

# **Graphical Design Tutorial for HDL Author - Graphics, HDL Designer - Graphics HDL Author - Pro and HDL Designer - Pro**

Software Version 2001.3

7 June 2001



**Copyright © Mentor Graphics Corporation 1996-2001.  
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### **RESTRICTED RIGHTS LEGEND 03/97**

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:  
Mentor Graphics Corporation  
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Web site: <http://www.hldesigner.com>  
Email: [hldesigner\\_support@mentor.com](mailto:hldesigner_support@mentor.com)

This is an unpublished work of Mentor Graphics Corporation.

# Trademark Information

The following names which appear in this documentation set are trademarks, registered trademarks or service marks of Mentor Graphics Corporation:

HDL Designer Series™, HDL Designer™, HDL Pilot™, HDL Detective™, HDL Author™, HDL2Graphics™, FPGA Advantage™, Interconnect Table™, Interface-Based Design™, IBD™, Inventra™, LeonardoInsight™, LeonardoSpectrum™, Mentor™, Mentor Graphics®, ModelSim®, ModuleWare™, Renoir™, Seamless® and Seamless CVE™.

The following names which appear in this documentation set are trademarks, registered trademarks or service marks of other companies:

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Exchange, FrameMaker and PostScript are registered trademarks of Adobe Systems Incorporated.

Altera, MegaWizard and MAX+PLUS are registered trademarks and Quartus a trademark of Altera Corporation.

ClearCase Attache is a trademark and ClearCase is a registered trademark of Rational Software Corporation.

DesignSync is a registered trademark of Synchronicity Incorporated.

FLEXIm is a trademark of Globetrotter Software, Incorporated.

Hewlett-Packard (HP), HP-UX and PA-RISC are registered trademarks of Hewlett-Packard Company.

Leapfrog, NC-Verilog, Verilog and Verilog-XL are trademarks and registered trademarks of Cadence Design Systems Incorporated.

Netscape is a trademark of Netscape Communications Corporation.

Quartus and APEX are trademarks of Altera Corporation.

SPARC is a registered trademark and SPARCstation is a trademark of SPARC International Incorporated.

SpyGlass is a trademark of Interra Inc.

Sun Microsystems and Sun Workstation are registered trademarks of Sun Microsystems Incorporated. Sun and SunOS are trademarks of Sun Microsystems Incorporated.

Synopsys, Design Analyzer, Design Compiler, FPGA Express, VCS, VCSi and VSS are trademarks of Synopsys Incorporated.

Synplify is a registered trademark of Synplicity Incorporated.

The Graphics Connection is a trademark of Square One.

Visual SourceSafe and Windows are trademarks of Microsoft Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Incorporated.

Xilinx is a registered trademark and Core Generator a trademark of Xilinx, Incorporated.

Other brand or product names that appear in the documentation are trademarks or registered trademarks of their respective holders.

# TABLE OF CONTENTS

<b>About This Manual</b> .....	x
Introduction.....	x
Copying Text From the Acrobat Viewer .....	xii
Example Designs .....	xii
<b>Chapter 1</b>	
<b>VHDL Timer Exercise</b> .....	1-1
Specification .....	1-1
Set Library Mapping.....	1-2
Set the Default Language.....	1-4
Create a Block Diagram.....	1-5
Edit the Title Block Template.....	1-6
Add Blocks .....	1-7
Add Embedded Blocks .....	1-8
Add Ports and Signals.....	1-9
Add a Bundle and Global Connector.....	1-11
Save the Block Diagram .....	1-12
Edit Block and Signal Names .....	1-14
Add an Embedded HDL Text View .....	1-18
Add a Panel and Edit the Title Block .....	1-20
Set State Machine Preferences.....	1-22
Create a Child State Diagram .....	1-24
Add States and Transitions .....	1-26
Save the State Diagram.....	1-27
Edit the States .....	1-28
Edit the Transitions.....	1-30
Create a Hierarchical State Diagram .....	1-32
Complete the Hierarchical State Diagram .....	1-34
Editing State Machine Properties .....	1-36
Set Generation Properties .....	1-39
Set Checks for HDL Generation.....	1-41
Generate HDL for the State Machine .....	1-42
Import the BCDCounter Design Unit.....	1-44
Create a Child Block Diagram.....	1-46

## TABLE OF CONTENTS [continued]

Edit the Generic Mapping.....	1-50
Add ModuleWare Components .....	1-51
Add a User Declaration .....	1-54
Create a Truth Table .....	1-56
Edit the Truth Table.....	1-57
Set Truth Table Properties .....	1-59
Browse the Timer Design .....	1-60
Generate HDL for the Hierarchy .....	1-61
Edit the Timer Symbol.....	1-63
Create a Test Bench .....	1-64
Import the Tester Design Unit .....	1-66
Instantiate the Imported Tester .....	1-67
Generate HDL for the Test Bench .....	1-68
Browse the Completed Design .....	1-70
Setup the Downstream Tools .....	1-71
Compile the Design .....	1-74
Invoke the ModelSim Simulator.....	1-76
Setup the Simulator Windows .....	1-77
Enable Animation .....	1-78
Simulate the Design.....	1-80
Review the Animation .....	1-82
Setup the Synthesis Tool .....	1-84
Run the Synthesis Flow .....	1-86
Using the Example VHDL Design .....	1-89
<b>Chapter 2</b>	
<b>Verilog Timer Exercise.....</b>	<b>2-1</b>
Specification .....	2-1
Set Library Mapping.....	2-2
Set the Default Language.....	2-4
Create a Block Diagram.....	2-5
Edit the Title Block Template.....	2-6
Add Blocks .....	2-7
Add Embedded Blocks .....	2-8
Add Ports and Signals.....	2-9

## TABLE OF CONTENTS [continued]

Add a Bundle and Global Connector .....	2-11
Save the Block Diagram .....	2-12
Edit Block and Signal Names .....	2-14
Add an Embedded HDL Text View .....	2-18
Add a Panel and Edit the Title Block .....	2-20
Set State Machine Preferences.....	2-22
Create a Child State Diagram .....	2-24
Add States and Transitions .....	2-26
Save the State Diagram.....	2-27
Edit the States .....	2-28
Edit the Transitions.....	2-30
Create a Hierarchical State Diagram .....	2-32
Complete the Hierarchical State Diagram .....	2-34
Editing State Machine Properties .....	2-36
Set Generation Properties .....	2-39
Set Checks for HDL Generation.....	2-41
Generate HDL for the State Machine .....	2-42
Import the BCDCounter Design Unit.....	2-44
Create a Child Block Diagram.....	2-46
Edit the Parameter Mapping .....	2-50
Add ModuleWare Components .....	2-51
Add a User Declaration .....	2-54
Create a Truth Table .....	2-56
Edit the Truth Table.....	2-57
Set Truth Table Properties .....	2-58
Add a Module Declaration.....	2-60
Browse the Timer Design .....	2-61
Generate HDL for the Hierarchy .....	2-62
Edit the Timer Symbol.....	2-64
Create a Test Bench .....	2-65
Import the Tester Design Unit .....	2-67
Instantiate the Imported Tester .....	2-68
Generate HDL for the Test Bench .....	2-69
Browse the Completed Design .....	2-71
Setup the Downstream Tools.....	2-72

## TABLE OF CONTENTS [continued]

Compile the Design .....	2-75
Invoke the ModelSim Simulator.....	2-77
Setup the Simulator Windows .....	2-78
Enable Animation .....	2-79
Simulate the Design.....	2-81
Review the Animation .....	2-83
Setup the Synthesis Tool .....	2-85
Run the Synthesis Flow .....	2-87
Using the Example Verilog Design .....	2-90
<b>Appendix A</b>	
<b>Using Text Design Tools .....</b>	<b>A-1</b>
Introduction.....	A-1
Create HDL Text for the Control Block .....	A-1
Create HDL Text for the DtoB Block.....	A-9
Importing the Tester Design Unit .....	A-12
Generating and Compiling the Design .....	A-12
<b>Appendix B</b>	
<b>Using Verilog-XL .....</b>	<b>B-1</b>
Introduction.....	B-1
Setup Verilog-XL .....	B-1
Invoke the Verilog-XL Simulator.....	B-3
Setup the SimWave Window .....	B-3
Enable Animation .....	B-4
Running the Verilog-XL Simulator .....	B-6
Review the Animation .....	B-7
<b>Appendix C</b>	
<b>Creating a VHDL Flow Chart.....</b>	<b>C-1</b>
Introduction.....	C-1
Create the Tester Flow Chart .....	C-2
Set Flow Chart Properties .....	C-3
Add a Start Point and Action Box .....	C-6



## TABLE OF CONTENTS [continued]

Add a Loop and an Associated Comment .....	C-7
Add an Action Box .....	C-11
Add a Hierarchical Action Box .....	C-12
Add a Decision Box .....	C-13
Add Wait Boxes .....	C-15
Copy the Decision Tree .....	C-17
Completing the Flow Chart .....	C-18

### Appendix D

<b>Creating a Verilog Flow Chart .....</b>	<b>D-1</b>
--	------------

Introduction .....	D-1
Create the Tester Flow Chart .....	D-2
Set Flow Chart Properties .....	D-3
Add a Start Point and Action Box .....	D-6
Add a Loop and an Associated Comment .....	D-7
Add an Action Box .....	D-11
Add a Hierarchical Action Box .....	D-12
Add a Decision Box .....	D-13
Add a Wait Box .....	D-15
Copy the Decision Tree .....	D-17
Completing the Flow Chart .....	D-18

---

# About This Manual

## Introduction


This manual provides a self-paced tutorial with step-by-step procedures for creating a simple timer design and [test bench](#) using [VHDL](#) or [Verilog](#).

The tutorial covers the basic procedures required to fully define and verify a design using graphical views. The full tutorial can be completed by users of the following [HDL Designer Series](#) graphical design tools:

- HDL Author - Graphics
- HDL Author - Pro
- HDL Designer - Graphics
- HDL Designer - Pro

The tutorial can also be completed by users of the HDL text design versions of these tools by importing [HDL text](#) views instead of creating the [graphic editor](#) views.

When new terminology is introduced, the keywords (shown in blue when this document is viewed online) are hyperlinked to a definition in the Glossary. (For example, the keywords [test bench](#), [VHDL](#) and [Verilog](#) in the first sentence above.)

User and reference information including an online version of the glossary can also be accessed at any time from the **Help Topics** index. The **Help Topics** provide a contents list, keyword index and full text search facility. In addition, many of the dialog boxes are linked to related help topics by  buttons.

The completed VHDL design can be compiled and simulated if ModelSim is available and the Verilog design can be compiled and simulated using ModelSim or Verilog-XL. Either design can be synthesized if the LeonardoSpectrum tools are available.

Although the procedures do not describe the use of other tools, the HDL Designer Series includes support for a range of alternative downstream tool interfaces and it should be possible to use the generated HDL with any of these interfaces.

However, you must consider any limitations of your external tool. In particular, some VHDL tools may not support the standard IEEE packages used in the tutorial and you should substitute an appropriate alternative package.

The tutorial assumes that users have some knowledge of the issues for digital hardware design and experience of the VHDL or Verilog language. It is possible to complete the tutorial without this knowledge by carefully copying the language syntax given in the procedures. However, a separate VHDL or Verilog training course is recommended in order to fully appreciate how the power of HDL design can be exploited using graphical design methods.

Procedures for completing the tutorial using VHDL are given in [Chapter 1](#) and procedures for using Verilog in [Chapter 2](#). The alternative procedures for use with the text design tools are described in [Appendix A](#). Procedures for using the Verilog-XL simulator are given in [Appendix B](#).

The main tutorial includes a test bench which uses on a flow chart view. The flow chart can be imported from example HDL code or can optionally be created by following the procedures in [Appendix C](#) and [Appendix D](#).









When pathnames (or window titles derived from pathnames) are shown in this tutorial, the PC convention (\) is used.

The examples shown in the tutorial pages have been laid out for maximum readability in the Acrobat viewer or on the printed page. When you are creating your own design, it is advisable to allow extra space between diagram objects so that you can easily route signals between them.

All user commands in the tutorial procedures are referenced using their menu path (shown in bold text) or toolbar button. However, many commands can also be accessed using keyboard shortcuts. Refer to the shortcut tables in the help pages for full lists of the available shortcuts. These lists can be accessed by choosing **Shortcuts** from the HDL Designer **Help** menu.

## Copying Text From the Acrobat Viewer

You can copy text from this document by choosing the  (**Text Select**) tool button or  shortcut key in the Acrobat viewer and choosing **Copy** from the Acrobat **Edit** menu (or using the + shortcut).

The text can be pasted into a text editor (or application dialog box) using the + shortcut or the **Paste** menu option if one is provided in the destination window. In the [graphic editors](#), you can use the **Paste Special** option to explicitly paste text from the system clipboard.



If you copy HDL text from a tutorial help page, check that punctuation characters are copied correctly. In particular, line feed characters may not be translated on UNIX systems and may need to be re-entered.

## Example Designs

Completed examples of the VHDL and Verilog tutorial designs are provided in the HDL Designer Series installation. These can be browsed for reference or can be compiled, simulated and animated directly if you modify their downstream library mapping to use a writable downstream directory.

Refer to the sections [“Using the Example VHDL Design” on page 1-89](#) or [“Using the Example Verilog Design” on page 2-90](#) for more information.

# Chapter 1

## VHDL Timer Exercise

This exercise creates a simple timer using [block diagrams](#) and a control [block](#) described as a hierarchical [state machine](#). A simple [truth table](#) is used to decode four-bit binary codes from the ten-bit input bus. The design is completed using a re-usable [component](#) described by a [HDL text](#) view.

A [test bench](#) is created using a [flow chart](#) which can be used as a test harness to simulate the generated VHDL for the timer design. The simulation results can be displayed as animation on the flow chart and state machine to assist in debugging the design. The verified timer design is then synthesized.

The instructions assume that a ModelSim simulator and the LeonardoSpectrum synthesis tools are available. However, the VHDL generated from the diagrams can also be used by other compatible downstream tools that are available on your system.

## Specification

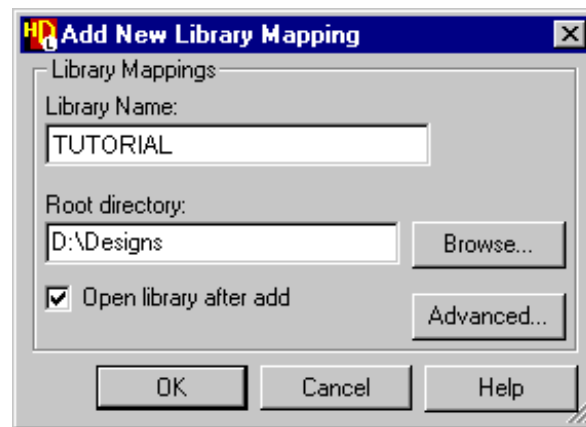
The timer outputs time data on two four-bit buses representing low and high values. There is also a logic output signal which triggers an audible alarm. The data input is provided on a ten-bit bus and control is provided by start, stop, reset and clock signals. These signals are summarized in the following table:

<b>Inputs</b>	<b>Outputs</b>
start (logic signal)	high (4-bit bus)
stop (logic signal)	low (4-bit bus)
reset (logic signal)	alarm (logic signal)
clk (logic signal)	
d (10-bit bus)	

## Set Library Mapping

A **library** is the logical location of the directories containing your design data. The source design, generated HDL and downstream data can be stored at any writable locations on your available file system but is typically saved below a common root directory.

To set library mapping, choose **New Library** from the **File** menu in the **design browser** window to display the Add New Library Mapping dialog box.

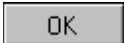


Enter a logical library name (for example, *TUTORIAL*) and specify the pathname for the root directory that will contain your library data (for example, *D:\Designs*).



By default, the root directory is set to `..\hds_scratch` which is created at `$HOME/hds_scratch` on UNIX or `<install_dir>\examples\hds_scratch` on Windows.

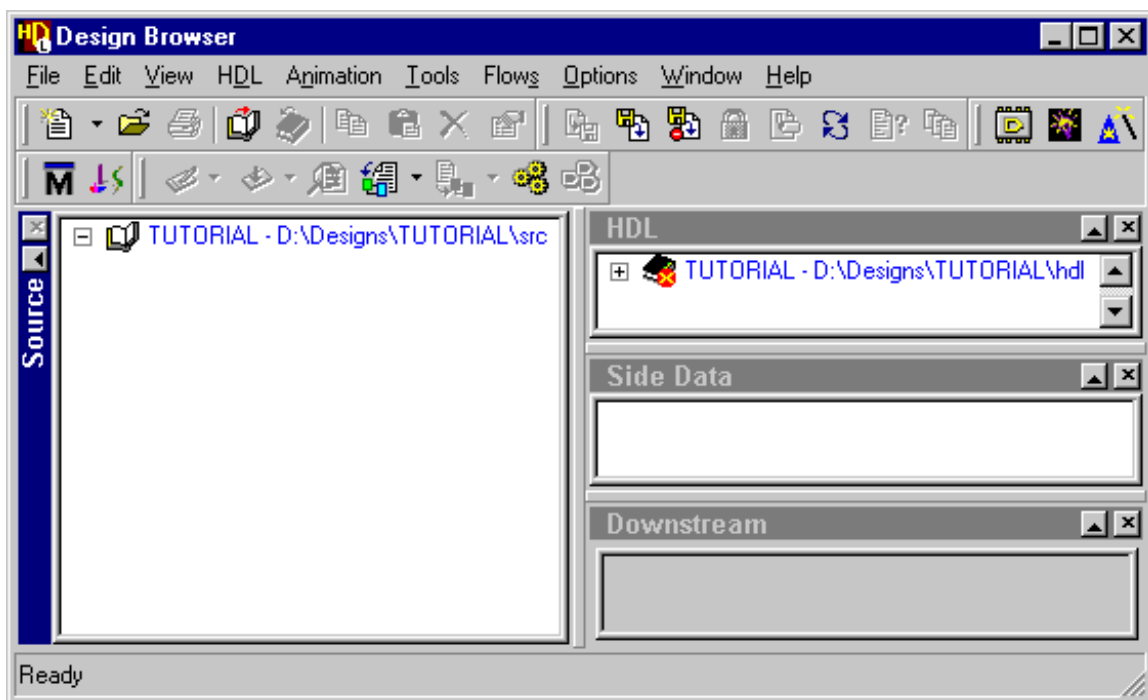
Library names and pathnames can be entered using upper, lower or mixed case but note that UNIX systems are case sensitive and the case used for pathnames should match the file structure. (On a PC, library names are case sensitive but pathnames are case insensitive.).

Check that the **Open library after add** option is set and use the  button to create and open the new library. Notice that the source design data directory is named `<root_directory>\TUTORIAL\src` and that the generated HDL directory is named `<root_directory>\TUTORIAL\hdl`.

The source design data directory is created (if it does not already exist) when you save a **design unit** to the *TUTORIAL* library (provided that the **parent** directory exists and you have write access to create a subdirectory at the specified location). The generated HDL directory is created when you generate HDL for the design. The mapping for the location of compiled data is defined automatically when you set up a downstream tool and the directories created when the design is compiled.

Your library definition is saved in an initialization file which is automatically saved in your **user directory** with the file name *hds.ini* and is read the next time that the HDL Designer Series tool is invoked.

The Source design data directory is displayed in the design browser as an open "book" and the HDL directory as a closed book similar to the following picture:

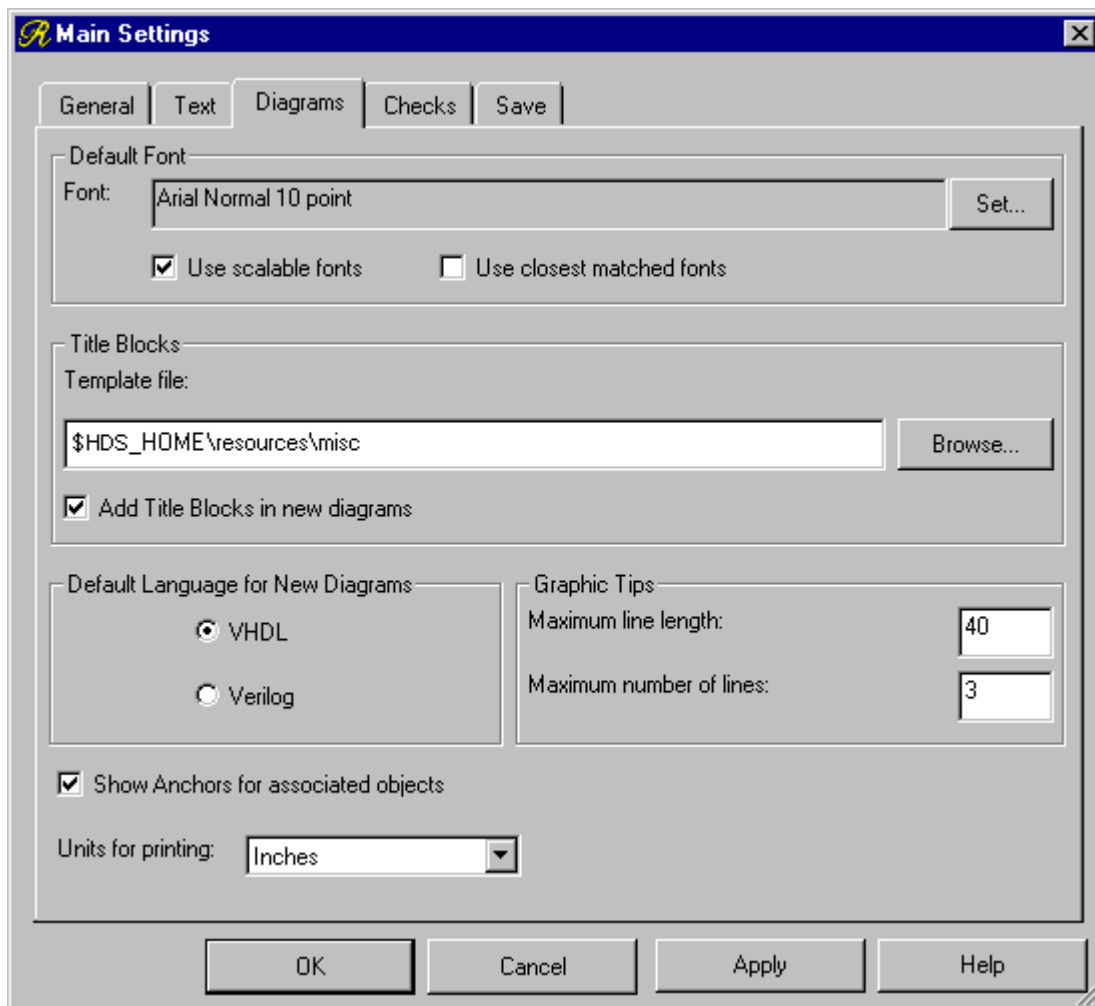


The Source and HDL pathnames are shown in blue text because the directories do not exist yet but will be updated in the browser when you create them by saving design units and generating HDL for the library. The Side Data and Downstream sub browsers are both empty at this stage unless any other libraries are open.

## Set the Default Language

A set of default preferences are loaded when you invoke a HDL Designer Series tool for the first time. There are separate tabbed dialog boxes for the main settings, VHDL and Verilog options, compile settings, HDL import options, version management settings and master preferences for each type of graphical diagram. The preference dialog boxes can be accessed from the **Options** menu.

Choose **Main** from the **Options** menu to display the Main Settings dialog box, select the **Diagrams** tab and ensure that **VHDL** is set as the default language to be used for new [graphic editor](#) views. Use the  button to confirm your choice.



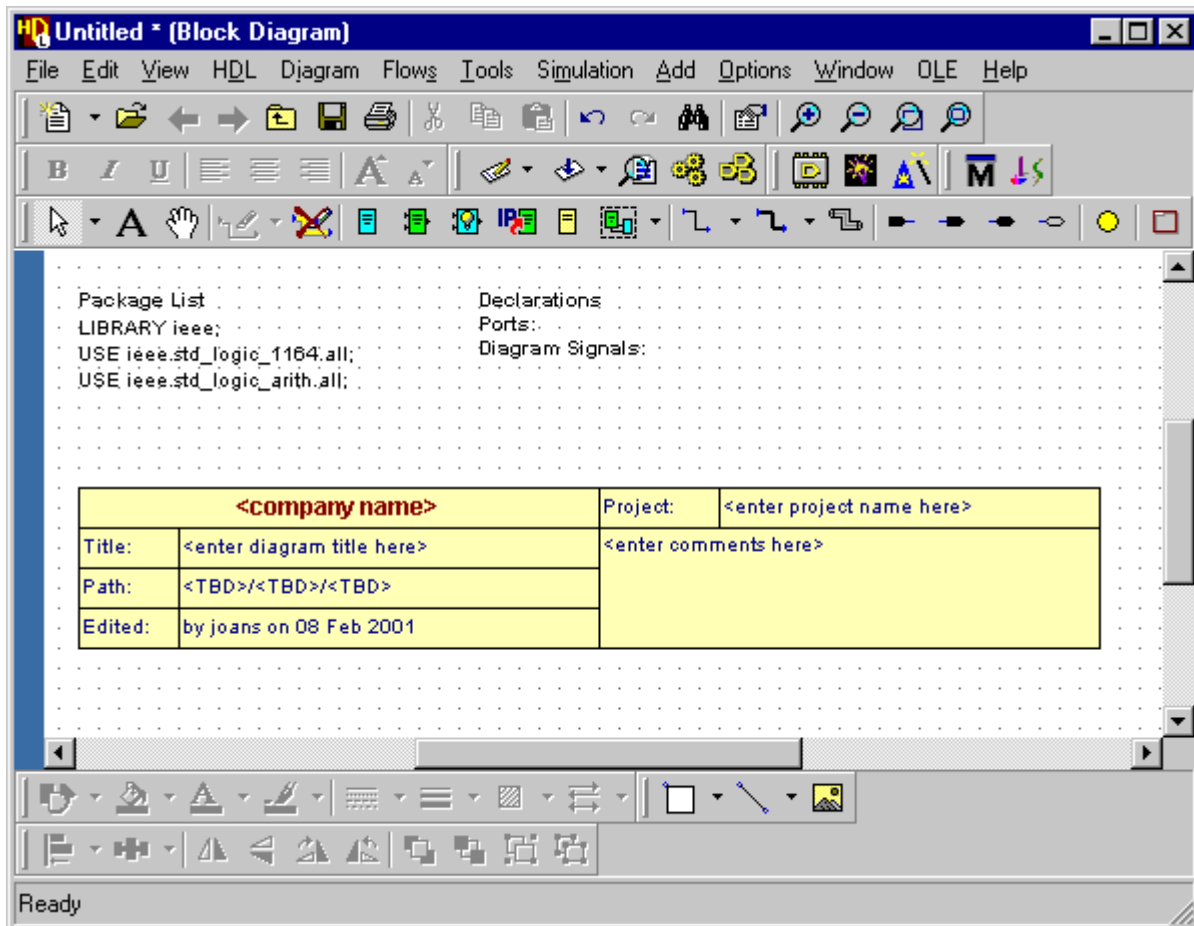
All other preferences can be left with their default values for this tutorial.



## Create a Block Diagram

Use the  button in the design browser window and select **Block Diagram**.

A new untitled [block diagram](#) is created.



The block diagram is a blank sheet except for a background grid, a [package list](#) (with the standard IEEE libraries *std\_logic\_1164* and *std\_logic\_arith* set by default), a copy of the default title block, and empty text fields with labels for Declarations, Ports and Diagram Signals.

Notice the five toolbars at the top and three toolbars at the bottom of the diagram. The toolbar buttons provide quick access to many of the most frequently used editing and formatting commands.

## Edit the Title Block Template

A title block is automatically added to all new diagrams if the **Add Title Blocks in new diagrams** option is set in your diagram preferences. The default title block is a template with default text for your company name, project name, diagram title and comments. The default title block incorporates internal variables which automatically enter your login name and the current date. Internal variables are also used to enter the logical pathname for the design. This path is initially shown as <TBD>/<TBD>/<TBD> but the internal variables are converted to show the library, design unit and view name when you save the diagram.

