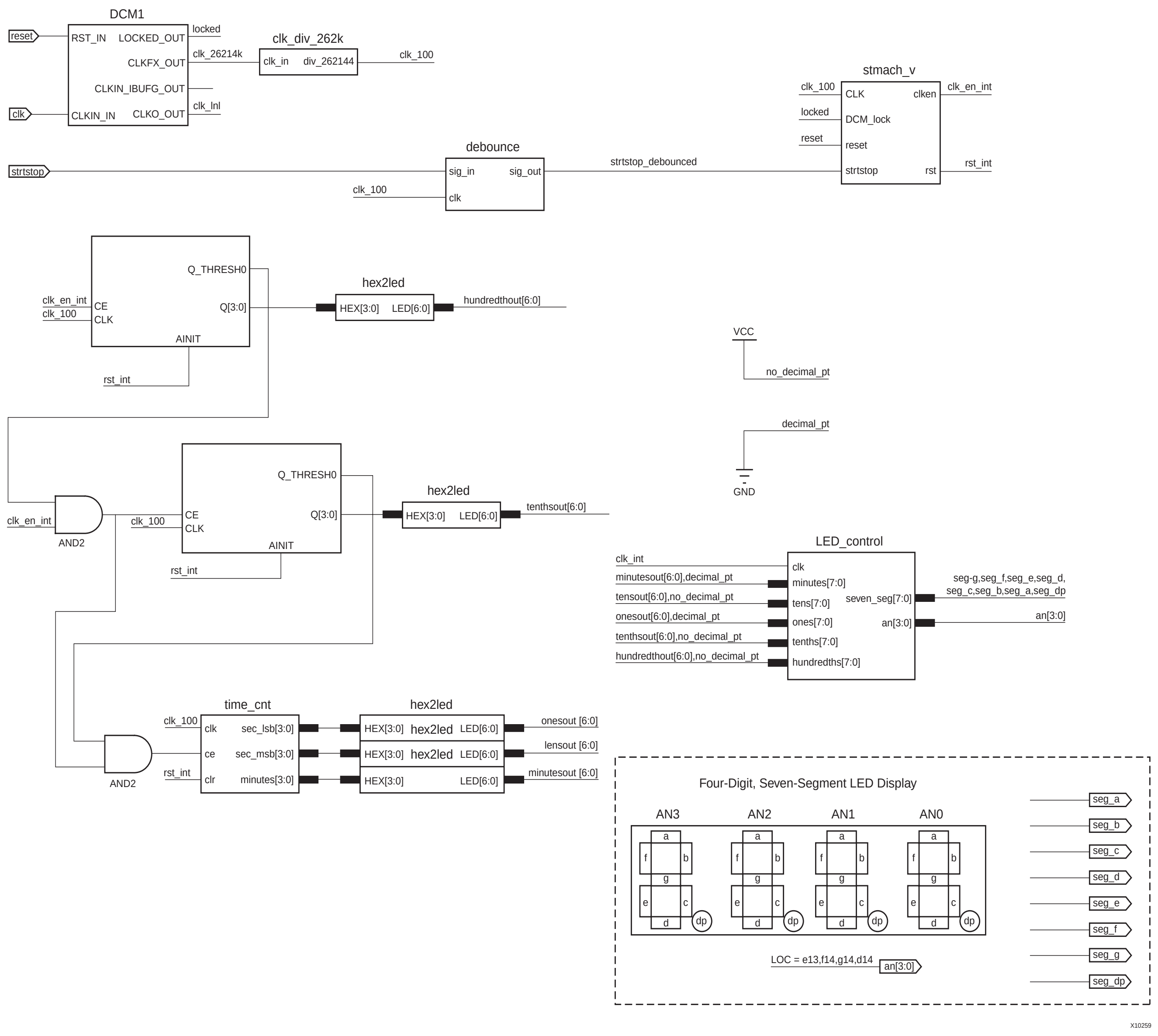


Figure 3-3: **Completed Watch Schematic**

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

Inputs

The following are input signals for the runner’s stopwatch design:

- **STRTSTOP**

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

- **RESET**

  Resets the stopwatch to 00.00.

- **CLK**

  System clock for the stopwatch design.

Outputs

The following are output signals for the design:

- **seg_a, seg_b, seg_c, seg_d, seg_e, seg_f, seg_g, seg_dp**

  These outputs drive the individual segments and the decimal point for all four digits of the stopwatch design. The digits of the stopwatch are displayed on 7-segment LED displays.

- **an(3:0)**

  This is a one-hot vector signal, which drives the anodes of the four 7-segment LED displays to determine which display will be lighted.

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 in this tutorial.

- **CLK_DIV_262k**

  Schematic-based macro that divides a clock frequency by 262,144.

- **DCM1**

  Clocking Wizard macro with internal feedback, frequency controlled output and duty-cycle correction. The CLKFX_OUT output converts the 50Mhz clock of the Spartan-3 demo board to 26.2144Mhz.

- **DEBOUNCE**

  Schematic module that implements a simplistic debounce circuit for the strtstop input signal.

- **HEX2LED**

  HDL-based macro that decodes each of the digit values from binary to 7-segment display format.

- **LED_control**

  Schematic module that controls the data multiplexing to the four 7-segment LED displays.

- **STMACH_V**

  State Machine macro that is defined and implemented in StateCAD.

- **TEN_CNT**

  CORE Generator 4-bit, binary encoded counter. This macro outputs a 4-bit code, which is decoded to represent the tenths and hundredths digits of the stopwatch.

- **TIME_CNT**

  Schematic-based module that counts from 0:0 to 9:59 decimal. This macro has three 4-bit outputs, which represent the minutes and seconds digits of the decimal value.

Note: The tutorial allows you to select either VHDL or Verilog for creating some of the functional blocks. Be consistent with your HDL language choice for functional blocks in a project. Because neither ModelSim Xilinx Edition (MXE) nor the ISE Simulator support mixed language simulation, one HDL choice is required in order to simulate the design in Chapter 4, "Behavioral Simulation." The example project is set up for VHDL simulation.

In order to select Verilog simulation, right-click on the device line (xc3s200-4ft256) and select **Properties**. In the properties window, change the Generated Simulation Language from VHDL to **Verilog**. Click **OK**.

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 Watch 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 Watch schematic available in the *wtut_sc* project is incomplete. In this tutorial, you will update the schematic in the Schematic Editor. After you have opened 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 in Project window.

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

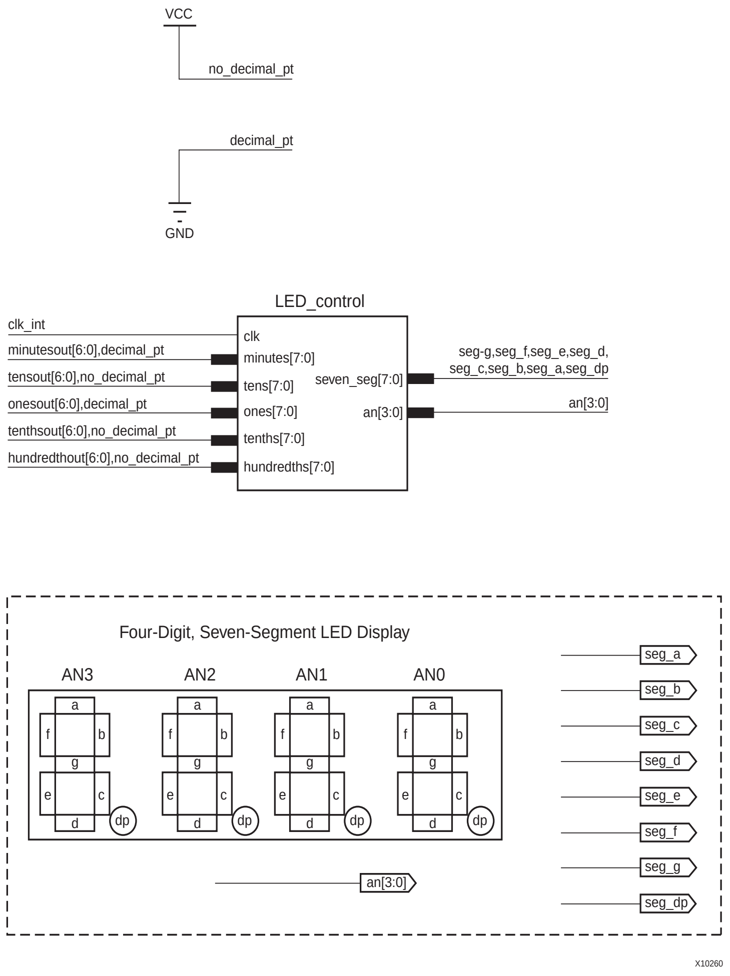


Figure 3-4: **Incomplete Watch Schematic**

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 the undock icon above the top right corner of the schematic.

After being undocked, the schematic window can be redocked by selecting the redock icon above the top right corner of the schematic.

Creating a Schematic-Based Macro

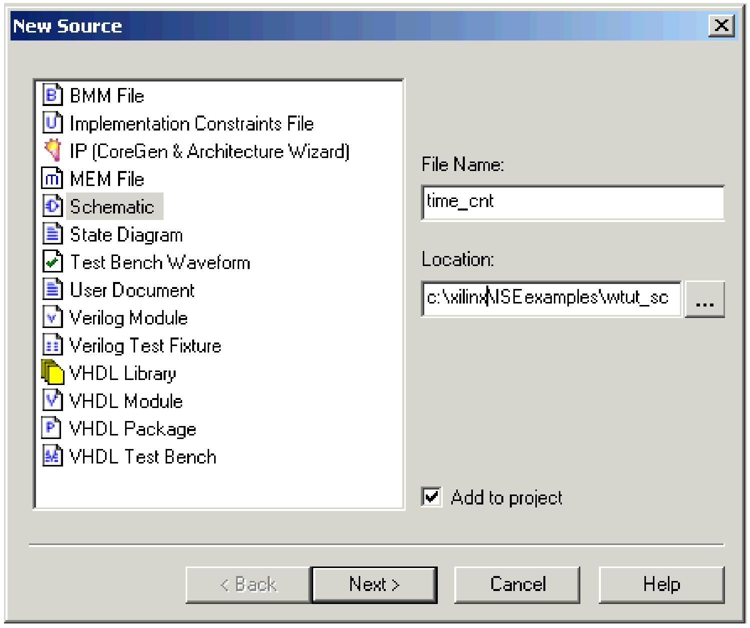
A schematic-based macro consists of a symbol and an underlying schematic. You can create either the underlying schematic or the symbol first. The corresponding symbol or schematic file can then be generated automatically.

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 three 4-bit outputs, which represent the minutes and seconds values of the stopwatch.

To create a schematic-based macro:

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



File Name: time_cnt

Location: c:\xilinx\ISEexamples\wtut_sc

☑ Add to project

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

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

2. Select **Schematic** as the source type.
3. Enter **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.

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.

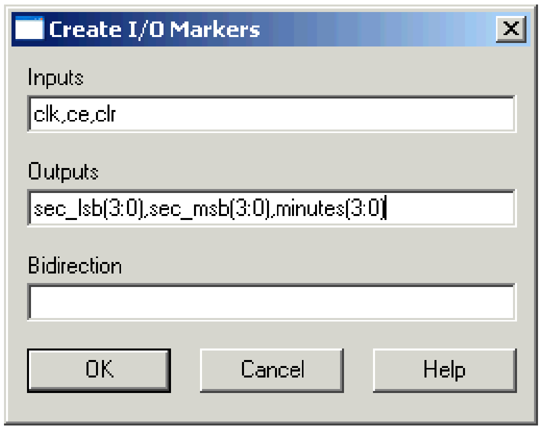
Adding I/O Markers

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 Options window opens.
2. In the Inputs box, enter **ce,clk,clr**.
3. In the Outputs box, enter **sec_lsb(3:0),sec_msb(3:0),minutes(3:0)**.



```
clk,ce,clr
sec_lsb(3:0),sec_msb(3:0),minutes(3:0)
```

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

4. Click **OK**. The six 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.

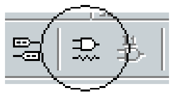


Figure 3-7: **Add Symbol Icon**

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

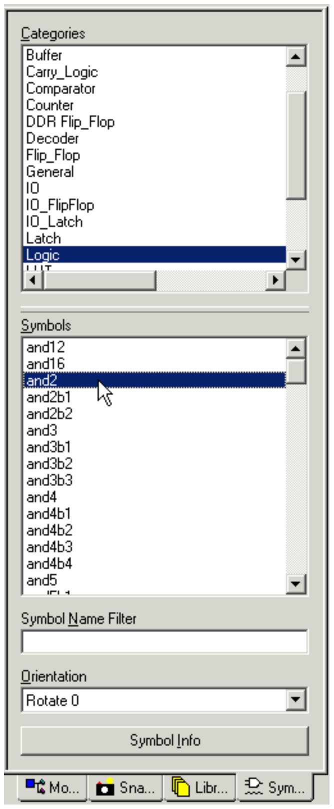


Figure 3-8: **Symbol Browser**

Note: To expand the Symbol Browser to see all contents, click the **Expand** icon at the top right corner of the Symbol tab.

Figure 3-9: **Symbol Tab Expand Icon**

The first component you will place is a CD4RE, a 4-bit BCD counter with clock enable, and synchronous clear.

2. Select the CD4RE component using one of two ways:
   - Highlight the **Counter** category from the Symbol Browser dialog box and select the component **CD4RE** from the symbols list.

     or

   - Select **All Symbols** and type **CD4RE** 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 CD4RE symbol.
4. Move the symbol outline to the location shown in Figure 3-10 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 the second CD4CE symbol 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.

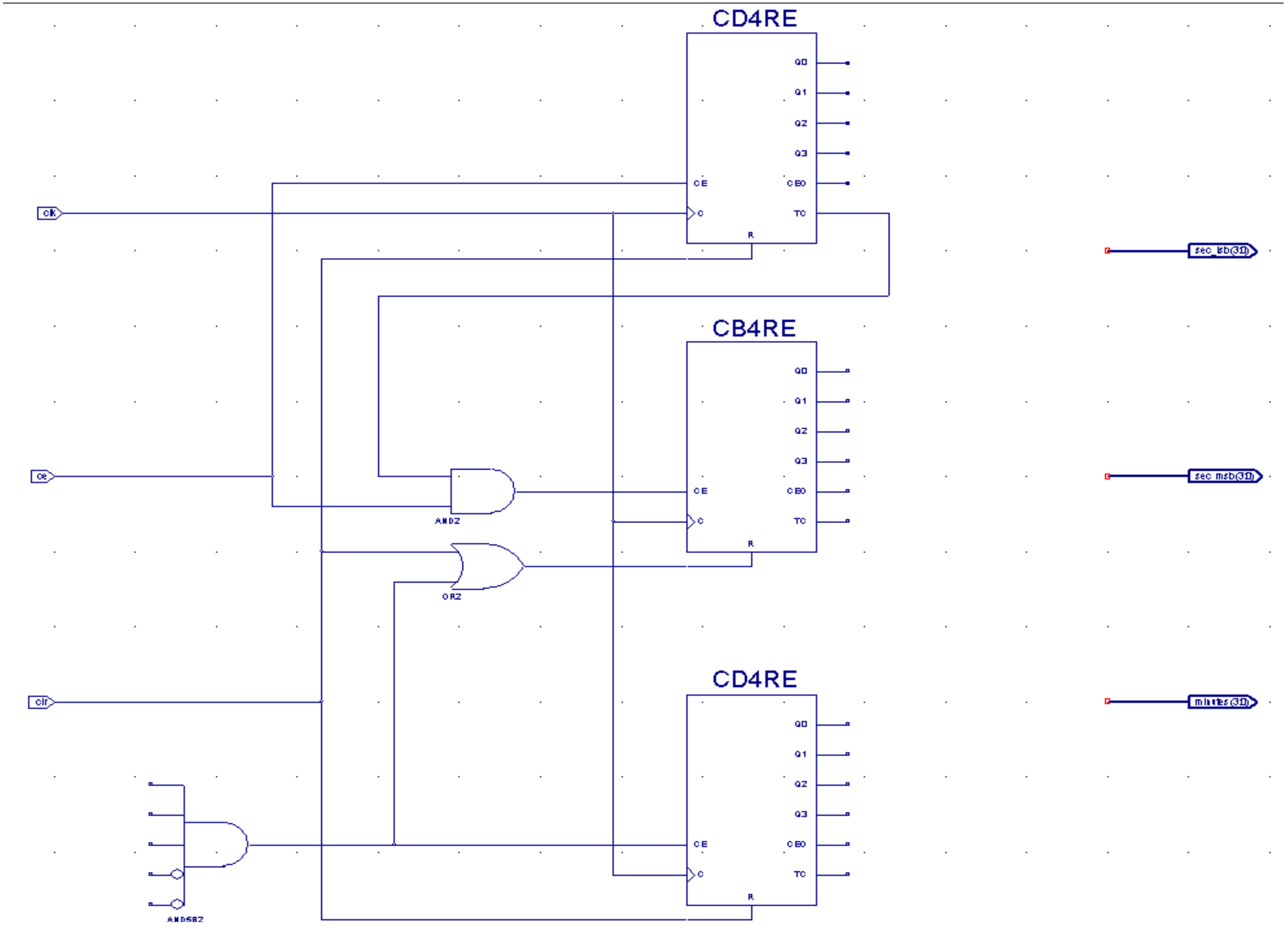


Figure 3-10: **Partially Completed time_cnt Schematic**

Placing the Remaining Components

Follow the steps above in "Adding Components to time_cnt" to place the following components on the schematic sheet:

- OR2
- CB4RE
- AND2
- AND5b2

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 each of these components, right-click on the component and select **Object Properties**. In the Object Properties window, select **Symbol Information**. Symbol information is also available in the Libraries Guides, accessible from the collection of software manuals on the web at http://www.xilinx.com/support/sw_manuals/xilinx7/.

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 AND2 and CB4RE components on the *time_cnt* schematic.

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

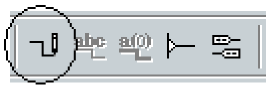


Figure 3-11: **Add Wires Icon**

2. Click the output pin of the AND2 and then click the destination pin CE on the CB4RE component. The Schematic Editor draws a net between the two pins.

Draw the nets to connect the remaining components as shown in the 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 which has been given a multi-bit name. To add a bus, use the methodology for adding wires and then add a multi-bit name. Once a bus has been created, you have the option of “tapping” this bus off to use each signal individually.

The next step is to create three buses called sec_lsb(3:0), sec_msb(3:0) and minute(3:0), each consisting of the 4 output bits of each counter, in the *time_cnt* schematic. The results can be found in the completed schematic.

To add the buses sec_lsb(3:0),sec_msb(3:0) and minute(3:0) to the schematic, perform the following steps:

1. Select **Add** → **Wire** or click the **Add Wires** icon in the Tools toolbar.
2. Click in the open space just above and to the right of the top CD4RE and then click again on pin of the sec_lsb(3:0) I/O marker. The wire should automatically be drawn as a bus with the name matching that of the I/O marker.
3. To verify this, zoom in. The bus is represented visually by a thicker wire.

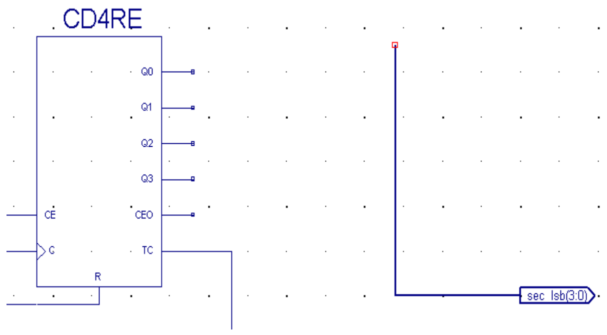


Figure 3-12: **Adding a Bus**

4. Repeat Steps 1 through 3 for the sec_msb(3:0) and minute(3:0) busses.
5. After adding the three 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 CB4RE and CD4RE counters to the buses. Use Bus Taps to tap off a single bit of a bus and connect it to another component.

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

To tap off a single bit of each bus:

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

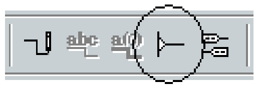


Figure 3-13: **Add Bus Tap Icon**

   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 correct orientation for the bus tap.
3. Place the tap on one of the three busses so that the wire side of the bus tap is pointing to an unconnected pin.
4. Repeat steps 1 to 3 to tapped off four bits from each of the three busses.

To connect each of the tap off bits:

1. Select **Add** → **Wire** or click the **Add Wire** icon in the Tools toolbar.
2. Draw a wire from each bus tap pin to the adjacent component pin.

3. Select **Add** → **Net Name** or click the **Add Net Name** icon in the Tools toolbar.
4. Type **sec_lsb(0)** in the Name field of the options toolbar.

   The net name is now at the end of your cursor.
5. Select **Increment the Name** in the Add Net Names Options dialog box.
6. With the Increment Name option selected, start at the top net and continue clicking down until you have named the fourth and final net sec_lsb(3).

Note: The Schematic Editor names the bus taps incrementally as they are drawn. Alternatively, name the first net sec_lsb(3) and select **Decrement the Name** in the Add Net Names Options dialog box, and nets are named from the bottom up.

7. Repeat Steps 4 through 6 for the sec_msb(3:0) bus and minute(3:0).
8. Press **Esc** to exit the Add Net Name mode.
9. Compare your *time_cnt* schematic with Figure 3-15 to ensure that all connections are made properly.

Note: It is the name of the wire that makes the electrical conection 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.

Adding Net Names

First, add a hanging wire to each of the five inputs of the AND5b2 component.

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.

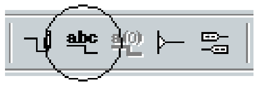


Figure 3-14: **Add Net Name Icon**

2. Type **minute_en** in the Name box of the Add Net Name options dialog box.

   The net name minute_en is now attached to the cursor.
3. Click the net attached to the output of the AND5b2 component.

   The name is then attached to the net. The net name will appear above the net if the name is placed on any point of the net other than an end point.
4. Type **msb_en** in the Name box of the Add Net Name options dialog box.

   The net name msb_en is now attached to the cursor.
5. Click the net attached to the output of the AND2 component.
6. With the net name msb_en still attached to the cursor, click on the net attached to the top input pin of the AND5b2 component.

Note: The two wires named msb_en 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 remaining AND5b2 inputs through net name association.

1. Select **Add** → **Net Name** or click the **Add Net Name** icon in the Tools toolbar.
2. Type **sec_msb(0)** in the Name box of the Add Net Name options dialog box.

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

3. Select **Increment the Name** in the Add Net Names options dialog box.

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

4. Click on one of the AND5b2 input nets which does not have an inversion bubble.
5. Click on the remaining three AND5b2 input nets of the so that the wires named sec_msb(1) and sec_msb(3) are attached to the inputs with inversion bubbles. Refer to Figure 3-15.

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

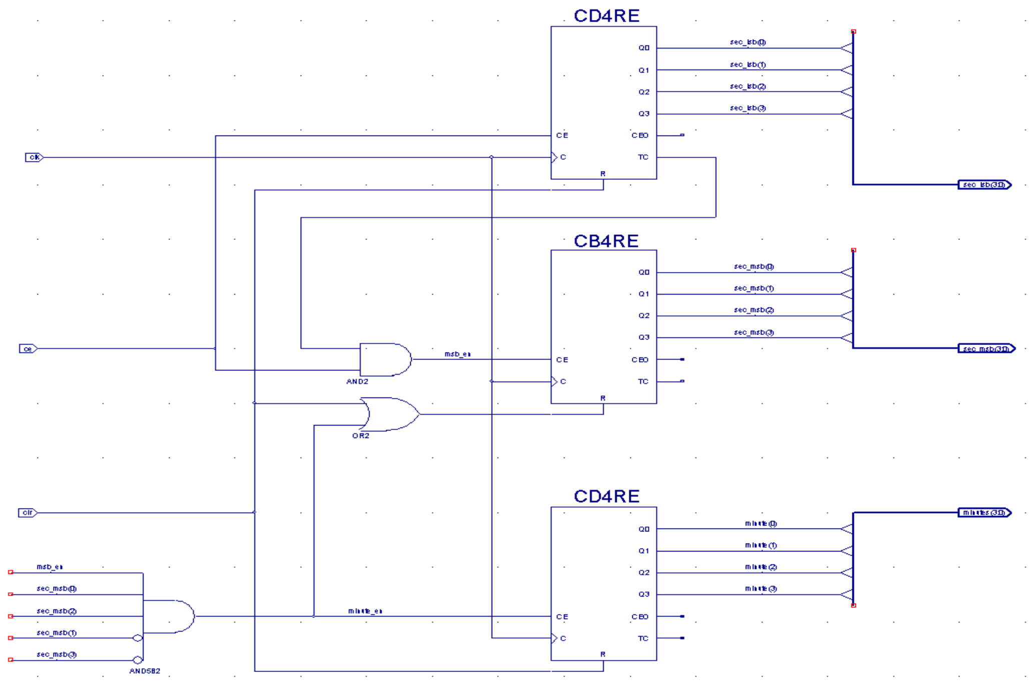


Figure 3-15: **Completed time_cnt Schematic**

Saving the Schematic

The *time_cnt* schematic is now complete.

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

Figure 3-16: **Save Icon**

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

2. 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 Watch 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. For this tutorial, you can follow either method.

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

1. In the Sources in Project window, select *time_cnt.sch.*
2. In the Processes for Source 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. Select **Tools** → **Symbol Wizard**.
2. In the Symbol wizard, select **Using Schematic** and select TIME_CNT in the schematic value field.
3. Click **Next**, then **Next**, then **Next**, and then **Finish** to use the wizard defaults.
4. 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 in Project window, double-click *stopwatch.sch* to open the schematic.
2. Select the **Add Symbol** icon.

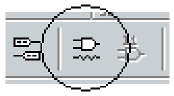


Figure 3-17: **Add Symbol Icon**

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 at approximately grid position [900,2100]. 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 pins of the time_cnt symbol. You will do this after adding other components to the Watch 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 counters, shift registers, RAM and multiplexers. You can customize and pre-optimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic, SRL16s, and distributed and block RAM.

In this section, you will create a CORE Generator module called ten_cnt. Ten_cnt is a 4-bit binary encoded counter. Two instances of ten_cnt will be used to produce the tenths and hundredths digits of the stopwatch's time value.

Note: WebPack users can not follow this part of the tutorial because the Core Generator tool is not delivered with the WebPack software. If you are using WebPack, please do the following:

- Copy the files from the wtut_sc\ten_cnt files directoty to the wtut_sc directory.
- Select **Project** → **Add Source**, select *ten_cnt.vhd* from the wtut_sc directory and click **Open**.
- Skip to "Creating a State Machine Module" section of the tutorial.

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 **ten_cnt** in the File Name field.
4. Click **Next**.
5. Double-click **Basic Elements - Counters**.
6. Select **Binary Counter**, click **Next** and click **Finish** to open the Binary Counter dialog box. This dialog box enables you to customize the counter to the design specifications.
7. Fill in the Binary Counter dialog box with the following settings:
   - Component Name: **ten_cnt**

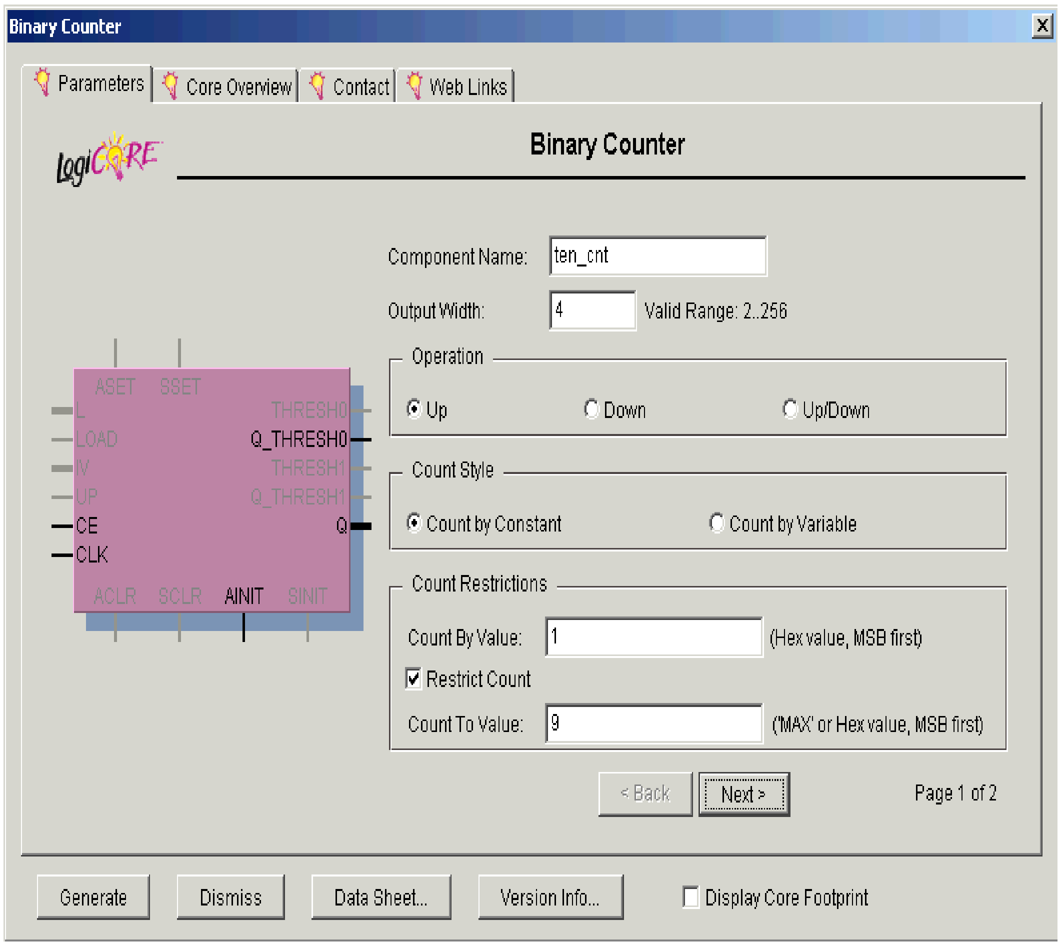
     Defines the name of the module.
   - Output Width: **4**

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

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

     Allows counting by a constant or a user-supplied variable.
   - Count Restrictions:
     - Count by value: **1**
     - Select **Restrict Count**
     - Count to value: **9**

This dictates the maximum count value.



Component Name: ten_cnt

Output Width: 4 Valid Range: 2..256

Operation: [✓ Up] Down Up/Down

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

Count Restrictions

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

[✓ Restrict Count]

Count To Value: 9 ('MAX' or Hex value, MSB first)

☐ Display Core Footprint

Figure 3-18: **CORE Generator Module Selector**

8. Click **Next**.
   - Threshold Options: Enable **Threshold 0** and set to **9**

     Signal goes high when the value specified has been reached.
   - Select **Registered**.
   - Click the **Register Options** button to open the Register Options dialog box.
9. Enter the following settings and then click **OK**.
   - Clock Enable: **Selected**
   - Asynchronous Settings: **Init** with a value of **0**
   - Synchronous Settings: **None**
10. Check that *only* the following pins are used (used pins will be highlighted on the symbol on the left side of the CORE Generator window):
    - **AINIT**
    - **CE**
    - **CLK**

- ♦ **Q**
- ♦ **Q_Thresh0**

11. Click **Generate**.

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

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

- ♦ **ten_cnt.sym**

  This file is a schematic symbol file.
- ♦ **ten_cnt.edn**

  This file is the netlist that is used during the Translate phase of implementation.
- ♦ **ten_cnt.vhd** and **ten_cnt.v**

  This file is the instantiation template that is used to incorporate the CORE Generator module into your source HDL.
- ♦ **ten_cnt.xco**

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

12. Click **Dismiss** to exit CORE Generator.

Creating a State Machine Module

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

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

Opening StateCAD

To open the partially complete diagram, first add the *stmach_v.dia* file to the project by selecting **Project** → **Add Source**. Then, double-click *stmach_v.dia* in the Sources in Project window. The state machine file is launched in StateCAD.

Note: You may see an error about using 8.3 file format. If so, click **OK** and select **File** → **Open** to open the *stmach_v.dia* file in StateCAD.

In the incomplete state machine diagram below:

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

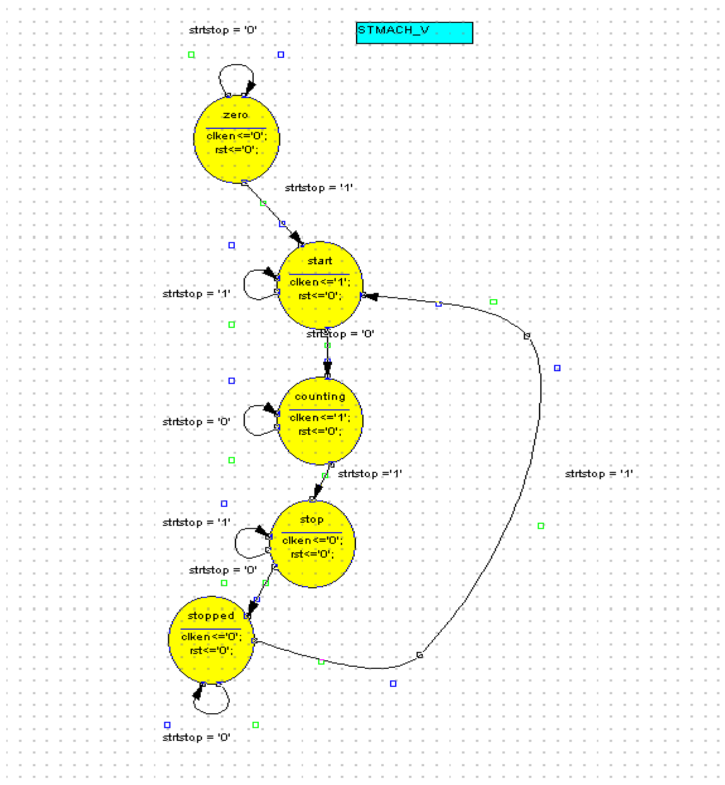


Figure 3-19: **Incomplete State Machine Diagram**

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.

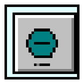


Figure 3-20: **Add State Icon**

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. Click the mouse to place the state bubble.

The state is given the default name, STATE0.

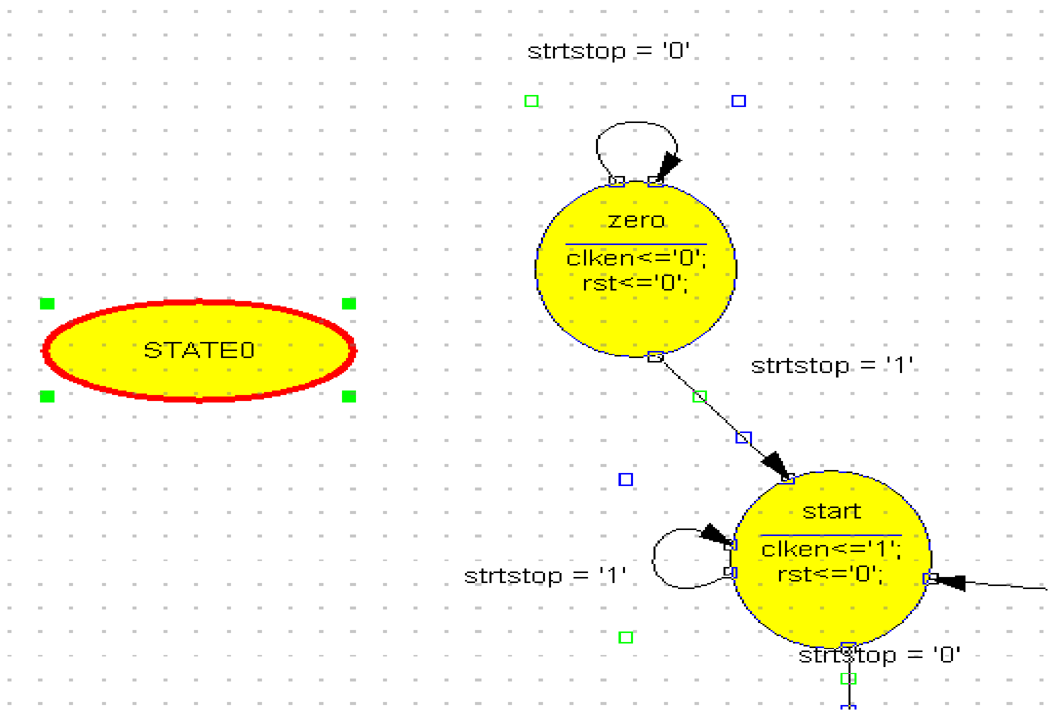


Figure 3-21: **Adding the CLEAR State**

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

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

4. Click **OK**.

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

Adding a Transition

A transition defines the movement between states of the state machine. Transitions are represented by arrows in the editor. You will 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 Transitions** icon in the vertical toolbar.

Figure 3-22: **Add Transitions Icon**

2. Double-click the **clear** state (one click to select it, and one click to start the transition.)
3. Click the **zero** state to complete the transition arrow.

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

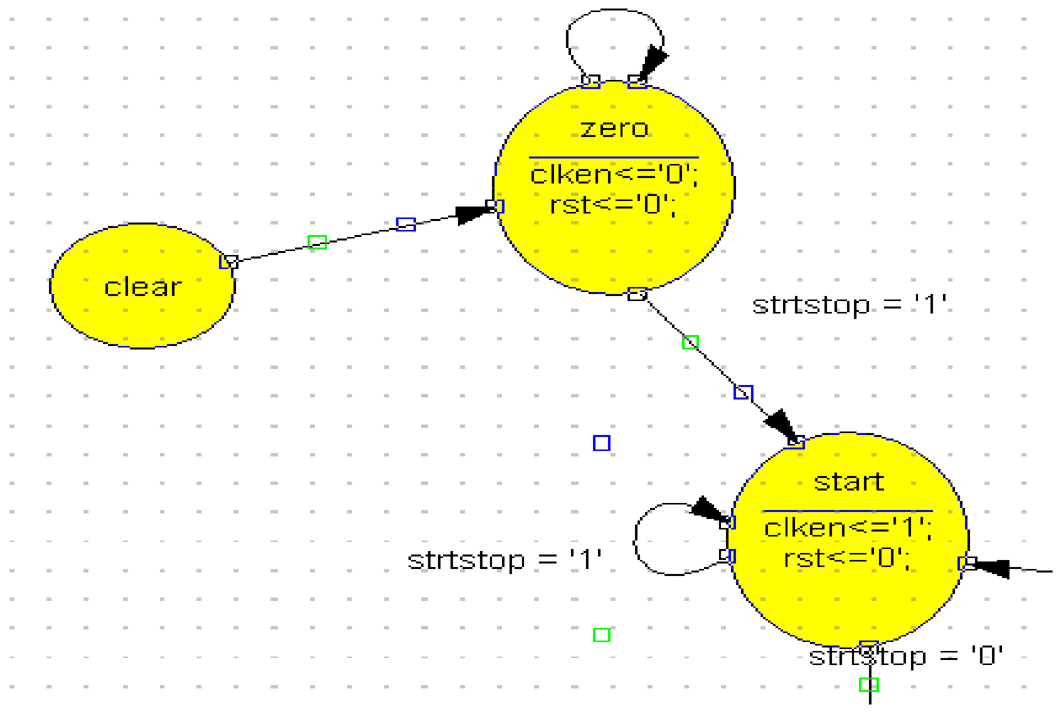


Figure 3-23: **Adding State Transition**

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

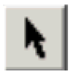


Figure 3-24: **Select Objects Icon**

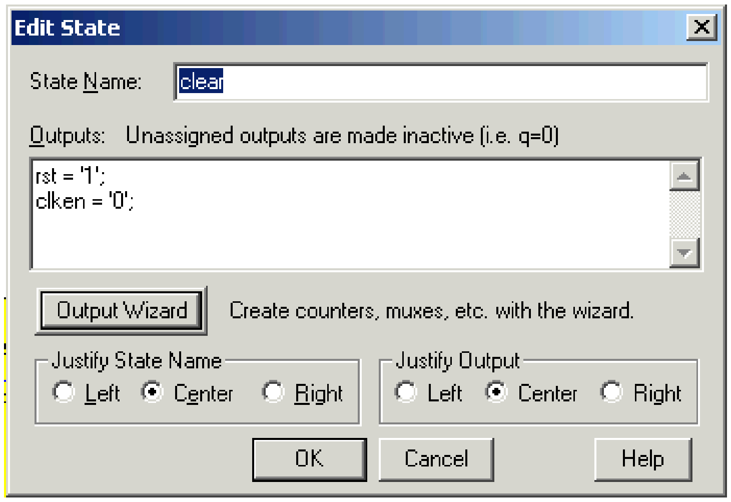
Adding a State Action

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

To add a state action:

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

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



State Name: clear

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

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

Create counters, muxes, etc. with the wizard.

Justify State Name: Left [✓ Center] Right

Justify Output: Left [✓ Center] Right

Figure 3-25: **Edit State Dialog Box**

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

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

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

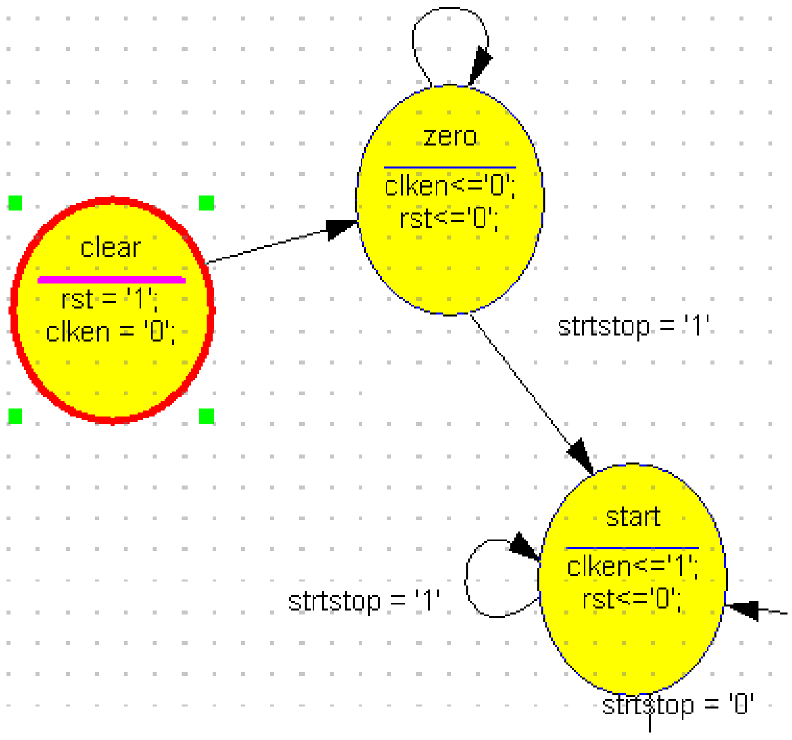


Figure 3-26: **Adding State Outputs**

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.

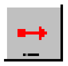


Figure 3-27: **Add Reset Icon**

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

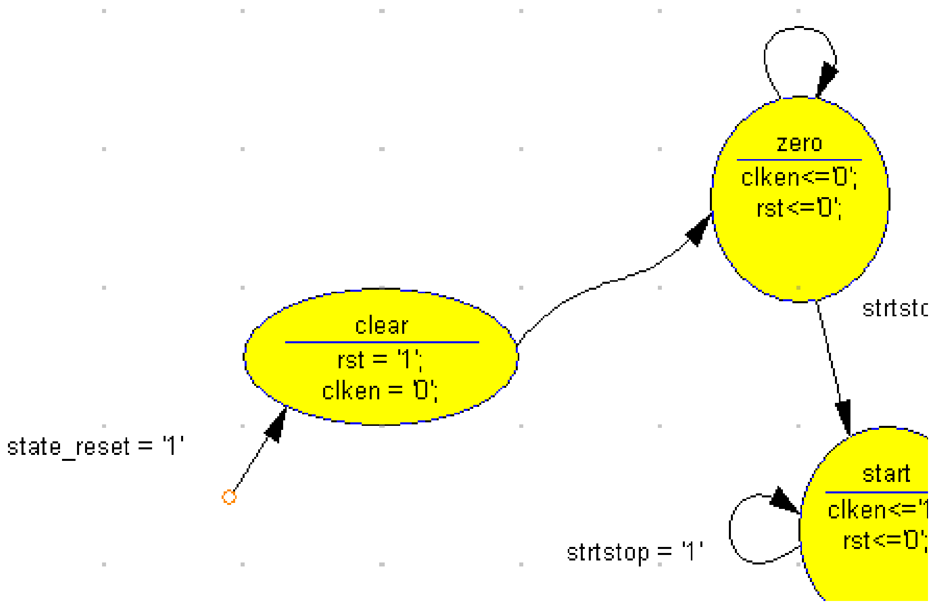


Figure 3-28: **Adding a Reset Condition**

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

Creating the State Machine Symbol

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

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

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

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

   The file *stmach_v.vhd* is added to the project in Project Navigator.
10. In the Sources in Project window, select *stmach_v.vhd*.
11. In the Processes for Source window, click the **+** beside Design Utilities to expand the hierarchy.
12. Double-click **Create Schematic Symbol**.

Note: To output a Verilog file in StateCAD, select **Options** → **Configuration** and change the Language selection to **Verilog**. In the instructions, STMACH_V.v replaces STMACH_V.vhd.

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. Select **Single DCM** in the Clocking hierarchy.

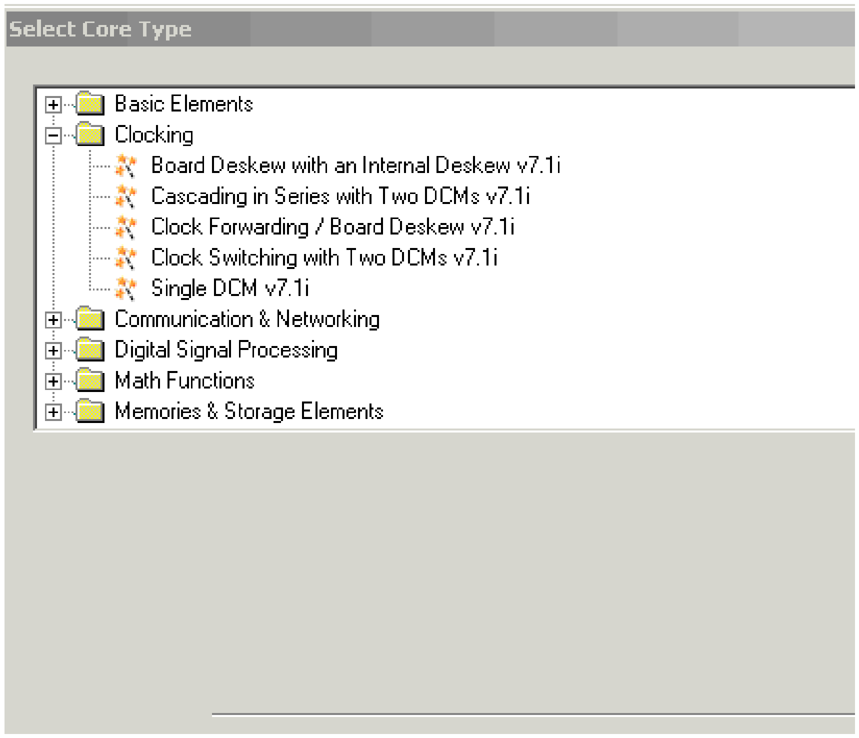
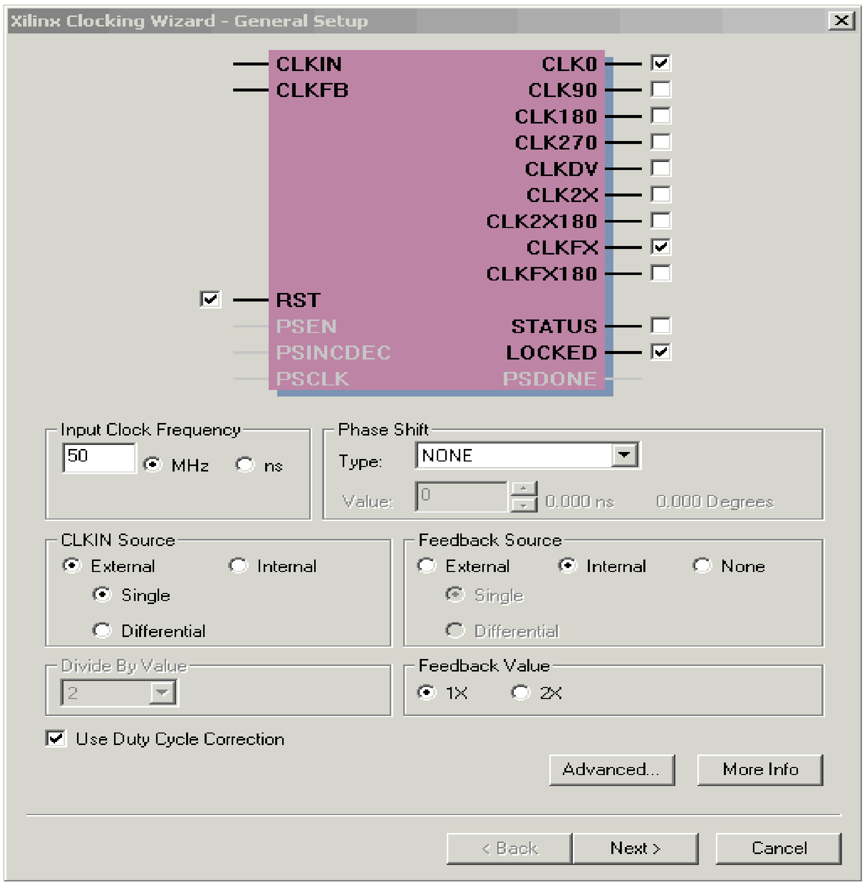

Select Core Type
Basic Elements
Clocking
Board Deskew with an Internal Deskew v7.1i
Cascading in Series with Two DCMs v7.1i
Clock Forwarding / Board Deskew v7.1i
Clock Switching with Two DCMs v7.1i
Single DCM v7.1i
Communication & Networking
Digital Signal Processing
Math Functions
Memories & Storage Elements

Figure 3-29: **Selecting Single DCM Core Type**

5. Click **Next** and click **Finish**.
6. Verify that **RST**, **CLK0** and **Locked** are selected.
7. Select the **CLKFX**.
8. Type **50** and select **Mhz** for the Input Clock Frequency.
9. Verify the following settings:
   - Clkin Source: **External, Single**
   - Feedback Source: **Internal**
   - Feedback Value: **1X**
   - Phase Shift: **None**
   - Duty Cycle Correction: **selected**



[✓ CLK0] [✓ CLKFX] [✓ RST] [✓ LOCKED]

Input Clock Frequency: 50 [✓ MHz] ns

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

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

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

Divide By Value: 2

Feedback Value: [✓ 1X] 2X

[✓ Use Duty Cycle Correction]

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**.

    An informational message about the LCK_cycle and the STARTUP_WAIT Bitgen options appears.
13. Click **OK**, click **Next**, click **Next**.
14. In the Clock Frequency Synthesizer window, type **26.2144** and select **Mhz** in the Use output frequency box.

$$(26.2144Mhz)/2^{18} = 100Hz$$

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

DCM1.xaw is added to the project.

Creating the DCM1 Symbol

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

1. In Project Navigator, in the Sources in Project window, select *DCM1.xaw*.
2. In the Processes for Source window, click the **+** beside Design Utilities to expand the hierarchy.
3. 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.

Next you will create a new HDL module. This macro serves to convert the 4-bit binary outputs of the ten_cnt and time_cnt modules into a 7-segment LED display format.

Using the New Source Wizard and ISE Text Editor

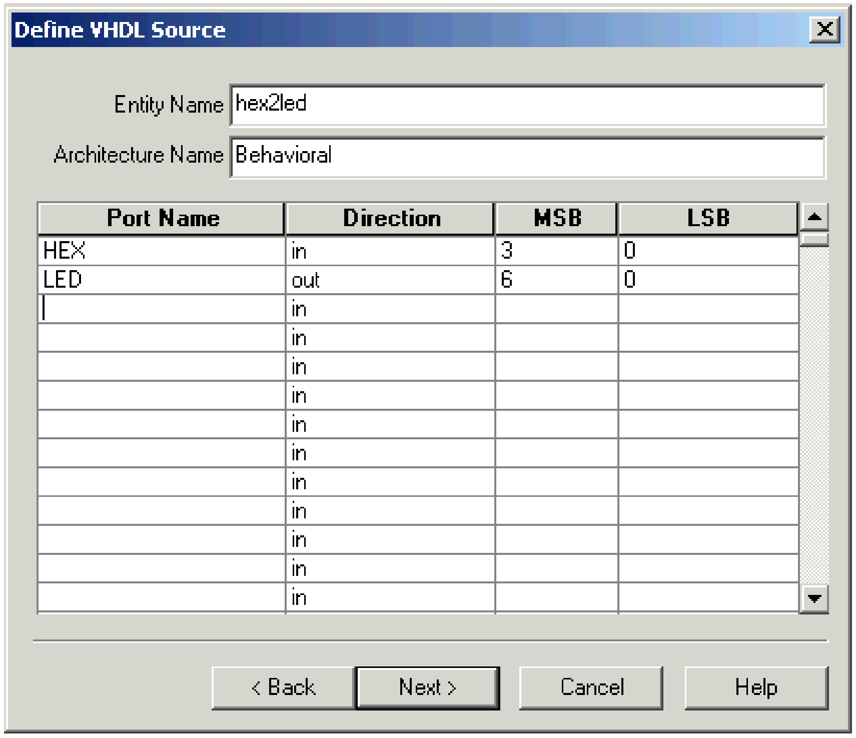
Enter the name and ports of the component in the New Source Wizard, and the wizard creates an HDL file that you can complete with the remainder of your code.

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

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

   The hex2led component has a 4-bit input port named HEX and a 7-bit output port named LED. First enter the port named HEX as follows:
5. Click in the Port Name field and type **HEX**.
6. Click in the Direction field and set the direction to **in**.
7. In the MSB field, enter **3**, and in the LSB field, enter **0**.
8. Repeat the previous steps for the LED(6:0) output bus. Be sure that the direction is set to **out**, MSB is set to **6** and LSB is set to **0**.

   The dialog box entries are displayed in Figure 3-31.