Click twice on <company name> in the default title block to display a popup edit box and replace the default text by the name of your company. Click twice on <enter project name here> and enter a name for your project (for example, HDL Designer Tutorial).

Display the **Diagrams** tab of the Main Settings dialog box (as described in “[Set the Default Language](#)” on page 1-4) and change the **Template file** pathname to a write-able location such as your [working directory](#) or home directory.

Select the title block by clicking with the mouse so that the selection handles are displayed and choose **Save Title Block** from the **File** menu.


<b>Mentor Graphics</b>		Project:	HDL Designer Tutorial
Title:	<enter diagram title here>	<enter comments here>	
Path:	<TBD>/<TBD>/<TBD>		
Edited:	by joans on 08 Feb 2001		


A dialog box is displayed with a warning that saving the title block cannot be undone. Click the  button to proceed.

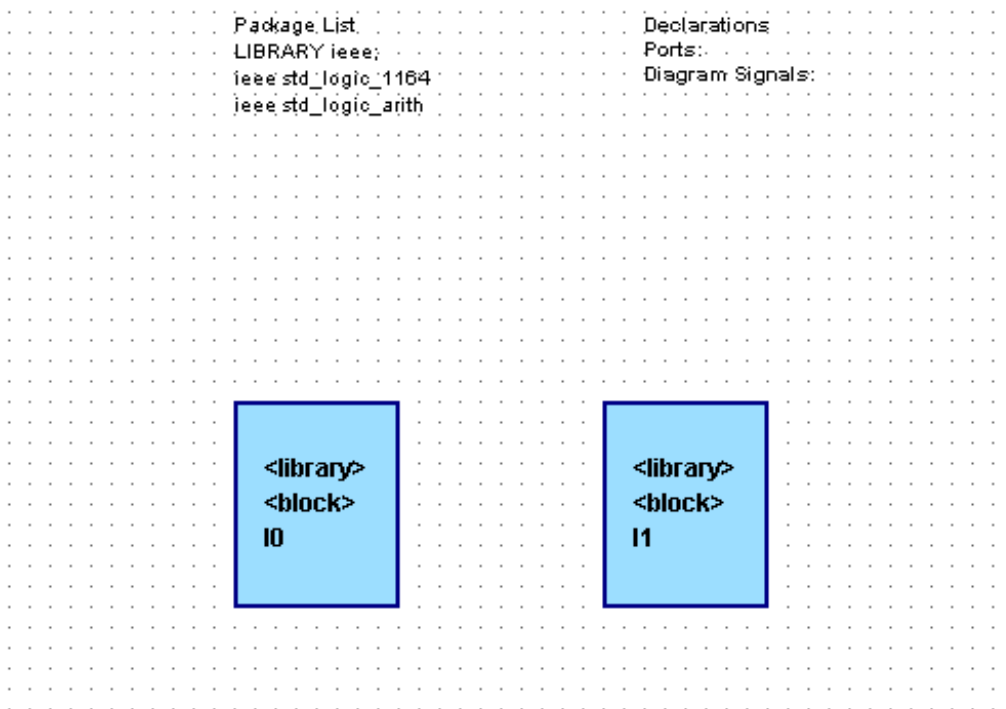
The title block is saved at the new location specified in your preferences and will be used as the default template in new diagrams.


Refer to the online help topics for information about how you can add and modify title blocks.




## Add Blocks

Move the title block to the bottom of your block diagram by dragging it with the  mouse button.


Use the  button to add two **blocks** on the diagram as shown in the picture below. The blocks are added with the default library `<library>`, the default name `<block>` and unique instance names (*I0* and *I1*)

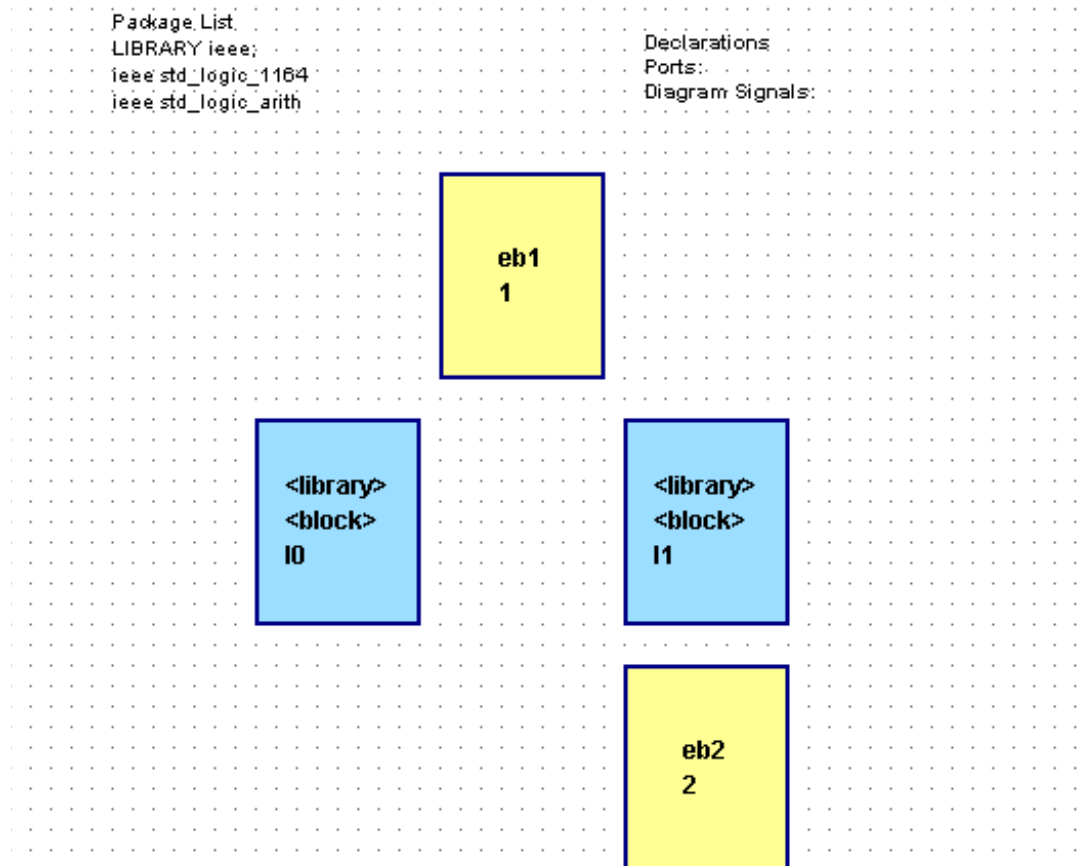


The command normally auto-repeats until you select another command or terminate the repeating command by using the right mouse button or  key. However, you can change the behavior of the toolbar buttons by setting the **Activate once only** preference in the **General** tab of the Main Settings dialog box as described on [page 1-6](#).

You can also use the  key with any toolbar button to toggle the repeat mode. For example, when **Remain active** is set,  +  adds a single block on a block diagram.

## Add Embedded Blocks

Use the  button to add two **embedded blocks** on your block diagram as shown in the picture below. The embedded blocks are added with unique default names (*eb1* and *eb2*) and numbers (*1* and *2*)


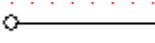

















The view describing a block must be saved as a uniquely named **design unit** in a library directory. However, the view describing an embedded block is saved as part of the parent block diagram and does not impose hierarchy when HDL is generated for your design. The name of an embedded block must be unique on the diagram and is used as a label in the generated HDL.


The blocks (*I0* and *I1*) will be used to define a child state machine and block diagram view. The embedded blocks (*eb1* and *eb2*) will be defined by concurrent assignment statements on the top level block diagram.

## Add Ports and Signals


You can use the following buttons to add [signals](#), [buses](#) and [ports](#) on a block diagram:

	Add a signal without a port	
	Add a signal with a port	
	Add a bus without a port	
	Add a bus with a port	
	Add an input port	
	Add an output port	
	Add a buffer port	
	Add a bidirectional port	

Signals or buses can be added between any existing [connectable items](#) on the diagram or left unconnected by double-clicking to terminate the [net](#) with a dangling [net connector](#). However, you can use the  pulldown on the buttons to change the default setting and terminate with a default port at unconnected end points. Notice that the toolbar button changes to show the current setting.


Choose **Signal with Port** and use the  button to connect three signals originating from the block on the left (instance *I0* in the picture) to the block on the right (instance *I1*) and one signal returning from *I1* to *I0*. The signals are added with unique names (*sig0*, *sig1*, *sig2* and *sig3*) and the default [type](#) *std\_logic*. Notice how declarations are automatically added to the list of Diagram Signals.



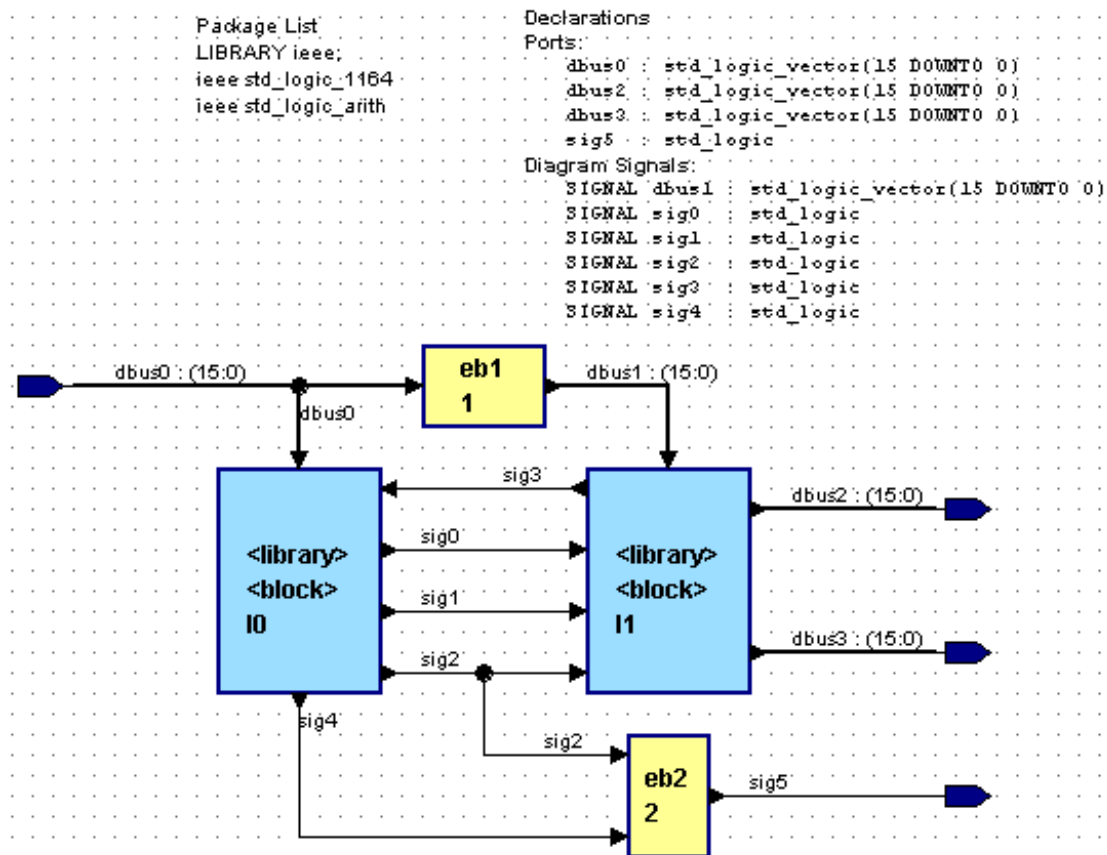
Allow two or more grid lines between each port or signal. You can resize objects by selecting a block or embedded block and dragging one of its resize handles. If necessary, you can drag text elements such as the signal name using the  mouse button.

Add a signal from *I0* to the embedded block *eb2* and another signal from a point on *sig2* terminating on the embedded block.


Add a signal from the embedded block terminating in space on the right side of your diagram. Notice that an output port is automatically added when you double-click at the end of the last signal and its declaration is added to the list of ports.



Choose **Bus with Port** and use the  button to add a bus from a **source** on the left side of your diagram with its **destination** on the upper embedded block *eb1*. A default input port is automatically created at the beginning of the bus. Add another bus starting from this bus and terminating on instance *I0*. Notice how both bus segments have the same default name *dbus0* but the default **bounds** (*15:0*) is shown (in abbreviated format) only on the first bus segment. The full declaration showing the default bus type and bounds *std\_logic\_vector(15 DOWNTO 0)* is added to the list of ports. See the online help topic Changing the Display of Signal Properties for information about the format for displaying signals and buses.



Add a bus (*dbus1*) from *eb1* to *I1*. Then add two buses (*dbus2* and *dbus3*) from *I1* terminated with default output ports by double-clicking on the right side of the diagram. Your diagram should now look similar to the picture below:

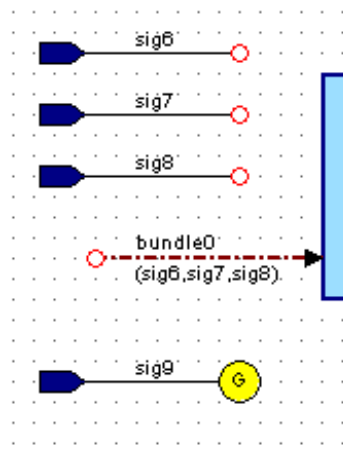




## Add a Bundle and Global Connector

Use the  button to add three signals on the left side of your diagram. Notice that a default input port is created at the source of each signal but a dangling [net connector](#) is drawn when you double-click at the end of each signal.

Select the three signals (by dragging a select rectangle with the  mouse button held down) and use the  button to connect a [bundle](#) containing these signals to block instance *I0* as shown in the picture below. Notice that the bundle has the default name *bundle0* and the three selected signals are automatically included in the bundle with their names listed under the bundle name.


Use the  button to add a [global connector](#) on your diagram below the bundle and use the  button to add a signal between the global connector and a default input port. (This will be a clock signal which is implicitly connected to every block on the diagram.)

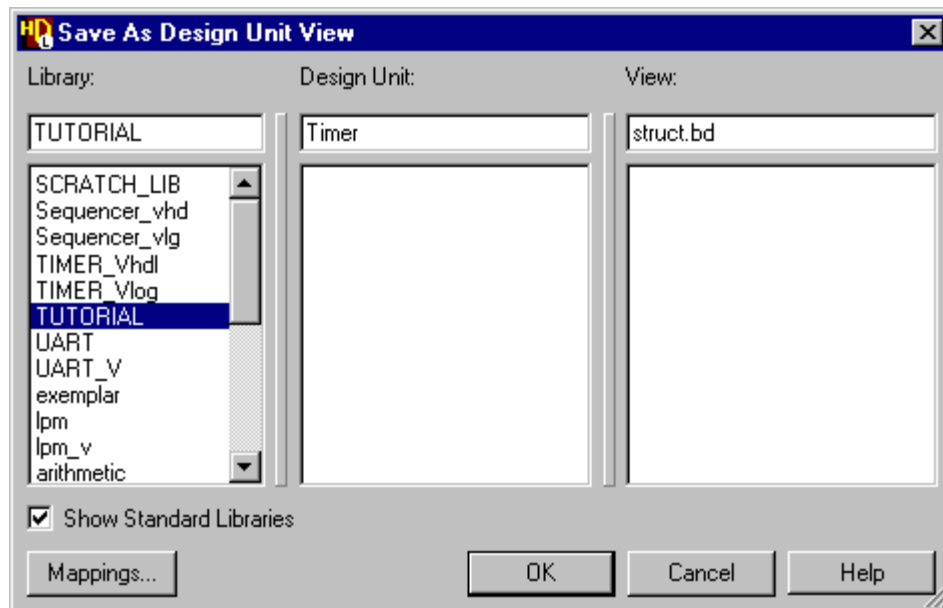


If you make a mistake when editing a diagram, you can use the  button to undo your last edit and the  button to redo an undo operation. You can also use commands from the **Align** cascade of the **Edit** menu to re-align and distribute objects on the diagram.

## Save the Block Diagram

Notice the asterisk (\*) character in the header of the block diagram editor window. This indicates that the diagram has been edited since it was last saved.

Use the  button to save the block diagram. The Save As dialog box is displayed which allows you to choose from the currently mapped libraries and specify the **design unit** and **design unit view** names. Choose the *TUTORIAL* library and enter design unit *Timer*. The dialog box should look similar to the example below:




The view name can be entered using any valid HDL identifier but normally defaults as follows:

struct.bd	block diagram
struct.ibd	interface-based design view
fsm.sm	state diagram
flow.fc	flow chart
tbl.tt	truth table
symbol.sb	symbol






If you omit the two-character extension it is automatically added to identify the type of diagram you are saving. The default leaf names can be changed by setting preferences. However, you should not change the extension (*.bd*, *.ibd*, *.sm*, *.fc*, *.tt* or *.sb*) or the design data file will not be recognized and cannot be reopened.

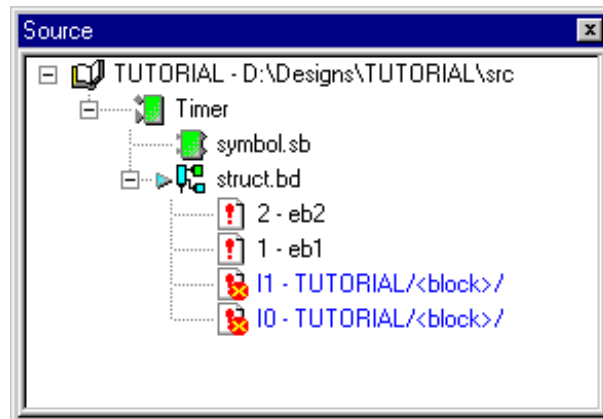



When you click the  button, your diagram is saved and the window title bar is updated to show the diagram pathname *TUTORIAL\Timer\struct*. This path is also added in the title block replacing the <TBD> used when the diagram was created. Notice that the asterisk (\*) character has been cleared in the block diagram header and the library name used on the blocks in your diagram has been updated to *TUTORIAL*.



If the design browser window is obscured, you can pop it to the front by selecting the **Design Browser** window from the **Windows** menu list in the block diagram window.

Click on the  icon for the *TUTORIAL* library in the [source browser](#) and notice that the view is expanded to display the *Timer* design unit. Click on the  icon for the *Timer* design unit to reveal that it contains a [symbol](#) and block diagram view. Click on the  icon for the *struct.bd* view to display the hierarchy of views instantiated as blocks and embedded blocks on the block diagram. The embedded blocks (*eb1* and *eb2*) are shown using the  icon to indicate that no views have been defined. The blocks (*I0* and *I1*) are also shown as undefined but with blue text and an  overlay indicating that no design units exist for their child views.

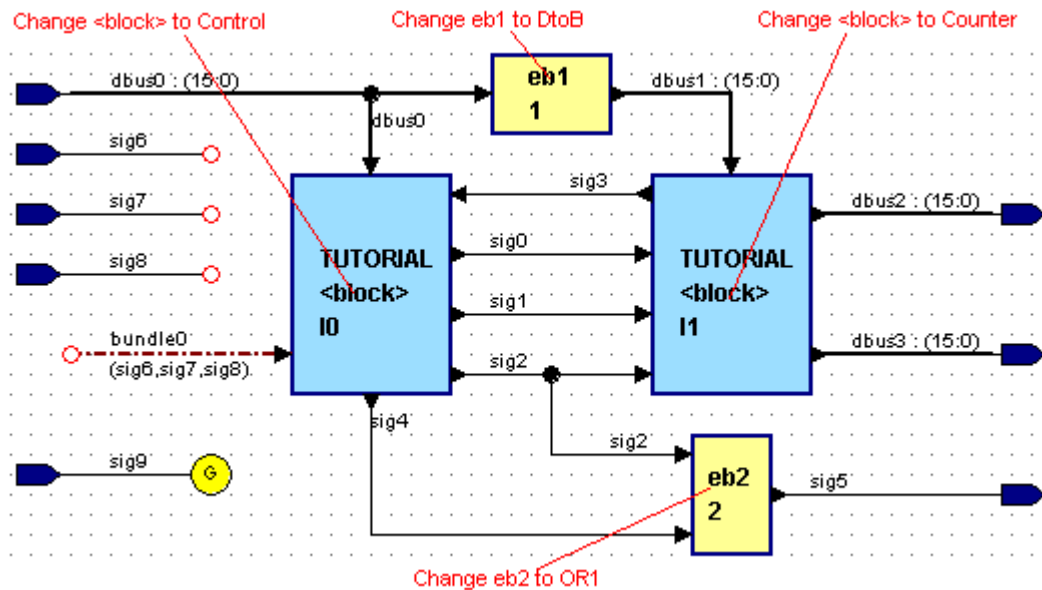


You can change the design browser layout and undock the source browser (shown above), [HDL browser](#), [side data browser](#) or [downstream browser](#) from the main window. Blue text and an  overlay in the source browser indicates that a view is not write-able. (In this case, because design units have not been created for the blocks. This convention is also used to show when you have read-only access.)

## Edit Block and Signal Names

You now have a completed top-level block diagram for the *Timer* design. However, the blocks and signals have default names.

Click on the text `<block>` in the lower block on the left (instance *I0* in the picture) and notice the small handles which indicate that the text object is selected. Click again and notice that the text is now highlighted and can be directly overwritten. Type the new name *Control* and click outside the text to complete the edit. Repeat this procedure to change the name of block instance *I1* to *Counter*, embedded block *eb1* to *DtoB* and embedded block *eb2* to *OR1*.




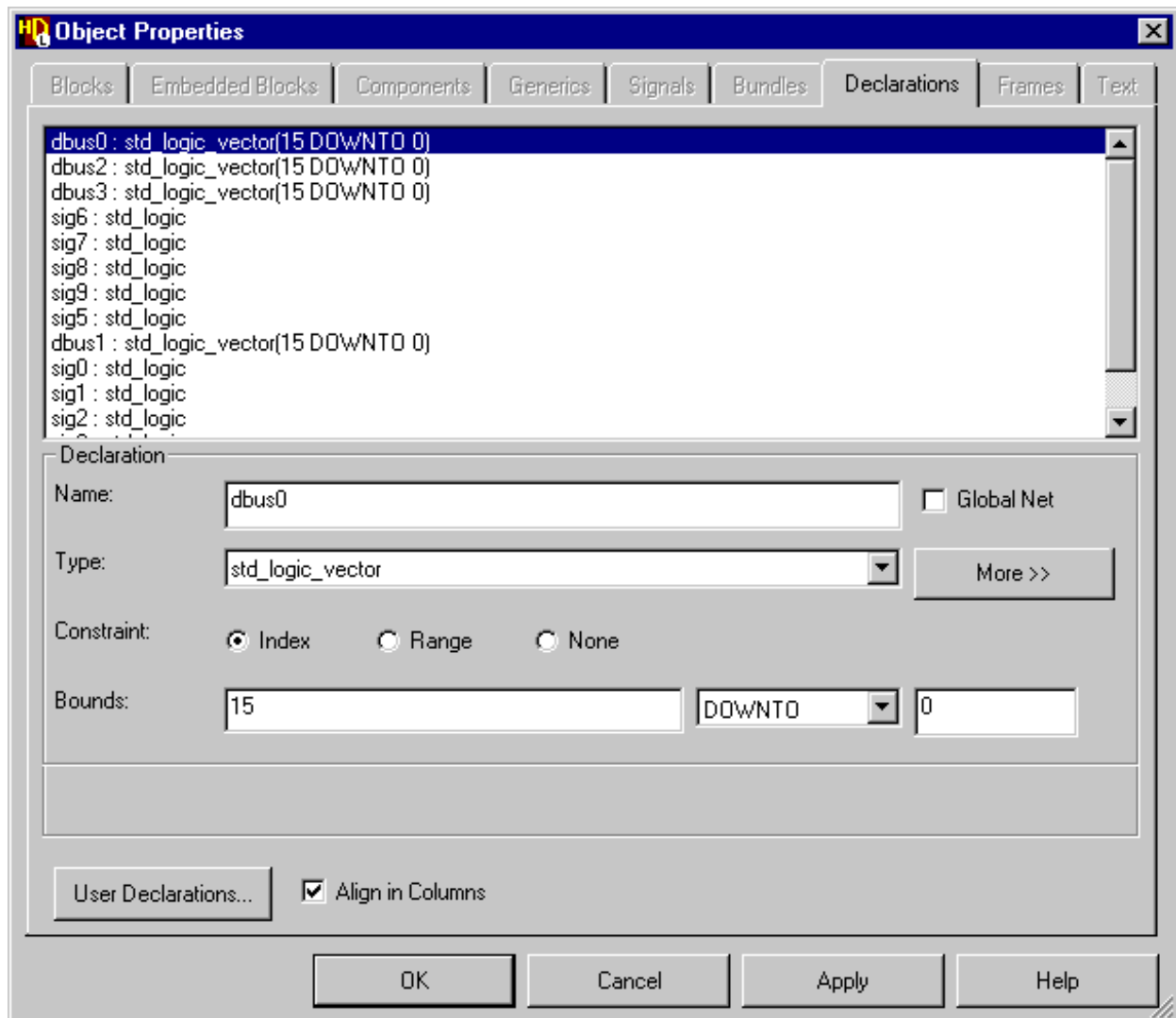
The text can be overwritten when it is highlighted. If you click again, the cursor changes to an I-beam which allows you to move the cursor in the text and edit individual characters.

While a text element is selected, an **anchor** that attaches the text to its associated object is visible and you can move the text independently to improve diagram clarity.

This text editing technique can also be used to edit the signal and bus names.

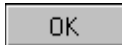
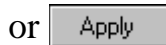
Alternatively, you can use a dialog box which allows you to edit the properties for a selected object.

Double-click on the existing declarations, use the  button or choose **Object Properties** from the **Edit** menu to display the Block Diagram Object Properties dialog box and choose the **Declarations** tab.



Notice that the port declarations are listed at the top of the dialog box and the other internal diagram signals at the bottom. Input ports are listed before the output ports, otherwise the declarations are listed in alphanumeric order.

You can choose one or more existing declarations in the dialog box and enter new values for any of the declaration fields. For example, choose `dbus1`, `dbus2` and `dbus3`, then enter a new *index* constraint with **bounds** `3 DOWNTO 0` to update all three buses while all other fields remain AS IS.

The changes are applied to the diagram when you click the  or  button. Notice that all occurrences on the diagram are updated including the declarations list, signals, buses and bundle contents and that the lists of port and signal declarations are sorted alphanumerically when the changes are applied to the diagram.

Use the dialog box to update the port and signal declarations as shown in the following tables.