Define VHDL Source

Entity Name: hex2led

Architecture Name: Behavioral

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

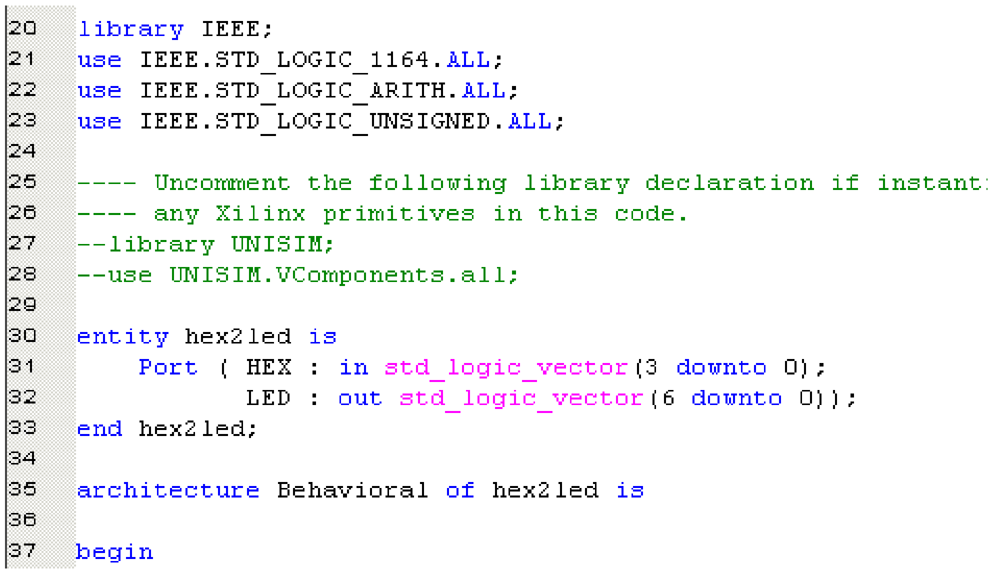
Figure 3-31: **New Source Wizard**

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

   A description of the module displays.

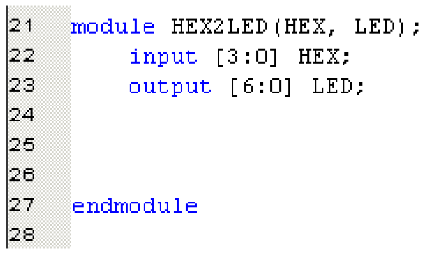
10. Select **Finish**.

    The HDL file opens in the ISE Text Editor.



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

---- Uncomment the following library declaration if instant
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of hex2led is

begin
```

Figure 3-32: **VHDL File**



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



endmodule
```

Figure 3-33: **Verilog File**

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

Using the Language Templates

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

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

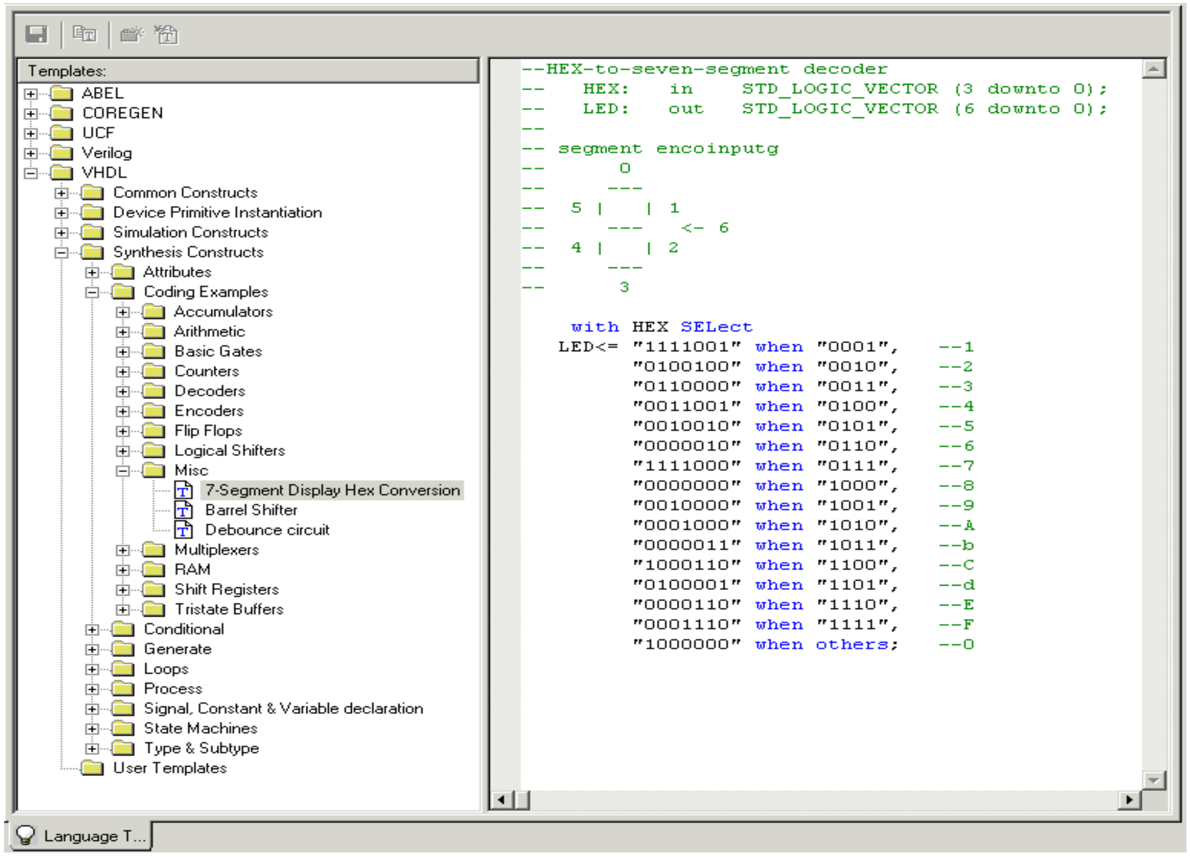
To invoke the Language Templates window and select a template for this tutorial:

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

   Each HDL language in the Language Template is divided into various sections according to the type of construct. To expand the view of any of these sections, click the **+** next to the topic. Click any of the listed templates to view the template contents in the right pane.

2. Click the **+** next to VHDL or Verilog to expland the template category for the HDL language you are using in this tutorial.
3. Click the **+** next to Synthesis Constructs to expand the category.
4. Click the **+** next to Coding Examples to expand the category.
5. Click the **+** next to Misc to expand the Miscellaneous category.
6. Click the template **7-Segment Display Hex Conversion** to preview the template contents in the right-hand pane.

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



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

    with HEX SELect
   LED<= "1111001" when "0001",   --1
         "0100100" when "0010",   --2
         "0110000" when "0011",   --3
         "0011001" when "0100",   --4
         "0010010" when "0101",   --5
         "0000010" when "0110",   --6
         "1111000" when "0111",   --7
         "0000000" when "1000",   --8
         "0010000" when "1001",   --9
         "0001000" when "1010",   --A
         "0000011" when "1011",   --b
         "1000110" when "1100",   --C
         "0100001" when "1101",   --d
         "0000110" when "1110",   --E
         "0001110" when "1111",   --F
         "1000000" when others;   --0
```

Figure 3-34: **Language Templates**

Adding the Language Template to Your File

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

To add the HEX2LED language template to your file:

1. In the Language Templates, click and drag the **7-Segment Display Hex Conversion** template into
   - the *hex2led.vhd* file under the architecture begin statement.

   or

   - the *hex2led.v* file under the module declaration.
2. Close the Language Templates window.

3. (Verilog only) After the input and output statements and before the hex2led converter that you just added, add the following line of code to the HDL file to allow an assignment.

   ```
   reg [6:0] LED;
   ```
4. (Verilog only) In the code, replace `<4-bit_hex_input>` with `HEX`, and replace `<7-seg_output>` with `LED`.

   You now have complete and functional HDL code.
5. Save the file by selecting **File** → **Save**.
6. Select *hex2led.vhd* or *hex2led.v* in the Sources in Project window.
7. In the Processes for Source window, click the **+** next to the Synthesize process to expand the hierarchy.
8. Double-click **Check Syntax** located in the Synthesize hierarchy.
   This launches the ISE Text Editor.
9. Close *hex2led.vhd* or *hex2led.v* and the Language Templates.

Creating the hex2led Symbol

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

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

You are now ready to place the hex2led symbol on the Watch schematic.

Placing the stmach, ten_cnt, clk_div_262k, DCM1, debounce, and hex2led Symbols

You can now place the stmach, ten_cnt, clk_div_262k, DCM1, debounce, and hex2led symbols on the Watch schematic (stopwatch.sch). If the schematic is already open in the Schematic Editor, ignore step 1.

1. In Project Navigator, double-click stop*watch.sch*. The schematic file opens in the Workspace.
2. Select **Add** → **Symbol** or click the **Add Symbol** icon from the Tools toolbar.

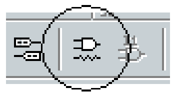


Figure 3-35: **Add Symbol Icon**

   This opens the Symbol Browser to the left of the Schematic Editor, which displays the libraries and their corresponding components to view the list of available library components.
3. View the list of available library components in the Symbol Browser.
4. Locate the project macros by selecting the project name in the Categories window.
5. Select the appropriate symbol, and add it to the Watch schematic in the approximate location as shown in Figure 3-36.

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

6. Save the schematic.

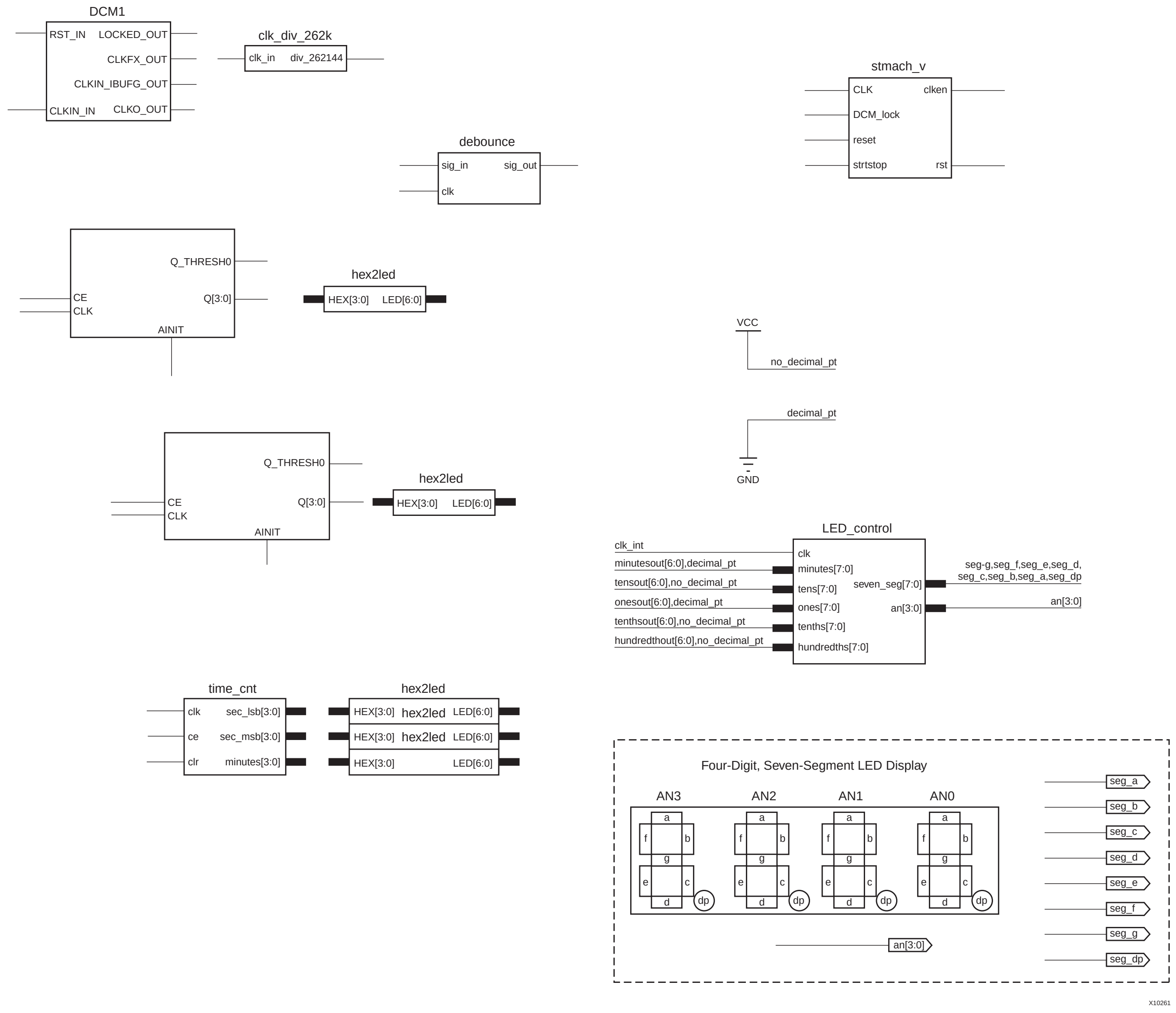


Figure 3-36: **Placing Design Macros**

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 clk_div_262k macro, which is a schematic-based user-created macro, and examine its components.

To push down into clk_div_262k from the top level stopwatch schematic:

1. Click clk_div_262k symbol in the schematic and select the **Hierarchy Push** icon. You can also right-click the macro and select **Push into Symbol**.

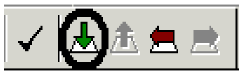


Figure 3-37: **Hierarchy Push Icon**

In the clk_div_262k schematic, you see a series of shift registers (SRL16). This macro illustrates how SRL16s can be used to create a clock divider of any power of 2 without using an excess number of flip-flops.

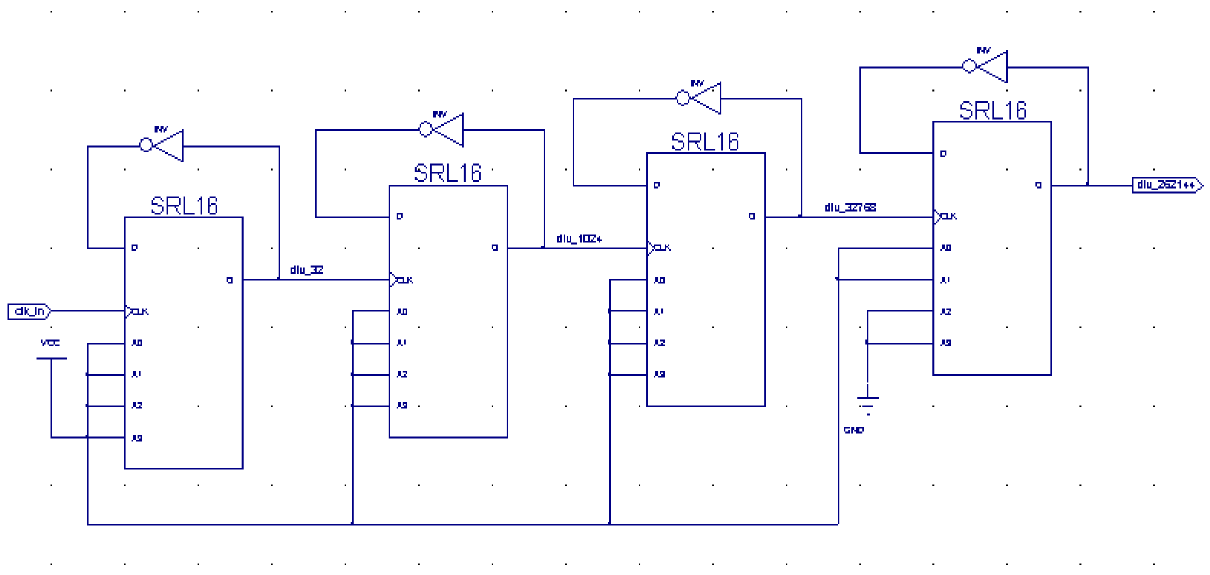


Figure 3-38: **clk_div_262k Schematic**

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**.

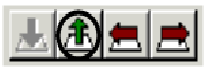


Figure 3-39: **Hierarchy Pop Icon**

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 HDL, the I/O markers are replaced with the appropriate pads and buffers.

Adding Input Pins

Next, add three input pins to the stopwatch schematic: CLK, RESET and STRTSTOP.

To add these components:

1. Draw a hanging wire to the two inputs of DCM1.
2. Draw a hanging wire to the sig_in input of the debounce symbol.

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

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

   The net name is now attached to the cursor.

3. Place the name on the leftmost end of the net as illustrated in Figure 3-40.
4. Repeat Steps 1 through 3 for the STRTSTOP and CLK pins.

   Once all of the nets have been labeled, add the I/O marker.

5. Select **Add** → **I/O Marker**.
6. In the Add I/O Marker Options dialog box, select **Add an input Marker** for an input signal direction.
7. Click and drag a box around the three labeled nets to place an input signal around each net name.

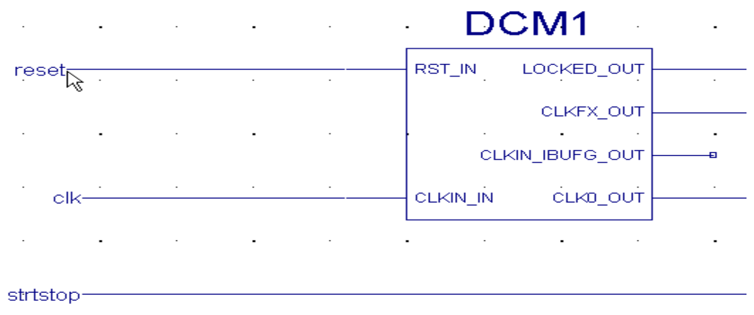


Figure 3-40: **Labeled Nets with I/O Markers**

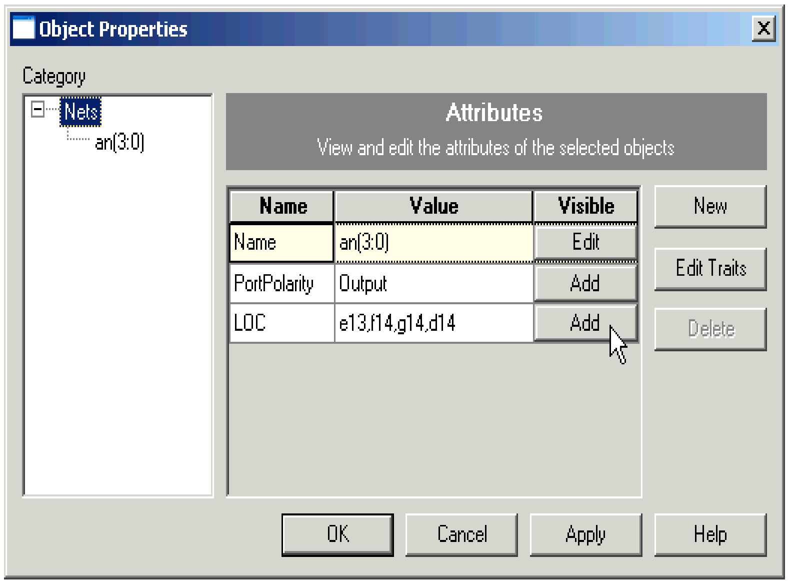
Assigning Pin Locations

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

For this tutorial, the inputs and outputs will be locked to specific pins in order to place and download the design to the Spartan-3 demo board. Because the tutorial Watch design is simple and timing is not critical, the example pin assignments will not adversely affect the ability of PAR to place and route the design.

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

1. Right-click on the an(3:0) net, and select **Object Properties** from the right-click menu.
2. Click the **New** button under Instance Attributes to add a new property.
3. Enter **LOC** for the Attribute Name and **e13,f14,g14,d14** for the Attribute Value.
4. Click **OK** to return to the Object Properties dialog box.



Name	Value	Visible
Name	an(3:0)	Edit
PortPolarity	Output	Add
LOC	e13,f14,g14,d14	Add

Figure 3-41: **Assigning Pin Locations**

5. To make the LOC attribute visible, select the **Add** button 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 an(3:0) bus.

   The above procedure constrains an(3) to pin e13, an(2) to pin f14 and so forth. The remaining pin location constraints will be added in "Using the Constraints Editor" and "Using the Pin-out Area Constraints Editor (PACE)" of Chapter 5, "Design Implementation"

Note: To turn off the Location constraint without deleting it, select the loc attribute, 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.

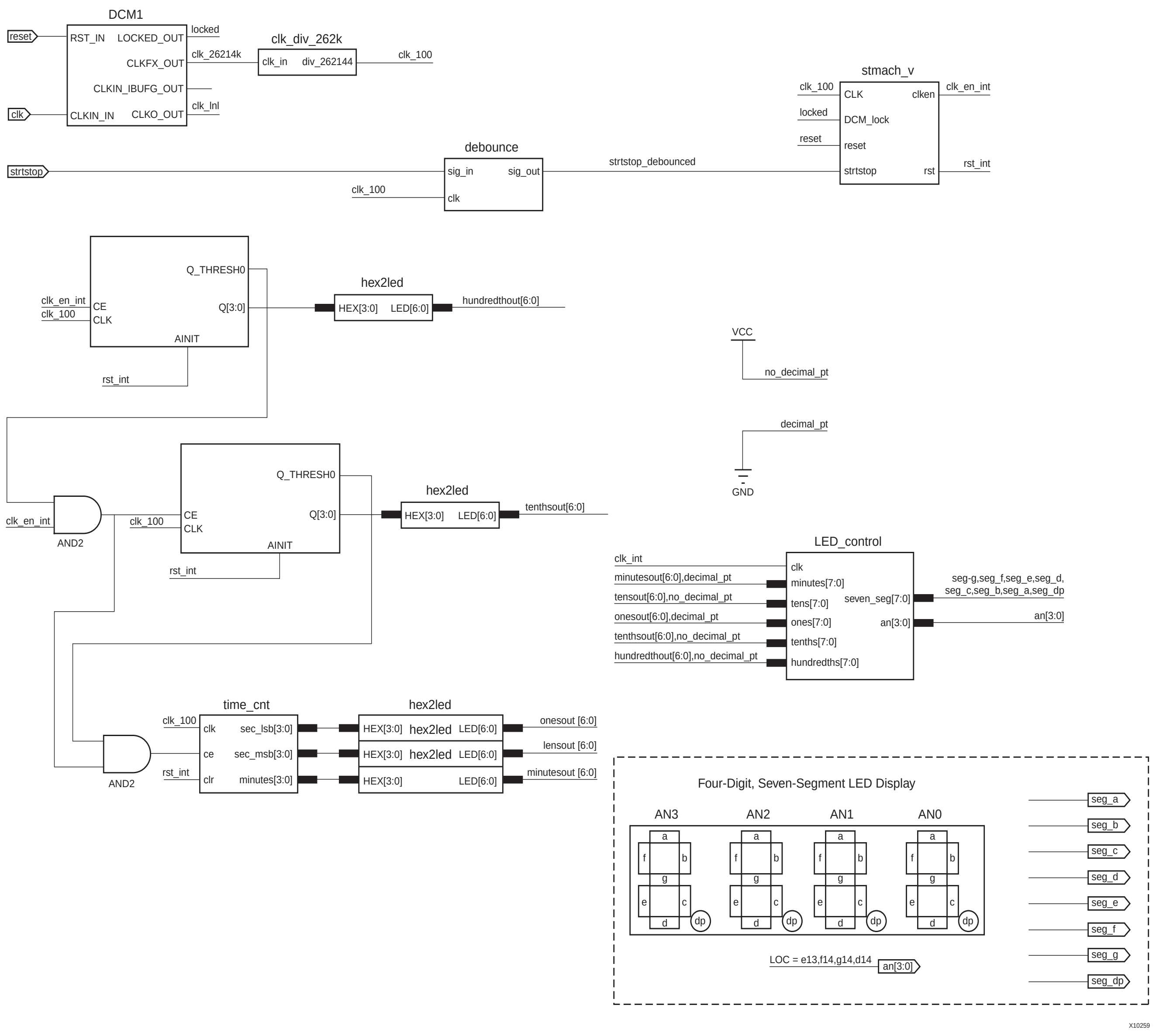


Figure 3-42: **Completed Watch Schematic**

To complete the schematic diagram:

1. Draw a wire between the CLKFX_OUT pin of DCM1 and the clk_in pin of the clk_div_262k macro (see "Drawing Wires."). Label the wire **clk_clk_26214k** (see "Adding Net Names").

2. Draw a hanging wire to the LOCKED_OUT and clk_out pins of DCM1 and name the wires **locked** and **clk_int** respectively. To terminate a hanging wire, double-click it. See "Drawing Wires" and "Adding Net Names".
3. Draw a hanging wire to the clk input of both the debounce and stmach_v macros wires (see "Drawing Wires") and 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_int and several other signals.

4. Draw wires between the sig_out pin of the debounce component and the strtstop pin of the STMACH_v macro (see "Drawing Wires"). Label the net **strtstop_debounced**.
5. Add hanging wires to the DCM_lock pin and the reset pin of the stmach_v macro. Name them **locked** and **reset**, respectively.
6. Place an AND2 component to the left of the lower ten_cnt macro. See "Adding Components to time_cnt."
7. Draw a wire to connect the output of the AND2 with the CE pin of the TEN_CNT macro. See "Drawing Wires."
8. Draw a wire to connect the Q_THRES0 pin of the upper TEN_CNT macro to one of the inputs to the AND2. See "Drawing Wires."
9. Draw a hanging wire to the clken output of the stmach_v component. Label the wire **clk_en_int**.
10. Draw a hanging wire to the ce pin of the upper TEN_CNT macro and another to the remaining input of the AND2 component. Name both wires **clk_en_int**.
11. Place a second AND2 component to the left of the TIME_CNT macro. See "Adding Components to time_cnt."
12. Draw a wire to connect the output of the AND2 with the CE pin of the TIME_CNT macro. See "Drawing Wires."
13. Draw a wire to connect the Q_THRES0 pin of the lower TEN_CNT macro to one of the inputs to the second AND2 component. See "Drawing Wires."
14. Draw a wire from the other input of the second AND2 component to the wire connected to the output of the first AND2 component.
15. Draw a hanging wire from the output of the clk_div_262k component and label this net **clk_100**.
16. Draw a hanging wire from the clk pin of the time_cnt macro and the clk pins of the two ten_cnt macros. See "Drawing Wires." Name the three newly added nets **clk_100**.
17. Draw hanging wires from the RST output pin of the STMACH macro, to the AINIT pins of the ten_cnt macros and the clr pin of the time_cnt macro. See "Drawing Wires." Label all four wires **RST_INT**.
18. Draw wires from the bus outputs of the ten_cnt and time_cnt macros to the inputs of the adjacent hex2led macros. See "Drawing Wires." Notice how the wire is automatically converted to a bus.
19. Draw hanging buses from each of the hex2led macro outputs.
20. Name the hex2led outputs nets as follows from top to bottom; hundredthsout(6:0), thenthsout(6:0), onesout(6:0), tensout(6:0), minutesout(6:0).

The schematic is now complete.

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

Chapter 4

Behavioral Simulation

This chapter contains the following sections.

Overview of Behavioral Simulation Flow

Xilinx® ISE™ provides an integrated flow with the ModelTech ModelSim simulator and the Xilinx ISE Simulator that allows simulations to be run from the Xilinx Project Navigator graphical user interface (GUI). The examples in this tutorial demonstrates how to use the integrated flow. Whether you use the ModelSim simulator or the ISE Simulator with this tutorial, the end result is the same.

For additional information about simulation and for a list of the other supported simulators, refer to Chapter 6 of the *Synthesis and Verification Guide*. 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/xilinx7/.

ModelSim Setup

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

ModelSim PE and SE

ModelSim PE and ModelSim SE are the full versions of the ModelSim product that can be purchased directly from ModelTech. In order to simulate with the ISE 7 libraries, use ModelSim 5.8 or later. Older versions may work but are not supported.

Note: For more information about purchasing ModelSim PE or SE version 5.8 or later, contact ModelTech.

ModelSim Xilinx Edition

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

MXE III 6.0a must be used with the ISE 7.1i software, as this is the only version for which the latest 7.1i libraries have been compiled.

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

http://www.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=MXE+III

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

http://www.xilinx.com/xlnx/xil_tt_product.jsp?sProduct=MXE+III

ISE Simulator Setup

ISE Simulator is automatically installed and setup with the ISE 7.1i installer.

Note: The ISE Simulator is available on Windows 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 tutorial design entry by following Chapter 2, "HDL-Based Design," or Chapter 3, "Schematic-Based Design." After you have completed one of these chapters, your design includes the required design files and is ready for simulation.

- **Test Bench File**

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

  Note: The ISE Simulator's Waveform Editor is available on Windows platforms only.

- **Xilinx Simulation Libraries**

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

Xilinx Simulation Libraries

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

Note: For a detailed description of each library, refer to Chapter 6 of the *Synthesis and Verification Design Guide*. This Guide is available with the collection of software manuals and is accessible from ISE by selecting **Help** → **Online Documentation**, or from the web at http://www.xilinx.com/support/sw_manuals/xilinx7/.

Updating the Xilinx Simulation Libraries

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

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

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

ModelSim PE or SE

If you are using ModelSim PE or SE, you will need to compile the simulation libraries with the updated models. Refer to Chapter 6 of the *Synthesis and Verification Design Guide*. 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/xilinx7/.

ModelSim Xilinx Edition III

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

Xilinx ISE Simulator

If you are using ISE Simulator, all the simulation libraries are precompiled and setup automatically. Each time a new ISE service pack is installed, the ISE Simulator will get 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 should appear 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 will be used.

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

ModelSim PE or SE

If you are using ModelSim PE or SE, you should have gone through the *Synthesis and Verification Design Guide* and used COMPXLIB to compile the libraries. During that process, 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.

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

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. You will see that all of the Xilinx simulation libraries are already mapped to the proper location.

ISE Simulator

The *modelsim.ini* 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 have the option of either adding a tutorial test bench file provided with this tutorial, or creating your own test bench file and adding 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 have been provided with this tutorial.

VHDL Design

After downloading the file to your project directory, add the tutorial VHDL test bench to the project in Project Navigator as follows:

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

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

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

Verilog Design

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

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

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

ISE recognizes the top-level design file associated with the test 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.

Note: 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.

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 in Project window, select the test bench file (*stopwatch_tb*).
2. In the Processes for Source window, click the **+** beside ModelSim Simulator to expand the process hierarchy.

Note: If the ModelSim Simulator processes do not appear, it means that either ModelSim is not 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 to 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.
```

The following simulation processes are available:

- **Simulate Behavioral Model**

  This process will start the design simulation.