Ports:

Old Name	New Name	Type	Constraint	Bounds
dbus0	d	std_logic_vector	index	9 DOWNTO 0
dbus2	low	std_logic_vector	index	3 DOWNTO 0
dbus3	high	std_logic_vector	index	3 DOWNTO 0
sig6	start	std_logic	none	none
sig7	stop	std_logic	none	none
sig8	reset	std_logic	none	none
sig9	clk	std_logic	none	none
sig5	alarm	std_logic	none	none

Diagram Signals:



Old Name	New Name	Type	Constraint	Bounds
dbus1	dat_in	std_logic_vector	index	3 DOWNTO 0
sig0	clear	std_logic	none	none
sig1	load	std_logic	none	none
sig2	hold	std_logic	none	none
sig3	zero	std_logic	none	none
sig4	beep	std_logic	none	none



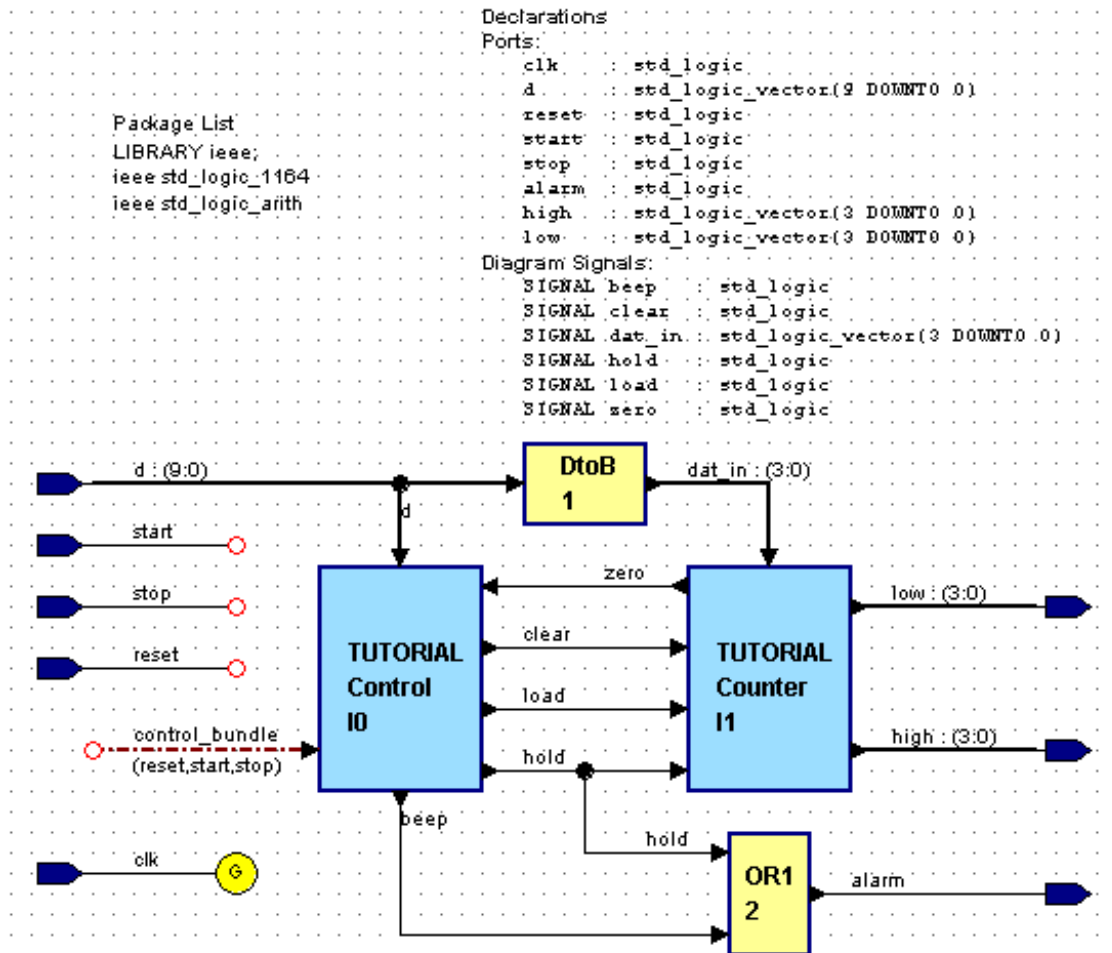
All occurrences of each signal name (including the bundle contents) are automatically updated when you use the **Declarations** tab. However, if you use, direct text editing (or the **Signals** tab) to change signal names, it may be necessary to check that all occurrences have been updated.



Select the bundle name and use direct text editing or the **Bundles** tab of the Object Properties dialog box to change the default bundle name to *control\_bundle*.




You can change the selection mode to select text or object shapes only by using the  pull-down menu on the  button.

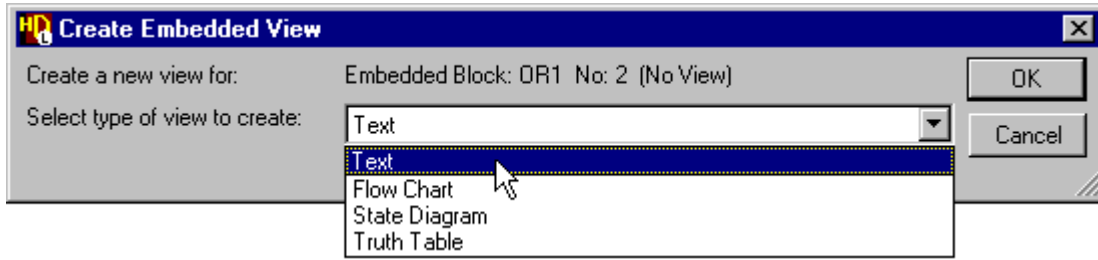
Your block diagram should now look similar to the following picture:




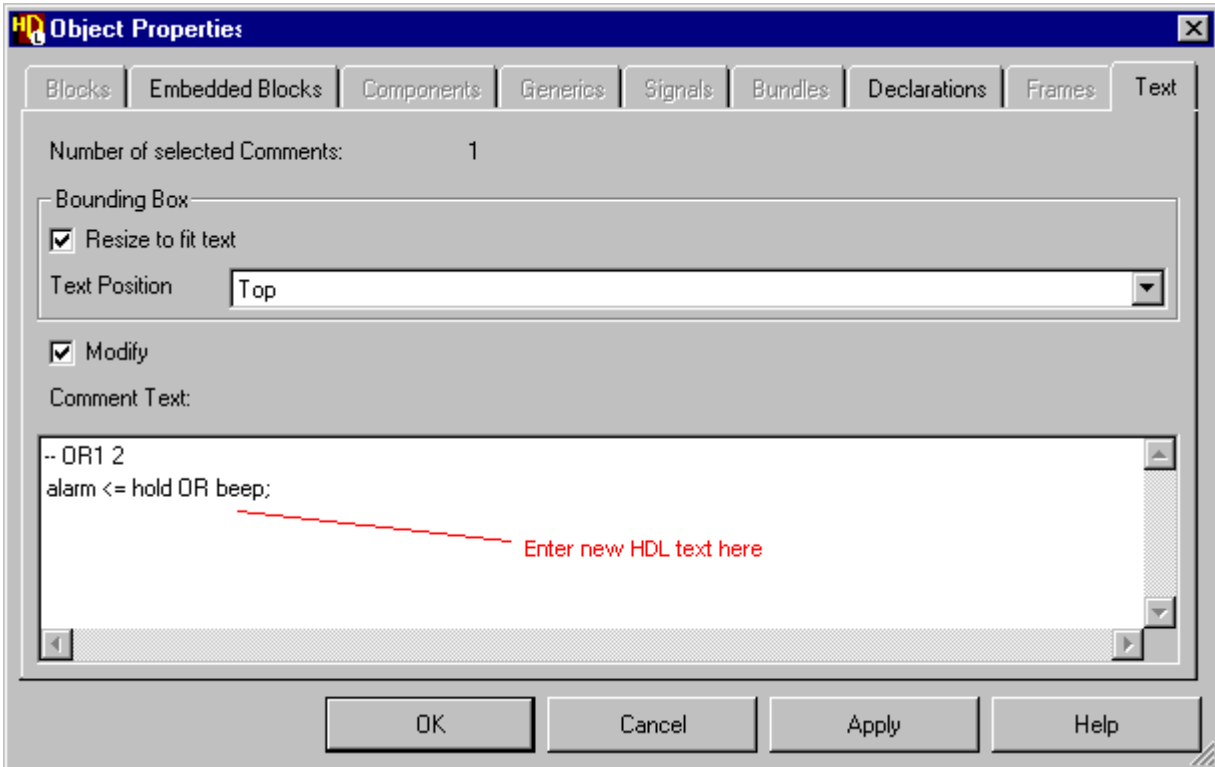
The  button on the dialog box allows you to disclose additional fields which allow you to modify other signal properties including 2D bounds, initial value, kind, VHDL attributes and synthesis constraints. The  button allows you to add additional user-entered [architecture declarations](#) to the structural VHDL. Refer to the “Editing VHDL Signal Declarations” help topic for more information about these features which are not used in this tutorial.

## Add an Embedded HDL Text View

Select the *OR1* embedded block and display the popup menu by clicking the  mouse button. Choose **New View** from the **Open** cascade in the popup menu to display the Create Embedded View dialog box. Choose Text from the pulldown list of views in the dialog box.



An embedded **HDL text** view containing default text is displayed on the block diagram adjacent to the embedded block. Select the text, re-display the Object Properties dialog box if necessary (using the  button) and choose the **Text** tab.



Check the **Resize to fit text** option and enter the following VHDL statement under the default `-- OR1 2` comment in the dialog box:

```
alarm <= hold OR beep;
```

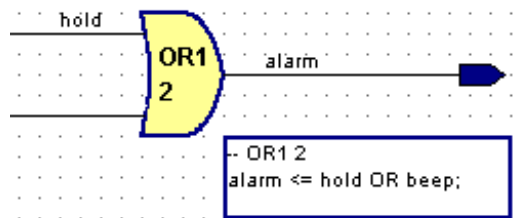
The modified HDL text is checked for syntax errors and applied to the diagram when you click the  or  button on the dialog box.

The functional blocks on the diagram are shown by default as simple rectangular shapes. However, it is sometimes useful to use logic notation when a block has a specific logical function. For example, in this block diagram, the *OR1* embedded block represents a logical OR function.

Select the embedded block and choose the pulldown  on the  button to display a palette of alternative shapes. Select  from the palette to apply a logical OR shape to the embedded block on the diagram.

You can also hide the port arrow heads by clearing the **Show Ports when connected** check box in the **Embedded Blocks** tab of the Object Properties dialog box.

The *OR1* embedded block should now look similar to the following picture:





- i It is also possible to indicate an active low (Not) or edge triggered clock signal. This feature can be used with the alternative shapes to represent extra functions such as an inverter, NAND, NOR or flip-flop. If required, you can rotate any block or component by 90 degree steps.

Refer to the “Logic Shape Notation” help topic for more information about these features.

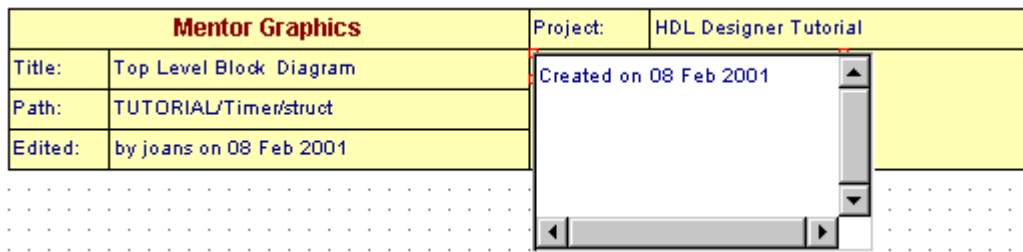
- i The logical OR function could also be implemented by a ModuleWare component similar to that used in [“Add ModuleWare Components”](#) on page 1-51.


## Add a Panel and Edit the Title Block

Use the  button to add a panel and hold down the  mouse button to drag the panel and enclose the graphical objects on your diagram. The panel is added with the default name `Panel0` and can be useful to outline areas of a diagram. For example, you can divide a large diagram into separate printable page-sized areas or use a panel to outline a view used for simulation or animation.


Complete the block diagram by editing the title and comments in the title block on the diagram. For example, enter the title `Top Level Timer Block Diagram` and a comment of the form: `Created by <your name> on <date>`.


The title block comprises a number of grouped [comment text](#) objects. Each comment text object can be edited directly by clicking twice on the text to display a text entry box.



 You can enter free-format text including line breaks and spaces which will be preserved on the diagram.

Click the  mouse button outside the entry box to complete the text entry.

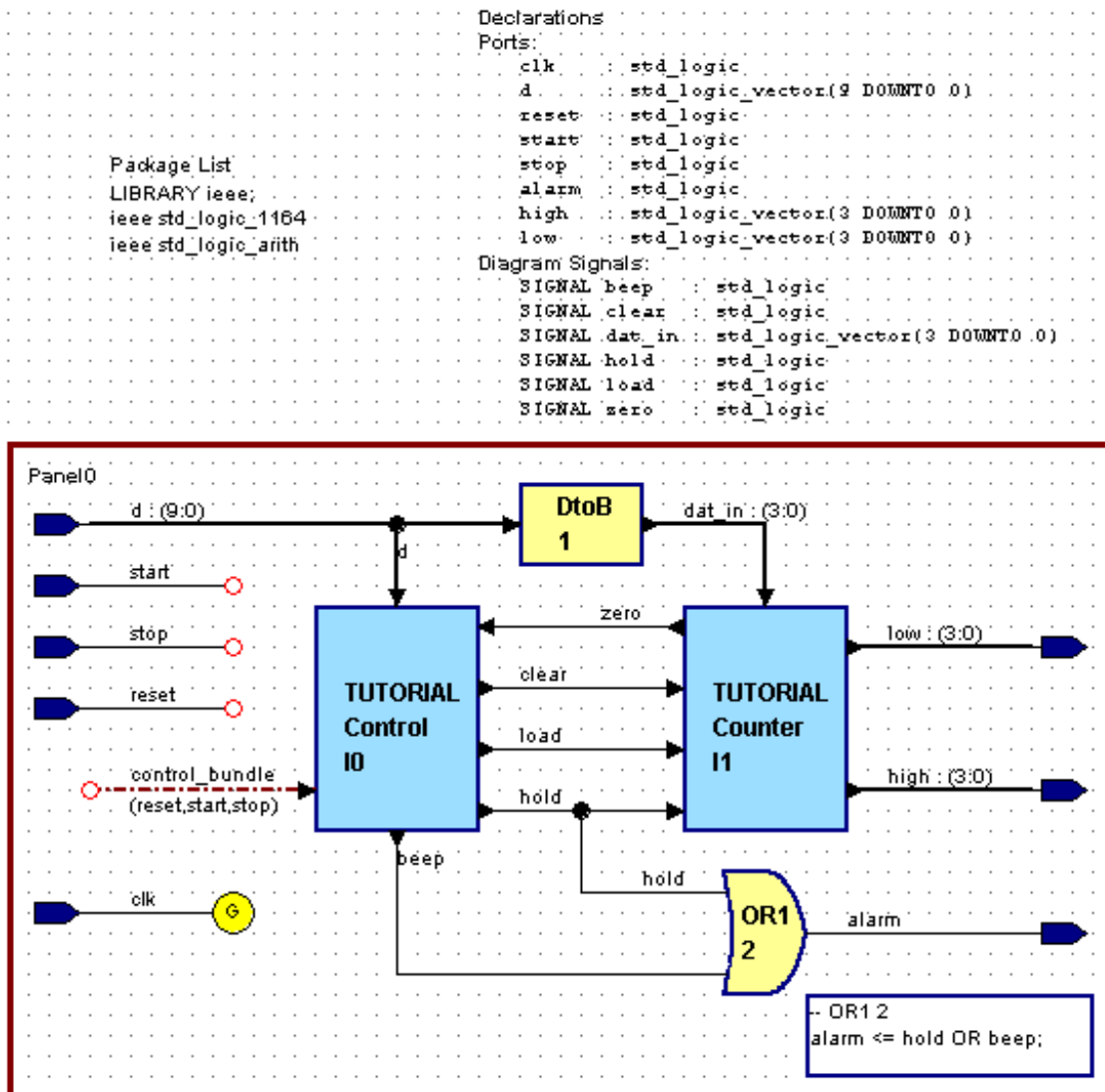
 You can also edit an existing comment text object by double-clicking to display the **Text** tab in the Object Properties dialog box. When comments are edited in the dialog box, it is possible to enter any special characters (for example accents or Kanji characters) which are supported on your system.

Use the  button to save the block diagram.



You have previously saved the diagram so you are not prompted for library and design unit names. However, you have changed the names of signals connected to input and output ports and the block diagram will be inconsistent with the [symbol](#) that was automatically created by the previous save. You are prompted whether to update the symbol. Click the  button to confirm.

The completed block diagram should look similar to the following picture:



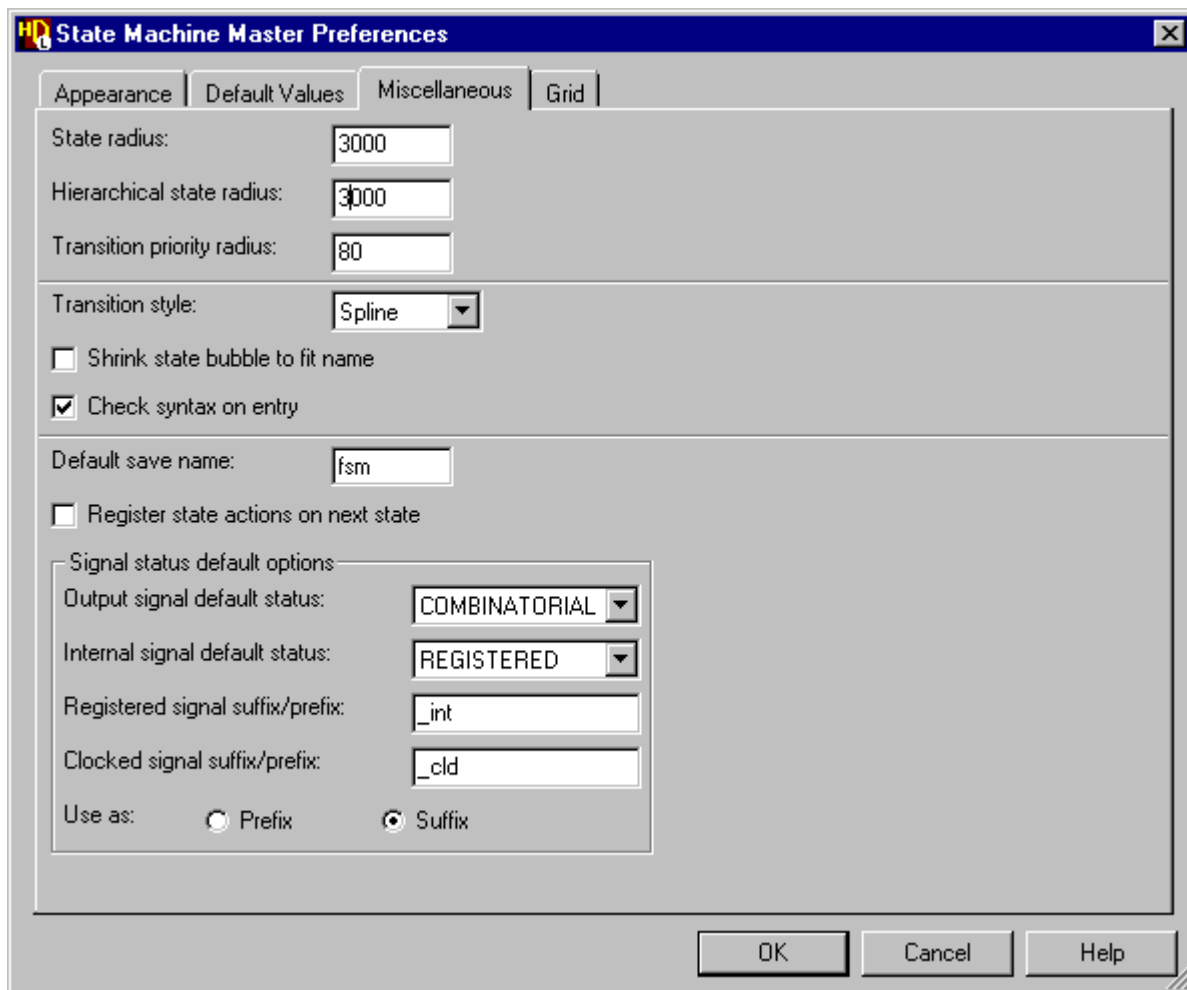
<b>Mentor Graphics</b>		Project:	HDL Designer Tutorial
Title:	Top Level Block Diagram	Created on	08 Feb 2001
Path:	TUTORIAL/Timer/struct		
Edited:	by joans on 08 Feb 2001		

The procedures in the following sections create a graphical state machine to describe the *Control* block. If you are using one of the HDL text design tools, refer to [Appendix A](#) for an alternative HDL text view of the *Control* block.

## Set State Machine Preferences

You will create a [state machine](#) view in the next procedure. You can set master preferences which modify the way new diagrams are drawn. As an example, for this tutorial it is suggested that you reduce the default size used for a [state](#).

Choose **State Machine** from the **Master Preferences** cascade of the **Options** menu in the design browser to display the State Machine Master Preferences dialog box and select the **Miscellaneous** tab:



You can set the minimum radius for states, [hierarchical states](#) and the [transition priority](#) object. The states will auto-size if the state name is larger than can be enclosed by the minimum radius. However, the minimum state size is overridden if you check **Shrink state bubble to fit name** which allows the states to shrink below this size if the state names are short.

Transitions on a [state diagram](#) are normally drawn with curved [splines](#) instead of the orthogonal [polylines](#) used for signals on a block diagram. However, the [transition](#) style can be changed to straight polylines.

Syntax checking on entry can be enabled or disabled (for example, if you want to enter non-HDL identifiers or comments while drafting a diagram). You can choose to register state actions on the next state instead of the current state.

You can also specify the default leaf save name for state diagrams, the default status for output and internal signals and the prefixes or suffixes used for the internal names of registered or clocked signals.

Change the [state](#) radius and [hierarchical state](#) radius values to 3000. This radius should be sufficient to enclose the state names used in this tutorial.

Examine the other preferences available in each tab of the State Machine Master Preferences dialog box.

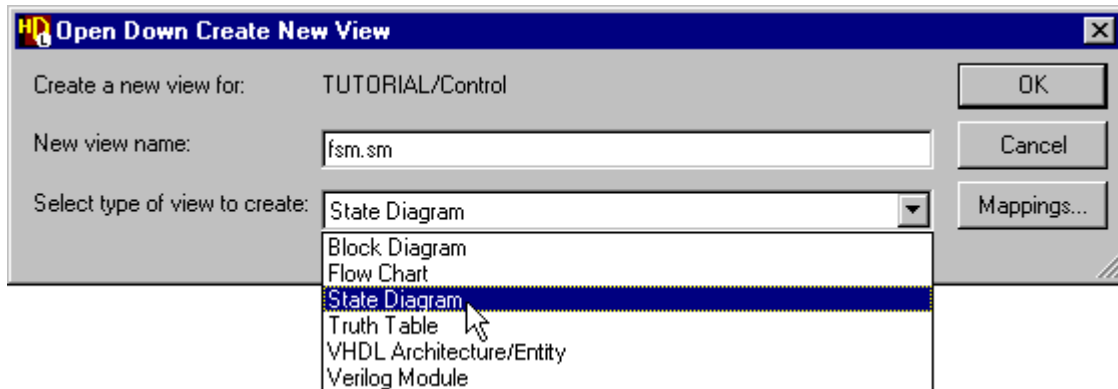
Use the  button to set the changed preference. You are prompted to confirm the change which will be used in the master preference next time you open a state diagram. Click the  button to confirm.



Notice that you can also display dialog boxes which allow you to set the master preferences for the block diagram, symbol, flow chart and truth table editors. You can use the  buttons in each dialog box for more information about these preferences. However, you are advised not to change any of the other preferences before you have completed this tutorial.

## Create a Child State Diagram

Move the cursor over the body of the *Control* block on the *Timer* block diagram, then press and release the **Right** mouse button to select the block and display the popup menu. Choose the **Open** cascade menu option **New View**. The Open Down Create New View dialog box is displayed:



Use the pulldown list to select the type of view you want to create. The view name defaults to *struct.bd* for a **block diagram**, *struct.ibd* for an **Interface-Based Design (IBD)** view, *flow.fc* for a **flow chart**, *fsm.sm* for a **state machine**, *tbl.tt* for a **truth table** or *untitled* for a **VHDL architecture** or **Verilog module** view. Alternatively, you can enter any other name of your choice or use the **Mappings...** button to modify the mapping for the current library.



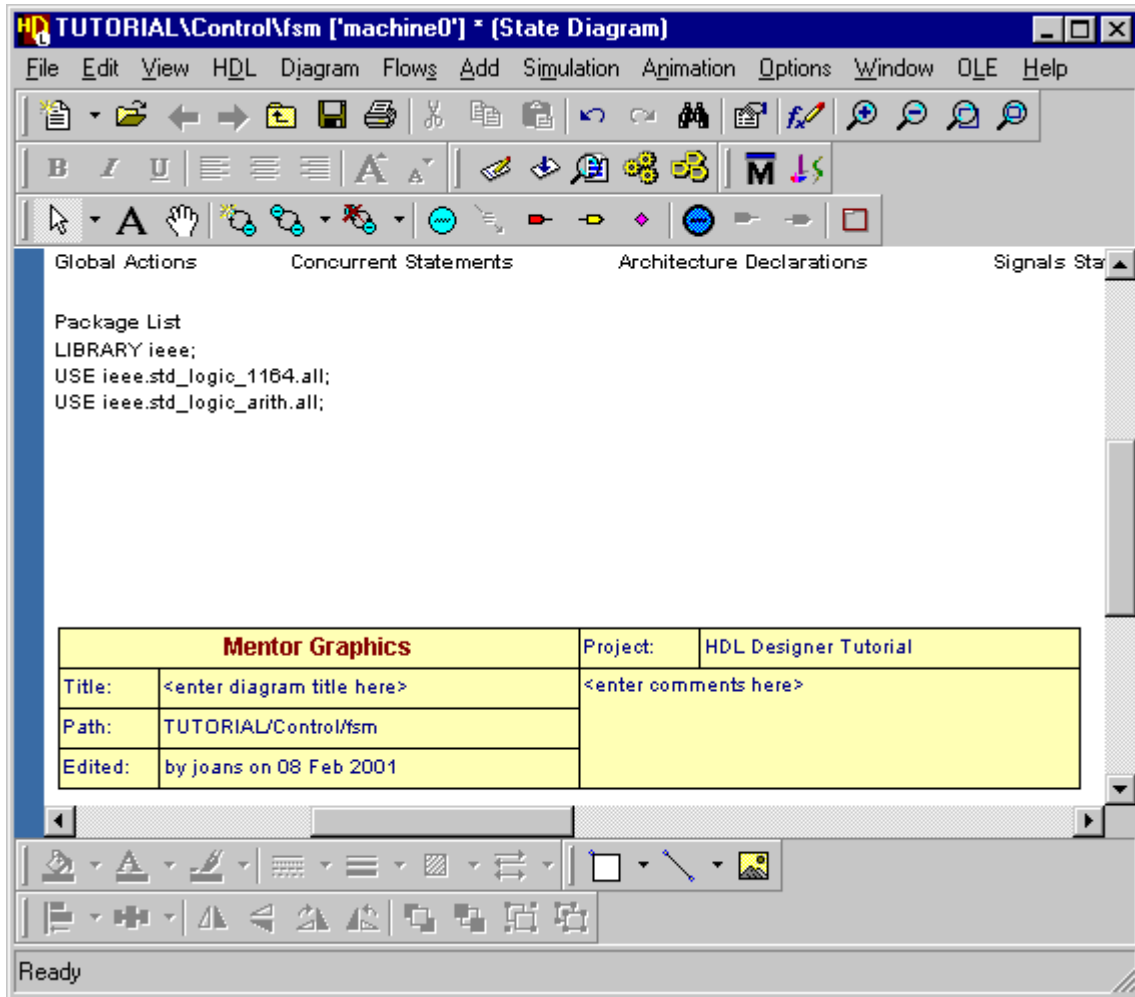
You need not enter the two character extension (*.bd*, *.fc*, *.sm* or *.tt*) for graphical views as the correct extension is automatically added. However, if you do enter any other extension you are warned that the file will not be recognized.

Select **State Diagram** from the pulldown list of views you can create and use the default view name *fsm.sm*.



Solid handles are displayed when the body of a block (or other re-sizable object) is selected. You can display the Open Down Create New View dialog box directly by double-clicking on the body of a block which has no views defined.