- **Generate Expected Simulation Results**

  This process is available only if you have a testbench waveform file from the ISE Simulator's Test Bench Waveform Editor. If you double-click this process, ModelSim will run in the background to generate expected results and display them in the ISE Simulator's Test Bench Waveform Editor. See "Creating a Test Bench Waveform Using the Waveform Editor."

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

  Simulates the netlist after the Translate (NGDBuild) implementation stage.

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

  Simulates the netlist after the Map implementation stage.

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

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

Specifying Simulation Properties

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:

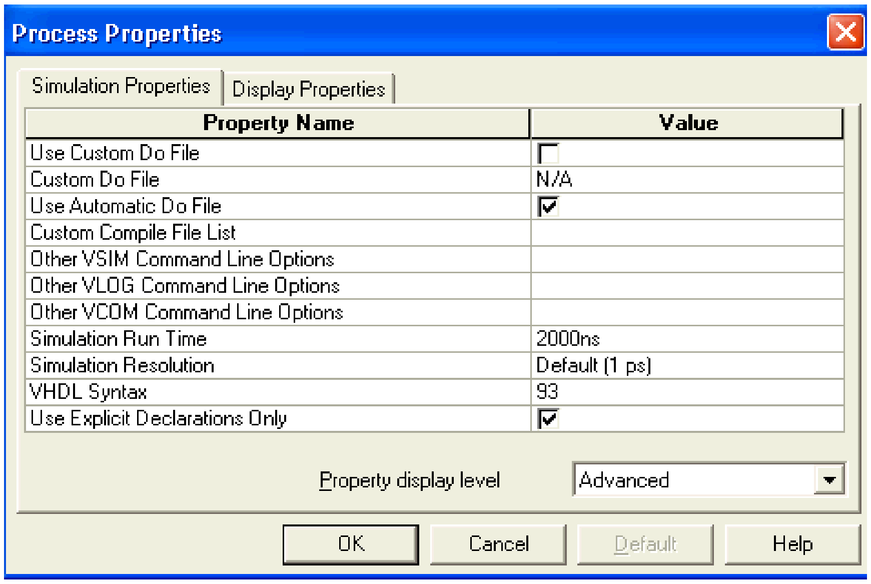
1. In the Sources in Project window, select the test bench file (*stopwatch_tb*).
2. Click the **+** sign next to **ModelSim Simulator** to expand the hierarchy in the Processes For Source window.
3. Right-click the **Simulate Behavioral Model** process.
4. Select **Properties**.

The Process Properties dialog box (Figure 4-1) displays.

5. In the Process Properties dialog box, set the Property display level to **Advanced**. This setting is right above the Help button.

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

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


Process Properties

Simulation Properties | Display Properties

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

Property display level: Advanced

Figure 4-1: **Behavioral Simulation Process Properties**

7. Click **OK** to continue.

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

Performing Simulation

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

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 in the Signal/Object window, select **Add** → **Wave** → **Selected Signals**.

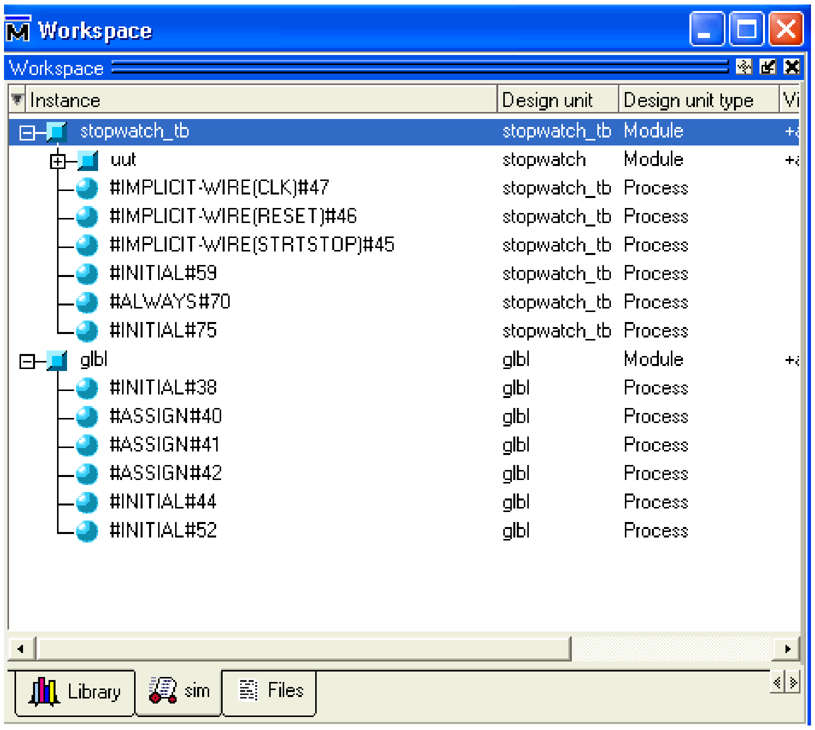
The following procedure explains how to add additional signals in the design hierarchy. In this tutorial, you will be adding the DCM signals to the waveform.

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 4-2: **Undock icon**

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.



Instance	Design unit	Design unit type	Vi
stopwatch_tb	stopwatch_tb	Module	+
uut	stopwatch	Module	+
#IMPLICIT-WIRE(CLK)#47	stopwatch_tb	Process	
#IMPLICIT-WIRE(RESET)#46	stopwatch_tb	Process	
#IMPLICIT-WIRE(STRTSTOP)#45	stopwatch_tb	Process	
#INITIAL#59	stopwatch_tb	Process	
#ALWAYS#70	stopwatch_tb	Process	
#INITIAL#75	stopwatch_tb	Process	
glbl	glbl	Module	+
#INITIAL#38	glbl	Process	
#ASSIGN#40	glbl	Process	
#ASSIGN#41	glbl	Process	
#ASSIGN#42	glbl	Process	
#INITIAL#44	glbl	Process	
#INITIAL#52	glbl	Process	

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

2. Select **DCM1** in the Structure / Instance window.

   Notice that 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:
   - RST_IN
   - CLKFX_OUT
   - CLK0_OUT
   - LOCKED_OUT

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

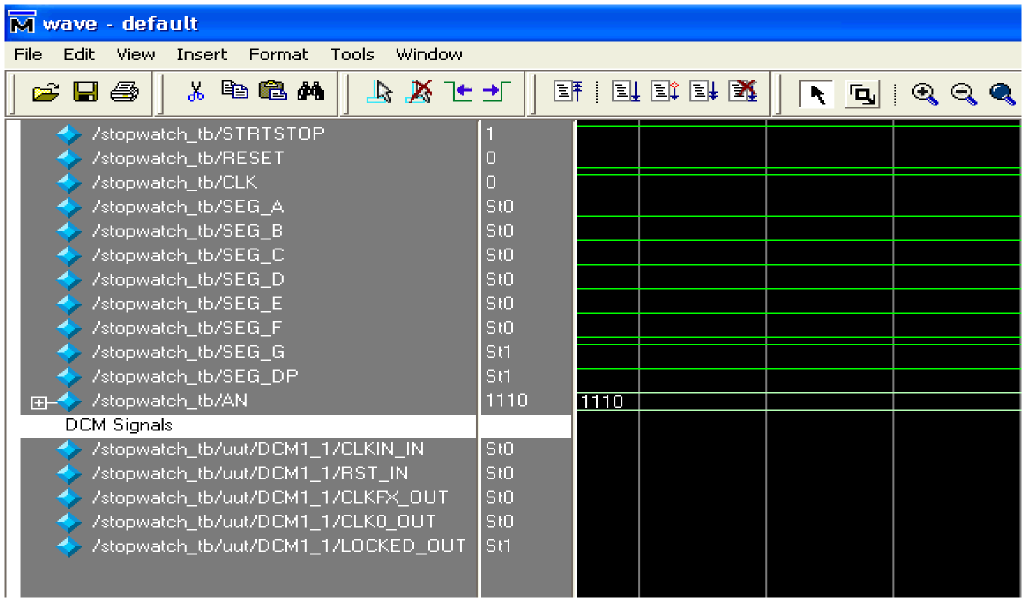
5. Right-click in the Signal / Object window and select **Add to Wave** → **Selected Signals**.

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. 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_IN signal.

After adding the DCM Signals divider, the waveform should look like Figure 4-4.



```
/stopwatch_tb/STRTSTOP                 1
/stopwatch_tb/RESET                    0
/stopwatch_tb/CLK                      0
/stopwatch_tb/SEG_A                    St0
/stopwatch_tb/SEG_B                    St0
/stopwatch_tb/SEG_C                    St0
/stopwatch_tb/SEG_D                    St0
/stopwatch_tb/SEG_E                    St0
/stopwatch_tb/SEG_F                    St0
/stopwatch_tb/SEG_G                    St1
/stopwatch_tb/SEG_DP                   St1
/stopwatch_tb/AN                       1110     1110
DCM Signals
/stopwatch_tb/uut/DCM1_1/CLKIN_IN      St0
/stopwatch_tb/uut/DCM1_1/RST_IN        St0
/stopwatch_tb/uut/DCM1_1/CLKFX_OUT     St0
/stopwatch_tb/uut/DCM1_1/CLK0_OUT      St0
/stopwatch_tb/uut/DCM1_1/LOCKED_OUT    St1
```

Figure 4-4: **Waveform After Adding DCM Signals Divider**

Notice that 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 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 rerun simulation in ModelSim:

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

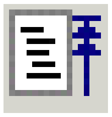


Figure 4-5: **Restart Simulation Icon**

The Restart dialog box opens.

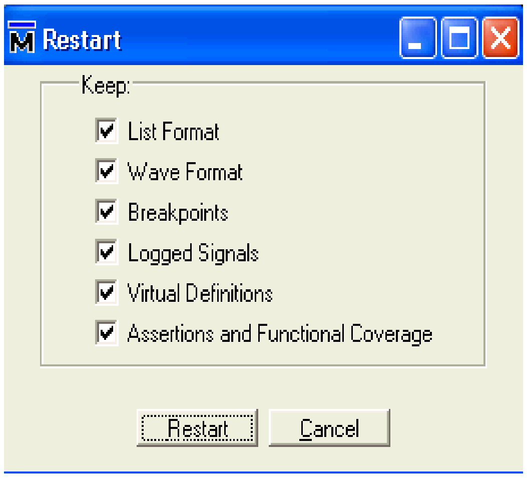
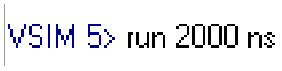


Figure 4-6: **Restart Dialog Box**

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



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

Figure 4-7: **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

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 will be 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 **Insert** → **Cursor**.
2. Click and drag this cursor to the first rising edge transition on the CLK0_OUT signal after the LOCKED_OUT signal has gone high.
3. Click the **Find Next Transition** icon twice to move the cursor to the next rising edge on the CLK0_OUT signal.

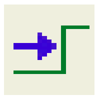


Figure 4-8: **Find Next Transition Icon**

4. 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 testbench, which in turn should be the DCM CLK0 output.

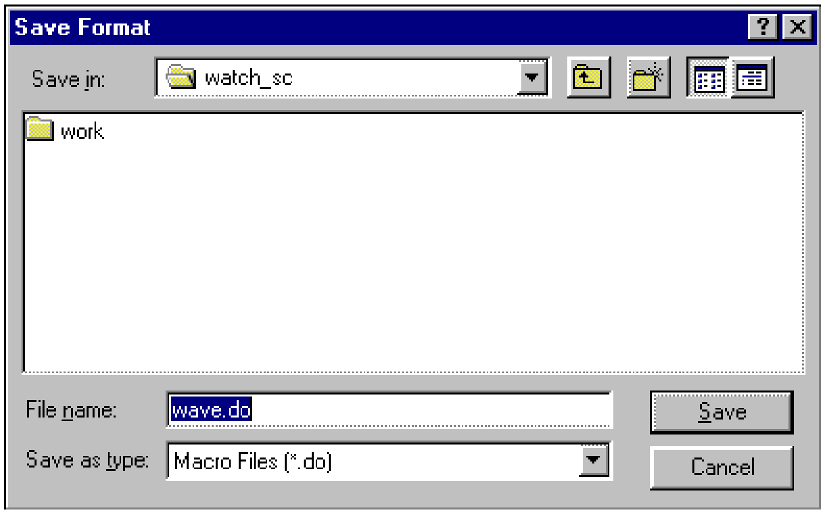
5. Measure CLKFX_OUT using the same steps as above. The measurement should read 38462 ps. This comes out to approximately 26 Mhz.

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.

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

1. In the Save Format dialog box, rename the filename from the default *wave.do* to **dcm_signal.do**.



Save in: watch_sc

File name: wave.do

Save as type: Macro Files (*.do)

Figure 4-9: **Save Format Dialog Box**

2. Click **Save**.

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

Your behavioral simulation is complete and you are ready to implement the design by following 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, in the Sources in Project window, right-click on the device line (xc3s200-4ft256) and select **Properties**. 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 in Project window, select the test bench file (*stopwatch_tb*).
2. Click the **+** beside Xilinx ISE Simulator to expand the process hierarchy.

The following simulation processes are available:

- **Check Syntax**

  This process with check for syntax errors in the testbench.

- **Simulate Behavioral Model**

  This process will start the design simulation.

- **Generate Expected Simulation Results**

  This process is available only if you have a testbench waveform file from ISE Simulator's Test Bench Waveform Editor . If you run this process, the ISE Simulator will run in the background to generate expected results and display them in the Waveform Editor. See "Creating a Test Bench Waveform Using the Waveform Editor."

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

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

Specifying Simulation Properties

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

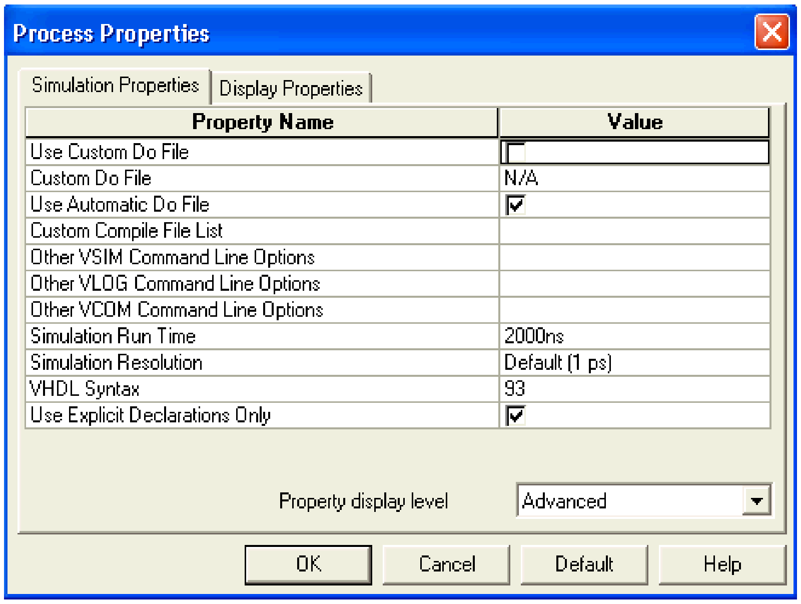
ISE allows you to set several ISE Simulator properties in addition to the simulation netlist properties. To see the behavioral simulation properties and to modify the properties for this tutorial:

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

   The Process Properties dialog box (Figure 4-10) displays.
5. In the Process Properties dialog box, set the Property display level to **Advanced**. This setting is right above the Help button.

   This global setting enables you to now see all available properties.
6. Change the Simulation Run Time to **2000 ns**.

7. Click **OK** to continue.



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

Property display level: Advanced

Figure 4-10: **Behavioral Simulation Process Properties**

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

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.

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. 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.

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

Figure 4-11 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.

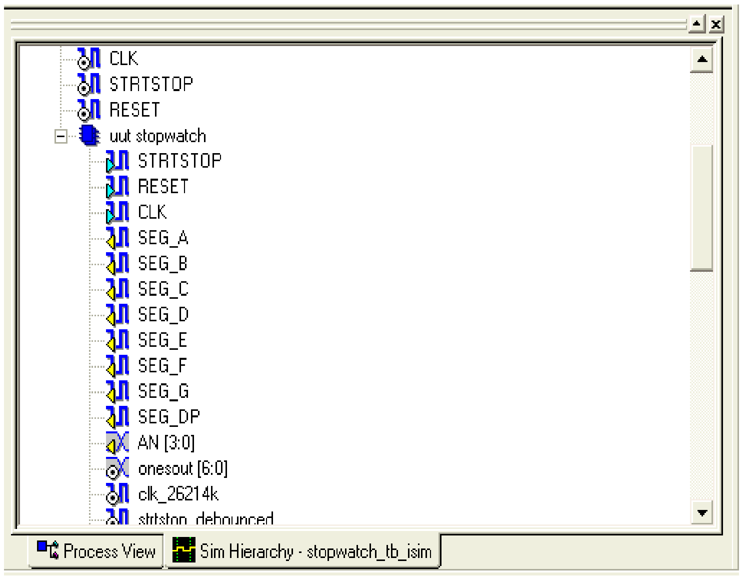


Figure 4-11: **Sim Hierarchy Window - Verilog flow**

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

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

6. Drag all the selected signals to the 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 will only record data for the signals that have been added to the waveform window while the simulation is running. Therefore, when new signals are added to the waveform window, you need to rerun the simulation for the desired amount of time.

Rerunning Simulation

To rerun the simulation in ISE Simulation:

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

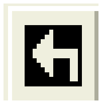


Figure 4-12: **ISE Simulator Restart Simulation Icon**

2. At the ISE Simulator command prompt, enter **run 2000 ns** and click **Enter**.

The simulation will run for 2000 ns. The waveforms for the DCM should now be 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 will be analyzed only after the LOCKED_OUT signal has gone high.

ISE Simulator has the capability to 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.

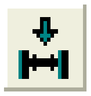


Figure 4-13: **Measure Marker Icon**

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 testbench, 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 and you are ready to implement the design by following Chapter 5, “Design Implementation.”

Creating a Test Bench Waveform Using the Waveform Editor

This section demonstrates how to use the Waveform Editor, which is a test bench creation tool in ISE 7. 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.

Note: The ISE Simulator Waveform Editor is available on Windows platforms only.

Creating a Test Bench Waveform Source

In this tutorial, create the testbench 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:

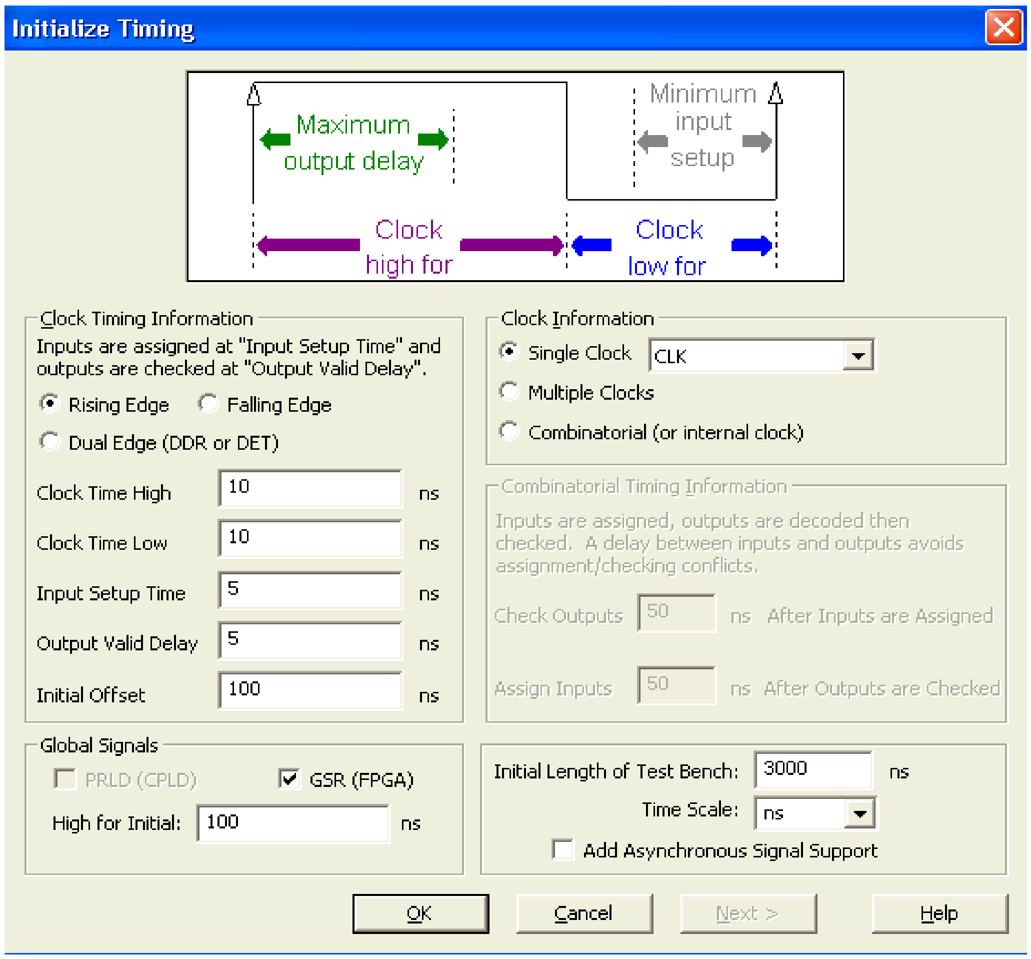
1. Select *time_cnt* in the Sources in Project window.
2. Select **Project** → **New Source**.
3. In the New dialog box, select **Test Bench Waveform** as the source type.
4. Type the name **time_cnt_tb**.
5. Click **Next**.

Note: In the Select dialog box, the *time_cnt* file is the default source file because it is selected in the Sources in Project window (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.

1. 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**
2. Select the **GSR (FPGA)** from the Global Signals section.
3. Change the Initial Length of Test Bench to **3000**.
4. Click **OK**.



Clock Timing Information
Inputs are assigned at "Input Setup Time" and outputs are checked at "Output Valid Delay".
[✓ Rising Edge] Falling Edge
Dual Edge (DDR or DET)
Clock Time High: 10 ns
Clock Time Low: 10 ns
Input Setup Time: 5 ns
Output Valid Delay: 5 ns
Initial Offset: 100 ns

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

Combinatorial Timing Information
Inputs are assigned, outputs are decoded then checked. A delay between inputs and outputs avoids assignment/checking conflicts.
Check Outputs 50 ns After Inputs are Assigned
Assign Inputs 50 ns After Outputs are Checked

Global Signals
PRLD (CPLD) [✓ GSR (FPGA)]
High for Initial: 100 ns

Initial Length of Test Bench: 3000 ns
Time Scale: ns
Add Asynchronous Signal Support

Figure 4-14: **Waveform Editor - Initialize Timing Dialog Box**

The ISE Simulator's Waveform Editor window 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 respectively.

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.

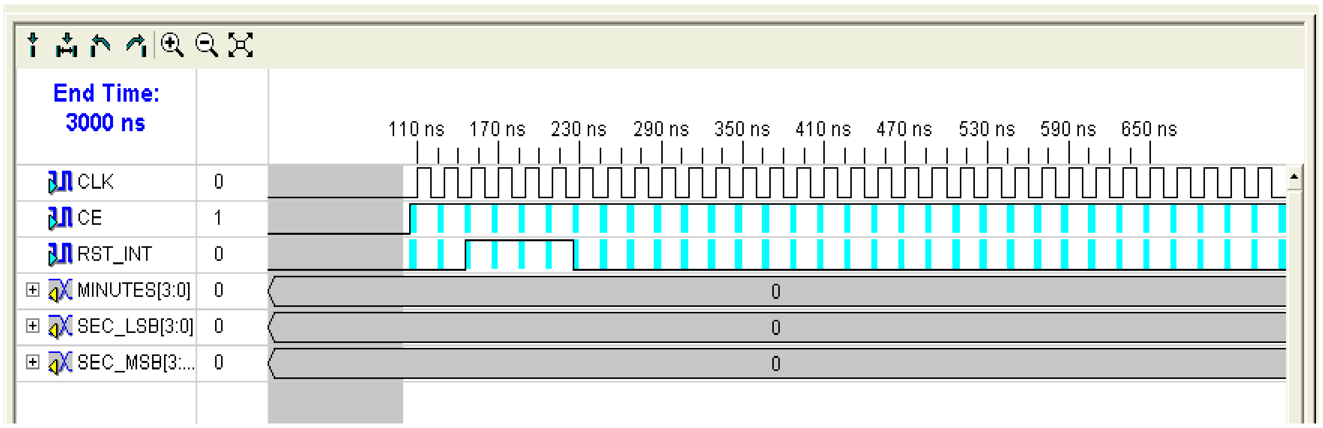


Figure 4-15: **Applying Stimulus in the Waveform Editor Window**

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. Double-click the **Generate Expected Simulation Results** process.

This runs the selected simulator in the background and displays the expected output values in the Waveform Editor.

Chapter 5

Design Implementation

This chapter contains the following sections.

Overview of Design Implementation

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

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

This chapter demonstrates the ISE Implementation flow. The front-end design has already been compiled in an EDA interface tool. For details about compiling the design, see Chapter 2, "HDL-Based Design" 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. There are three inputs to the system: CLK, RESET and SRTSTP. This system generates three seven-bit outputs for output to three seven-segment LED displays.

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. You may not have a User Constraint File (UCF) to which you will add design constraints in this chapter.

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

1. 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® website, 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/tutorials7.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
`stopwatch.edn`, `stopwatch.edf` or `stopwatch.ngc`	Input netlist file (EDIF)
`ten_cnt.edn`	Counter netlist file (EDIF)
`stopwatch.ucf`	User Constraints File

3. Open ISE.
   a. On a workstation, enter `ise &`
   b. On a PC, select **Start** → **Programs** → **Xilinx ISE 7** → **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 Module Type.
   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:
      - **Spartan3** for the Device Family
      - **xc3s200** for the Device
      - **-4** for the Speed Grade, **ft256** for the Package
   i. Click **Next**.
   j. Click **Finish**.

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

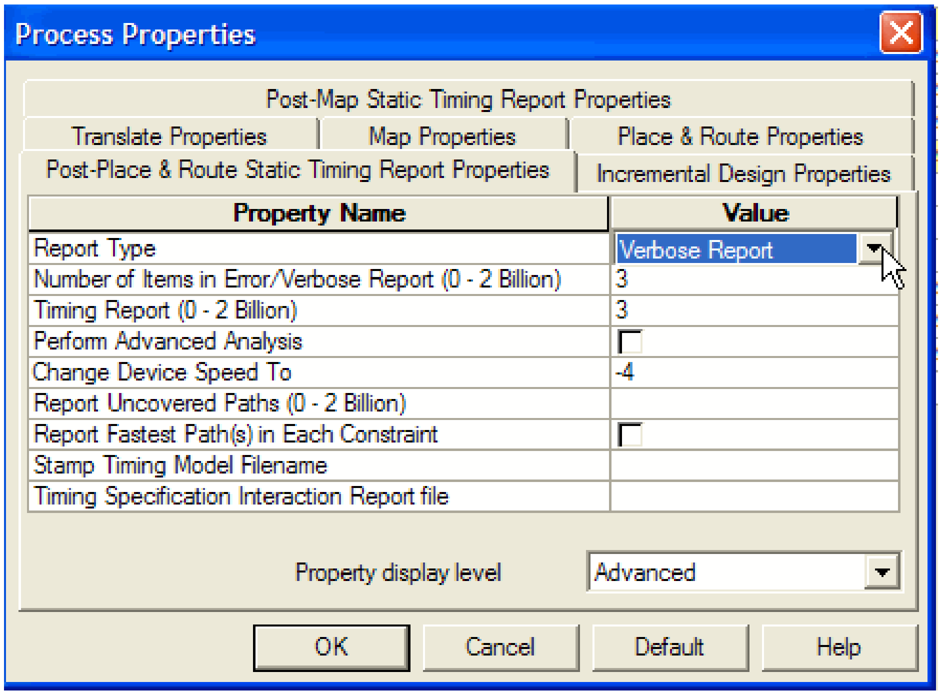
Specifying Options

This section describes how to set some 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 in the Project window, select the **stopwatch** top-level file.
2. In the Processes for Source window, 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 tabs, each contains properties for a different aspect of design implementation.


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

Property Name	Value
Report Type	Verbose Report
Number of Items in Error/Verbose Report (0 - 2 Billion)	3
Timing Report (0 - 2 Billion)	3
Perform Advanced Analysis	☐
Change Device Speed To	-4
Report Uncovered Paths (0 - 2 Billion)	
Report Fastest Path(s) in Each Constraint	☐
Stamp Timing Model Filename	
Timing Specification Interaction Report file	

Property display level: Advanced

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

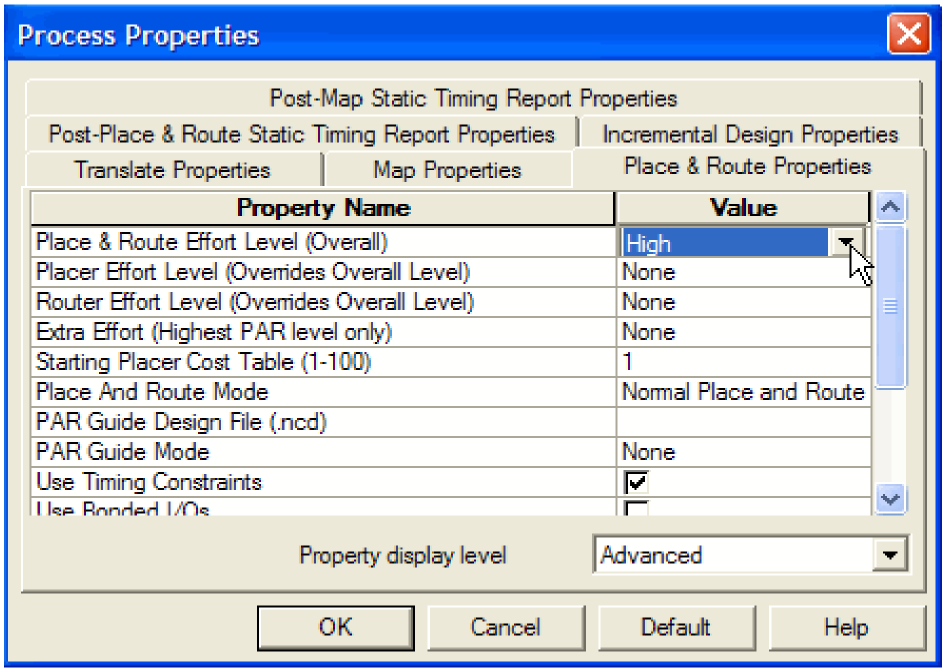
4. Ensure that you have set the Property display level to **Advanced**. This setting is right above the Help button.

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

   This report will be generated after Place and Route is completed.
9. Click the **Place & Route Properties** tab.
10. Change the Place & Route Effort Level (overall) to **High**.

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


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

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

Property display level: Advanced

Figure 5-2: **Place & Route Properties**

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

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

To launch the Constraints Editor:

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

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

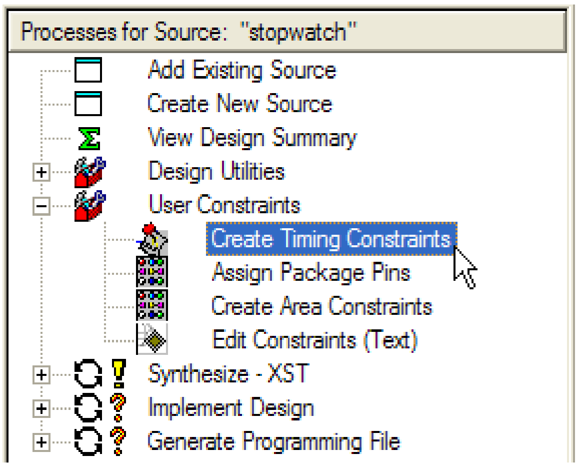


Figure 5-3: **Edit Implementation Constraints**

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 program NGDBuild performs the following functions:

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

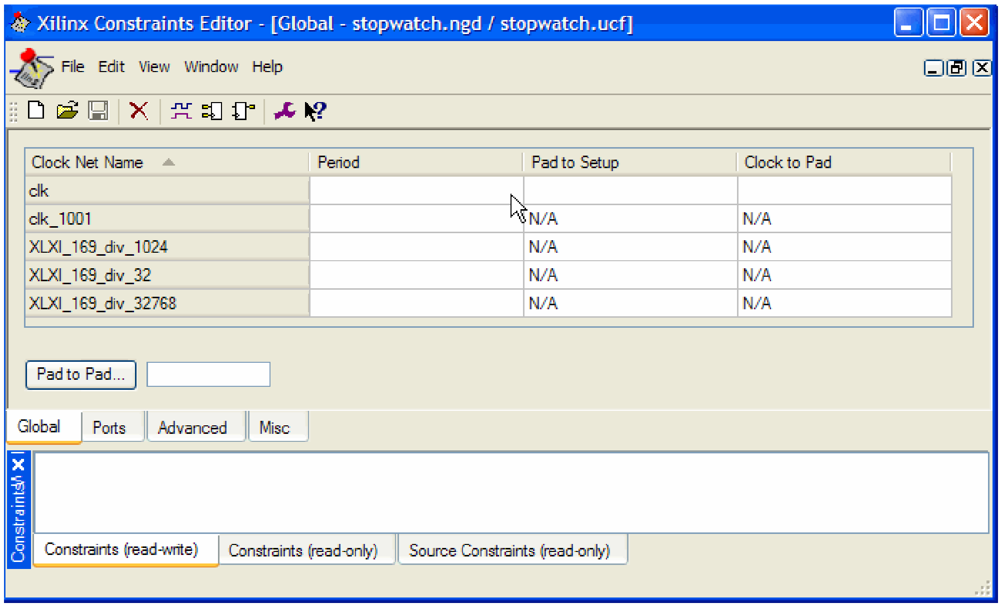
- **Corresponding UCF (User Constraint File)**

  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* file and *stopwatch.ucf* files are automatically read into the Constraints Editor.

The Global tab appears in the foreground of the Constraints Editor window. This window automatically displays all the clock nets in your design, and enables you to define the associated period, pad to setup, and clock to pad values. Note that many of the internal names will vary depending on the design flow and synthesis tool used.

In this section, a PERIOD and TIMEGRP OFFSET IN constraint are going to be written in the UCF and used during implementation.



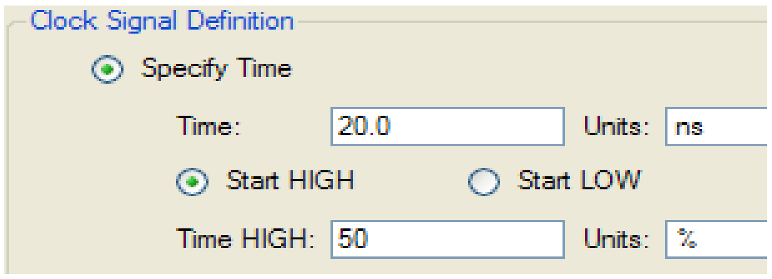
Clock Net Name	Period	Pad to Setup	Clock to Pad
clk			
clk_1001		N/A	N/A
XLXI_169_div_1024		N/A	N/A
XLXI_169_div_32		N/A	N/A
XLXI_169_div_32768		N/A	N/A

Figure 5-4: **Constraints Editor - Global Tab**

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 **Specific Time** is selected.

   It enables you to define an explicit period for the clock.
3. Enter a value of **20.0** in the Time text box.
4. Verify that **ns** is selected from the Units pull-down list.


Clock Signal Definition

[✓ Specify Time]

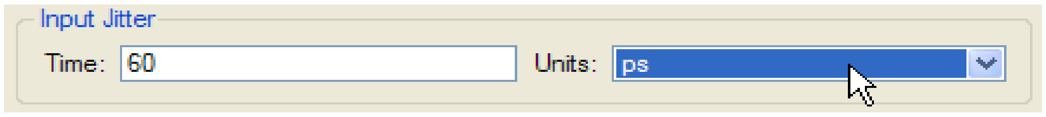
Time: 20.0 Units: ns

[✓ Start HIGH] Start LOW

Time HIGH: 50 Units: %

Figure 5-5: **PERIOD Constraint Values**

5. For the Input Jitter section, enter a value of **60** in the Time text box.
6. Verify that **ps** is selected from the Units pull-down list.



Input Jitter
Time: 60
Units: ps

Figure 5-6: **INPUT JITTER Constraint Value**

7. Click **OK**.

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

Note: When you double-click in a cell, a dialog box opens and enables you to specify a constraint.

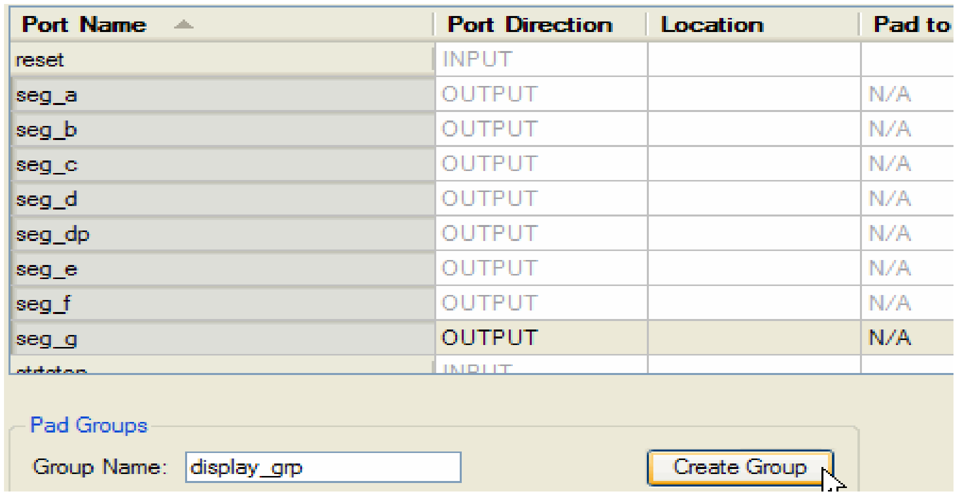
8. Select the **Ports tab** from the Constraints Editor window.

   The left hand side displays a listing of all the current ports.

9. Select **seg_a** in the Port Name Column.
10. Hold the **Shift** key and select **seg_g**.

    This selects the elements for creating a grouped offset.

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

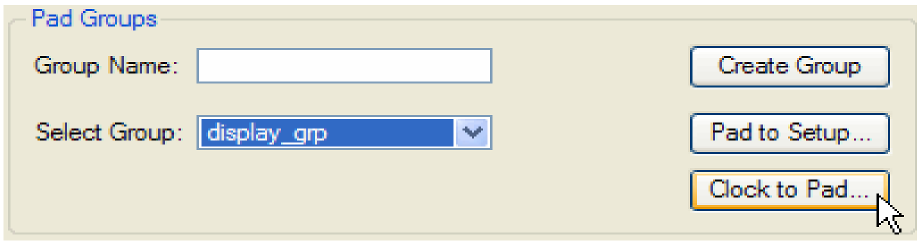


Port Name	Port Direction	Location	Pad to
reset	INPUT		
seg_a	OUTPUT		N/A
seg_b	OUTPUT		N/A
seg_c	OUTPUT		N/A
seg_d	OUTPUT		N/A
seg_dp	OUTPUT		N/A
seg_e	OUTPUT		N/A
seg_f	OUTPUT		N/A
seg_g	OUTPUT		N/A

Pad Groups
Group Name: display_grp
Create Group

Figure 5-7: **Selected Elements of a Grouped OFFSET**

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

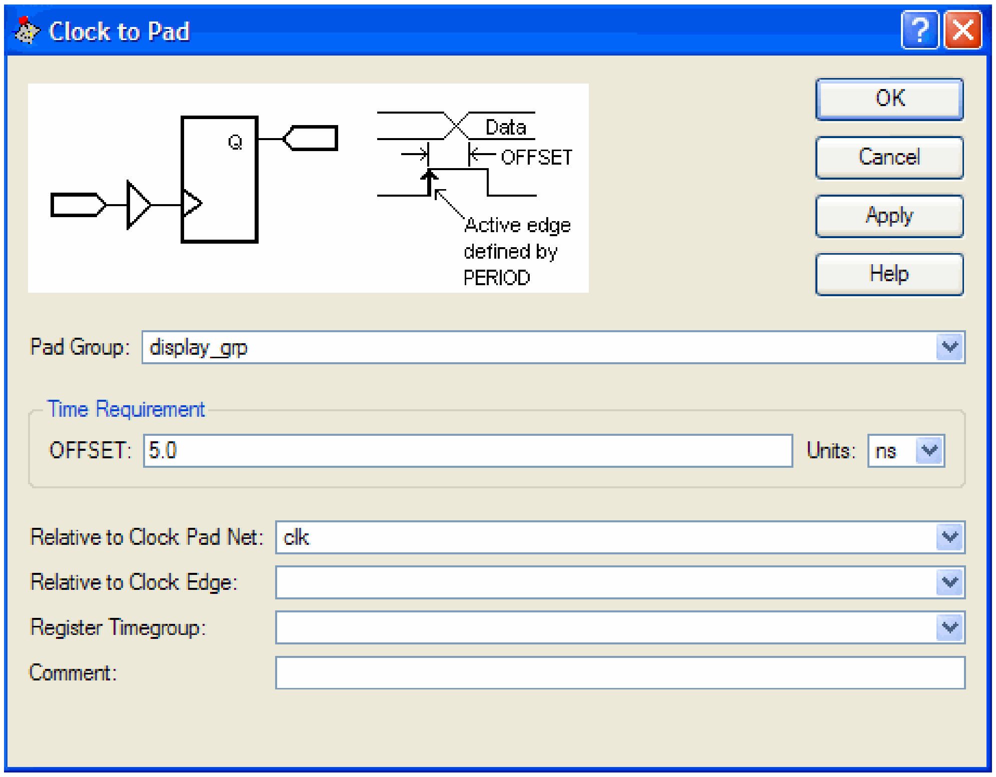


Pad Groups
Group Name:
Select Group: display_grp
Create Group
Pad to Setup...
Clock to Pad...

Figure 5-8: **Selecting the Group that was created to use in an OFFSET**

The Clock to Pad dialog box opens.

14. Enter **5.0 ns** for the Timing Requirement.
15. Select **clk** in the Relative to Clock Pad Net field.



Pad Group: display_grp

Time Requirement

OFFSET: 5.0 Units: ns

Relative to Clock Pad Net: clk

Relative to Clock Edge:

Register Timegroup:

Comment:

Figure 5-9: **Clock to Pad Dialog**

16. Click **OK**
17. Select **File** → **Save**.

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

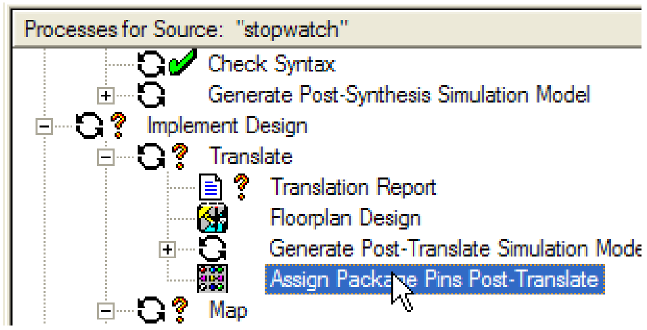
Using the Pin-out Area Constraints Editor (PACE)

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

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

1. In the Processes for Source window, click the **+** next to Implement Design to expand the process hierarchy.
2. Click the **+** next to Translate to expand the process hierarchy.
3. 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.

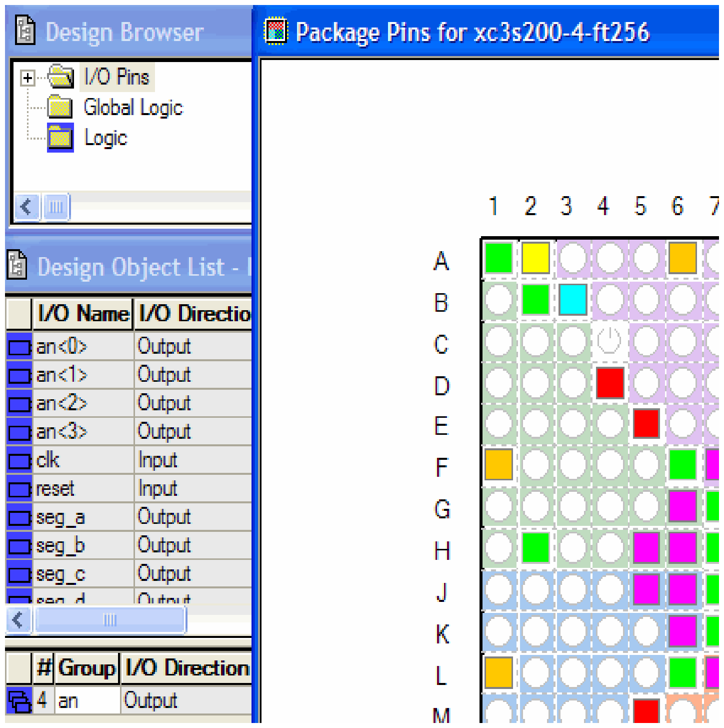


Processes for Source: "stopwatch"

- Check Syntax
- Generate Post-Synthesis Simulation Model
- Implement Design
  - Translate
    - Translation Report
    - Floorplan Design
    - Generate Post-Translate Simulation Mode
    - Assign Package Pins Post-Translate
  - Map

Figure 5-10: **Edit Package Pin Placement**

This process launches PACE.



Design Browser

- I/O Pins
- Global Logic
- Logic

Design Object List - I

I/O Name	I/O Directio
an<0>	Output
an<1>	Output
an<2>	Output
an<3>	Output
clk	Input
reset	Input
seg_a	Output
seg_b	Output
seg_c	Output
seg_d	Output

#	Group	I/O Direction
4	an	Output

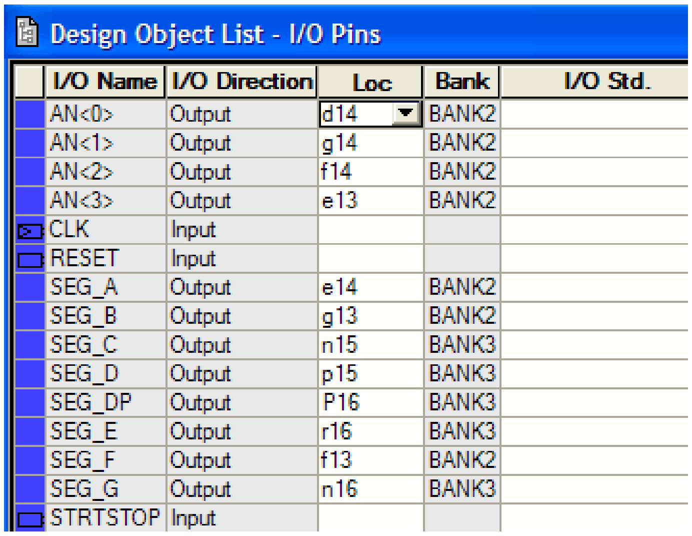
Figure 5-11: **Pin-Out Area Constraints Editor (PACE) When Launched**

4. In PACE, select the **Package View Tab** to open the Package Pins window.

   This window shows the graphical representation of the device package.

5. Select the Design Object List window.

   This window displays all the IOs in the Design.
6. In the Design Object List window, scroll down to the "seg _*"nets.
7. To enter the pin locations, click the Pin Location text box associated with each of the following signals:
   - seg_a → E14
   - seg_b → G13
   - seg_c → N15
   - seg_d → P15
   - seg_dp → P16
   - seg_e → R16
   - seg_f → F13
   - seg_g → N16


Design Object List - I/O Pins

I/O Name	I/O Direction	Loc	Bank	I/O Std.
AN<0>	Output	d14	BANK2	
AN<1>	Output	g14	BANK2	
AN<2>	Output	f14	BANK2	
AN<3>	Output	e13	BANK2	
CLK	Input			
RESET	Input			
SEG_A	Output	e14	BANK2	
SEG_B	Output	g13	BANK2	
SEG_C	Output	n15	BANK3	
SEG_D	Output	p15	BANK3	
SEG_DP	Output	P16	BANK3	
SEG_E	Output	r16	BANK3	
SEG_F	Output	f13	BANK2	
SEG_G	Output	n16	BANK3	
STRTSTOP	Input			

Figure 5-12: **Pin Locations Typed in PACE**

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

8. In the Design Object List window, click and drag the following signals to the specific location in the Package Pin window:
   - clk → T9
   - reset → L13
   - strtstop → M13

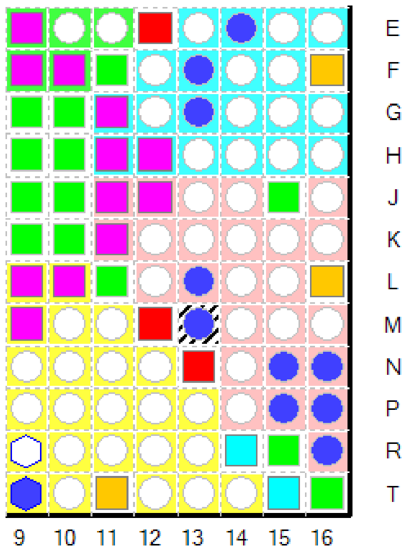


Figure 5-13: **Drag and Drop IOs in the Package Pins Window**

9. If using a third-party synthesis tool, change LOC values for AN<0>, AN<1>, AN<2> and AN<3> to d14, g14, f14, and e13, respectively.
10. Once the pins are locked down, select **File** → **Save**. The changes made in PACE are saved in the *stopwatch.ucf* file in your current working directory.
11. *For Verilog/VHDL designs only:* In the Save dialog box, select the XST Defaults for the Bus Delimiter Dialog.
12. To exit PACE, select **File** → **Exit**.

Mapping the Design

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

1. In the Processes for Source window, right-click on **Map**.
2. Select **Run** from the menu.

   Note: This can also be accomplished by double-clicking Map.

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

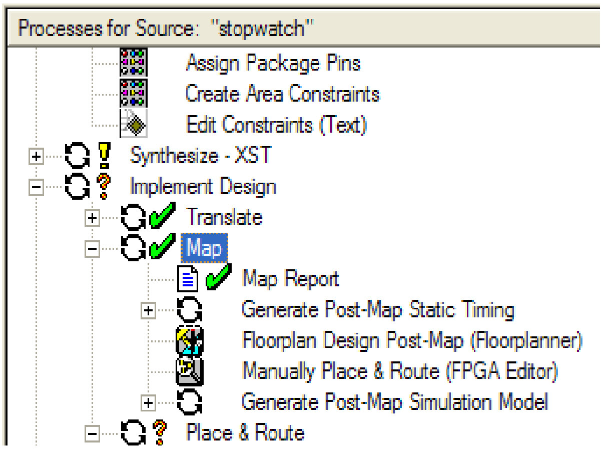

Processes for Source: "stopwatch"
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
Implement Design
Translate
Map
Map Report
Generate Post-Map Static Timing
Floorplan Design Post-Map (Floorplanner)
Manually Place & Route (FPGA Editor)
Generate Post-Map Simulation Model
Place & Route

Figure 5-14: **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 Through 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. For detailed information on the Map report, refer to the *Development System Reference Guide*. 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/xilinx7/.

To view a report:

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

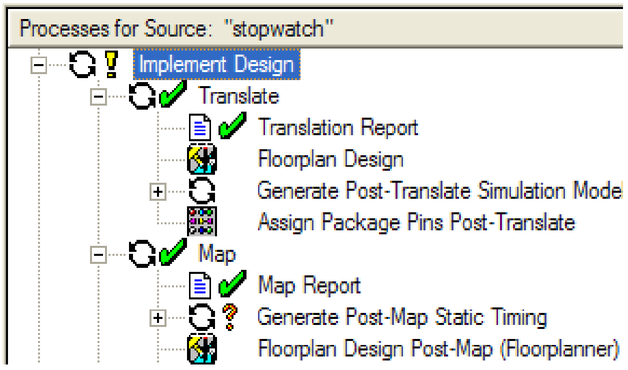

Processes for Source: "stopwatch"
Implement Design
Translate
Translation Report
Floorplan Design
Generate Post-Translate Simulation Model
Assign Package Pins Post-Translate
Map
Map Report
Generate Post-Map Static Timing
Floorplan Design Post-Map (Floorplanner)

Figure 5-15: **Translation Report and Map Report**

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

Using Timing Analysis to Evaluate Block Delays After Mapping

After the design is mapped, use the Logic Level Timing Report to evaluate the logical paths in the design. Because the design is not yet placed and routed, actual routing delay information is not available. The timing report describes the logical block delays and estimated routing delays. The net delays 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).