A new [state diagram](#) (*TUTORIAL\Control\fsm ['machine0']*) is created as a [child](#) view of the *Control* block:




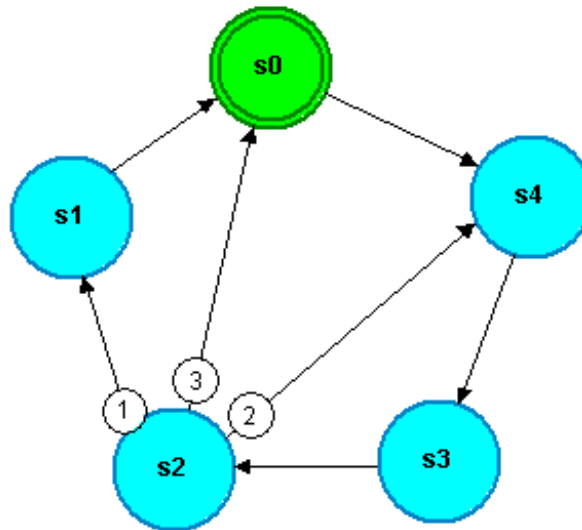
A default state machine name (*machine0*) is appended to the design unit and view names but can be changed by choosing **Rename Concurrent State Machine** from the **Diagram** menu.

The state diagram is a blank sheet except for the default VHDL [package list](#) and labels for [global actions](#), [concurrent statements](#), [architecture declarations](#), [signals status](#), [process declarations](#) and [state register statements](#). The state diagram also includes the default title block which you saved as a template in an earlier topic.

## Add States and Transitions

Use the  button to add five **states** on your state diagram. The states are added with default names *s0*, *s1*, *s2*, *s3* and *s4*. Notice that the first state you add is assumed to be the **start state** and is drawn in green with a double outline. The other states are drawn in cyan with a single outline.

Use the  button to add **transitions** between the states as shown in the picture below. Notice that when you add more than one transition leaving a state, the **transition priority** is indicated by a number associated with the **transition arc**.




The priorities are initially assigned in the order that you add the transitions but will be re-assigned in a later topic if necessary.




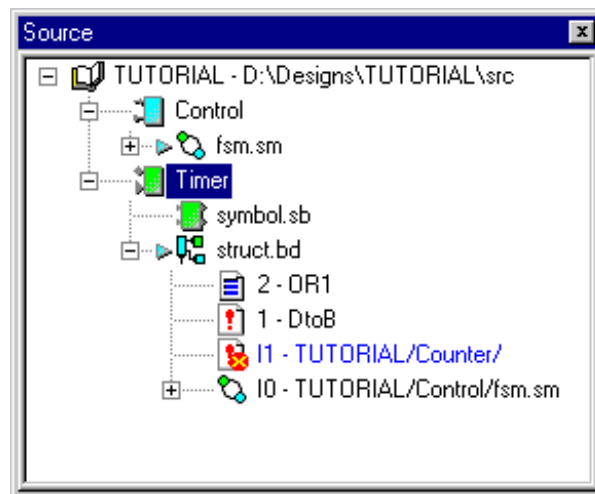
If you add a transition in the wrong direction, you can easily change its direction by choosing **Reverse Direction** from the popup or **Diagram** menu. Note that a popup description (known as a **graphic tip**) is displayed when the cursor is stationary over an object. In particular, when the cursor is over a transition, the source state and the destination state are named even if the states are outside the current window.



## Save the State Diagram

Use the  button to save the state diagram. The state diagram was created as a **child** view from its **parent** block diagram and is saved using the library, design unit and view names specified when it was created. The source browser view is updated to display the *Control* design unit.

 You can pop the design browser window to the front by selecting it from the **Windows** menu list in an editor window.


The *Control* design unit is shown as a block in the browser because its interface is defined by the connections on its parent block diagram. The *Timer* design unit is shown as a component because it has no parent block diagram and its interface is defined by a symbol. Click on the  icon for the *Control* design unit to expand the design unit revealing that it contains a state diagram view.

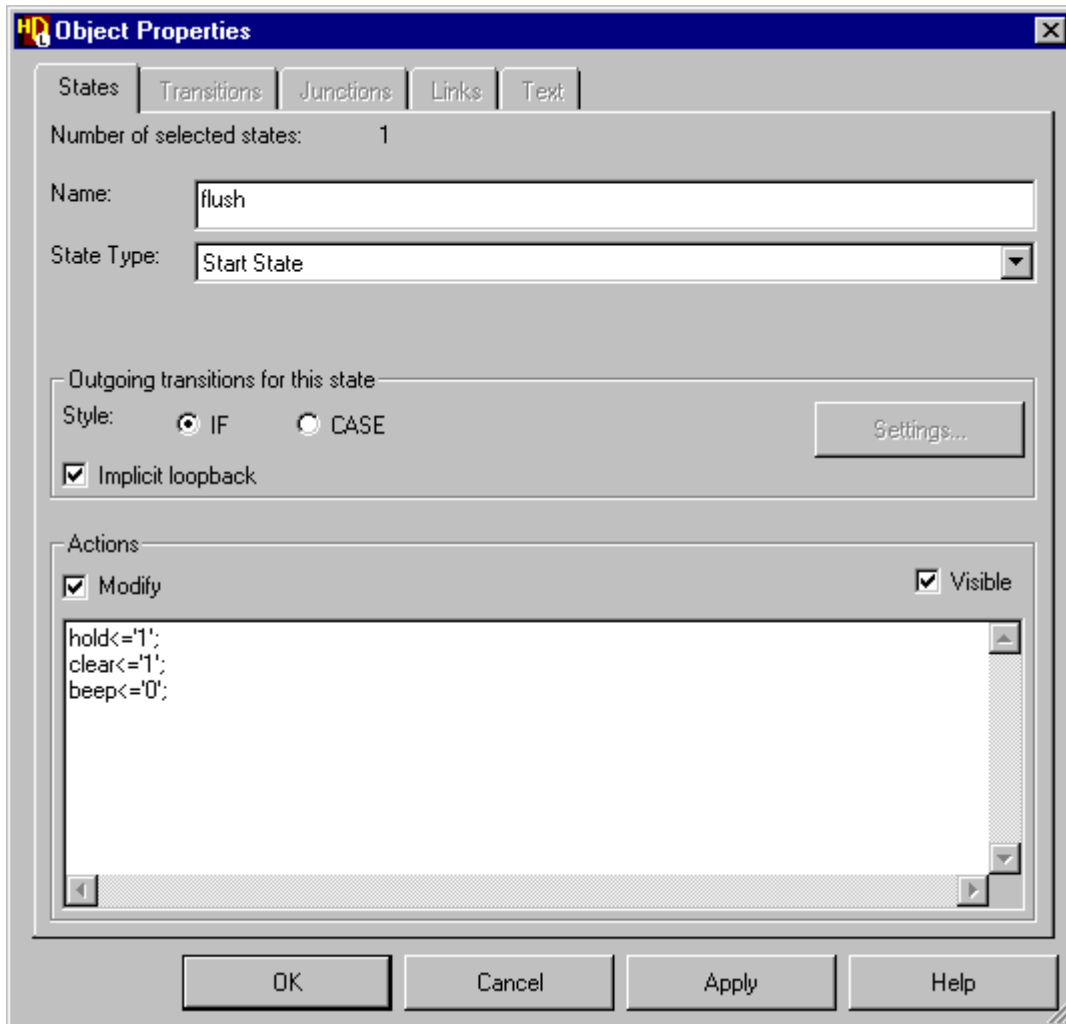


Notice also that the icon used for instance *IO* in the hierarchy for the *struct.bd* view has changed to  indicating that it is now described by a state diagram. (If the hierarchy is not already displayed, click on the  icon for the *struct.bd* view.)

The *ORI* embedded block is shown as a text view but the *DtoB* embedded block and the *Counter* block are still undefined.

## Edit the States

Select the start state (*s0* in the picture on [page 1-26](#)) and use the  button to display the **States** tab of the State Machine Object Properties dialog box. (Alternatively, you can display the dialog box by choosing **Object Properties** from the **Edit** menu or double-clicking on a state.)



The **States** tab allows you to enter a name and actions text for one or more selected states on a state diagram. You can also change the visibility of state actions and (when a single state is selected) change the state to a [start state](#) or a [hierarchical state](#).



Use the dialog box to enter the following state names and **actions** replacing the default state names *s0* to *s4* in the picture on [page 1-26](#):

Old Name	New Name	Style	Actions
s0	flush	IF	hold<='1'; clear<='1'; beep<='0';
s1	count	IF	(no actions)
s2	getkey	IF	hold<='1';
s3	load_t	IF	hold<='1';
s4	load_u	IF	hold<='1'; load<='1';

Notice that the VHDL Expression Builder dialog box is automatically displayed when you start to enter the actions. The dialog box can be used to choose from lists of the available port or local signal names, operators and example values. For example, choose the *hold* signal,  button, value '1' and  button to enter the action *hold* <= '1';.

The syntax for state actions is automatically checked and any errors reported on entry. VHDL statements must be terminated by a semicolon (;) character. Line breaks and spaces can be used for clarity and will be preserved on the diagram.

If the name is larger than the state, it auto-resizes to fit the new name. You can also resize a state by selecting the state and dragging one of its resize handles but you cannot make it smaller than the enclosed name.





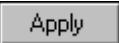



You can also edit the state name or actions by direct text editing on the diagram or copy a state and paste its state actions into another state by choosing the **Paste State Actions** option from the **Paste Special** cascade in the popup menu.

## Edit the Transitions

Add **conditions** to the state diagram transitions as shown in the following table:

Origin	Destination	Priority	Condition	Actions
count	flush	1	stop='1'	none
getkey	flush	3	stop='1'	none
getkey	count	1	start='1'	none
getkey	load_u	2	d/=ZEROS	none
flush	load_u	1	d/=ZEROS	none
load_u	load_t	1	(none)	none
load_t	getkey	1	d/=ZEROS	none



Hold down the  mouse button and select the two transitions entering the state *flush* by dragging the cursor across them (or by using  +  mouse button). Use the  button to display the **Transitions** tab of the State Machine Object Properties dialog box. Enter the condition text *stop='1'* in the dialog box using the expression builder and click the  button to add this condition to both of the selected transitions.

A confirmation dialog box is displayed warning that the transition from state *getkey* to state *count* has no condition and is not the lowest priority. Click the  button to acknowledge the warning.

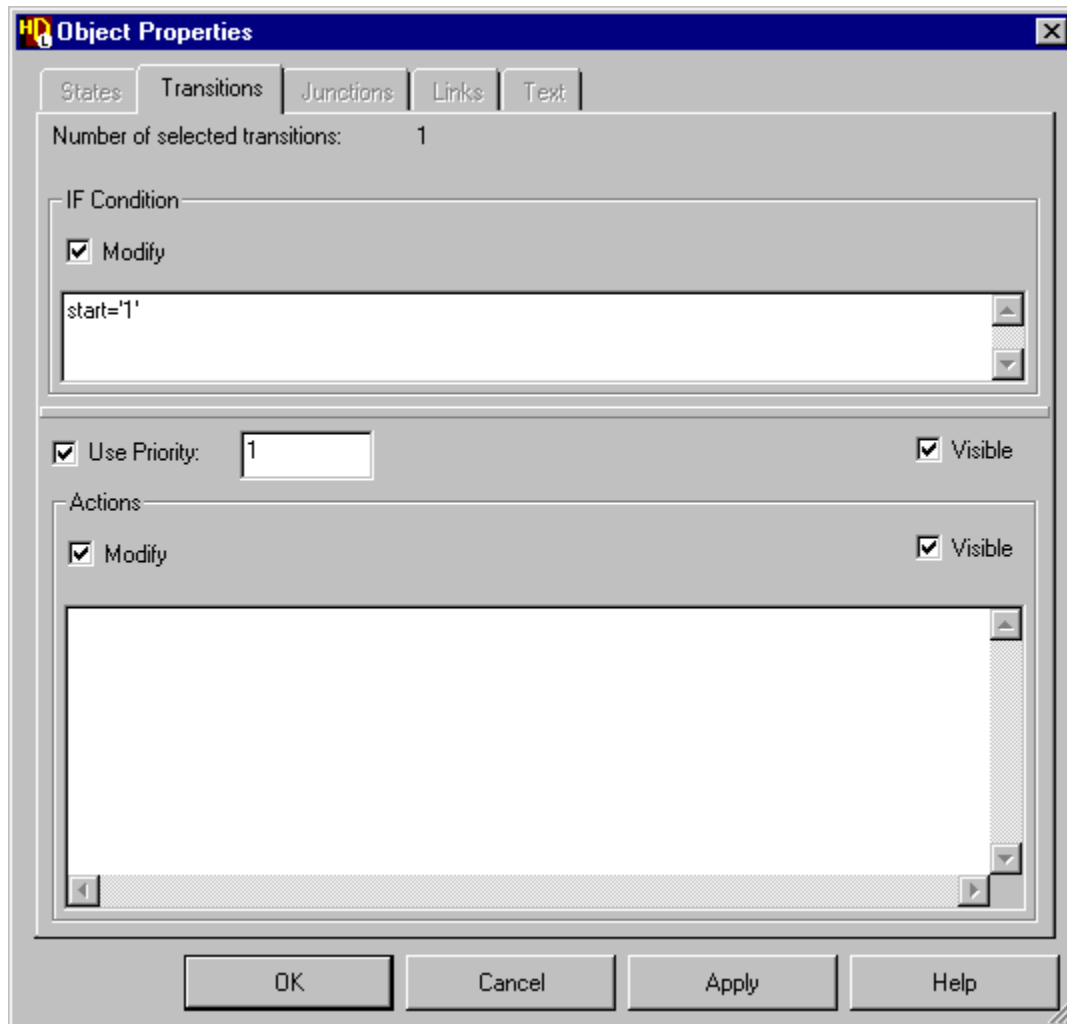
Repeat this procedure for the *start='1'* and *d/=ZEROS* conditions.




The condition syntax is checked on entry. Any valid VHDL fragment can be entered using line breaks and indentation to improve the legibility on the diagram if required.

Ensure that the transition priorities leaving the state *getkey* are the same as those shown in the table. You can change a transition priority in the **Transitions** tab of the State Machine Object Properties dialog box. Alternatively, you can use the  button to zoom in or the  button to view an area and use direct text editing to change the priority.

When you change a transition priority, the priority of the other transitions leaving the same state are automatically adjusted.

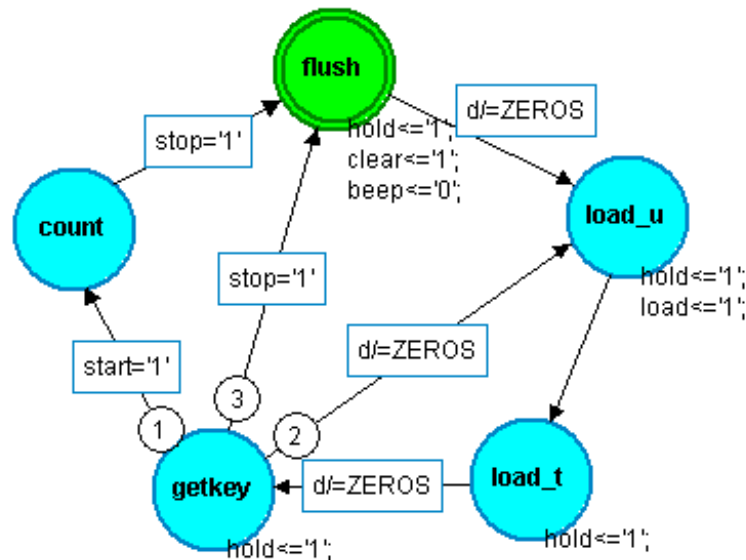


If you have zoomed in, use the  button to view all of the state diagram.

Use the  button to save your changes to the state diagram.

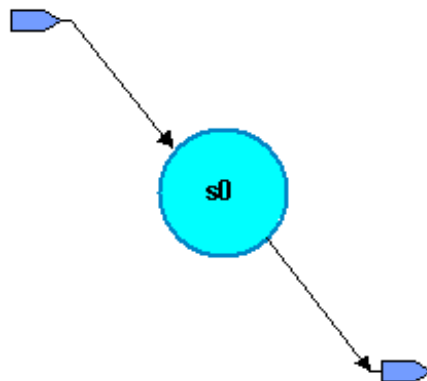
## Create a Hierarchical State Diagram

The state diagram should look similar to the following picture.




Select the *count* state and choose **Hierarchical State** in the **Change To** cascade from the **Diagram** menu. Alternatively, select the *count* state and choose the Hierarchical State pulldown option for the State Type in the **States** tab of the State Machine Object Properties dialog box. The *count* state is redrawn as a **hierarchical state** with a triple outline and darker fill color.



Double-click on the hierarchical state (or use the **Open Down** option from the popup menu) to create the new hierarchical child state diagram. A new child state diagram window is initialized with a default state *s0* connected to an **entry point** and **exit point**.

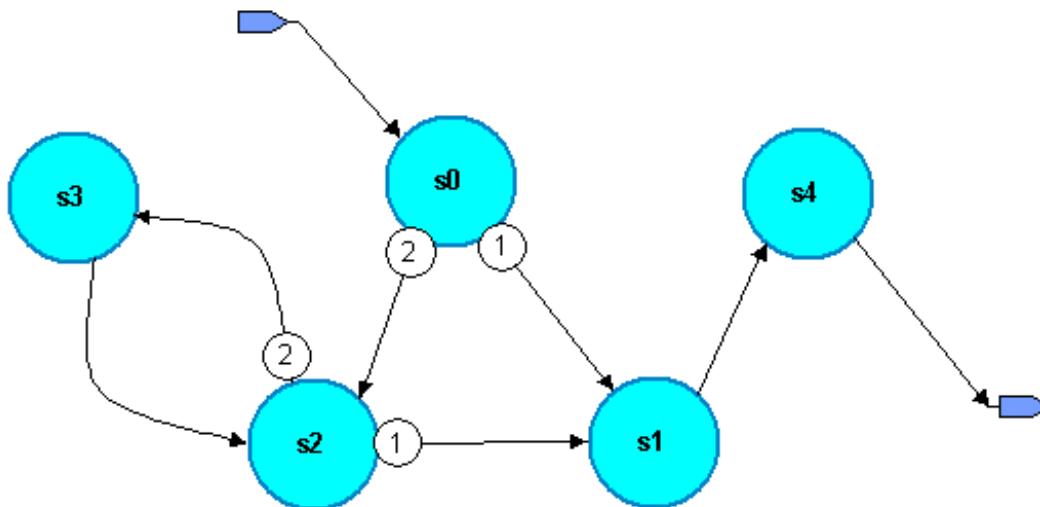


Notice that the name of the hierarchical state (*count*) is included in the window title: *TUTORIAL\Control\ fsm[:count 'machine0']*. This naming convention shows that the child diagram is a partial view of the parent diagram.

Select the exit point and choose **Change to State** from the **Diagram** menu. While the new state is selected, drag with the  mouse button and release the mouse button with the ghosted state to the left and below the first state. Use the **Copy Here** option from the popup menu to make a copy of the state at the cursor position.

Repeat this procedure to add two more states on the diagram. This is an alternative method for adding objects which can be useful when you want to add an object with the same or similar properties and attributes to an existing object.

Use the  button to add a new exit point and the  button to connect transitions between the states as shown in the following picture:



You can add route points while routing a transition by clicking at several points between states to create a smooth arc as shown in the picture between states *s2* and *s3*

## Complete the Hierarchical State Diagram

Use the **States** tab of the State Machine Object Properties dialog box to rename the states and add state actions as shown below:

Old Name	New Name	Style	Actions
s0	standby	IF	hold<='1';
s1	alarm	IF	hold<='1'; clear<='1'; beep<='1';
s2	counting	IF	(no actions)
s3	suspended	IF	hold<='1';
s4	end_count	IF	hold<='1'; clear<='1';

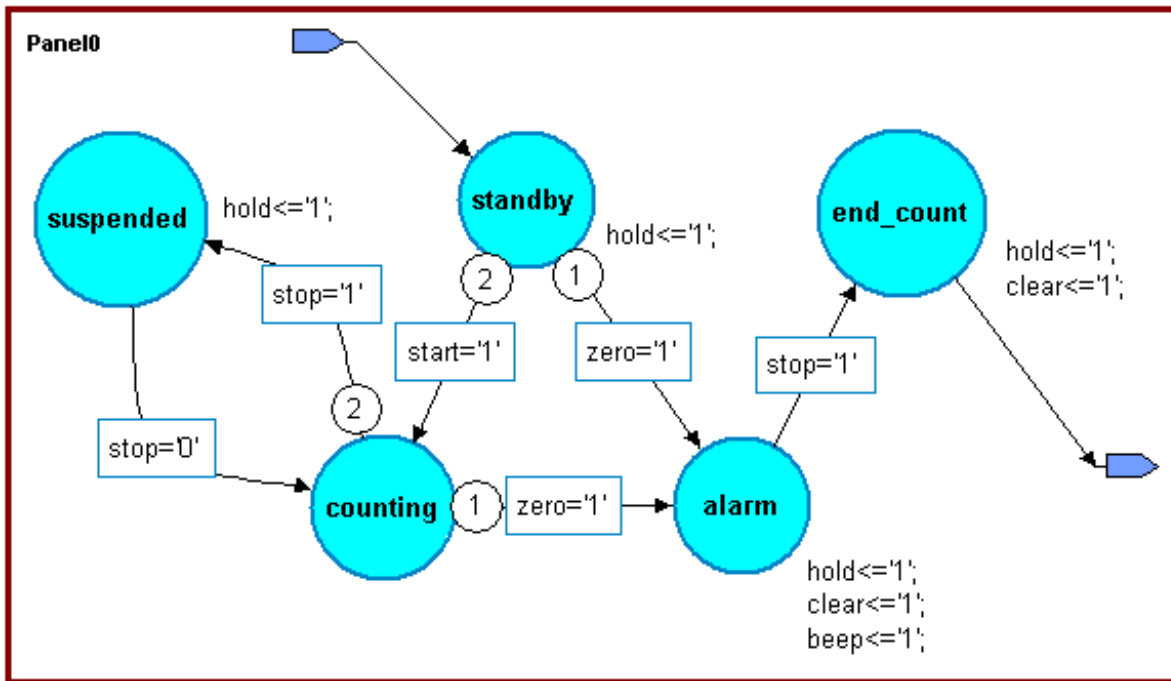
Use the **Transitions** tab to add the following conditions:

Origin State	Destination State	Priority	Condition	Actions
suspended	counting	1	stop='0'	none
counting	suspended	2	stop='1'	none
counting	alarm	1	zero='1'	none
standby	counting	2	start='1'	none
standby	alarm	1	zero='1'	none
alarm	end_count	1	stop='1'	none

A confirmation box may be displayed warning that a transition you have not yet edited has no condition and is not the lowest priority. Click the  button to acknowledge the warning.

Complete the hierarchical state diagram by editing the title and comments in the title block and using the  button to add a panel around the graphical objects on your diagram. This panel can be used to set the diagram view when you animate the state diagram later in this tutorial.


The state diagram should look similar to the picture below:



<b>Mentor Graphics</b>		Project:	HDL Designer Tutorial
Title:	Counter state machine	Child hierarchical view of the count state in the Counter state machine.	
Path:	TUTORIAL/Control/fsm		
Edited:	by joans on 08 Feb 2001		



Ensure that the transition priorities are as shown and that you include the terminating semi-colon when you enter the state actions.


Use the  button to save the state diagram.

Although it is displayed as a separate window, a child hierarchical diagram is a partial view of a state machine and the parent and child diagrams are saved as a single design unit view (*TUTORIAL\Control\fsm.sm*) although the name in the title bar is *TUTORIAL\Control\fsm[:count 'machine0']*.

## Editing State Machine Properties

There are a number of properties associated with a state machine which can be edited using the State Machine Properties dialog box. The dialog box can be accessed by choosing **State Machine Properties** from the **Diagram** or popup menu in the parent or child state diagram. It can also be accessed by double-clicking over one of the labels which are displayed in the parent diagram of a hierarchical state machine.


Use the  button (or choose the **Open Up** option from the **File** menu) to re-display the parent state diagram if it is not already displayed.

Double-click the  mouse button over the Global Actions, Concurrent Statements or State Registers label to display the **Statement Blocks** tab of the State Machine Properties dialog box. There are no [global actions](#), [concurrent statements](#) or [state register statements](#) required in this design. Hide the labels for these objects on the state diagram by clearing the **Visible** check box for Global Actions, Concurrent Statements and State Register Statements in the **Statement Blocks** tab.

Choose the **Process Declarations** tab. As there are no [process declarations](#) required in this design, hide this label by clearing the **Visible** check box in the **Process Declarations** tab.

Choose the **Architecture Declarations** tab (by double-clicking over the Declarations label on the state diagram) and enter the following constant declaration in the free-format entry box:

```
Constant ZEROS : std_logic_vector := "0000000000";
```

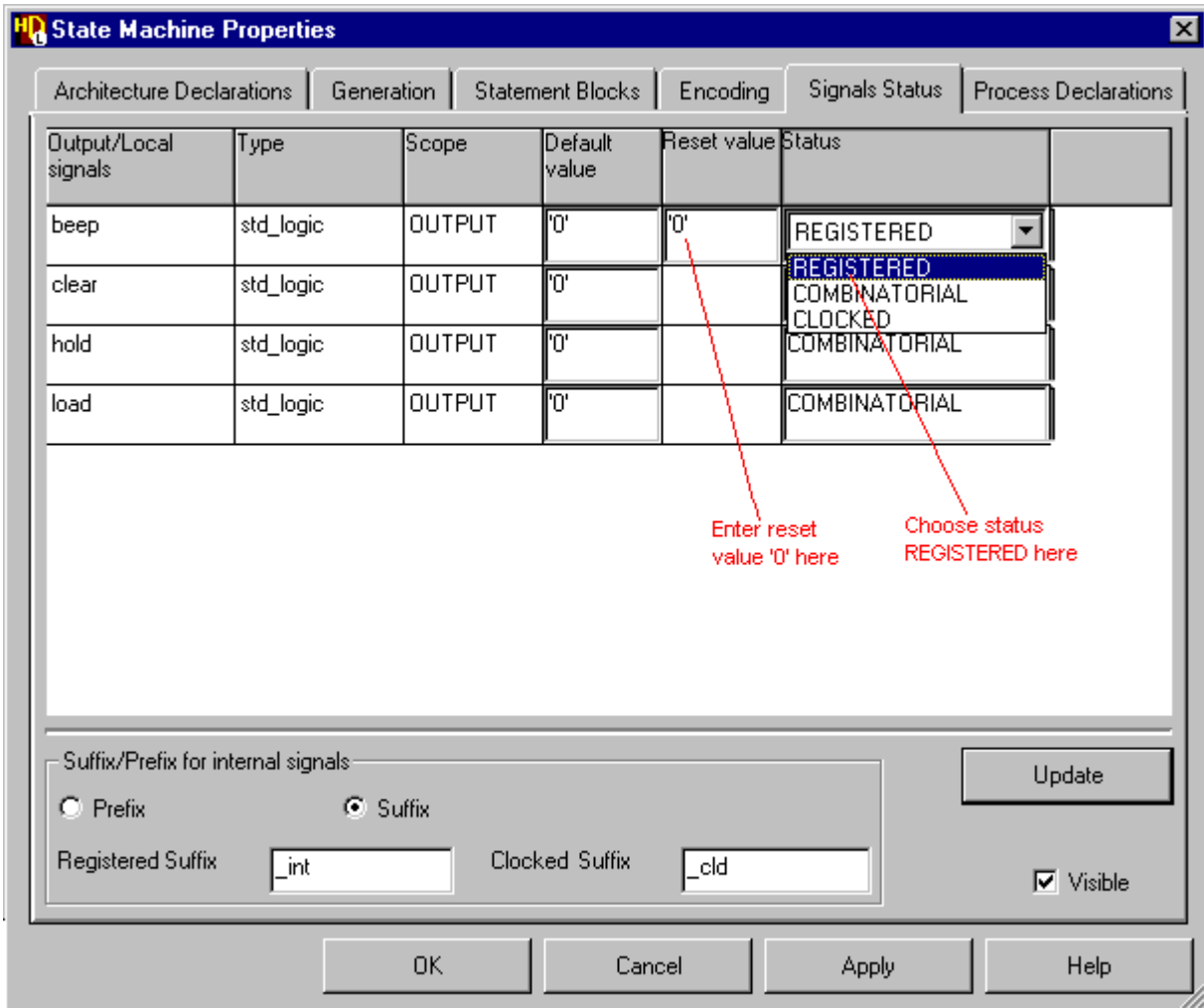
Click the  button to update the diagram. The constant declaration is added below the Architecture Declarations label on the diagram and will be included as [architecture declarations](#) when HDL is generated for the state machine.