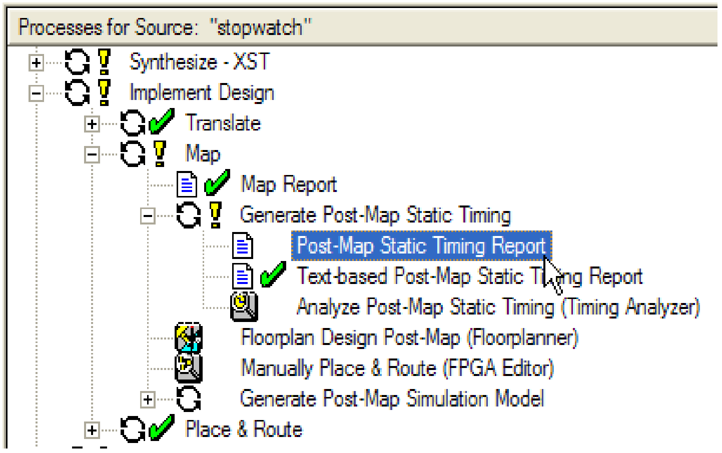
Report Paths in Timing Constraints Option

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 tutorial 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 for Source window, 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 **Post-Map Static Timing Report**. Timing Analyzer automatically launches and displays the report.



Processes for Source: "stopwatch"
Synthesize - XST
Implement Design
Translate
Map
Map Report
Generate Post-Map Static Timing
Post-Map Static Timing Report
Text-based Post-Map Static Timing Report
Analyze Post-Map Static Timing (Timing Analyzer)
Floorplan Design Post-Map (Floorplanner)
Manually Place & Route (FPGA Editor)
Generate Post-Map Simulation Model
Place & Route

Figure 5-16: **Post-Map Static Timing Report**

At the top of this report, you will find the period timing constraint and the minimum period obtained by the tools after mapping. The report contains only three paths per timing constraint, and as a result, you can see a breakdown of the first three paths which contains the component and routing delays. Notice at the end of each path, the percentage of logic verse the percentage of routing is reported. The unplaced floors listed are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

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

Even if you do not generate a Logical Level Timing Report, PAR still processes a design based on the relationship between the block delays, floors, and timing specifications for the design. For example, if a PERIOD constraint of 8 ns is specified for a path, and there are block delays of 7 ns and unplaced floor net delays of 3 ns, PAR stops and generates an error message. In this example, PAR fails because it determines that the total delay (10 ns) is greater than the constraint placed on the design (8 ns). The Post-Map Static Timing Report will list any pre-PAR timing violations.

Placing and Routing the Design

The design can be placed and routed after the mapped design is evaluated. Evaluation verifies that block delays are reasonable given the design specifications.

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

To run PAR, in the Processes for Source window, double-click **Place & Route**.

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

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

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.

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 using FPGA Editor:

1. Launch FPGA Editor in the expanded Place & Route hierarchy by double-clicking **View/Edit Routed Design (FPGA Editor)**.

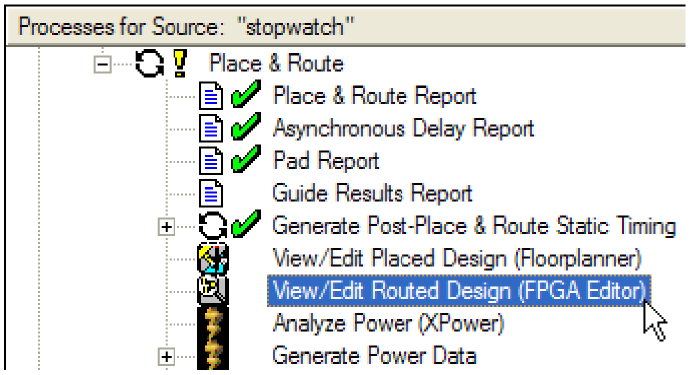

Processes for Source: "stopwatch"
Place & Route
Place & Route Report
Asynchronous Delay Report
Pad Report
Guide Results Report
Generate Post-Place & Route Static Timing
View/Edit Placed Design (Floorplanner)
View/Edit Routed Design (FPGA Editor)
Analyze Power (XPower)
Generate Power Data

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

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

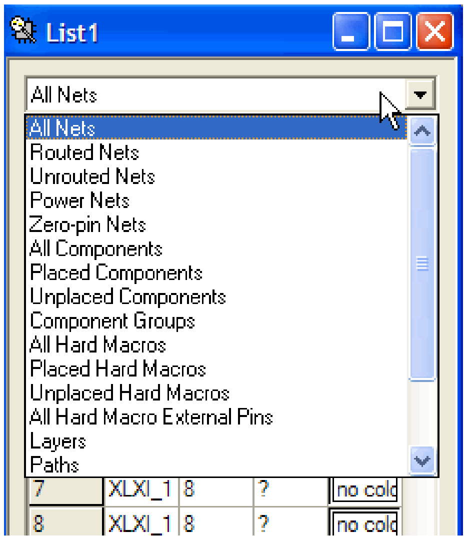


Figure 5-18: **List Window in FPGA Editor**

3. Select the **clk_int** (Clock) net to see the fanout of the clock net.

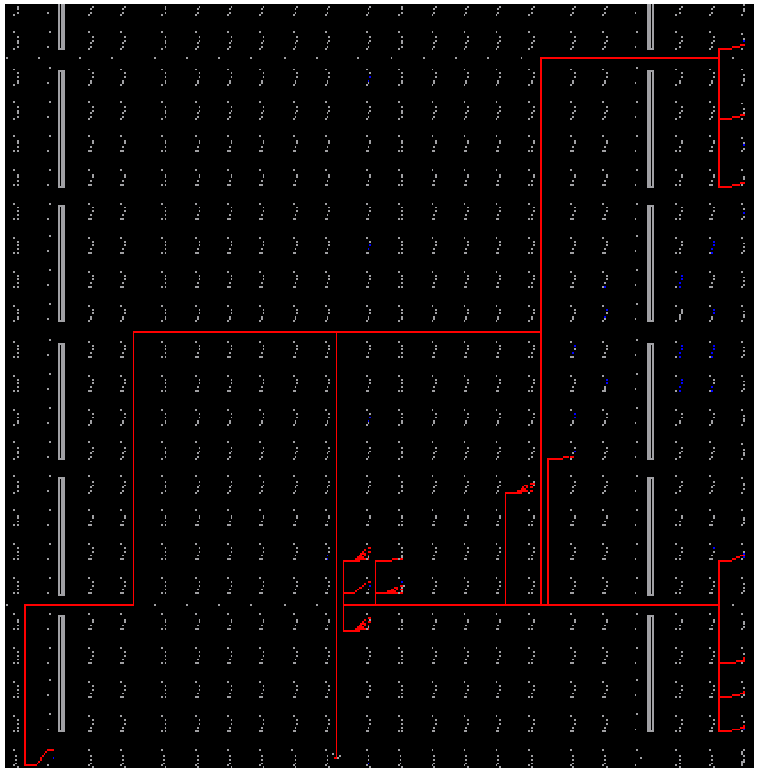


Figure 5-19: **Clock Net**

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

Evaluating Post-Layout Timing

After the design is placed and routed, a Post Layout Timing Report is generated by default to verify that the design meets your specified timing goals. This report evaluates the logical block delays and the routing delays. The net delays are now reported as actual routing delays after the Place and Route process. To display this report:

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

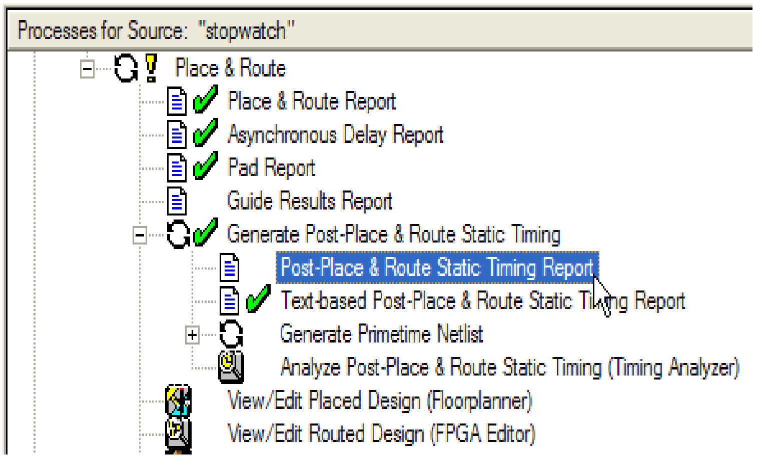

Processes for Source: "stopwatch"
Place & Route
Place & Route Report
Asynchronous Delay Report
Pad Report
Guide Results Report
Generate Post-Place & Route Static Timing
Post-Place & Route Static Timing Report
Text-based Post-Place & Route Static Timing Report
Generate Primetime Netlist
Analyze Post-Place & Route Static Timing (Timing Analyzer)
View/Edit Placed Design (Floorplanner)
View/Edit Routed Design (FPGA Editor)

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

The following is a summary of the Post-Place & Route Static Timing Report.

- The minimum period value increased due to the actual routing delays.
- After the Map step, logic delay contributed to about 80% of the minimum period attained. The post-layout report indicates that the logical delay value decreased somewhat. The total unplaced floors estimate changed as well. Routing delay after PAR now equals about 31% of the period.
- The post-layout result does not necessarily follow the 50/50 rule previously described because the worst case path primarily includes component delays. After the design is mapped, block delays constitute about 80% of the period.

After place and route, the 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 this to be reduced any further is unrealistic. In general, you can reduce excessive block delays and improve design performance by decreasing the number of logic levels in the design.

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

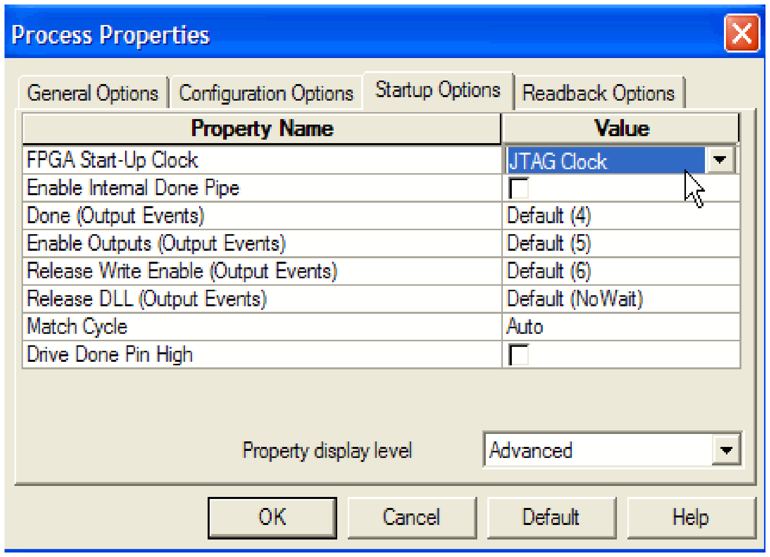
Creating Configuration Data

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

In this tutorial, you will be creating 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.

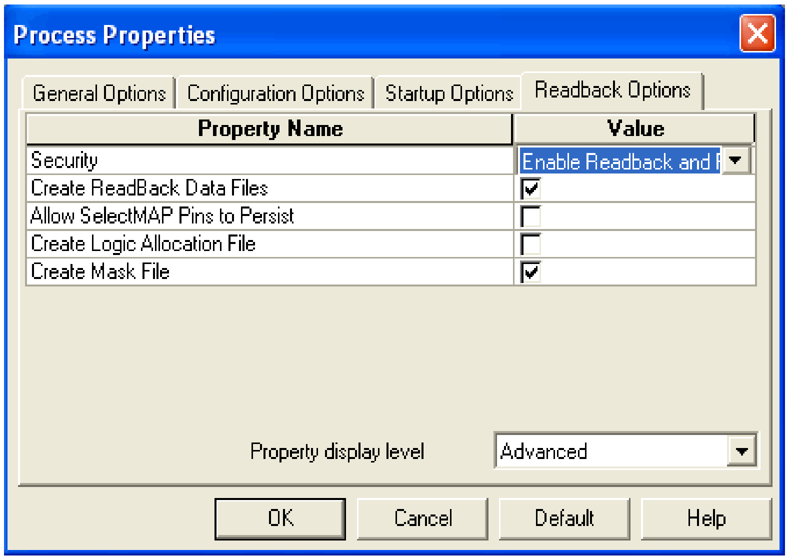


Property Name	Value
FPGA Start-Up Clock	JTAG Clock
Enable Internal Done Pipe	☐
Done (Output Events)	Default (4)
Enable Outputs (Output Events)	Default (5)
Release Write Enable (Output Events)	Default (6)
Release DLL (Output Events)	Default (NoWait)
Match Cycle	Auto
Drive Done Pin High	☐

Figure 5-21: **Process Properties Startup Options Tab**

3. Click the **Startup Options** tab.
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.



Property Name	Value
Security	Enable Readback and I
Create ReadBack Data Files	✓
Allow SelectMAP Pins to Persist	☐
Create Logic Allocation File	☐
Create Mask File	✓

Figure 5-22: **Process Properties Readback Options Tab**

5. Click the **Readback Options** tab.
6. Change the Security property to **Enable Readback and Reconfiguring**.
7. Leave the remaining options in the default setting.
8. Click **OK** to apply the new properties.
9. Double-click **Generate Programming File** to create a bitstream of this design.

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

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

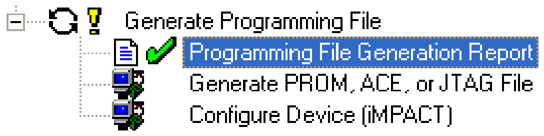

Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)

Figure 5-23: **Programming File Generation Report**

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.

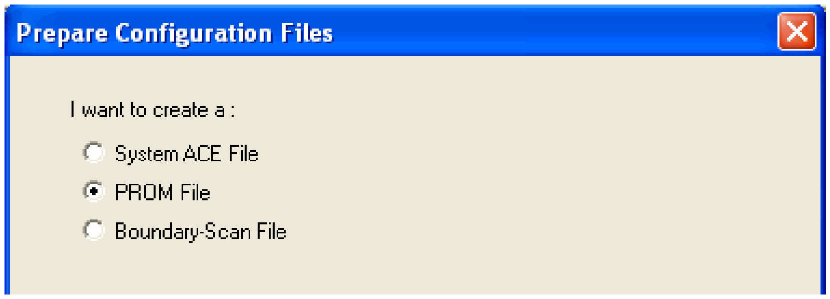
In iMPACT, a wizard enables you to create a PROM file and to:

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

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

1. Launch iMPACT from Project Navigator by double-clicking **Generate PROM, ACE, JTAG File**, located under the Generated Programming File process.

   A wizard opens.
2. In the Prepare Configuration Files dialog box, under “I want to create a:”, select **PROM File**.
3. Click **Next**.

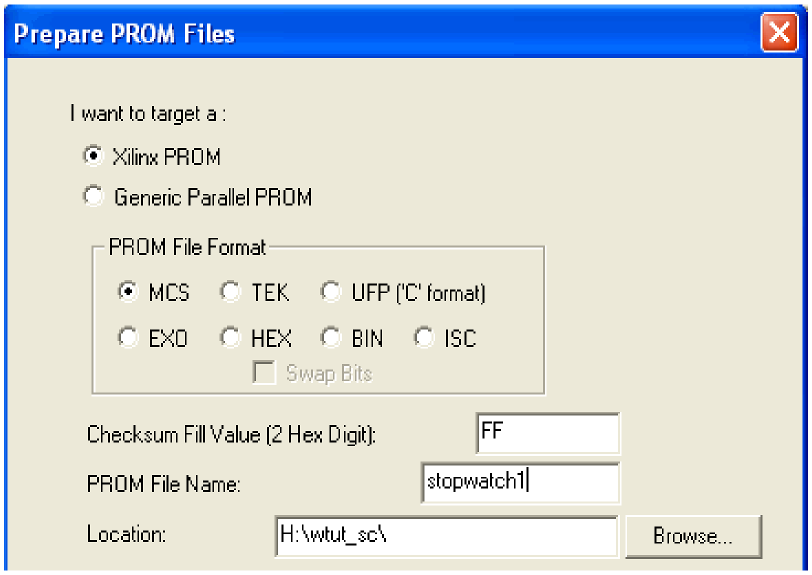


I want to create a :

System ACE File [✓ PROM File] Boundary-Scan File

Figure 5-24: **Prepare Configuration Files Dialog**

4. In the Prepare PROM Files dialog box:
   a. Under “I want to target a:”, select **Xilinx Serial PROM**.
   a. Under PROM File Format, select **MCS**.
   b. For PROM file name, type **stopwatch1**.



I want to target a :

(●) Xilinx PROM
( ) Generic Parallel PROM

PROM File Format
(●) MCS ( ) TEK ( ) UFP ('C' format)
( ) EXO ( ) HEX ( ) BIN ( ) ISC
☐ Swap Bits

Checksum Fill Value (2 Hex Digit): FF
PROM File Name: stopwatch1
Location: H:\wtut_sc\

Figure 5-25: **Prepare PROM Files Dialog**

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

   Note: If you have more data than space available in the PROM, you must split the data into several individual PROMs with the Split PROM option. In this case, only a single PROM is needed.

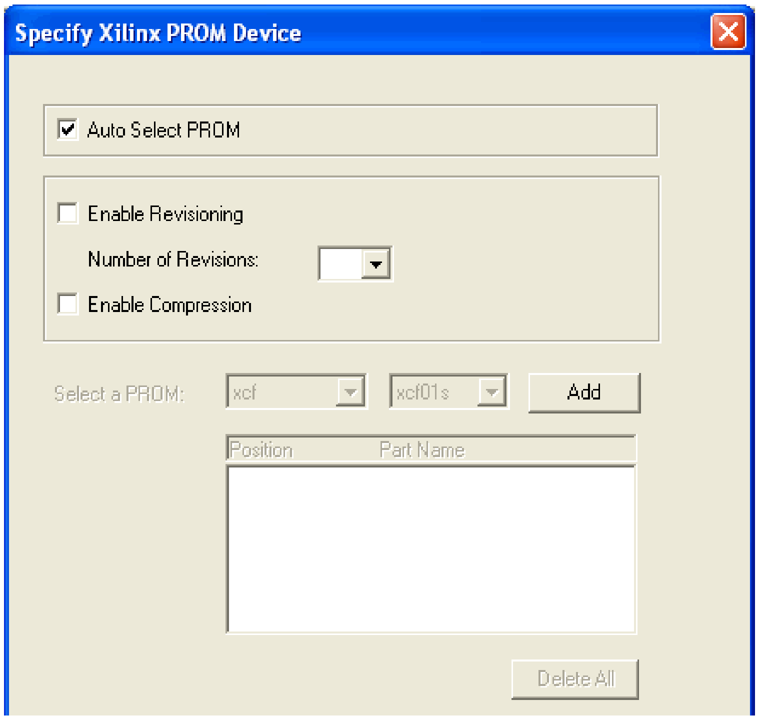


Figure 5-26: **Specify Xilinx Serial PROM Device Dialog Box**

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

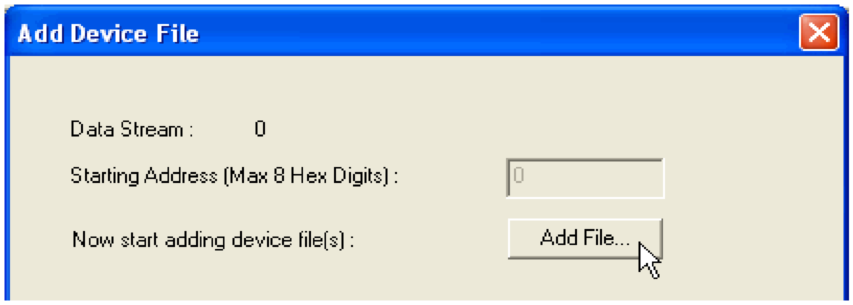

Add Device File
Data Stream : 0
Starting Address (Max 8 Hex Digits) :
0
Now start adding device file(s) :
Add File...

Figure 5-27: **Add Device File Dialog Box**

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. Click **Finish**.
12. When asked to generate a file now, click **Yes**.

    iMPACT displays the PROM associated with your bit file.
13. To close iMPACT, select **File** → **Exit**.

This completes this chapter of the tutorial. For more information on this design flow and implementation methodologies, see the iMPACT Help, available from the iMPACT application by 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.

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 hierarchy in the Processes for Source window. 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 a 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** → **Online Documentation**, or from the web at http://www.xilinx.com/support/sw_manuals/xilinx7/. 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"

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 fully verify the design, the design should be analyzed both statically and dynamically.

In this chapter, you will perform a timing simulation using the ModelSim simulator.

Note: You can use ISE Simulator for timing simulation; however, this chapter does not cover timing simulation using the ISE Simulator. A signal search capability is required for this tutorial, but is not available in the ISE Simulator.

Getting Started

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

Required Software

In addition to Xilinx ISE™ 7, you must have a ModelSim simulator installed. Refer to Chapter 4, "Behavioral Simulation" for information on installing and setting up ModelSim.

Required Files

The timing simulation flow requires the following files:

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

  This chapter assumes that you have completed Chapter 5, "Design Implementation," and thus, have a placed and routed design. A tool called NetGen will be used in this chapter to create a simulation netlist from the placed and routed design which will be used to represent the design during the Timing Simulation.

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

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

- **Xilinx Simulation Libraries**

  For timing simulation, the SIMPRIM library is 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 the Xilinx simulation libraries, see to "Xilinx Simulation Libraries" in Chapter 4.

Timing Simulation Using ModelSim

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

ISE also runs NetGen to generate a simulation netlist from the placed and routed design.

Specifying Simulation Process Properties

To set the simulation process properties:

1. In the Sources in Project window, select the test bench file.
2. In the Processes for Source window, 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 to 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.
```

Click **Apply** and **OK** to set the Preferences.

3. Right-click **Simulate Post-Place & Route VHDL (Verilog) Model**.

4. Select **Properties**.

   The Process Properties dialog box displays.

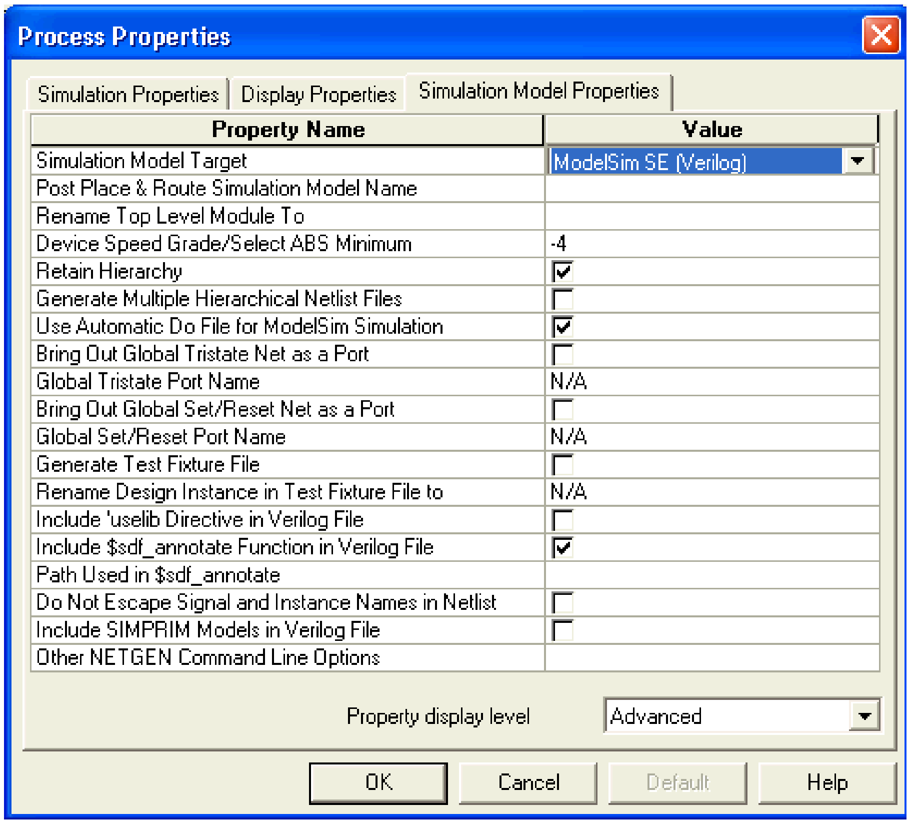
5. Click the **Simulation Model Properties** tab.

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

6. Ensure that you have set the Property display level to **Advanced**. This setting is right above the Help button.

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

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



Property Name	Value
Simulation Model Target	ModelSim SE (Verilog)
Post Place & Route Simulation Model Name	
Rename Top Level Module To	
Device Speed Grade/Select ABS Minimum	-4
Retain Hierarchy	✓
Generate Multiple Hierarchical Netlist Files	☐
Use Automatic Do File for ModelSim Simulation	✓
Bring Out Global Tristate Net as a Port	☐
Global Tristate Port Name	N/A
Bring Out Global Set/Reset Net as a Port	☐
Global Set/Reset Port Name	N/A
Generate Test Fixture File	☐
Rename Design Instance in Test Fixture File to	N/A
Include 'uselib Directive in Verilog File	☐
Include $sdf_annotate Function in Verilog File	✓
Path Used in $sdf_annotate	
Do Not Escape Signal and Instance Names in Netlist	☐
Include SIMPRIM Models in Verilog File	☐
Other NETGEN Command Line Options	

Property display level: Advanced

Figure 6-1: **Simulation Model Properties**

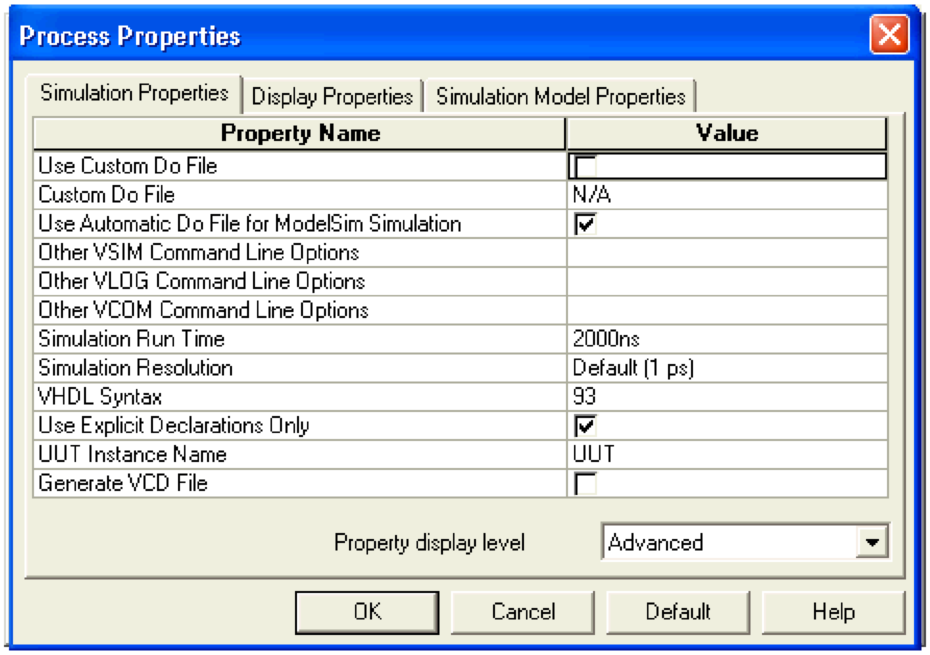
7. Click the **Display Properties** tab.

   This tab gives you control over the MTI (ModelSim) simulation windows. By default, three windows open when timing simulation is launched from ISE. They are the Signal window, the Structure window, and the Wave window. For more details on ModelSim Simulator windows, refer to the *ModelSim User Manual*.

8. Click the **Simulation Properties** tab.

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

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



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

Property display level: Advanced

Figure 6-2: **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 VHDL Model** or **Simulate Post-Place and Route Verilog Model** in the Processes for Source window.

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

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**.

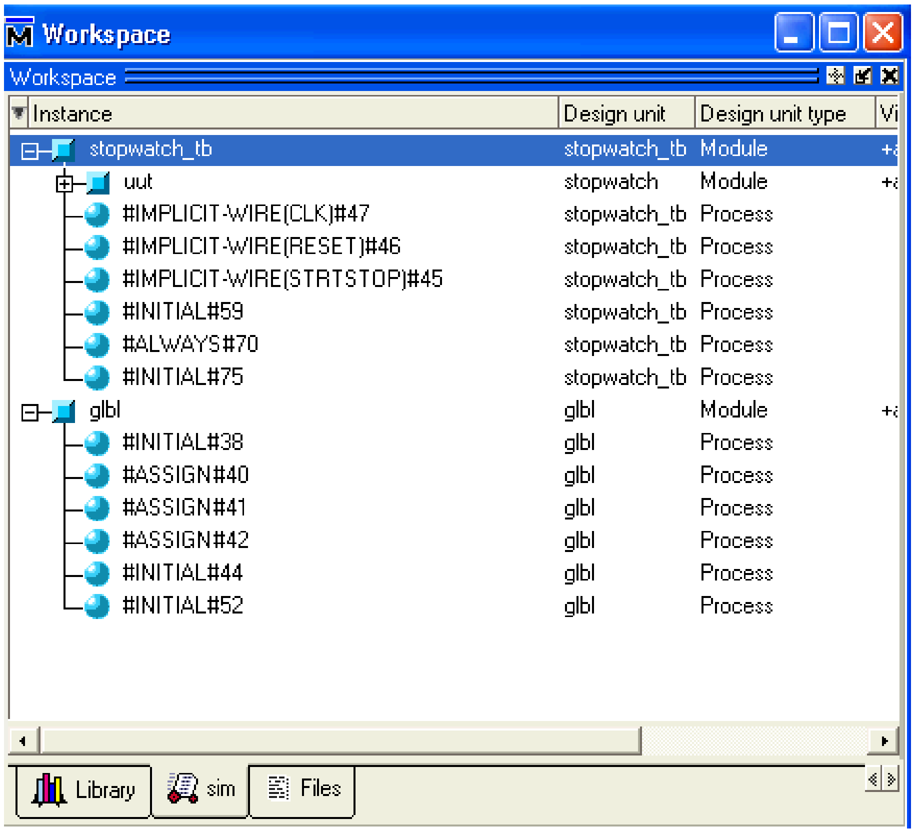
The following procedure explains how to add additional signals in the design hierarchy. In this tutorial, you will be adding the DCM signals to the waveform.

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 Verilog flow. The graphics and the layout of the Structure / Instance window for a schematic or VHDL flow may appear different.



Instance	Design unit	Design unit type	Vi
stopwatch_tb	stopwatch_tb	Module	+
uut	stopwatch	Module	+
#IMPLICIT-WIRE(CLK)#47	stopwatch_tb	Process	
#IMPLICIT-WIRE(RESET)#46	stopwatch_tb	Process	
#IMPLICIT-WIRE(STRTSTOP)#45	stopwatch_tb	Process	
#INITIAL#59	stopwatch_tb	Process	
#ALWAYS#70	stopwatch_tb	Process	
#INITIAL#75	stopwatch_tb	Process	
glbl	glbl	Module	+
#INITIAL#38	glbl	Process	
#ASSIGN#40	glbl	Process	
#ASSIGN#41	glbl	Process	
#ASSIGN#42	glbl	Process	
#INITIAL#44	glbl	Process	
#INITIAL#52	glbl	Process	

Figure 6-4: **Structure/Instance Window - Verilog 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 in **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.

9. Right-click and select **Add to Wave** → **Selected Signals**.

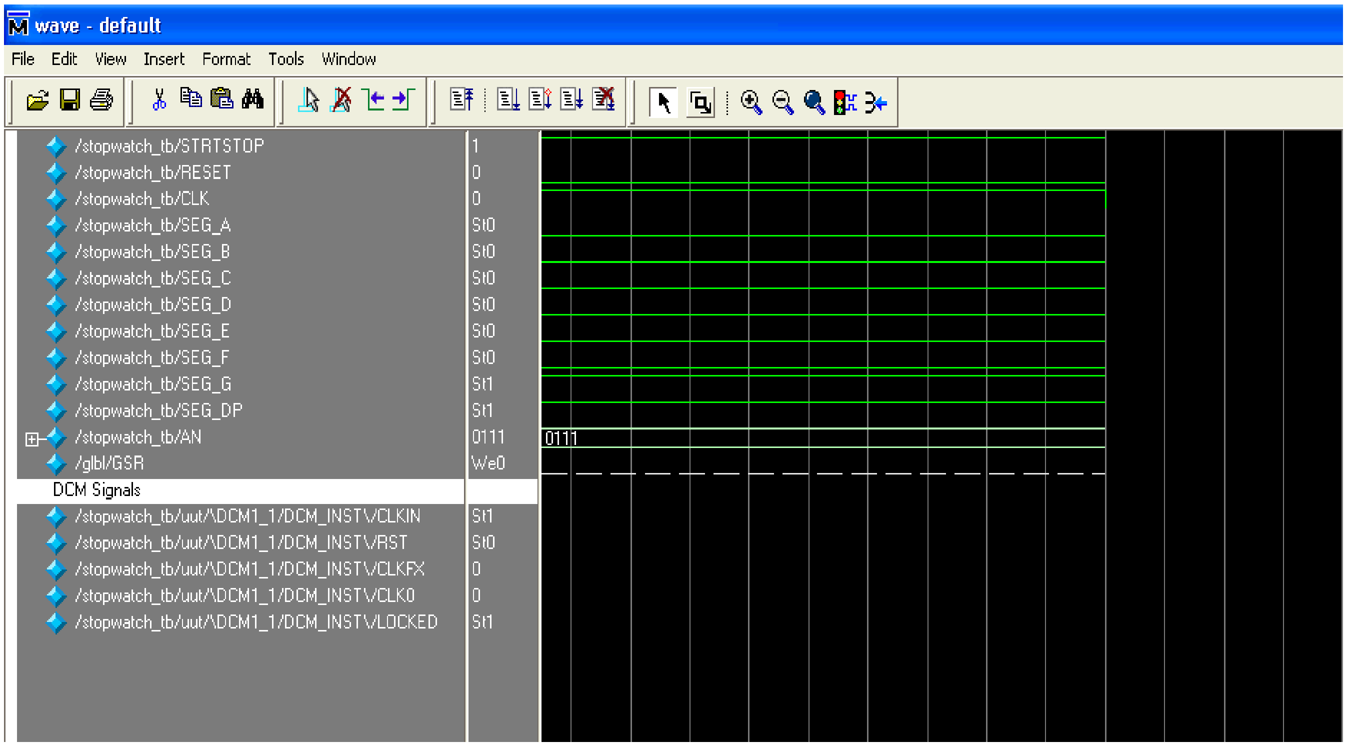
Adding Dividers

Modelsim also 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. To do so, select **Tools** → **Preferences**. In the Display Signal Path box, enter **2** and click **OK**.

The waveform should look as shown in Figure 6-5.



```
/stopwatch_tb/STRTSTOP                      1
/stopwatch_tb/RESET                         0
/stopwatch_tb/CLK                           0
/stopwatch_tb/SEG_A                         St0
/stopwatch_tb/SEG_B                         St0
/stopwatch_tb/SEG_C                         St0
/stopwatch_tb/SEG_D                         St0
/stopwatch_tb/SEG_E                         St0
/stopwatch_tb/SEG_F                         St0
/stopwatch_tb/SEG_G                         St1
/stopwatch_tb/SEG_DP                        St1
/stopwatch_tb/AN                            0111   0111
/glbl/GSR                                   We0
DCM Signals
/stopwatch_tb/uut/\DCM1_1/DCM_INST\/CLKIN   St1
/stopwatch_tb/uut/\DCM1_1/DCM_INST\/RST     St0
/stopwatch_tb/uut/\DCM1_1/DCM_INST\/CLKFX   0
/stopwatch_tb/uut/\DCM1_1/DCM_INST\/CLK0    0
/stopwatch_tb/uut/\DCM1_1/DCM_INST\/LOCKED  St1
```

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.

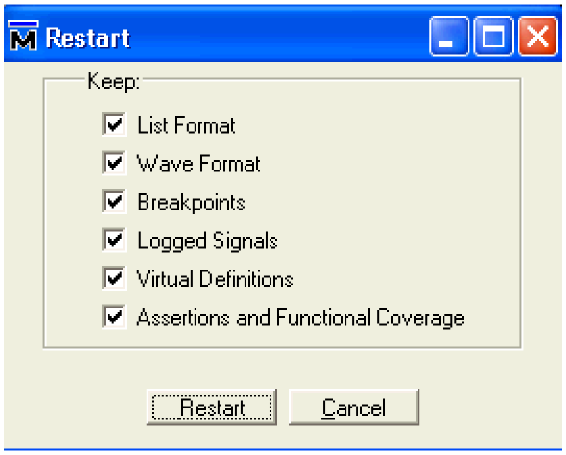
Rerunning Simulation

To restart and re-run the simulation:

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

Figure 6-6: **Restart Simulation Icon**

The Restart dialog box opens.



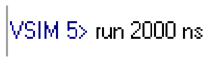
Restart

Keep:

- ☑ List Format
- ☑ Wave Format
- ☑ Breakpoints
- ☑ Logged Signals
- ☑ Virtual Definitions
- ☑ Assertions and Functional Coverage

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 at the ModelSim command prompt**

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 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.

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

To measure the CLK0:

1. Select **Insert** → **Cursor**.
2. Click and drag the cursor to the rising edge transition on the CLK0 signal after the LOCKED signal has gone high.
3. Click the **Find Next Transition** icon twice to move the cursor to the next rising edge on the CLK0 signal.

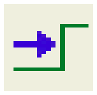


Figure 6-9: **Find Next Transition Icon**

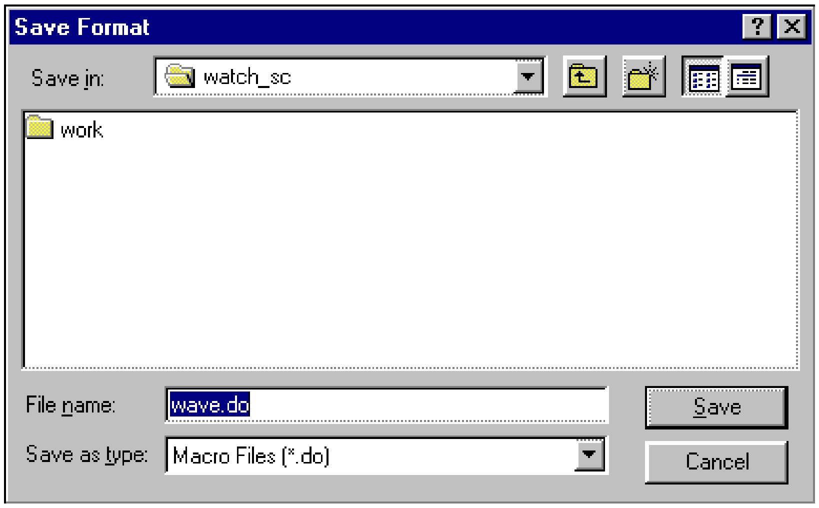
Look at the bottom of the waveform the see the distance between the two cursors. The measurement should read 20000 ps. This converts to 50 Mhz, which is the input frequency from the testbench, which in turn should be the DCM CLK0 output.

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**.



Save Format

Save in: watch_sc

work

File name: wave.do

Save as type: Macro Files (*.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."

Chapter 7

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 USB cable. 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"
- "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
- Spartan™/-II/-IIE/XL/3/3E
- XC4000™/E/L/EX/XL/XLA/XV
- CoolRunner™XPLA3/-II
- XC9500™/XL/XV
- XC18V00P
- XCF00S
- XCF00P

Download Cable Support

Parallel Cable III & 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 Hardware** → **Xilinx Parallel Cable IV**.

USB Platform Cable

The USB 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 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 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 and Spartan™/-II/-IIE/3/3E)
- Select Map—FPGAs (Virtex™/-II/-II PRO/-E/4 and Spartan™/-II/-IIE/3/3E)
- Desktop —FPGAs (Virtex™/-II/-II PRO/-E/4 and Spartan™/-II/-IIE/3/3E)

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/tutorials7.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. Ensure 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 ISE, double-click **Configure Device (iMPACT)** in the Processes for Source window (see Figure 7-1).

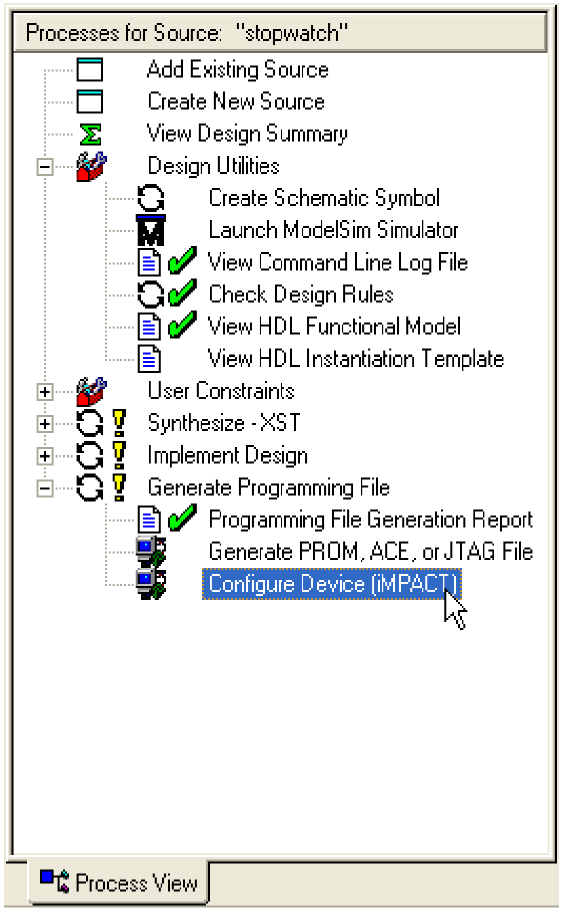

Processes for Source: "stopwatch"
Add Existing Source
Create New Source
View Design Summary
Design Utilities
Create Schematic Symbol
Launch ModelSim Simulator
View Command Line Log File
Check Design Rules
View HDL Functional Model
View HDL Instantiation Template
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)
Process View

Figure 7-1: **Opening iMPACT from ISE**

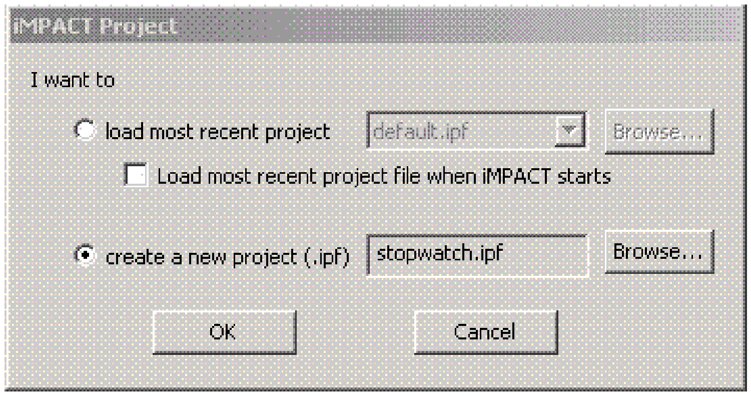
Opening iMPACT stand-alone

To open iMPACT without going through an ISE project, use one of the following methods.

- PC — Click **Start** → **Xilinx ISE 7** → **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.


iMPACT Project

I want to

( ) load most recent project — default.ipf

[ ] Load most recent project file when iMPACT starts

(•) create a new project (.ipf) — stopwatch.ipf

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. Enter a new project filename in the appropriate directory. For this tutorial, enter any name for the project
4. Click **OK**.
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

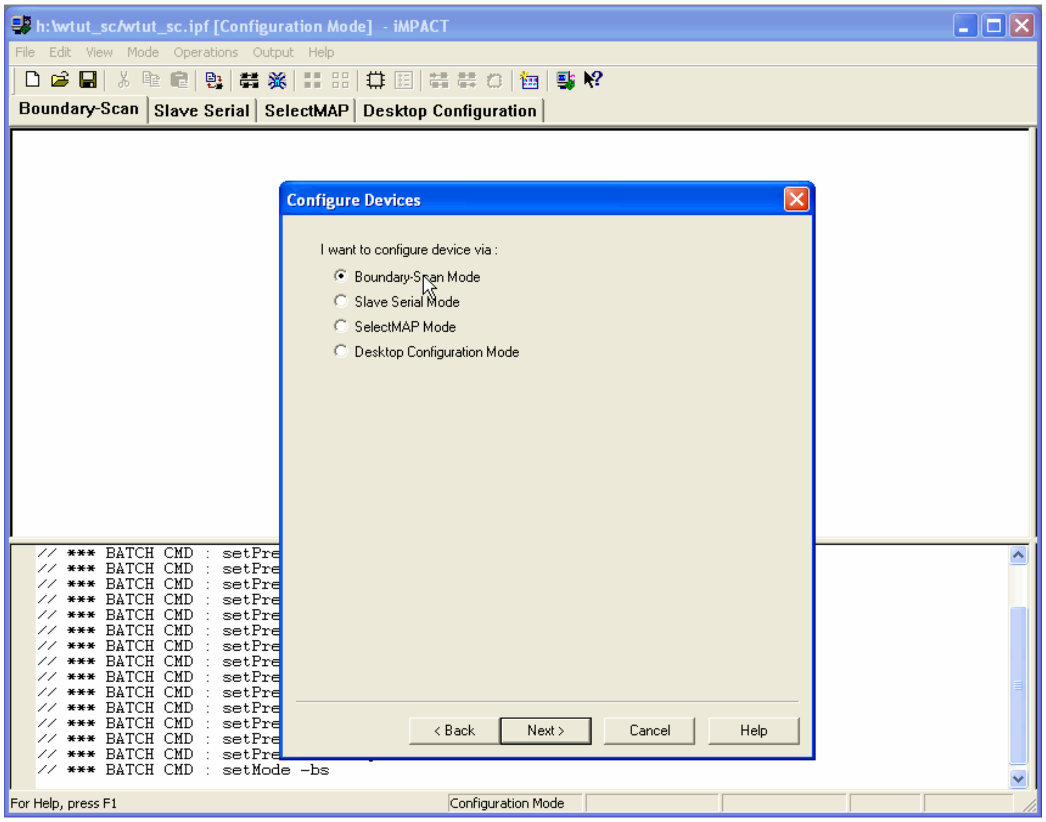
For this tutorial, you will be using the Boundary Scan Configuration Mode. Boundary Scan Configuration Mode allows you to perform Boundary Scan Operations on any chain comprising JTAG compliant devices. The chain can consist of both Xilinx® and non-Xilinx devices; however, limited operations will be 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. *For iMPACT stand-alone only:* Select **Configuration Mode** in the initial dialog box.
2. Select **Configure Devices** in the Operation Mode Selection dialog box.
3. Click **Next**.
4. Select **Boundary Scan Mode** in the Configure Devices dialog box.



I want to configure device via :

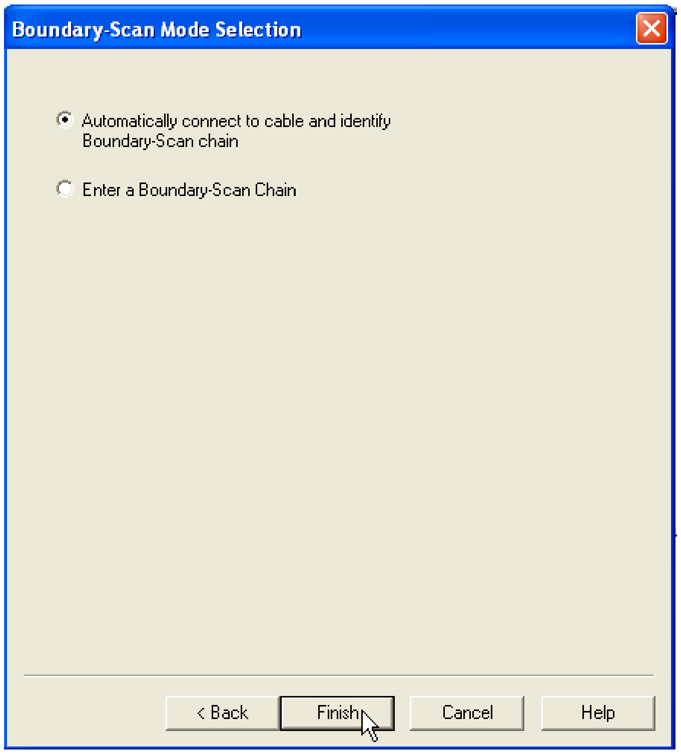
[✓ Boundary-Scan Mode]
Slave Serial Mode
SelectMAP Mode
Desktop Configuration Mode

```
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setPre
// *** BATCH CMD : setMode -bs
```

Figure 7-3: **Initial Window When Opening iMPACT**

5. Click **Next**.

6. Select **Automatically connect to cable and identify Boundary-Scan chain** in the Boundary Scan Mode Selection dialog box.


Boundary-Scan Mode Selection

(● Automatically connect to cable and identify Boundary-Scan chain)

○ Enter a Boundary-Scan Chain

Figure 7-4: **Selecting automatic boundary scan from Wizard**

7. Click **Finish**.

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.

Note: In Figure 7-4, you will notice 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.

Assigning Configuration Files

After initializing a chain, the software prompts you for a configuration file (see Figure 7-5). The configuration file is used to program the device. There are several types of configuration files.

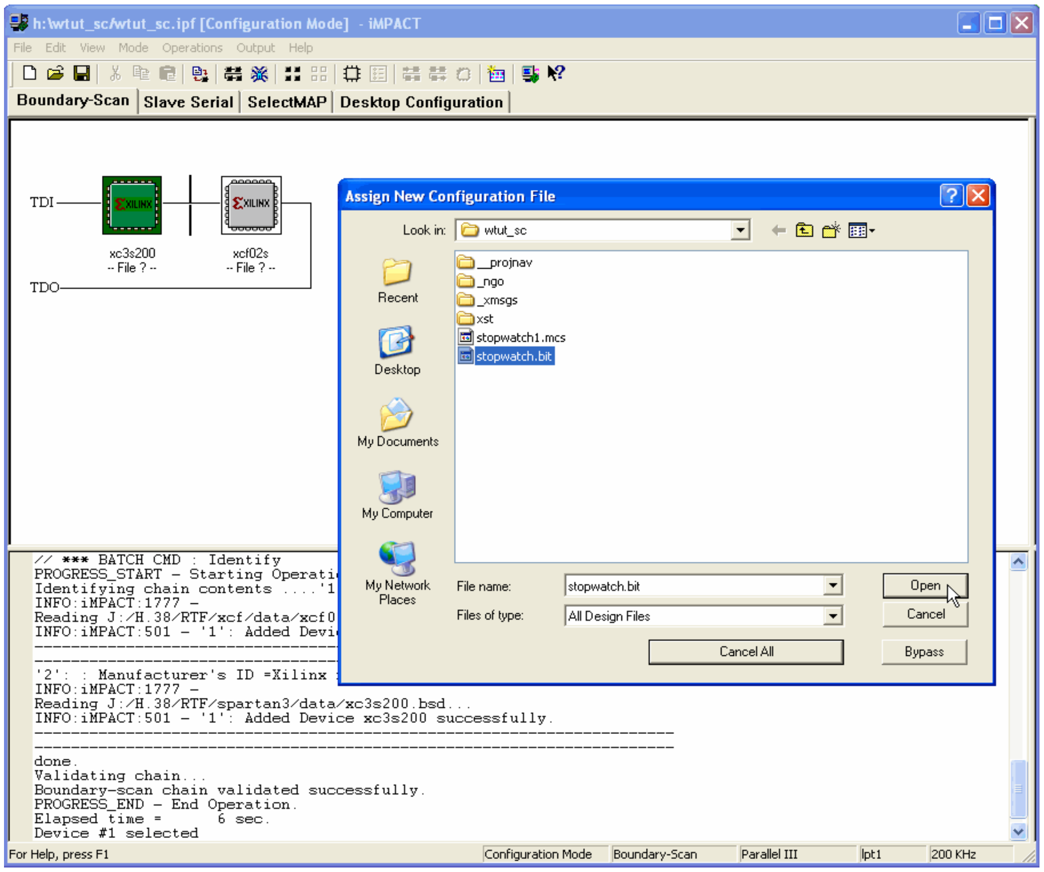
- A Bitstream file (*.bit, *.rbt, *.isc) is used to configure an FPGA.
- A JEDEC file (*.jed,*.isc) is used to configure a CPLD.
- A PROM file (*.mcs, .exo, .hex, or .tek) is used to configure a PROM.

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**.



File name: stopwatch.bit

Files of type: All Design Files