Re-display the State Machine Properties dialog box if necessary and choose the **Signals Status** tab (by double-clicking over the Signals Status label on the state diagram).



Notice how the output signals are listed in the dialog box with the default value set to '0' and the status *COMBINATORIAL*.

Click on the Status field for the *beep* signal and choose *REGISTERED* from the drop down box. Click in the Reset Value field and enter the value '0'.

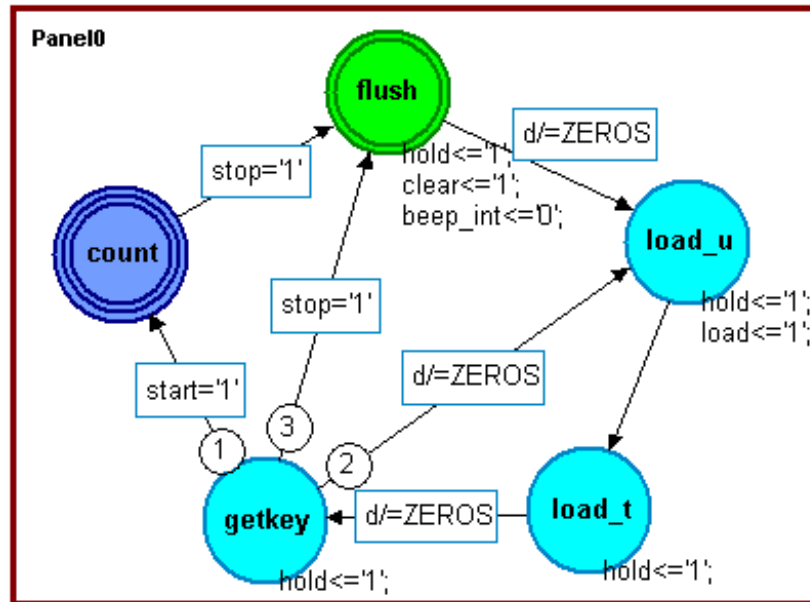


Any locally declared internal signals would be shown with the status REGISTERED. Bidirectional or buffer signals are treated as output signals.

The dialog box also allows you to change the suffix or prefix used for internal registered or clocked signal names. For this tutorial, suffixes are used with the default values *\_int* and *\_cld*.

Confirm the dialog box by clicking the  button.

Complete the state diagram by editing the title and comments in the title block and using the  button to add a panel around the graphical objects on your diagram. This panel can be used to set the diagram view when you animate the state diagram later in this tutorial. The final state diagram should look similar to the picture below:



**Architecture Declarations**

Constant ZEROS : std\_logic\_vector := "0000000000";

**Signals Status**

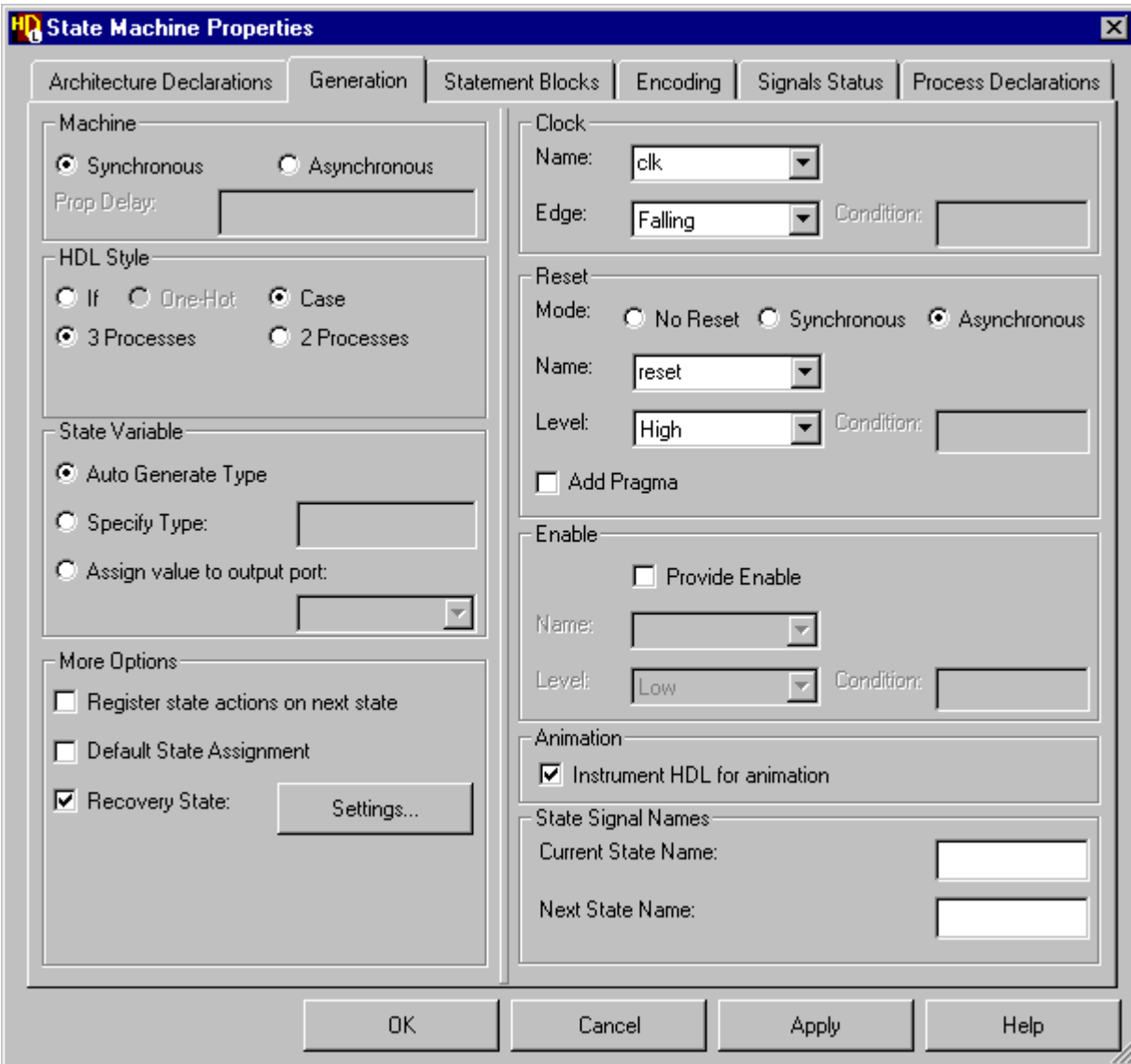
SIGNAL	SCOPE	DEFAULT	RESET	STATUS	Package List
beep	OUT	'0'	'0'	REG	
clear	OUT	'0'		COMB	ieee std_logic_1164
hold	OUT	'0'		COMB	ieee std_logic_arith
load	OUT	'0'		COMB	

Notice that all occurrences of the *beep* signal (in the *flush* state and in the *alarm* state on the child hierarchical state diagram) have been replaced by the internal signal name *beep\_int* using the default suffix *\_int*.


The State machine Properties dialog box also provides tabs for setting HDL generation characteristics and state machine encoding. State machine encoding is not used in this tutorial and the encoding scheme should be set to **None**. You can use the  buttons for more information about each tab of the State Machine Properties dialog box.

## Set Generation Properties

Re-display the State Machine Properties dialog box if necessary and choose the **Generation** tab. This tab sets generation characteristics used for HDL generation.



Choose the **Synchronous** option in the Machine box. Use the default options **Case** and **3 Processes** for HDL Style and **Auto Generate Type** for the State Variable.

Check that the **Recovery State** is set to `<start_state>` by using the  button to display the Recovery State Settings dialog box.



This entry automatically assigns the start state or you can choose from a pulldown list of states and specify recovery state actions.



Leave the **Register state actions on next state** and **Default State Assignment** options unset.


Choose *clk* from the pulldown list of clock signal names and *Falling* from the Edge pulldown list. Set an **Asynchronous** reset and choose the signal name *reset* with a *High* level from the pulldown lists. Leave the **Provide Enable** options unset.

If a ModelSim simulator is available, set the **Instrument HDL for animation** option. This option adds additional code to the generated HDL which is used for state machine animation later in this tutorial.



This additional code is surrounded by translation control pragmas which ensure that it is ignored by downstream synthesis tools.

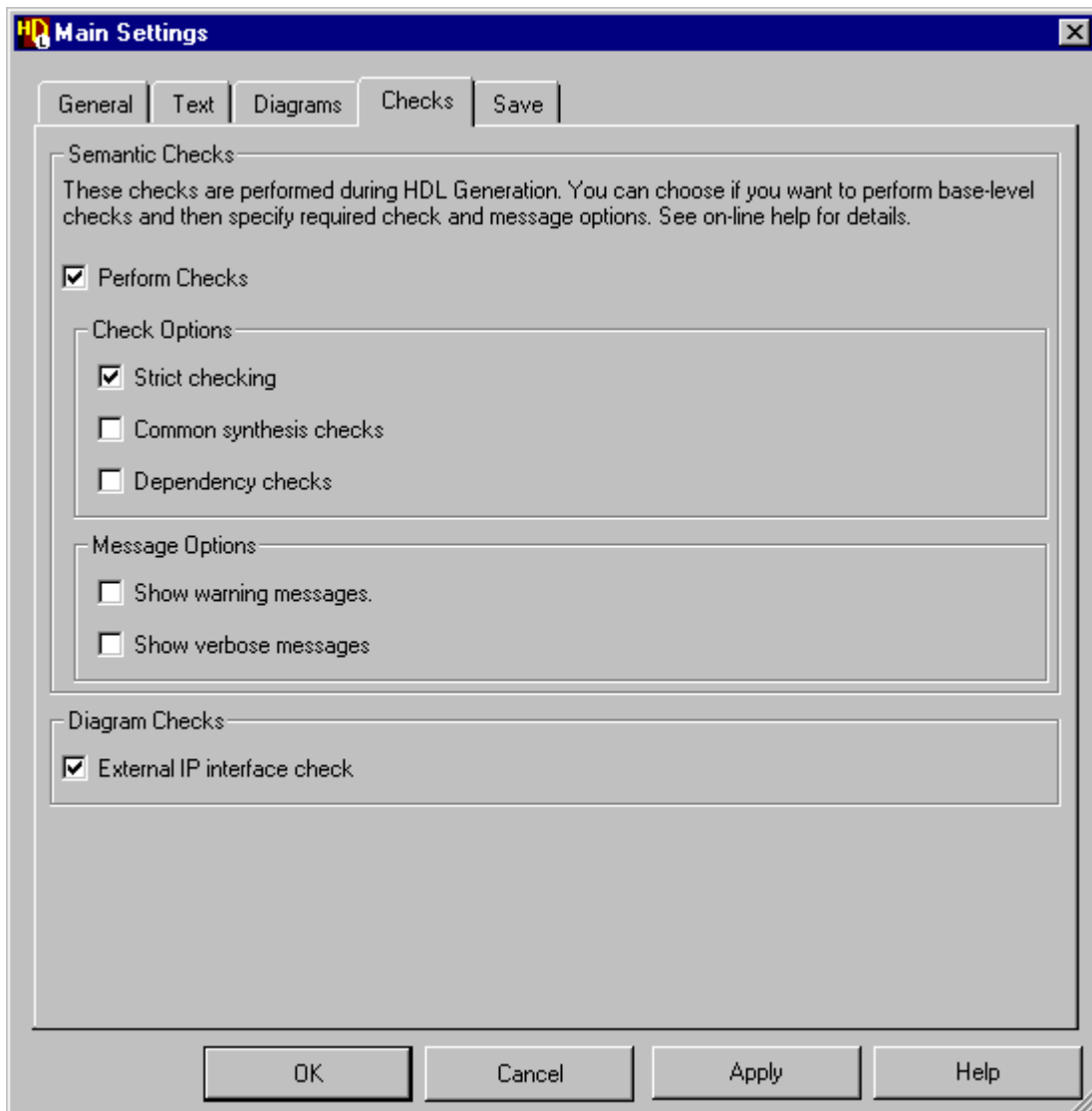
Use the  button (or  followed by ) to apply these generation characteristics to your state machine and dismiss the dialog box.

Use the  button to save the state diagram.

## Set Checks for HDL Generation

Although the *Timer* design is not yet complete, you can now generate HDL for the *Control* state machine.

You can set the level of checks performed when HDL is generated by using the **Checks** tab in the Main Settings dialog box which can be accessed by choosing **Main** from the **Options** menu.

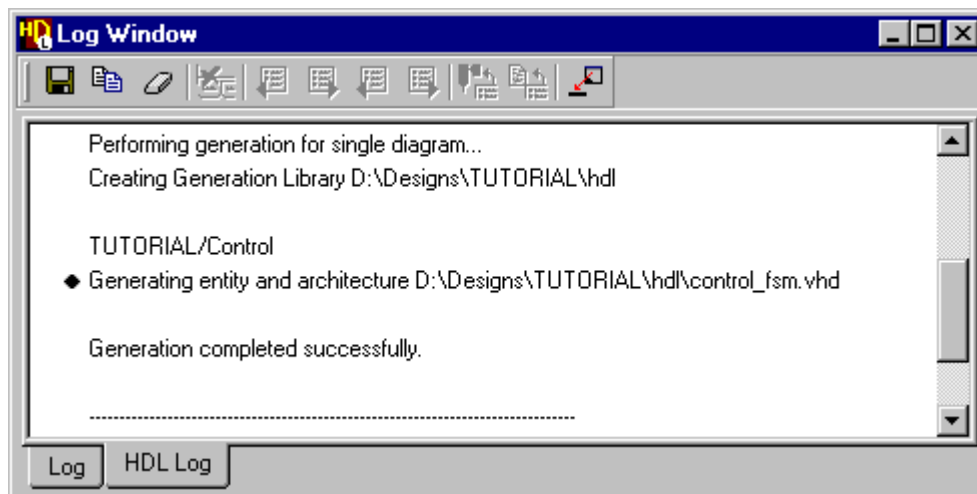


For this tutorial, the **Strict checking** option should be selected.




## Generate HDL for the State Machine


Choose the **Generate** option from the **HDL** menu (or use the  button) in the state diagram window.


The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation.




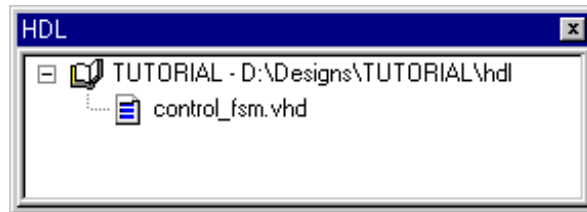
This example shows a combined file is generated for the **VHDL entity** and **VHDL architecture body**. Separate files may be generated if you have changed the preferences for VHDL file options.

If there are any errors, you can move to the next error message using the  button or the previous error using the  button. You can display the source graphics corresponding to the error by double-clicking on the error message (or using the  button when the error is selected).

Alternatively, you can use the  button when the error is selected to view the generated HDL. If your text editor supports a line number argument, the HDL line corresponding to the error is selected automatically.

If there are no errors, you can use the toolbar buttons to view the source graphics or generated HDL corresponding to any message which is marked by the  icon in the log window. For example, the “Generating entity and architecture ...” message shown above.


Any files in the generated HDL directory can also be opened from the [HDL browser](#). Click on the  icon for the *TUTORIAL* library in the HDL browser and notice that the view is expanded to reveal the generated [VHDL architecture body](#) file for the *Control* design unit.




You can open this file by double-clicking on the generated file name or by choosing **Open** from the popup menu.



If you have changed your preferences to create separate VHDL entity and architecture body files, then both files are shown.

You can also view the generated HDL files for the active diagram by using the  button in any editor window or when the diagram view is selected in the source browser.

If you are using the default text editor (ESview on Windows or NEdit on UNIX), you can cross-reference from an error message to the corresponding line in the generated HDL file. You can also use the  button or the **Show HDS Source** menu command in the text editor to cross-reference from any line in the generated HDL to the corresponding graphics object.





These options can also be set up for other user customizable external text editors. Refer to the “Setting the Text Editor” help topic for more information.

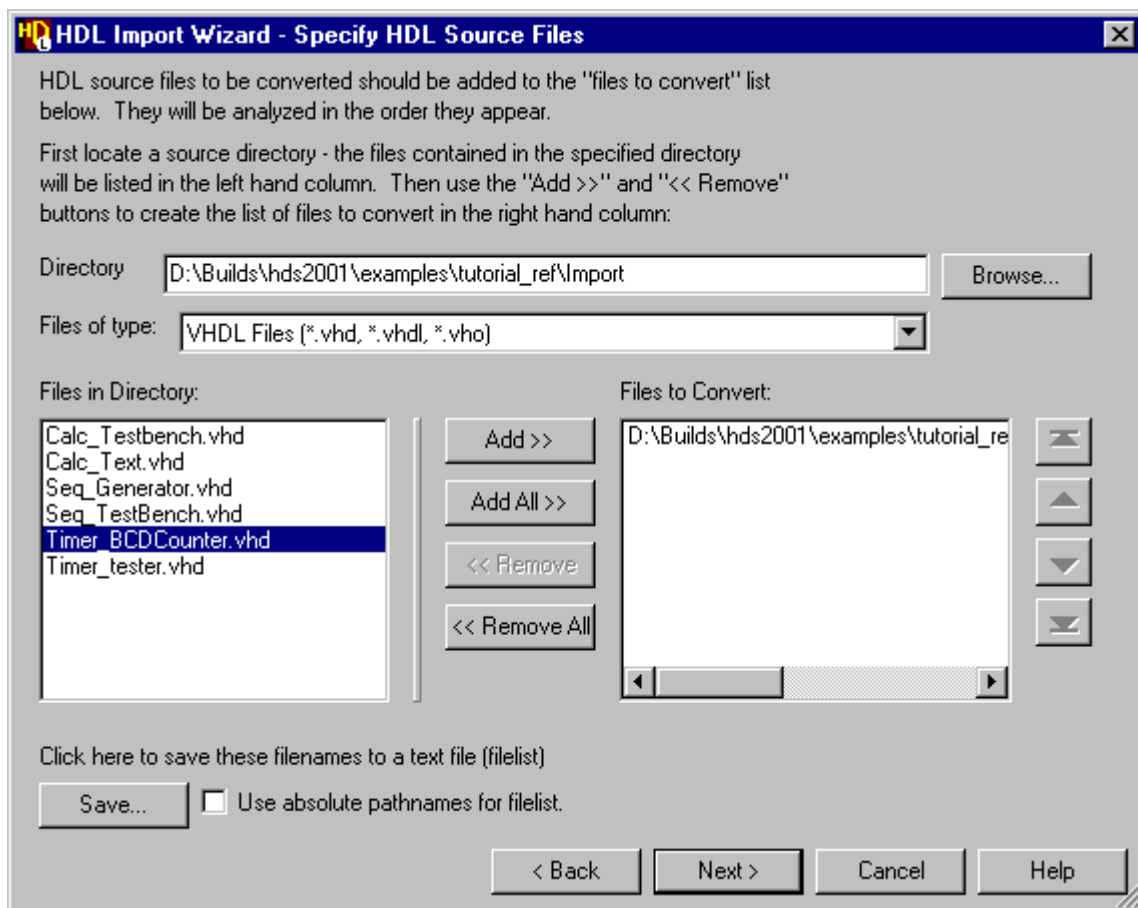
Close the text editor after you have viewed the generated HDL.


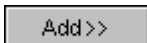
Use the **Close Window** option from the **File** menu to close the parent (*fsm* [*'machine0'*]) and child (*fsm* [*:'count* *'machine0'*]) views of the *Control* state machine.

## Import the BCDCounter Design Unit

You have now defined the *Control* block by drawing a graphical state machine. The *Counter* block will be defined by instantiating two instances of a HDL text component in a child block diagram.

Choose the pulldown  on the  button and select the  option from the palette or choose **Text HDL Import** from the **HDL Import** cascade of the **HDL** menu to display the HDL Import wizard. Choose **Specify HDL files** in the first page of the wizard and click the  button to display the Specify HDL Source Files page.



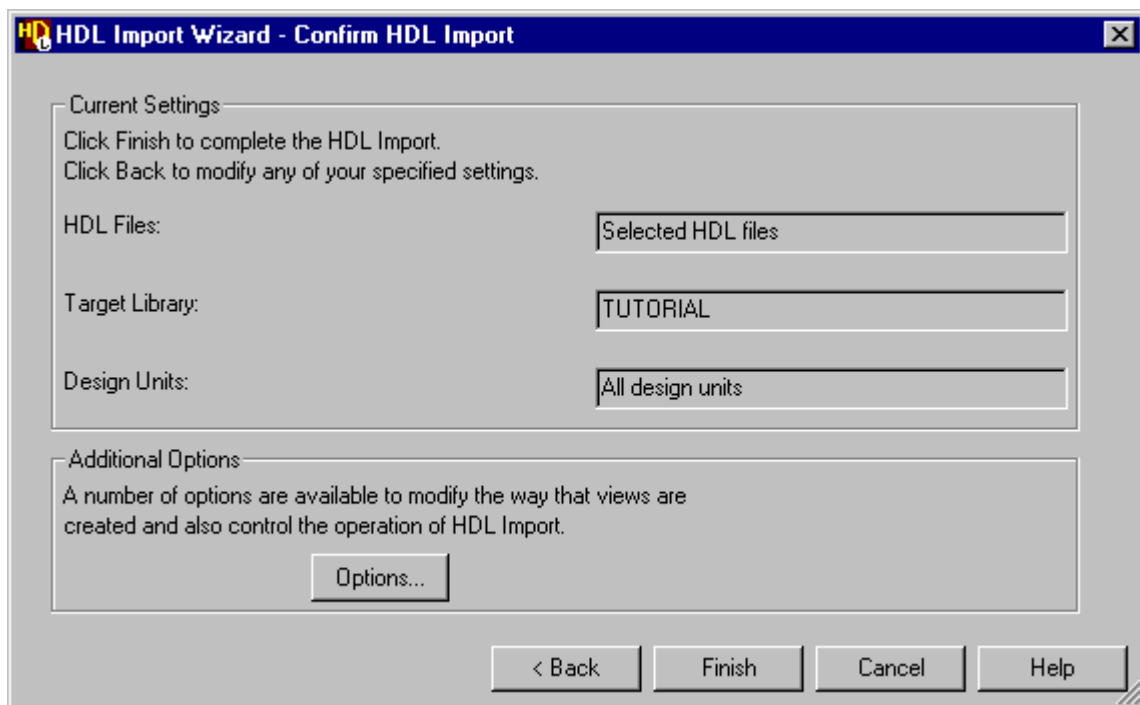
Choose VHDL files from the pulldown filter for Files of type and use the  button to locate the `..\examples\tutorial_ref\Import` installation subdirectory. Select the `Timer_BCDCounter.vhd` file and add it to the list of files for conversion by using the  button.



Click the  button on the wizard to display the Design Units page. This page allows you to select from a list of design units found in the source HDL files. In this case, there is only the *BCDCounter* and the **All** option should be selected.

Click the  button on the wizard to display the Target Libraries page. This page allows you to select the default target library and add libraries if there are any instantiations for black box components in the source HDL code.

Ensure that your *TUTORIAL* library is selected as the target library and click the  button to display the Confirm HDL Import page.



Click the  button in this page to complete the HDL import. The following completion message should be displayed in the HDL log window:

```

** Creating component BCDCounter in library TUTORIAL
** Creating HDL view spec.vhd of component BCDCounter in
    library TUTORIAL

HDL Import complete
-----
1 HDS design unit saved,
1 component
  1 HDL view
-----

```

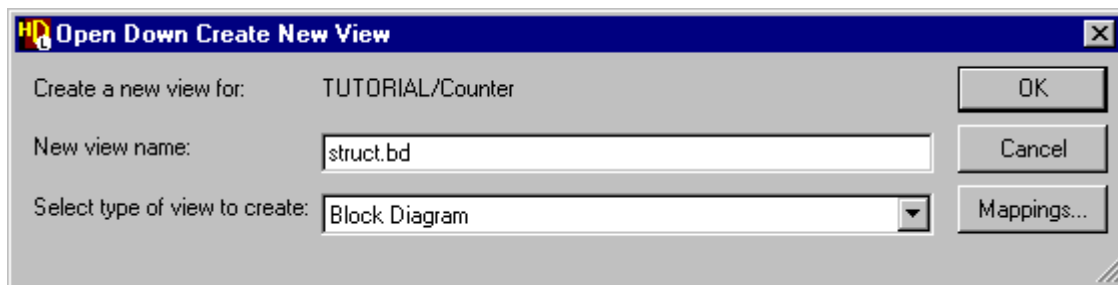
## Create a Child Block Diagram

Display the *Timer* block diagram.



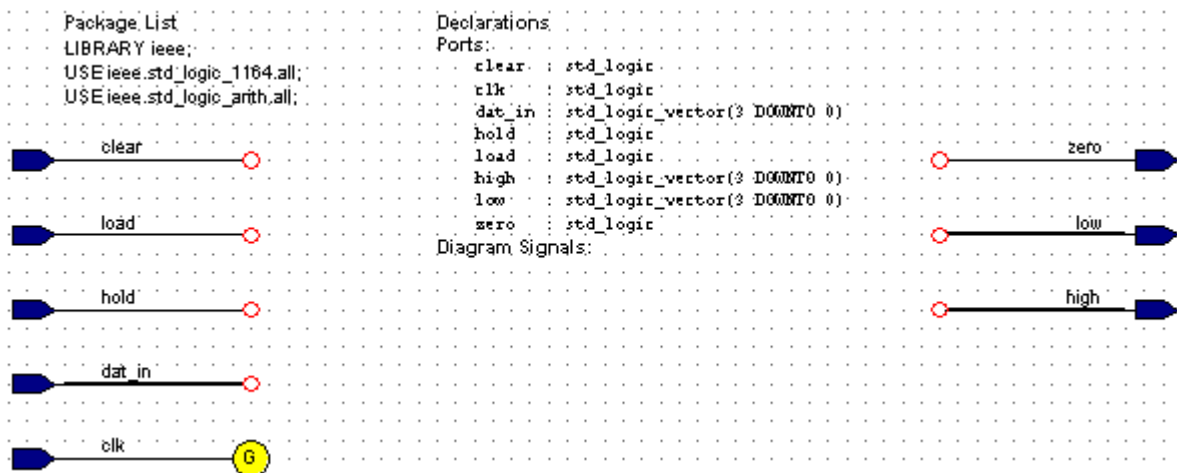
If the window is obscured, you can pop it to the front by selecting the window name from the **Windows** menu list in any other window.


Double-click (or choose **Open** from the popup menu) on the body of the *Counter* block to display the Open Down Create New View dialog box.

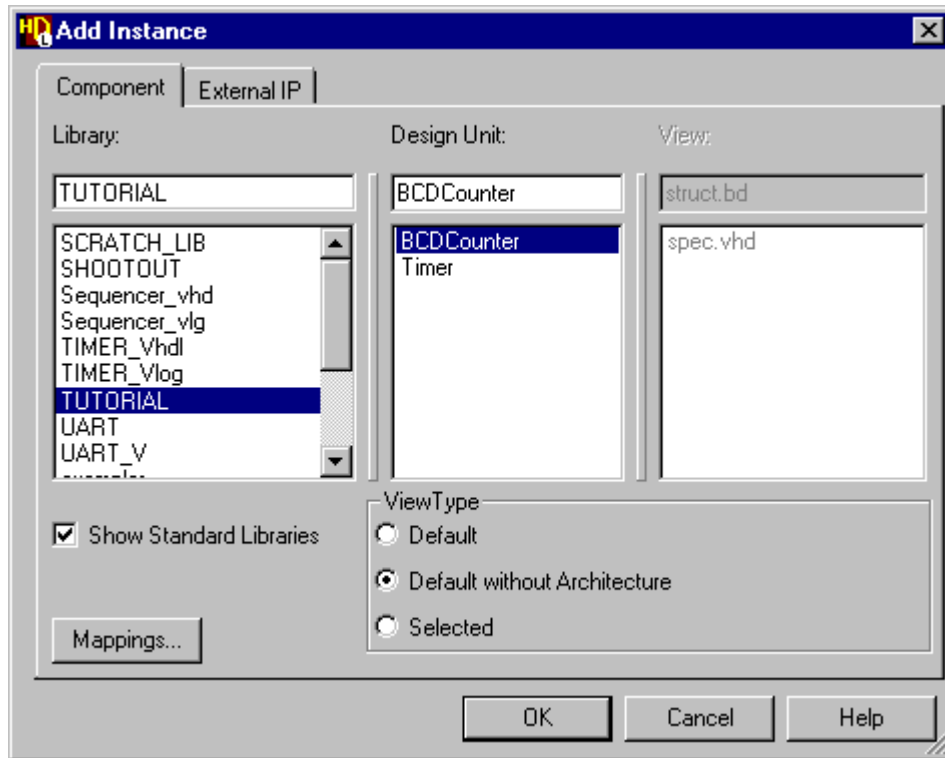


Select **Block Diagram** from the pulldown list of views you can create and accept the default view name *struct.bd*.

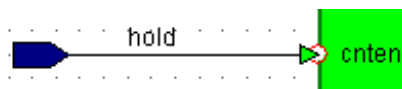
A new block diagram (*TUTORIAL\Counter\struct*) is created as a child view of the *Counter* block. The child block diagram is initialized with declarations and ports corresponding to the connected signals and buses on the parent diagram (including a global connector for the *clk* signal). Notice how inputs are initialized on the left and outputs on the right.




Use the  button to display the Add Instance dialog box. Use the dialog box to add two instantiations (*I0* and *I1*) of the *BCDCounter* component placing one below the other on the diagram but allowing several grid lines for connections between them.





Move the *hold* signal so that it overlaps the *cnten* port on instance *I0*.

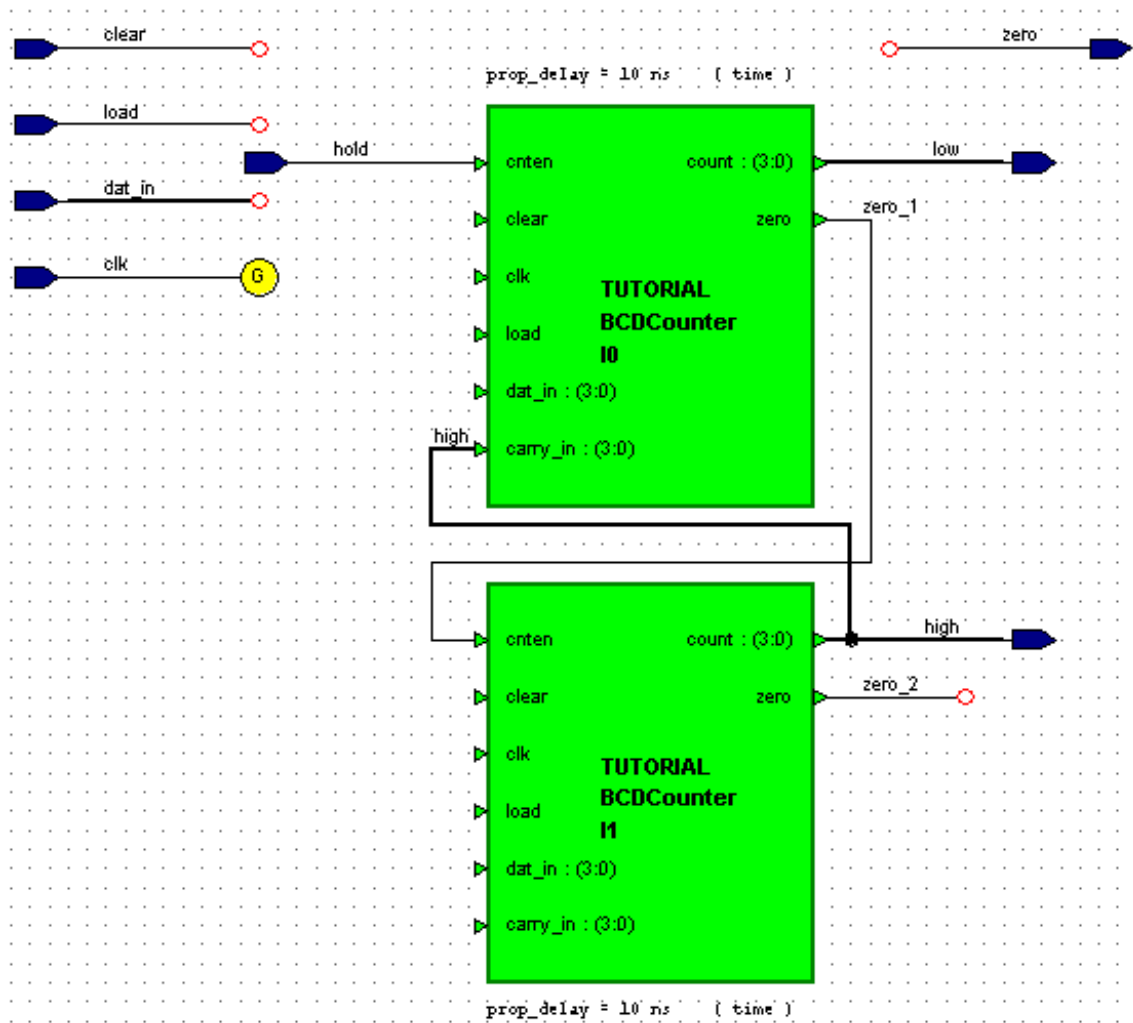


Similarly, move the *low* signal over the *count* output port on instance *I0* and the *high* signal over the *count* output port on instance *I1*. Select both instances and choose **Connect** from the popup menu to make the connections.


Use the  button to connect a signal between the *zero* output port on *I0* and the *cnten* port on *I1*. This signal is automatically named *zero\_1* since the output signal *zero* already exists on the diagram.

Use the  button to connect a bus between a point on the *high* output bus and the *carry\_in* port on *I0*.

Use the  button to connect a stub signal to the zero output port on I1. This signal is automatically named *zero\_2* since the signals *zero* and *zero\_1* already exist on the diagram. The block diagram should now look similar to the following picture:



Select the global connector and choose **Delete** from the popup menu taking care not to delete the stub *clk* signal and input port.

Select instance *I0* and choose the **Add Signal Stubs** command from the popup menu. You are warned that the nets *clear*, *clk*, *dat\_in* and *load* already exist on the diagram. Click the  button to acknowledge the warning.

The signal stubs are added when you accept the warning and the matching signals are implicitly connected by name.

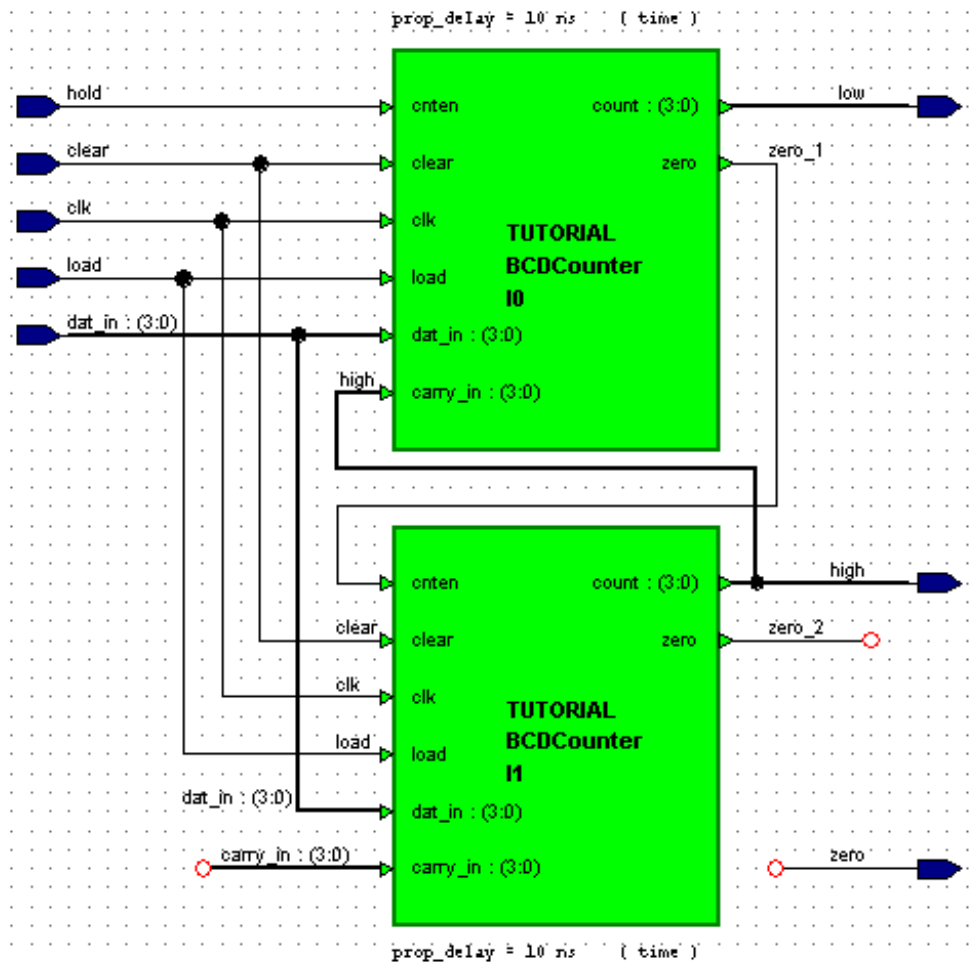
Select the input port signals *clear*, *load*, *dat\_in* and *clk* then choose **Autoconnect** from the **Autoroute** cascade of the popup menu. The signals are automatically connected to the matching stub signals on instance *I0*.



If nothing is selected on the diagram, autoconnect attempts to re-route all connections on the diagram. Ensure that only the required signals are selected if you do not want to change all the connections on a diagram.


Select instance *I1* and choose the **Add Signal Stubs** command from the popup menu to add stub signals for the remaining ports on this instance. Select these signals and choose **Autoconnect**. Click the  button and acknowledge the warning that the nets already exist on the diagram to complete the connections.

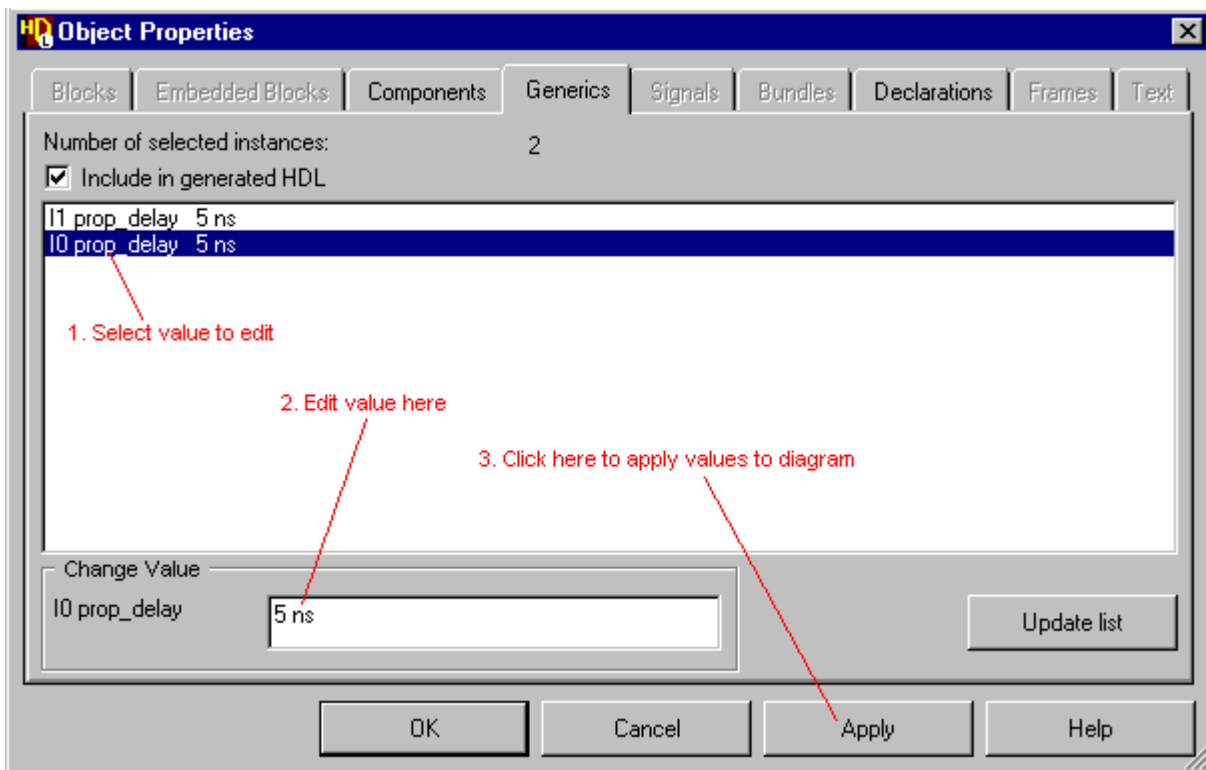
The block diagram should now look similar to the following picture:



## Edit the Generic Mapping

When you instantiated the *BCDCounter* components a VHDL generic mapping for the propagation delay (*prop\_delay*) was placed above the component. The generic is declared in the *symbol* defining the component and can be mapped to different values for each instance of a component. The generic mapping is a text object which can be moved independently by dragging it with the **Left** mouse button. For example, the generic mapping for instance *I1* has been moved below the instance in the picture on the previous page.


For this tutorial, edit both values to be *5 ns*. Use the **Shift** key with the **Left** mouse button to select both instances of the *BCDCounter* component. Then use the  button to display the Block Diagram Object Properties dialog box and choose the **Generics** tab.

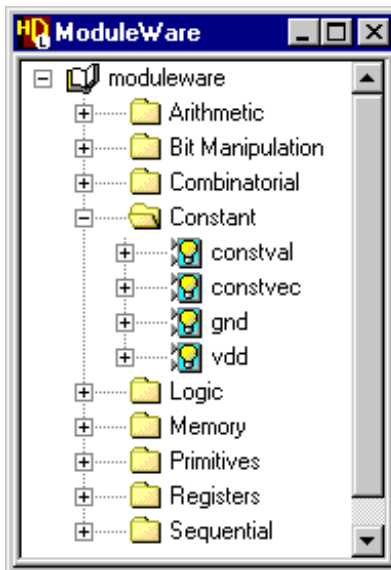



Select the entry for instance *I0* and edit its value in the Change Value box. Repeat for the value on instance *I1*. Then use the **Apply** button to update both values on the block diagram.


## Add ModuleWare Components

Although you have routed the *Counter* block diagram, several signals have been left unconnected. These will be connected by using **ModuleWare** components.


Display the ModuleWare browser by choosing **ModuleWare Browser** from the **Window** menu. The browser displays the contents of the *moduleware* library which is divided into a number of folders. Click on the  icon to expand the folder for the *Constant* category::



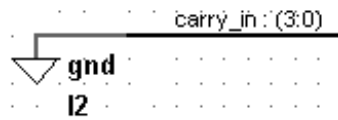
Use the  mouse button to drag the *gnd* component over the *Counter* block diagram and release the button to place it near the *carry\_in* input to instance *I1*. The ModuleWare instance is added with a default instance name (*I1*).

-  You can also add a ModuleWare component by selecting it in the Add Instance dialog box or by dragging it from the *moduleware* library when it is displayed in the source browser.

Use the mouse to position the stub signal on the *gnd* component over the dangling connector on the *carry\_in* net and choose **Connect** from the popup menu.

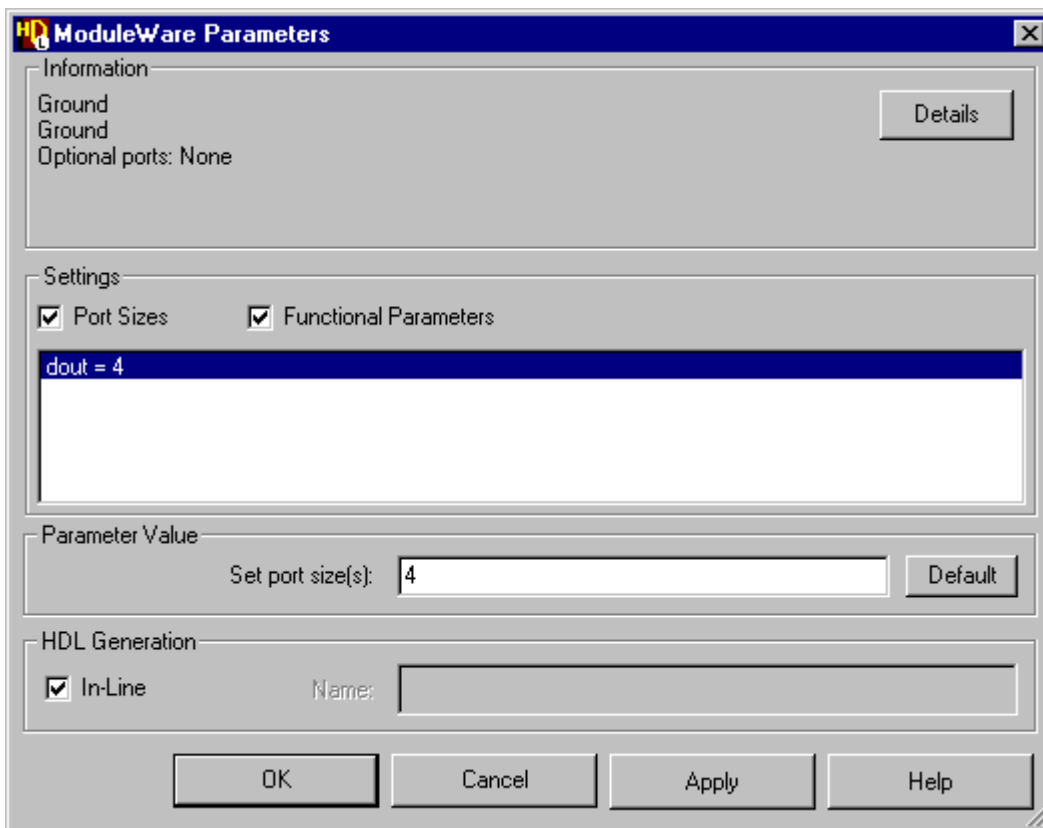
-  The port arrow head on the stub signal for the ModuleWare component instance automatically disappears when you make the connection.

The *I2* ModuleWare instance should now look similar to the following picture.



Notice that the *carry\_in* net is a 4-bit bus. The stub signal on the ModuleWare instance has a default width of 1 and must be set to match the width of the connected net.

Double-click over the *gnd* instance (or choose **ModuleWare Parameters** from the popup menu) to display the ModuleWare Parameters dialog box:



Select the *dout* signal and set its port size to 4. Check that the **In-Line** option is set for HDL Generation and confirm the dialog box by clicking the **OK** button.

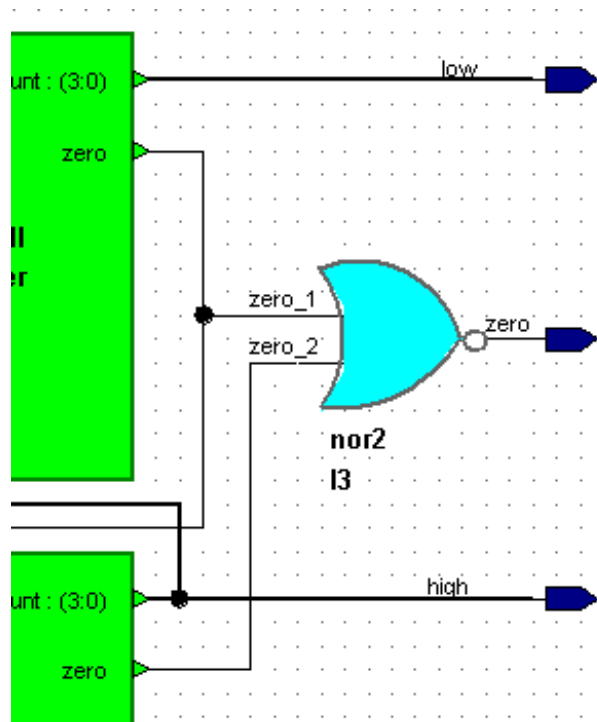


HDL can be generated in-line (within the same file as the view containing the ModuleWare instance) or as a separate file when this option is unset.



Repeat this procedure to add an instance of the *nor2* component from the *Logic* folder of the *moduleware* library near the *zero\_1* output from instance *I0* on your block diagram.

Connect this instance to the *zero\_1*, *zero2* and *zero* nets as shown in the following picture:





Notice that this instance is connected to 1-bit signals and you do not need to modify its port sizes.

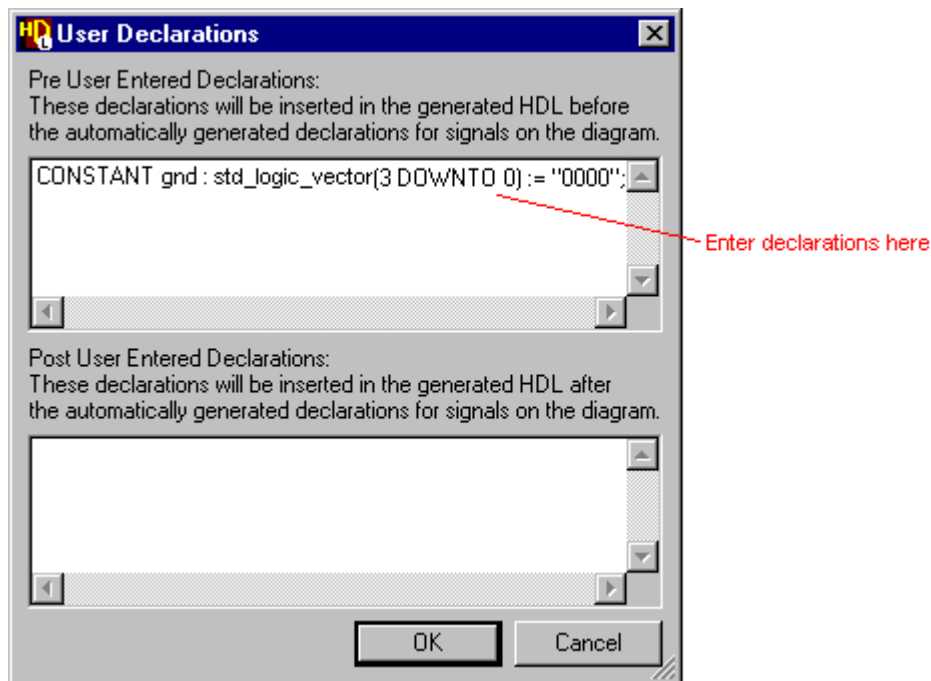


This function could also be implemented by instantiating a 2-input OR gate and using the ModuleWare Parameters dialog box to set the output port with the enumerated type *ActiveLow*. The logical NOR function could also be implemented by an embedded block similar to that used in [“Add an Embedded HDL Text View”](#) on page 1-18.

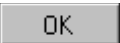
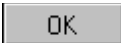
## Add a User Declaration

Double-click over the Declarations label on the diagram (or use the  button) to display the **Declarations** tab of the block diagram Object Properties dialog box. Click the  button to display a free format entry dialog box. Define the *gnd* signal by entering the following constant declaration in the Pre User Entered Declarations section of the dialog box:

```
CONSTANT gnd : std_logic_vector(3 DOWNT0 0) := "0000";
```




The syntax is automatically checked and any errors reported on entry. You can enter any valid VHDL statements which must be terminated by a semicolon (;) character. Line breaks and spaces can be used for clarity and will be preserved on the diagram.

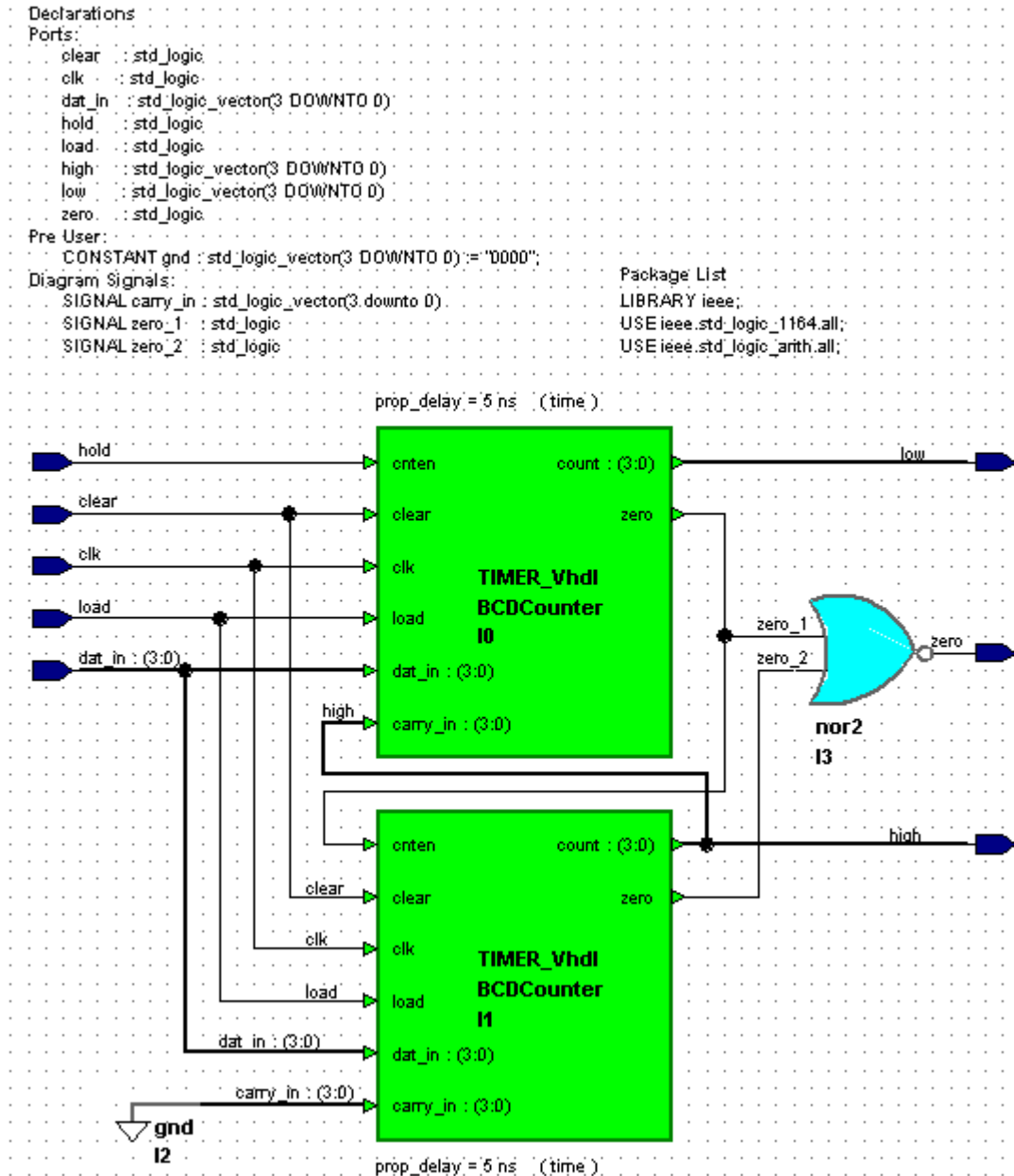
Click on the  button to confirm the user declaration and click the  button in the Block Diagram Object Properties dialog box to add the new user declaration on the diagram.