```
// *** BATCH CMD : Identify
PROGRESS_START - Starting Operati
Identifying chain contents ....'1
INFO:iMPACT:1777 -
Reading J:/H.38/RTF/xcf/data/xcf0
INFO:iMPACT:501 - '1': Added Devi
---------------------------------
---------------------------------
'2': : Manufacturer's ID =Xilinx
INFO:iMPACT:1777 -
Reading J:/H.38/RTF/spartan3/data/xc3s200.bsd...
INFO:iMPACT:501 - '1': Added Device xc3s200 successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
done.
Validating chain...
Boundary-scan chain validated successfully.
PROGRESS_END - End Operation.
Elapsed time =      6 sec.
Device #1 selected
```

Figure 7-5: **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 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.

When the software prompts you to select a configuration file for the second device (XCF02S):

4. Select the MCS file from your project working directory.
5. Click **Open**.

Saving the Project File

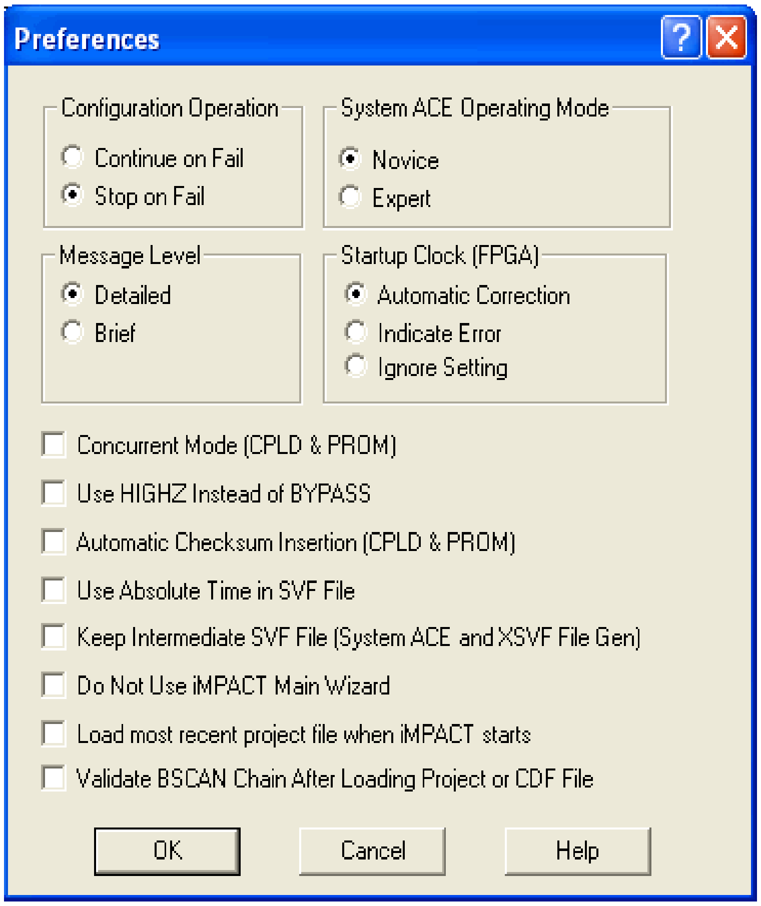
Once the chain has been fully described and configuration files are assigned, you should save your iMPACT Project File (IPF) for later use. To do this, select **File** → **Save Project As**. The Save As dialog box appears and you can browse and save your project file accordingly. To restore the chain when 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 7-6. Click **Help** for a description of the Preferences.

In this tutorial, keep the default values and click **OK**.


Preferences

Configuration Operation
○ Continue on Fail
● Stop on Fail

System ACE Operating Mode
● Novice
○ Expert

Message Level
● Detailed
○ Brief

Startup Clock (FPGA)
● Automatic Correction
○ Indicate Error
○ Ignore Setting

☐ Concurrent Mode (CPLD & PROM)
☐ Use HIGHZ Instead of BYPASS
☐ Automatic Checksum Insertion (CPLD & PROM)
☐ Use Absolute Time in SVF File
☐ Keep Intermediate SVF File (System ACE and XSVF File Gen)
☐ Do Not Use iMPACT Main Wizard
☐ Load most recent project file when iMPACT starts
☐ Validate BSCAN Chain After Loading Project or CDF File

Figure 7-6: **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.

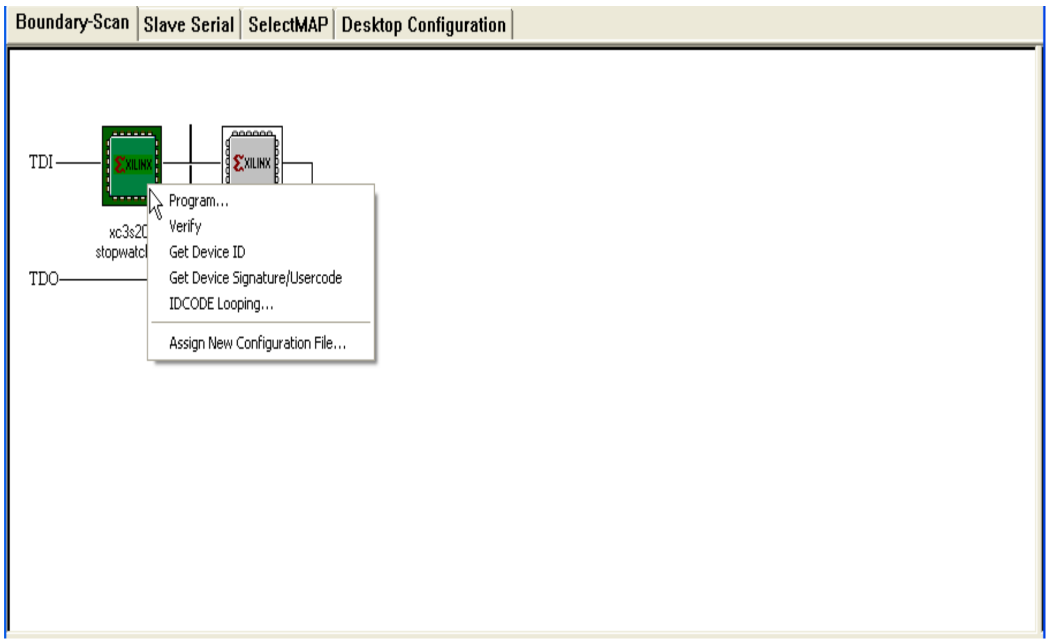


Figure 7-7: **Available Boundary Scan Operations for an XC3S200 Device**

The software accesses the IDCODE for this Spartan-3 device. The result is displayed in the log window (see Figure 7-8).

```
Device #1 selected
// *** BATCH CMD : ReadIdcode -p 1
Validating chain...
Boundary-scan chain validated successfully.
'1': IDCODE is '00000001010000010100000010010011'
'1': IDCODE is '01414093' (in hex).
'1': : Manufacturer's ID =Xilinx xc3s200, Version : 0
```

Figure 7-8: **Log Window Showing Result of Get Device ID**

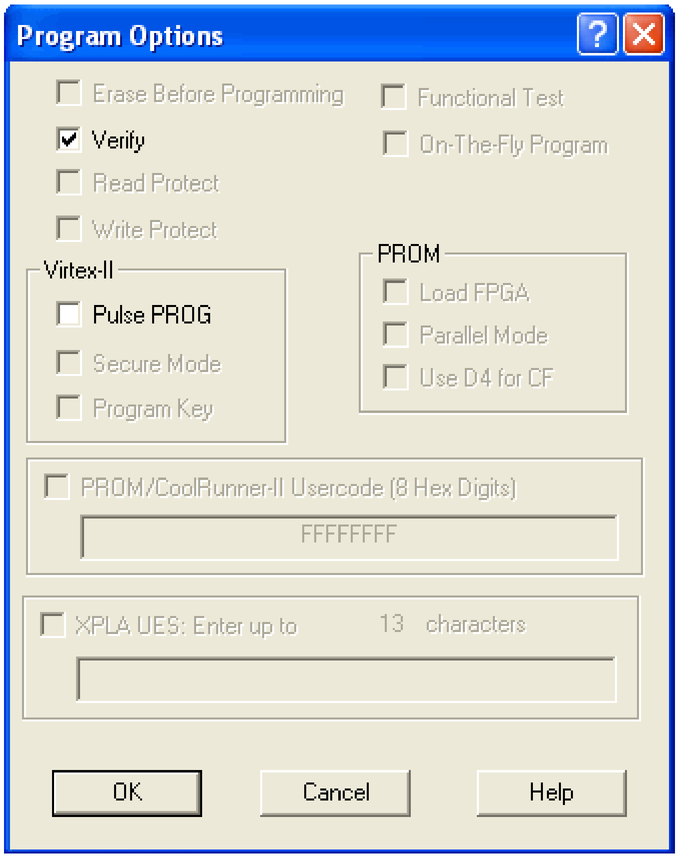
3. Right-click on the XC3S200 device
4. Select **Program** from the right-click menu.

The Program Option window appears (see Figure 7-9).

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.



Program Options

- [ ] Erase Before Programming
- [x] Verify
- [ ] Read Protect
- [ ] Write Protect
- [ ] Functional Test
- [ ] On-The-Fly Program

Virtex-II
- [ ] Pulse PROG
- [ ] Secure Mode
- [ ] Program Key

PROM
- [ ] Load FPGA
- [ ] Parallel Mode
- [ ] Use D4 for CF

- [ ] PROM/CoolRunner-II Usercode (8 Hex Digits)

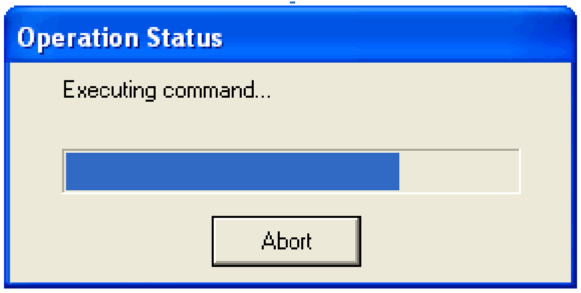
FFFFFFFF

- [ ] XPLA UES: Enter up to 13 characters

Figure 7-9: **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.

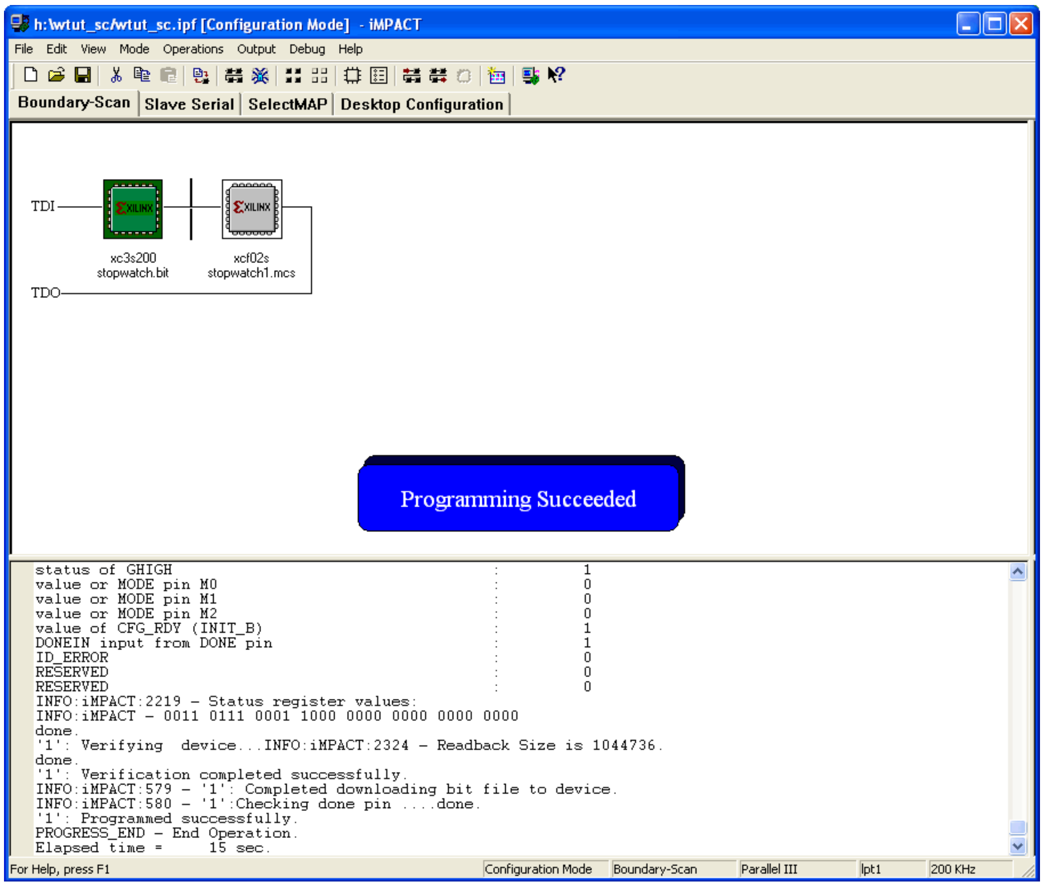


Operation Status

Executing command...

Figure 7-10: **Operation Status**

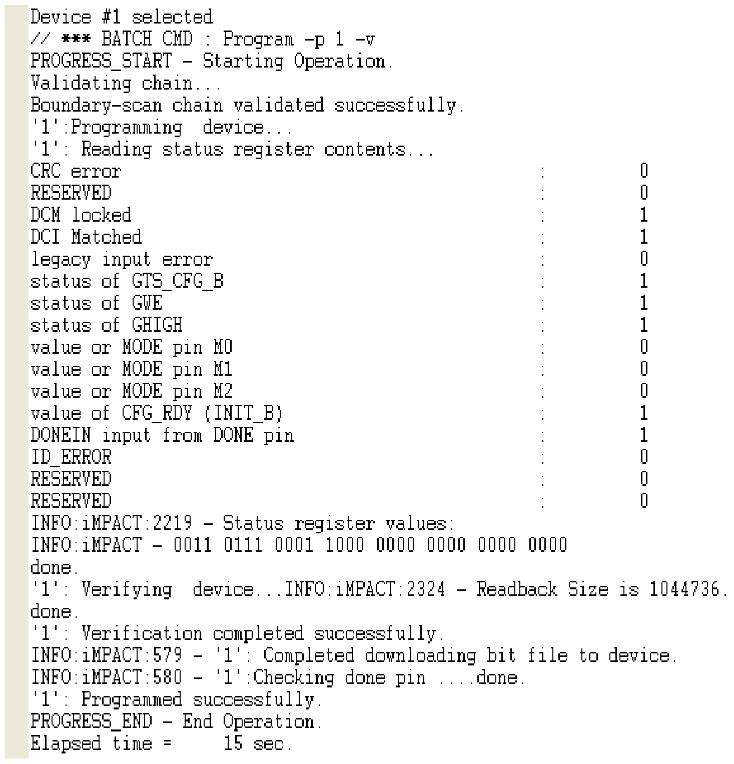
When the Program operation completes, a large blue message appears showing that programming was successful (see Figure 7-11). This message disappears after a couple of seconds.



```
status of GHIGH                                        :         1
value or MODE pin M0                                   :         0
value or MODE pin M1                                   :         0
value or MODE pin M2                                   :         0
value of CFG_RDY (INIT_B)                              :         1
DONEIN input from DONE pin                             :         1
ID_ERROR                                               :         0
RESERVED                                               :         0
RESERVED                                               :         0
INFO:iMPACT:2219 - Status register values:
INFO:iMPACT - 0011 0111 0001 1000 0000 0000 0000 0000
done.
'1': Verifying  device...INFO:iMPACT:2324 - Readback Size is 1044736.
done.
'1': Verification completed successfully.
INFO:iMPACT:579 - '1': Completed downloading bit file to device.
INFO:iMPACT:580 - '1':Checking done pin ....done.
'1': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =     15 sec.
```

Figure 7-11: **Programming Succeeded**

The log window also shows that the programming completed successfully and shows all of the operations that were performed (see Figure 7-12).



```
Device #1 selected
// *** BATCH CMD : Program -p 1 -v
PROGRESS_START - Starting Operation.
Validating chain...
Boundary-scan chain validated successfully.
'1':Programming  device...
'1': Reading status register contents...
CRC error                                        :         0
RESERVED                                         :         0
DCM locked                                       :         1
DCI Matched                                      :         1
legacy input error                               :         0
status of GTS_CFG_B                              :         1
status of GWE                                    :         1
status of GHIGH                                  :         1
value or MODE pin M0                             :         0
value or MODE pin M1                             :         0
value or MODE pin M2                             :         0
value of CFG_RDY (INIT_B)                        :         1
DONEIN input from DONE pin                       :         1
ID_ERROR                                         :         0
RESERVED                                         :         0
RESERVED                                         :         0
INFO:iMPACT:2219 - Status register values:
INFO:iMPACT - 0011 0111 0001 1000 0000 0000 0000 0000
done.
'1': Verifying  device...INFO:iMPACT:2324 - Readback Size is 1044736.
done.
'1': Verification completed successfully.
INFO:iMPACT:579 - '1': Completed downloading bit file to device.
INFO:iMPACT:580 - '1':Checking done pin ....done.
'1': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =     15 sec.
```

Figure 7-12: **Log Window Showing Successful Configuration of the FPGA**

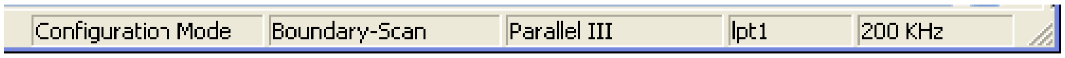
Your design has been programmed and has been verified. The board should now be working and should allow you to start, stop and reset the runner's stopwatch.

Troubleshooting Boundary Scan Configuration

Verifying Cable Connection

When an error occurs during a Boundary Scan operation, first verify first that the cable connection is established and that the software autodetect function is working. If after plugging in the cable into the board and into your machine a connection is still not established, 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 allows 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 7-13).



Configuration Mode	Boundary-Scan	Parallel III	lpt1	200 KHz

Figure 7-13: **Cable Connection Successful**

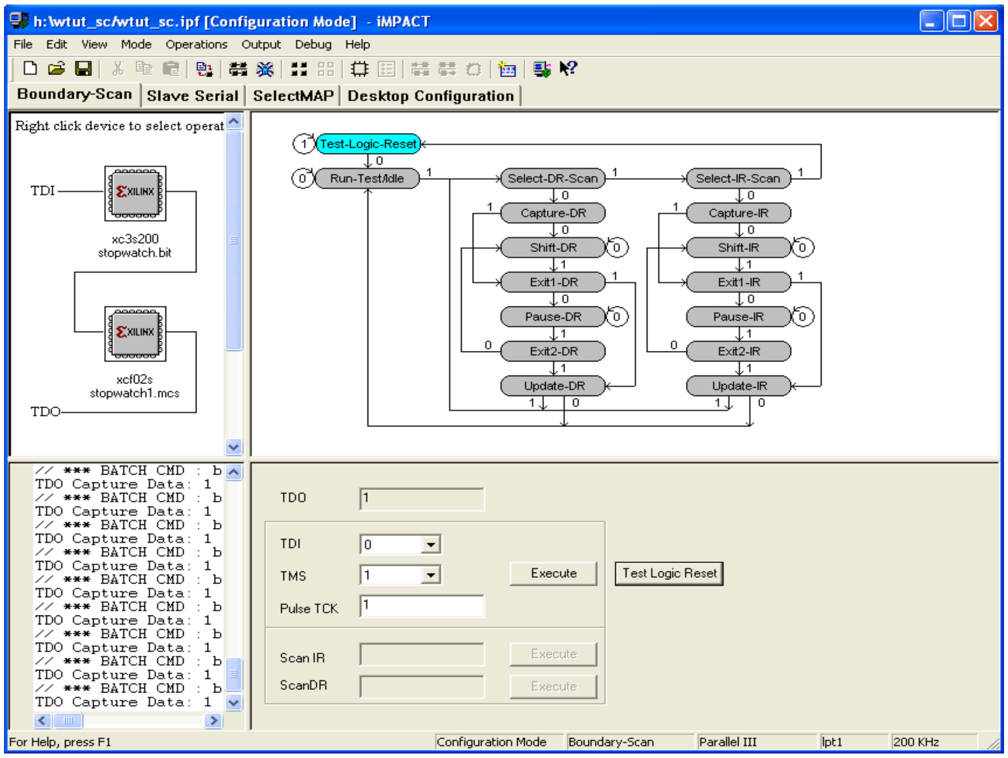
If a cable is connected to the system and the cable autodetection fails, refer to Xilinx Answer Record #15742. Go to http://www.xilinx.com/support and search for “15742”.

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 7-14). This can be used for testing commands and verifying that the chain is set up correctly. To use this feature, select **Debug** → **Start Debug Chain** in iMPACT.



```
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
// *** BATCH CMD : b
TDO Capture Data: 1
```

TDO 1

TDI 0

TMS 1

Pulse TCK 1

For Help, press F1 | Configuration Mode | Boundary-Scan | Parallel III | lpt1 | 200 KHz

Figure 7-14: **Debug Chain**

For help, 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.

iMPACT supports the creation of device programming files in three formats, SVF, XSVF, 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.

In this section, all of the configuration information is written to the SVF file.

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 **Mode** → **File Mode** to indicate that you are creating a programming file.
2. Select the **SVF-STAPL-XSVF** tab.
3. Since you've already defined the chain in the BSCAN tab, a dialog box will appear to allow the chain to be copied to the SVF-STAPL-XSVF tab.
4. Click **Yes**.
5. Another dialog box now appears. Click **Ok** to create a SVF file.
6. A new window will pop up requesting a file name for the SVF file.
7. Enter **getid** for the file name.
8. Click **Save**.

Manual JTAG chain setup for SVF generation

For this tutorial, you do not need to setup a chain manually because a daisy chain is setup and saved in"Using Boundary Scan Configuration Mode."

The chain can be manually created or modified as well. To do this,

1. Ensure that you are in Configuration Mode (**Mode** → **Configuration Mode**).
2. Ensure that you are in Boundary Scan Mode (click the **Boundary-Scan tab**).

   You can now add one device at a time.

3. Right-click on an empty space in the iMPACT window and select **Add Xilinx Device** or **Add Non-Xilinx device**.

   The device is added where the large cursor is positioned.

To add a device between existing devices just click on the line between them and then add the new device.

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.

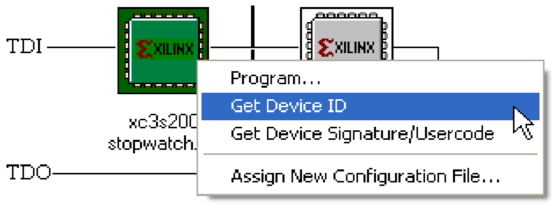
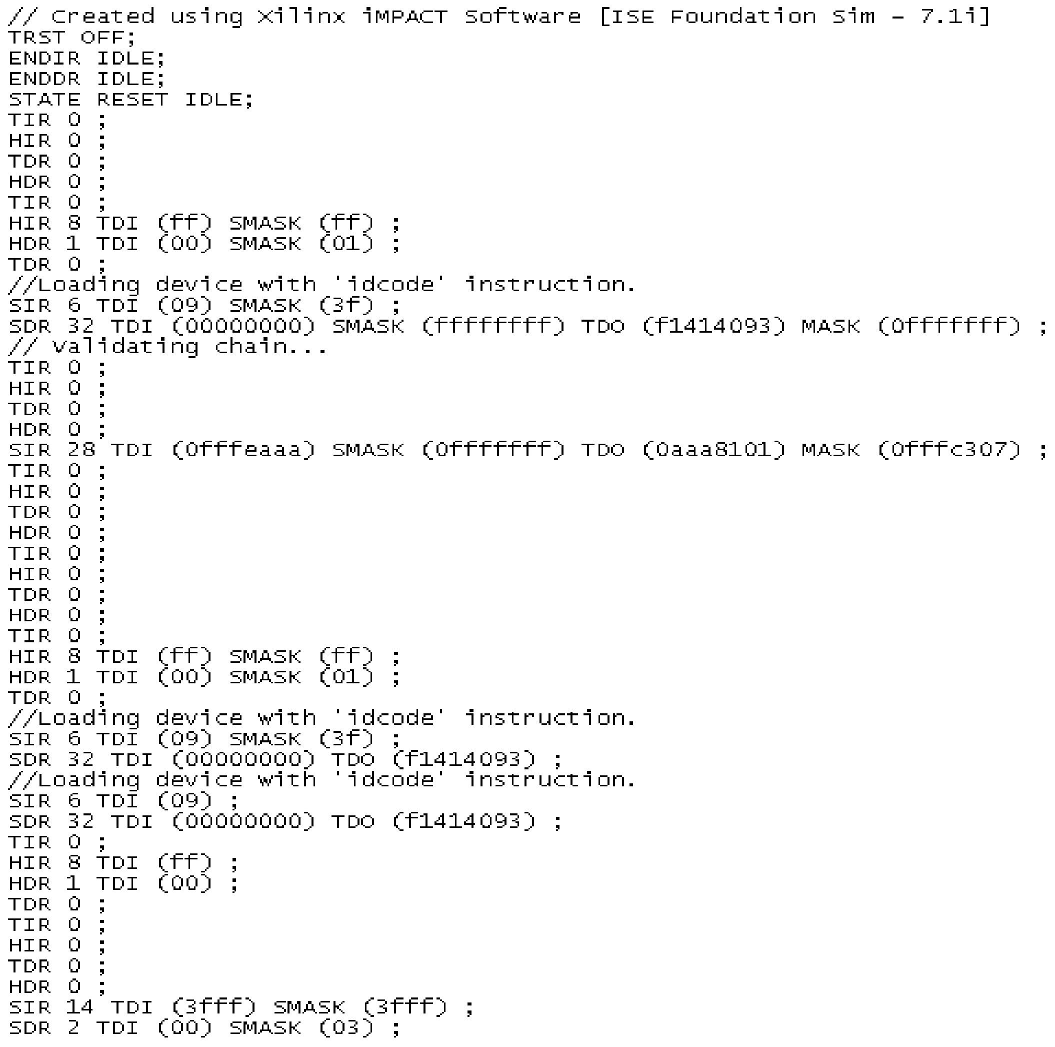

TDI
TDO
XILINX
XILINX
xc3s200
stopwatch.
Program...
Get Device ID
Get Device Signature/Usercode
Assign New Configuration File...

Figure 7-15: **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, you must open the SVF in a text editor outside of the iMPACT software. Figure 7-16 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 (0fffffff) ;
// Validating chain...
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 28 TDI (0fffeaaa) SMASK (0fffffff) 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 7-16: **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.

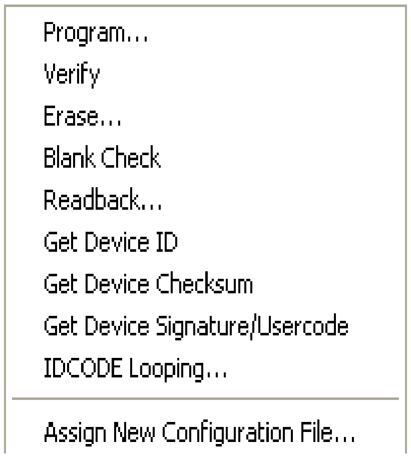


Figure 7-17: **Available Boundary Scan Operations for a XCF00S Device**

The instructions and configuration data needed to Program the second device are added to the 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**.

The file is closed and no more information can be written to it.

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 the chain, adding only Xilinx devices.
- Assign the SVF file to the chain.
- Right-click and select **Execute**.

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, click the Slave Serial Tab at the top of the iMPACT window.

SelectMAP Configuration Mode

With iMPACT, Select MAP 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 Select MAP Configuration Mode, click the Select MAP tab at the top of the iMPACT window. Only the MultiPRO cable can be used for Select MAP Configuration.

Note: These modes cannot be used with the Spartan-3 Starter Kit.

Xilinx ISE 7 Software Manuals and Help

Click one of the following links to view the documents:

PDF Collection	View and **print** the collection with Adobe Acrobat Reader within your Web browser.
Compressed PDF	**Download** a PDF file, then view the downloaded file with Adobe Acrobat Reader locally.
HTML Collection	View and **search** the collection within your Web browser.

UPDATE! **ISE Help** is reorganized and includes the following updates:

- **Project Navigator basics** added to the *Getting Started* section, such as the new "Project Navigator Overview" and "Source File Types" topics
- **Overview and strategic information** added to the *FPGA Design* and *CPLD Design* sections
- **Quick links** to additional resources added to the *Software Reference Information* and *Device Reference Information* sections, including links to individual tools Help, user guides, and device reference information
- **Top 10 customer questions** included in the new *Troubleshooting* section
- **Common search and index** for all Help in the collection

NEW! The following architecture-specific **Libraries Guides** are now available, in both HDL and schematic versions:

- *Virtex™-4 Libraries Guide for HDL Designs*
- *Virtex-4 Libraries Guide for Schematic Designs*
- *Spartan™-3E Libraries Guide for HDL Designs*
- *Spartan-3E Libraries Guide for Schematic Designs*

Note The original version of the Libraries Guide is available for information on other architectures. Future Libraries Guides will be published in the architecture-specific format.

For best results, use Netscape® Communicator 7 and higher or Microsoft® Internet Explorer 6 and higher to view these files. To view manuals from previous releases, go to support.xilinx.com Documentation.

Send your Software Manuals feedback to isedocs@xilinx.com

Get Acrobat Reader Adobe to correctly view the PDF files.