The pre user entered declarations are added to the declarations list on the diagram between the port and signal declarations.

Use the  button to save the block diagram.

The *Counter* block diagram should look similar to the following picture.



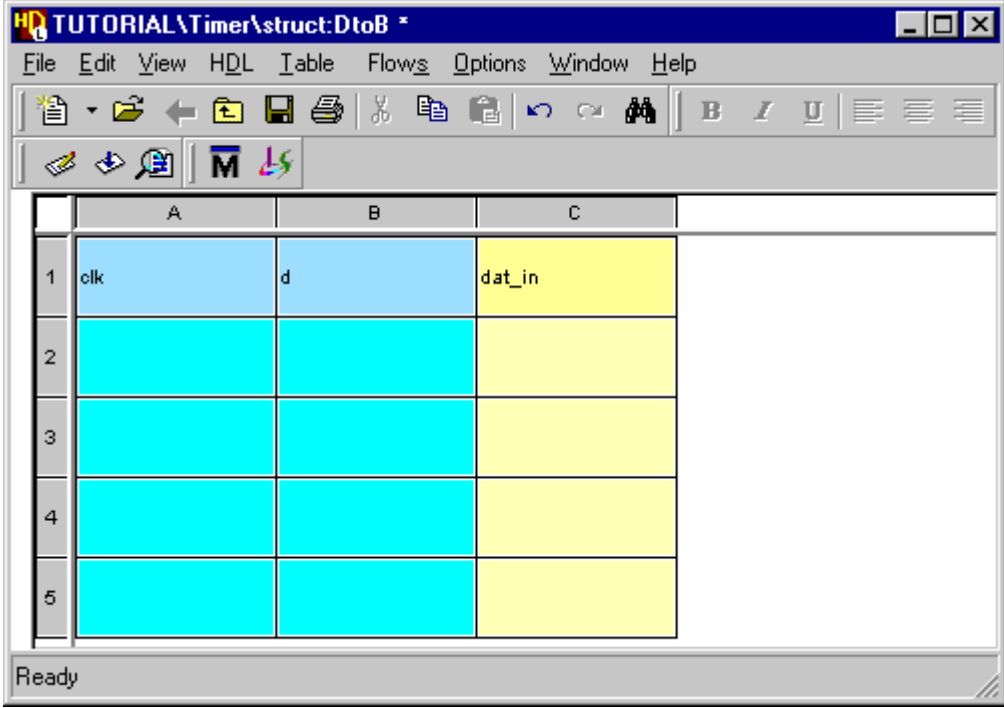
## Create a Truth Table

Use the  button (or choose the **Open Up** option from the **File** menu) in the *Counter* block diagram to display its parent diagram (the *Timer* block diagram).

You have now defined a state diagram view for the *Control* block and a block diagram view for the *Counter* block. The *Timer* design is completed by using a [truth table](#) to describe the *DtoB* embedded block.

Truth table views are not available if you are using the HDL text design tools. However, alternative procedures for using a HDL text view of the *DtoB* embedded block are given in [Appendix A](#).

Double-click over the *DtoB* embedded block to display the Create Embedded View dialog box and select **Truth Table**. A new truth table is created as an embedded view (*TUTORIAL\Timer\struct:DtoB*). The table is initialized as a matrix of four rows with two input columns (for the *clk* and *d* inputs) and one output column (for the *dat\_in* output).



	A	B	C
1	clk	d	dat_in
2			
3			
4			
5			

## Edit the Truth Table

Click the mouse in either of the blue input columns (A or B) and use the **Add Column** option from the popup menu to add another eight input columns (C to J).

Click the mouse in one of the blue input rows (rows 2, 3, 4 or 5) and use the **Add Row** option to add another seven rows (6 to 12) to the truth table.





You can add multiple rows or columns by selecting more than one cell to add the corresponding number of rows or columns.







Rename the input columns to represent each bit ( $d(9)$  down to  $d(0)$ ) of the  $d$  input bus and enter the value '1' in one row for each column. (The  $clk$  signal is not used and can be renamed.)

Enter the following values in the  $dat\_in$  output column:

```
"0000"
"0001"
"0010"
"0011"
"0100"
"0101"
"0110"
"0111"
"1000"
"1001"
"0000"
```

To add text to a cell, click in the cell, enter the text and click in any other cell to confirm the entry. You can edit the text in an existing cell by double-clicking the cell and using the edit cursor to change individual text characters.

You can also use the  button to copy and the  button to paste text between cells and re-size the rows or columns by dragging the control sash in the non-scrolling area.

You can format text in single or multiple selected cells by using the  (bold),  (italic) or  (underline) buttons and align text using the  (align center),  (align left) or  (align right) buttons.

The completed truth table should look similar to the picture below.

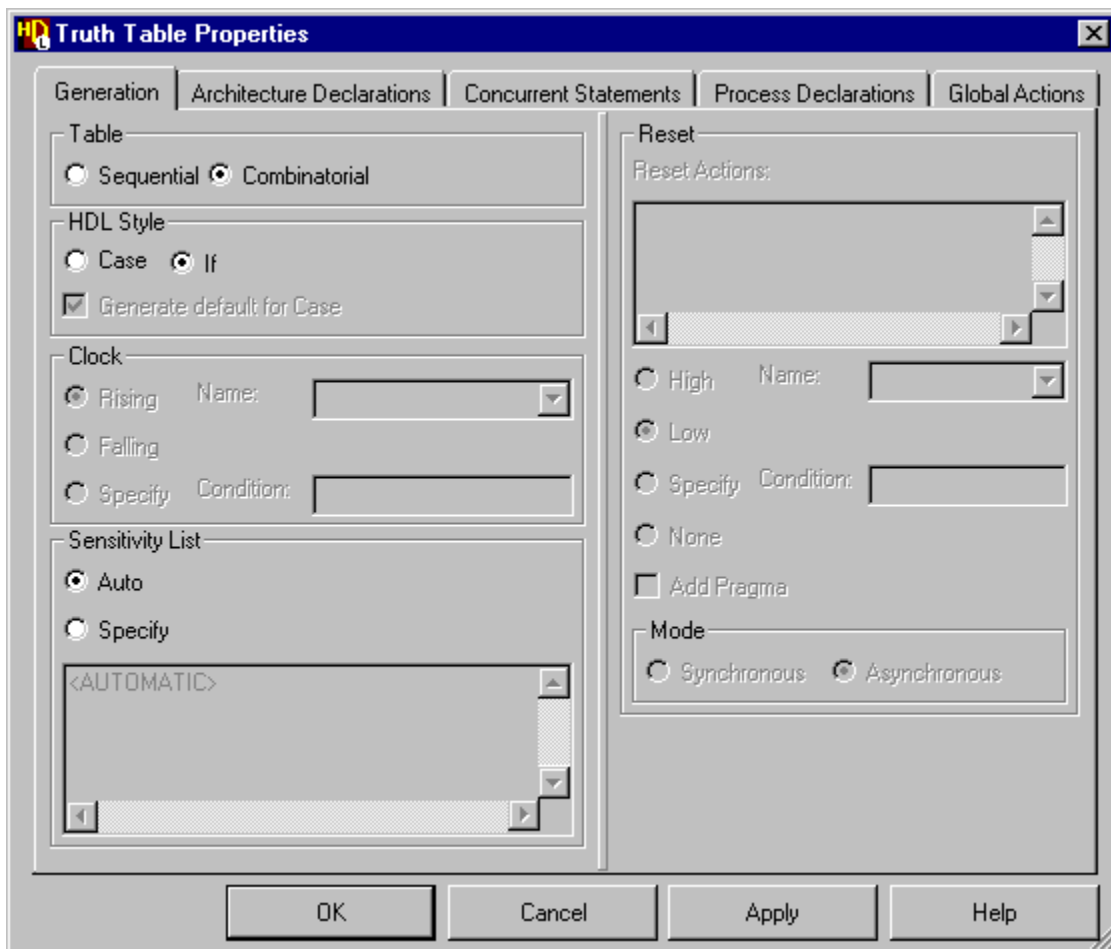
	A	B	C	D	E	F	G	H	I	J	K
1	d(9)	d(8)	d(7)	d(6)	d(5)	d(4)	d(3)	d(2)	d(1)	d(0)	dat_in
2									'1'		"0000"
3									'1'		"0001"
4								'1'			"0010"
5							'1'				"0011"
6						'1'					"0100"
7					'1'						"0101"
8				'1'							"0110"
9			'1'								"0111"
10		'1'									"1000"
11	'1'										"1001"
12											"0000"



The last (empty) row in a truth table represents an ELSE condition and assigns the output expression a known value when none of the input expressions are true.

## Set Truth Table Properties


Choose **Truth Table Properties** from the **Table** menu to display the Truth Table Properties dialog box. Select the **Generation** tab to display the HDL generation characteristics for the truth table. Notice that the Sensitivity List is set to **Auto**. When this option is set, the sensitivity list is automatically created by HDL generation. Check that the default options for **Combinatorial** and **If** style are set and use the  button to confirm the dialog box.

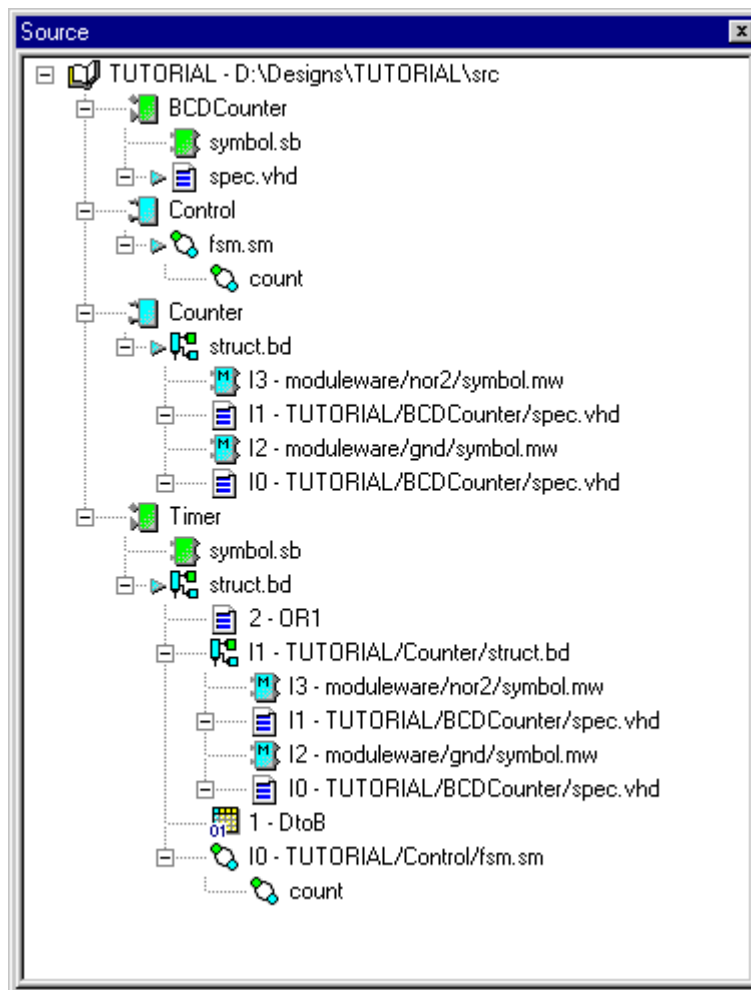


The tabs for Architecture Declarations, Concurrent Statements, Process Declarations and Global Actions can be left empty for this tutorial.

Save the truth table. This also saves the *Timer* block diagram since the truth table is an embedded view within the same design unit as the block diagram.

## Browse the Timer Design




Examine the completed design in the source browser using the  icons to expand each design unit. You can expand the *BCDCounter* component to show it contains a VHDL architecture (*spec.vhdl*) and a symbol (*symbol.sb*). You can expand the *Control* block to show it contains a state machine (*fsm.sm*) and also expand the state machine to show the child hierarchical state machine (*count*). You can expand the *struct.bd* view below the *Counter* block to show it contains two instantiations of the *BCDCounter*. The fully expanded design should look similar to the following picture:



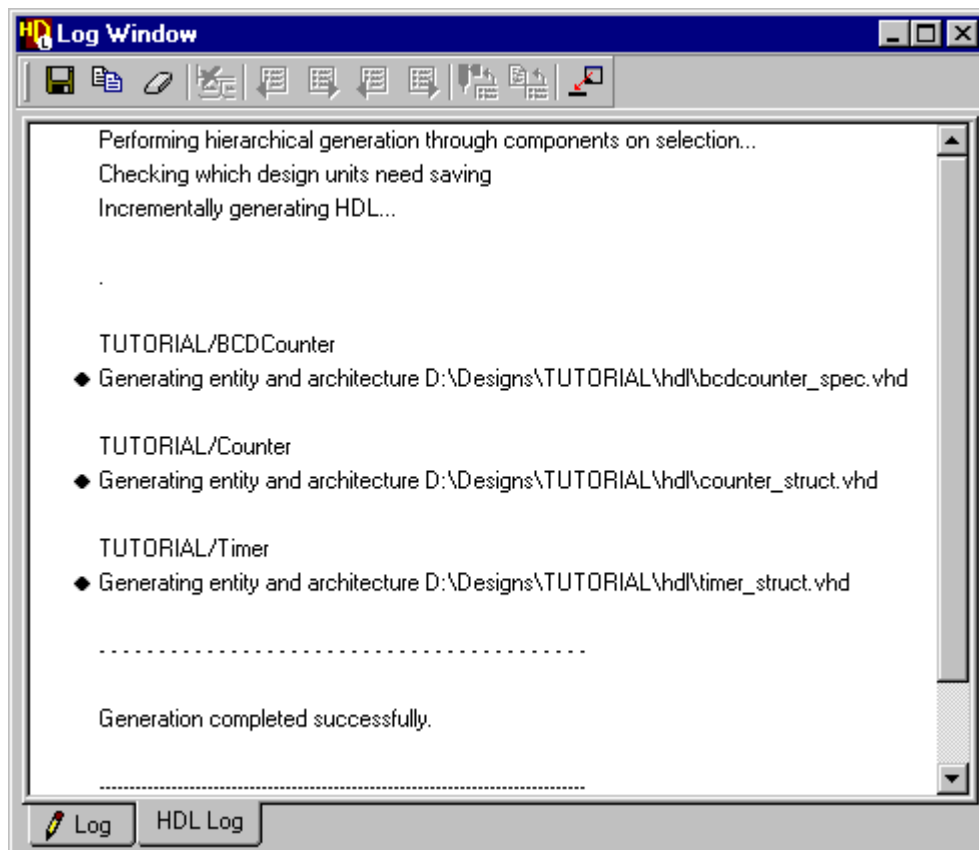
Notice that you can expand the *Timer* component to display the complete hierarchy including the HDL text views describing the *BCDCounter* and the *control* state machine.



## Generate HDL for the Hierarchy




Select the *Timer* design unit in the design browser window and choose the pulldown  on the  button. Select the  option from the palette (or choose the **Generate Through Components** option from the **HDL** menu).


The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation.



This example shows a combined file generated for the **VHDL entity** and **VHDL architecture body**. Separate files may be generated if you have changed the preferences for VHDL file options.



The *Control* design unit was generated earlier in this tutorial and is not generated unless it has been changed. HDL is generated for all other design units. You are prompted to save any design units which have not been saved since they were edited.


If there are any errors, you can move to the next error message using the  button or the previous error using the  button. You can display the source graphics corresponding to the error by double-clicking on the error message (or by using the  button when the error is selected).

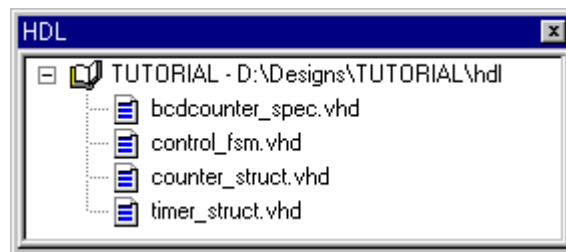
Alternatively, you can use the  button when the error is selected to view the generated HDL. If your text editor supports a line number argument, the HDL line corresponding to the error is selected automatically.

The embedded truth table view is generated as part of the *Timer* design unit. If any syntax errors are issued for the truth table, the corresponding cells are highlighted. For example, if HDL generation encounters an expression which is not defined in the diagram interface, the corresponding input cells are highlighted.



If there are no errors, you can use the toolbar buttons to view the Graphics or HDL corresponding to the “Generating...” message in the HDL Log window.

 You can cross-reference to the graphics or HDL from any message which is marked by the  icon in the log window.


Any files in the generated HDL directory can also be opened from the HDL browser. Click on the  icon for the *TUTORIAL* library in the HDL browser and notice that the view is expanded to reveal the generated files for the each design unit.



You can open any of these files by double-clicking on the generated file name or by choosing **Open** from the popup menu.


 You can also view the generated HDL files for the active diagram by using the  button in any editor window or when the diagram view is selected in the source browser.

## Edit the Timer Symbol

Use the  button (or choose the **Interface** option from the **Open** cascade of the **File** menu) in the *Timer* block diagram. A symbol editor window is opened which shows the external interface for the *Timer* design as a default symbol.

This symbol is used when the *Timer* design is instantiated as a component in a higher level design.

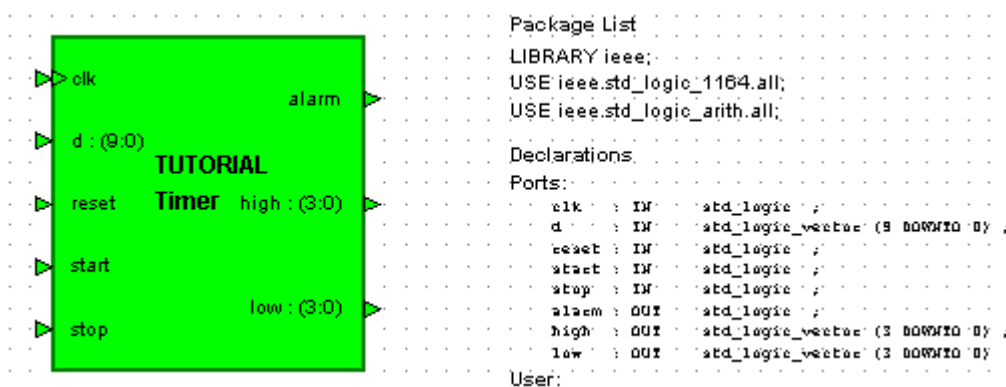
Ports representing the interface connections are shown in default locations but you can select and drag the ports to any location around the body of the symbol. You may also want to resize the symbol body.

 You can redistribute the ports evenly by selecting the symbol body and choosing **Equidistant Ports** from the popup menu.

The full port declarations are shown as a list in the symbol editor and are not shown on the individual ports. However, you can set preferences to control whether type and bounds information is displayed or set the properties visible for an individual port. See the “Changing the Visibility of Symbol Port Properties” help topic for more information.

Select the *clk* port and choose **On** from the **Clock** cascade to apply an edge triggered clock indicator to the port.

The edited symbol should look similar to the picture below:

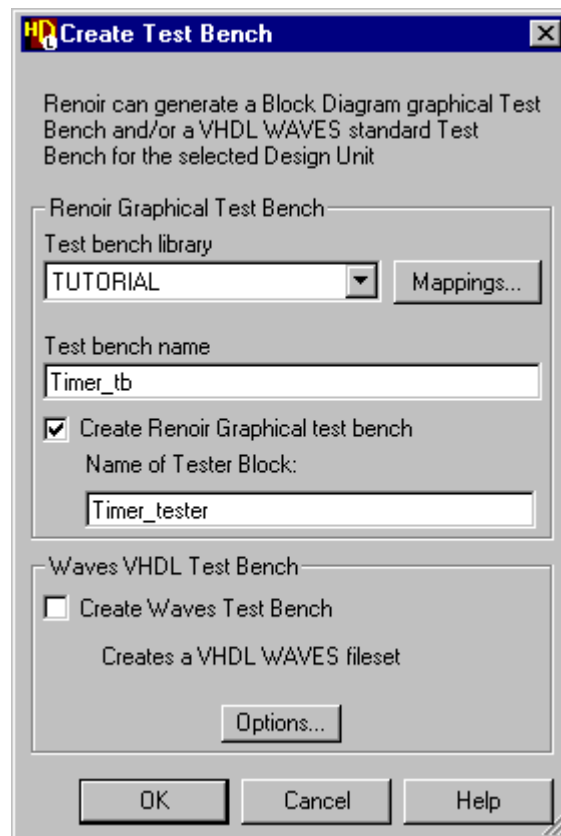


Close and save the symbol.


## Create a Test Bench

Select the *Timer* design unit in the design browser and choose **Create Test Bench** from the **New** cascade in the **File** menu to display the Create Test Bench dialog box. Notice that the dialog box defaults to the *TUTORIAL* library, with the test bench name derived by adding *\_tb* to the *Timer* design unit and the default tester block by adding *\_tester*.

Check that the **Create Graphical test bench** check box is set and confirm the dialog box by clicking the  button.



Notice that a *Timer\_tb* design unit is added to the source browser window.

Click on the  icon for the *Timer\_tb* design unit in the source browser to expand the view and reveal that a symbol and block diagram view (with the default name *struct.bd*) have been created. Double click on the *struct.bd* view to open the block diagram.

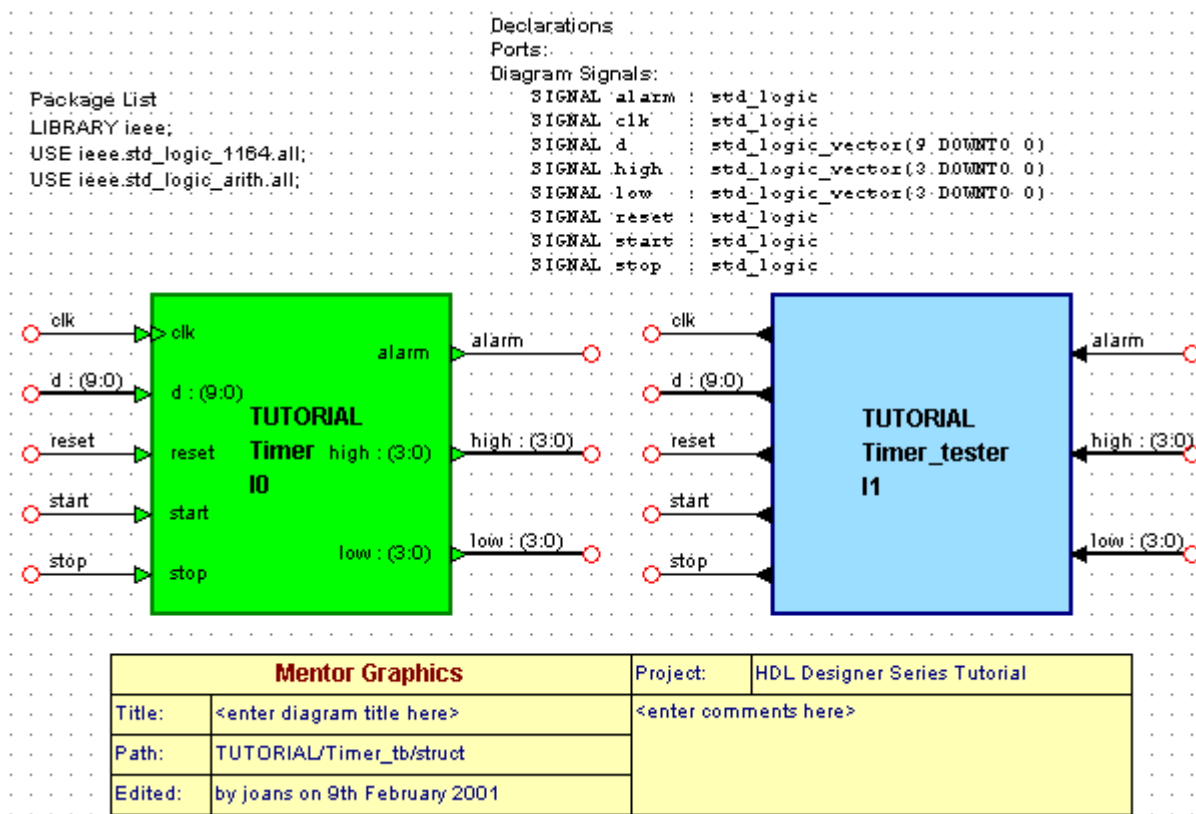
Notice that the *Timer* design has been instantiated as a component with stub signals connected to the input and output ports. The adjacent *Timer\_tester* block has corresponding output and input ports which are implicitly connected by name to the stub signals on the component.



The port locations are identical on the *Timer* component and *Timer\_tester* block but their directions are reversed. Hence the output ports are on the left and input ports on the right of the block.

All the signals and buses are declared as diagram signals since a test bench typically has no external ports.

The test bench block diagram should look similar to the following picture.




It is not necessary to explicitly connect signals and buses between each port on the component and the tester block as these are implicitly connected by name.

## Import the Tester Design Unit

The *Timer\_tester* block should provide test stimulus to the *Timer* design unit and monitor its output. This can be achieved by using a [flow chart](#) or HDL text view.

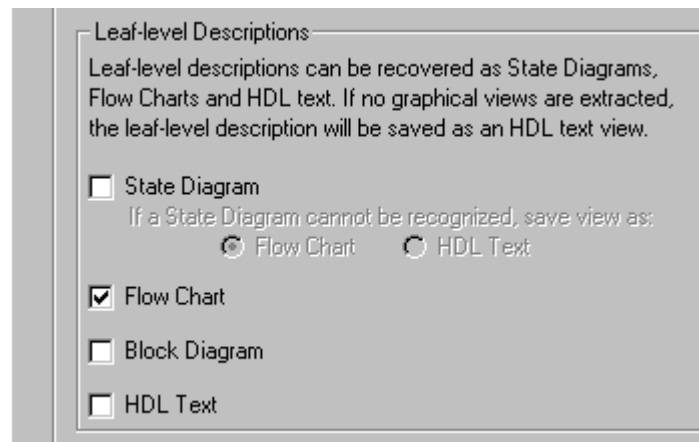
For, this tutorial, the *Timer\_tester* design unit can be imported from the *examples* installation subdirectory using a similar procedure to that used to “[Import the BCDCounter Design Unit](#)” on page 1-44. However, alternative procedures for [Creating a VHDL Flow Chart](#) are provided in [Appendix C](#).

Use the  button (or choose **Full HDL Import** from the **HDL Import** cascade of the **HDL** menu) in the design browser to display the full HDL Import wizard, choose **Specify HDL files** in the first page and select the *Timer\_tester.vhd* file in the Specify HDL Source Files page of the wizard.



The previous HDL import directory and files are saved as preferences. You should only need to remove the *Timer\_BCDCounter.vhd* file and add the *Timer\_tester.vhd* file.

Set the **Flow Chart** option in the Leaf-level Descriptions section of the Choose View Styles page.




This page is only displayed if you are using the graphics design tools and have selected the **Full HDL Import** option. If you are using the text design tools, the tester design unit is imported as a HDL text view.


Use the  button on the Confirm HDL Import page to import the flow chart.

## Instantiate the Imported Tester

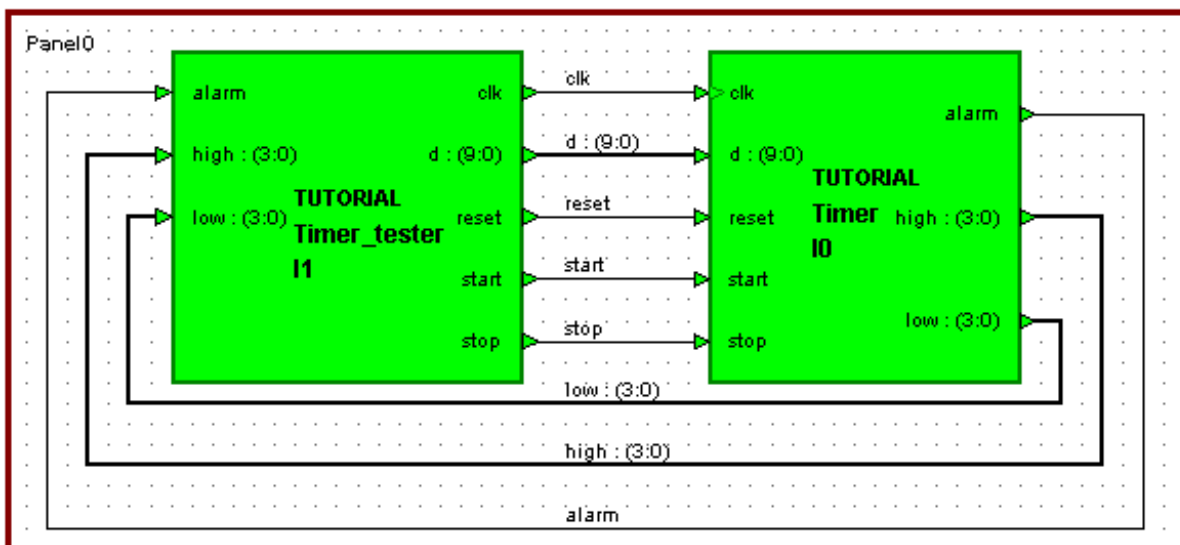
Re-open the *Timer\_tb* block diagram if it is not still open. Select the *Timer\_tester* block including all the connected signal stubs and delete them using the **Del** key.

Use the  key to display the Add Component dialog box and instantiate the imported *Timer\_tester* component in the block diagram to the left of the *Timer* instance. Notice that the *Timer\_tester* and *Timer* components have matching ports but they are located on opposite sides because they have the inverse direction.

Choose **Add Signal Stubs** from the popup menu and add stub signals to the *Timer\_tester* component. The matching nets on the two components are now implicitly connected by name. However, you may like to connect them explicitly by choosing **Autoconnect** from the **Autoroute** cascade of the **Diagram** menu.

Complete the test bench by editing the title and comments in the title block and using the  button to add a panel around the components. This panel can be used to set the diagram view when you simulate your test bench later in this tutorial.

The final test bench should look similar to the picture below:



<b>Mentor Graphics</b>		Project:	HDL Designer Series Tutorial
Title:	<enter diagram title here>	<enter comments here>	
Path:	TUTORIAL/Timer_tb/struct		
Edited:	by joans on 9th February 2001		

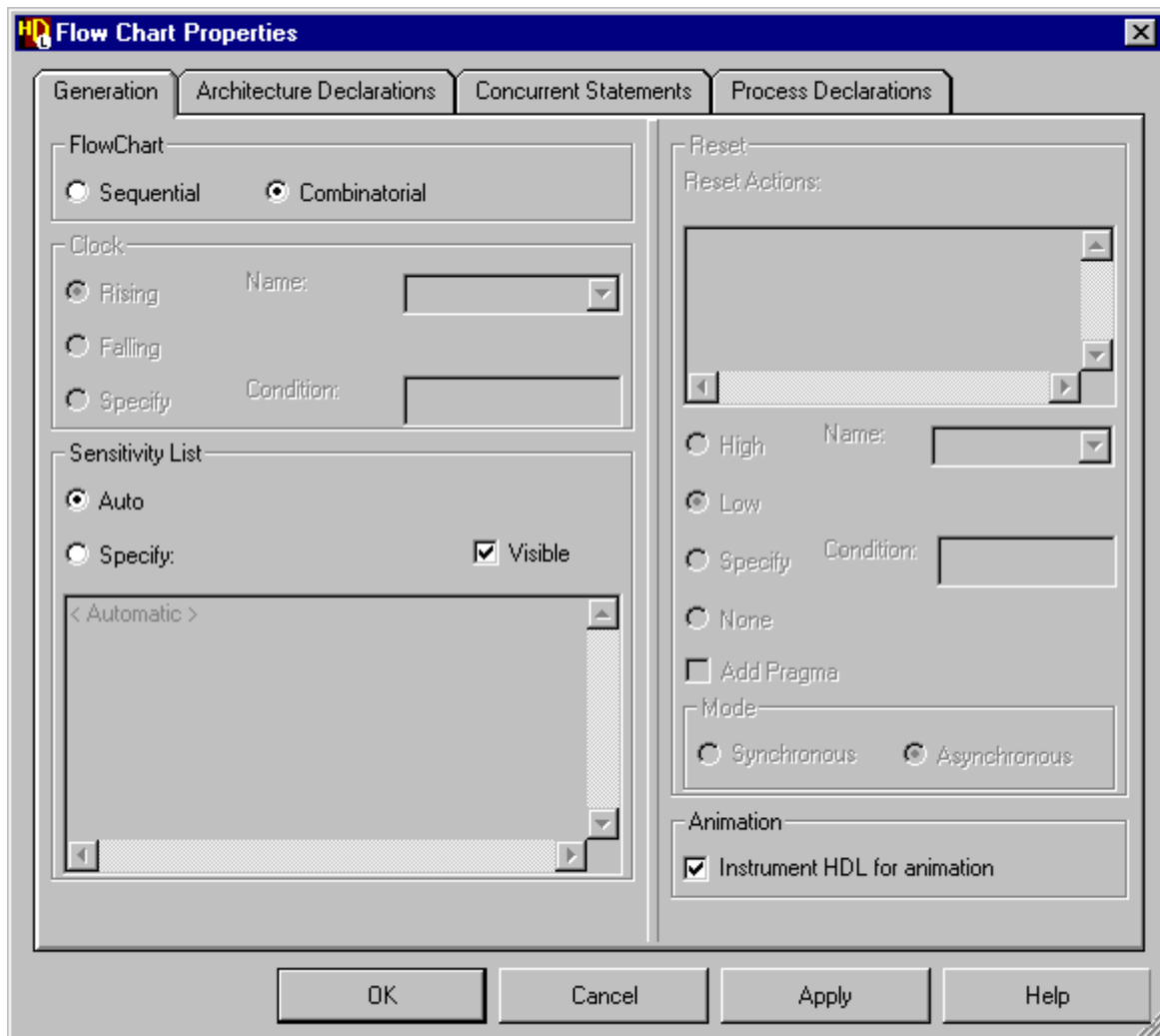
## Generate HDL for the Test Bench

If the *Timer\_tester* component is described by a flow chart, double-click on its instance in the test bench to open the flow chart view and choose **Flow Chart Properties** from the **Edit** menu to display the Flow Chart Properties dialog box.




If you have recovered the *Timer\_tester* component as a HDL text view, all properties for the view are defined in the HDL text and do not need to be set using a dialog box.

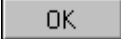
Choose the **Generation** tab and check that the **Combinatorial** option is set.


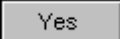




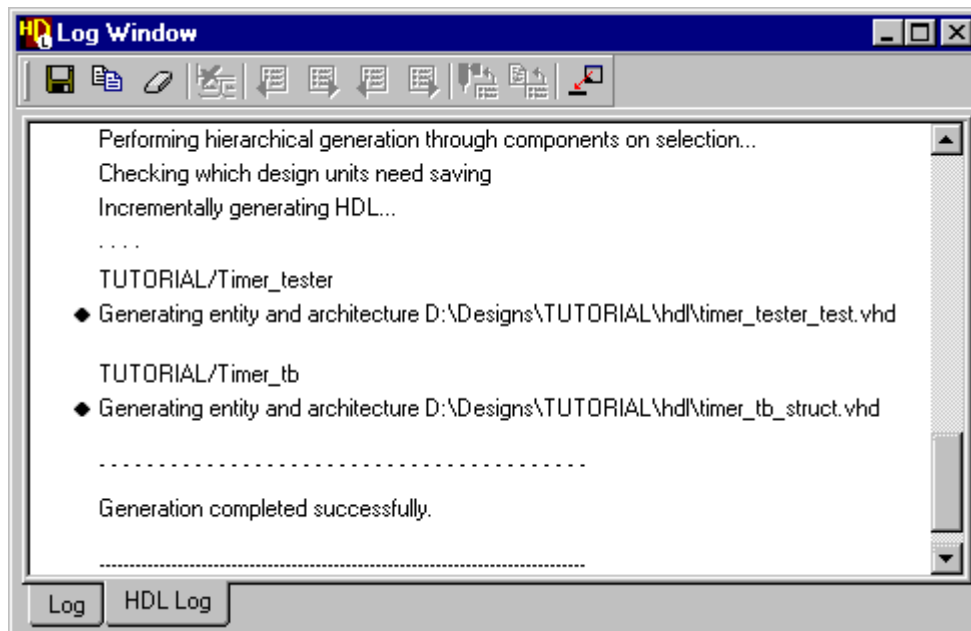
If a ModelSim simulator is available, set the **Instrument HDL for animation** option. This option adds additional code to the generated HDL which is used for flow chart animation later in this tutorial.

 The additional animation code is surrounded by translation control pragmas which ensure that it is ignored by downstream synthesis tools.

Use the  button to confirm the dialog box.

Select the *Timer\_tb* design unit in the design browser and choose the  button (or choose the **Generate Through Components** option from the **HDL** menu). All views to be generated must have been saved and if you have not saved the test bench you are prompted whether to continue. Confirm the dialog box by clicking the  button.

VHDL is generated for the test bench and tester design units (but not the *Timer* design hierarchy since these design units have been generated earlier in this tutorial and do not need to be regenerated unless they have been modified).



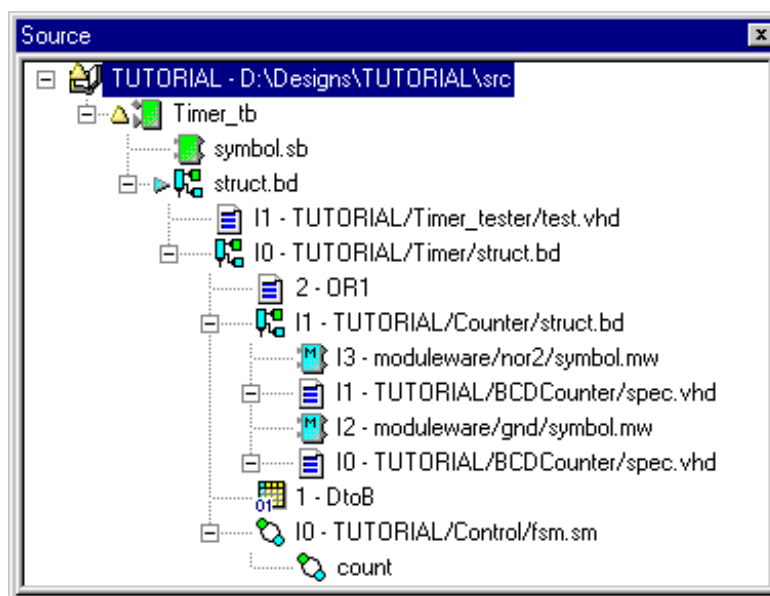
The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation. If there are any errors, you can display the corresponding graphics by double-clicking on the error message as described in the section [“Generate HDL for the Hierarchy” on page 1-61](#).

## Browse the Completed Design

View the hierarchy of the *Timer\_tb* design unit in the design browser by choosing **Expand All** from the design browser **View** menu or by using the  $\oplus$  icons to expand each design unit and display the full hierarchy which also includes all views of the *Timer* design.

Select the *Timer\_tb* design unit in the source browser and choose **Toggle Top Marker** from the **Edit** menu. Notice that the  $\triangle$  icon is displayed adjacent to the design unit. Notice that the  $\triangle$  icon is also shown against the *BCDCounter* and *Timer\_tester* design units. This is because they were automatically marked as top level design units when they were imported. Use **Toggle Top Marker** to unset the marker for these design units.

Select the *TUTORIAL* library and choose **Show Marked Design Units** from the **View** menu. Notice that all the unmarked design units are hidden in the design browser. However, all the design unit views are accessible from the *Timer\_tb* hierarchy.



You can restore the hidden design units by choosing **Show All Design Units** from the **View** menu.

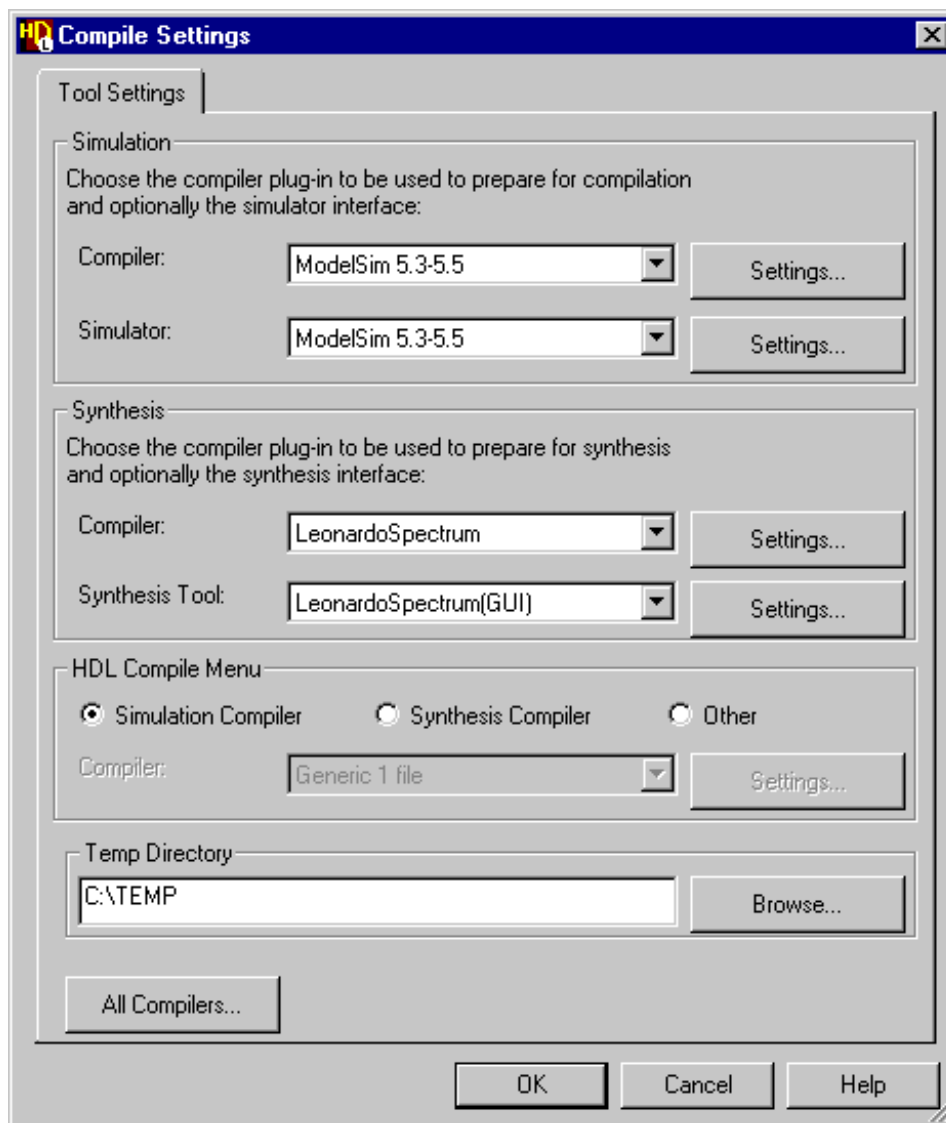
## Setup the Downstream Tools

For this tutorial, it is assumed that a ModelSim compiler is available. You can alternatively prepare data for use with other compatible downstream tools.



If you have installed the HDL Designer Series tool as part of FPGA Advantage, the ModelSim simulator and LeonardoSpectrum synthesis tools will already have been set up automatically.

Choose **Compile** from the **Options** menu to display the **Tool Settings** tab of the Compile Settings dialog box.

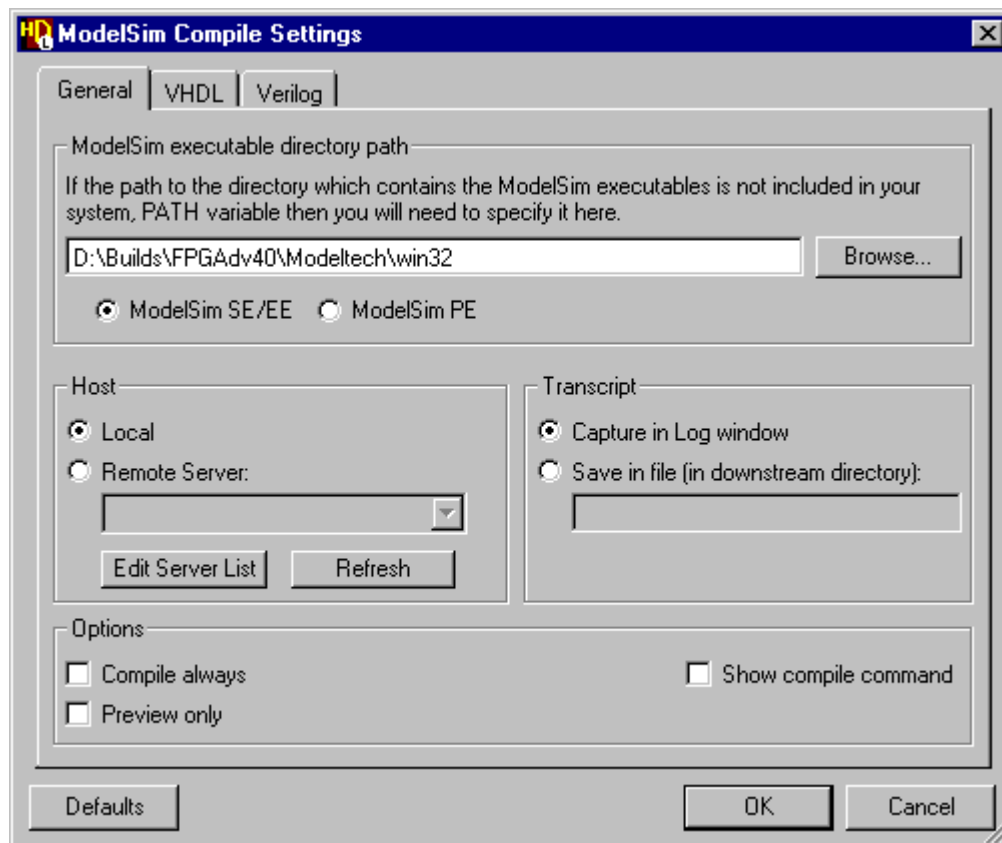


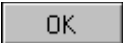
Choose a simulation compiler and simulator that is available on your workstation.



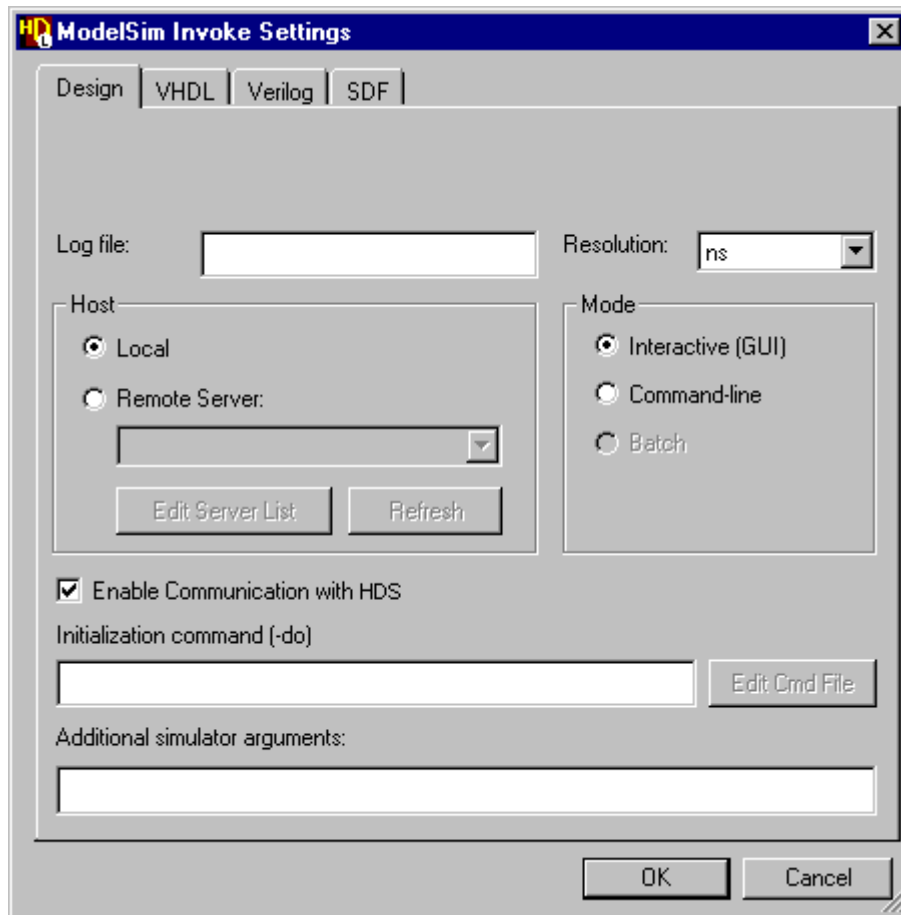
Select the ModelSim 5.3-5.5 compiler and ModelSim 5.3-5.5 simulator if version 5.3 or later of the ModelSim SE/EE or PE simulator is available.

Use the simulation compiler  button to display the tool settings for your compiler. For example, the ModelSim Compile Settings dialog box:




Enter the pathname to the simulation compiler executable if it is not already accessible using your PATH environment variable. If you are using ModelSim, ensure that the correct edition (Special or Elite Edition: SE/EE; Personal Edition: PE) is selected. All other options can normally be left in their default state. Use the  button to confirm and close the dialog box.

Use the Simulator  button to display the default invoke settings for your simulator. For example, the ModelSim Invoke Settings dialog box:






Accept the default resolution and check the **Enable Communication with HDS** and **Interactive** check boxes are selected. The other entry fields can be left in their default state. Use the  button to confirm the dialog box.

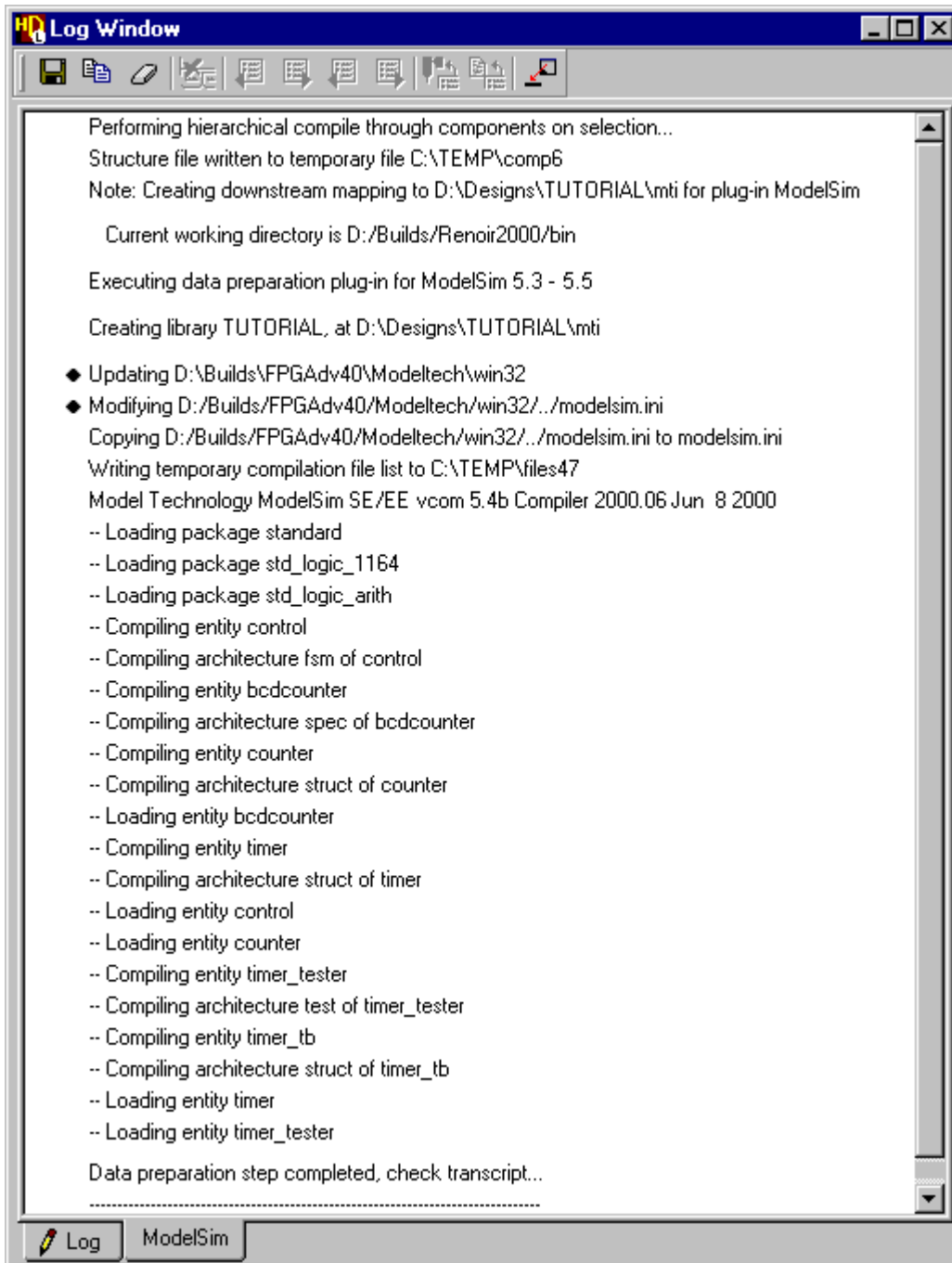
 Refer to the online help topics for information about setting options for other supported downstream tools.

On both PC or UNIX systems, ensure that the temporary directory specified in the Compile Settings dialog box (which normally defaults to `/tmp` on UNIX or to `C:\TEMP` on PC systems) exists.

Use the  button to confirm the Compile Settings dialog box.

## Compile the Design

Select the *Timer\_tb* design unit in the design browser and choose the pulldown  on the  button. Select the  option from the palette to **Compile Through Components**. The progress of the compilation is shown in the HDL Log Window.



```
HD Log Window
Performing hierarchical compile through components on selection...
Structure file written to temporary file C:\TEMP\comp6
Note: Creating downstream mapping to D:\Designs\TUTORIAL\mti for plug-in ModelSim

Current working directory is D:/Builds/Renoir2000/bin

Executing data preparation plug-in for ModelSim 5.3 - 5.5


Creating library TUTORIAL, at D:\Designs\TUTORIAL\mti




◆ Updating D:\Builds\FPGAAdv40\Modeltech\win32
◆ Modifying D:/Builds/FPGAAdv40/Modeltech/win32/./modelsim.ini
Copying D:/Builds/FPGAAdv40/Modeltech/win32/./modelsim.ini to modelsim.ini
Writing temporary compilation file list to C:\TEMP\files47
Model Technology ModelSim SE/EE vcom 5.4b Compiler 2000.06 Jun 8 2000
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package std_logic_arith
-- Compiling entity control
-- Compiling architecture fsm of control
-- Compiling entity bcdcounter
-- Compiling architecture spec of bcdcounter
-- Compiling entity counter
-- Compiling architecture struct of counter
-- Loading entity bcdcounter
-- Compiling entity timer
-- Compiling architecture struct of timer
-- Loading entity control
-- Loading entity counter
-- Compiling entity timer_tester
-- Compiling architecture test of timer_tester
-- Compiling entity timer_tb
-- Compiling architecture struct of timer_tb
-- Loading entity timer
-- Loading entity timer_tester

Data preparation step completed, check transcript...



Log ModelSim
```

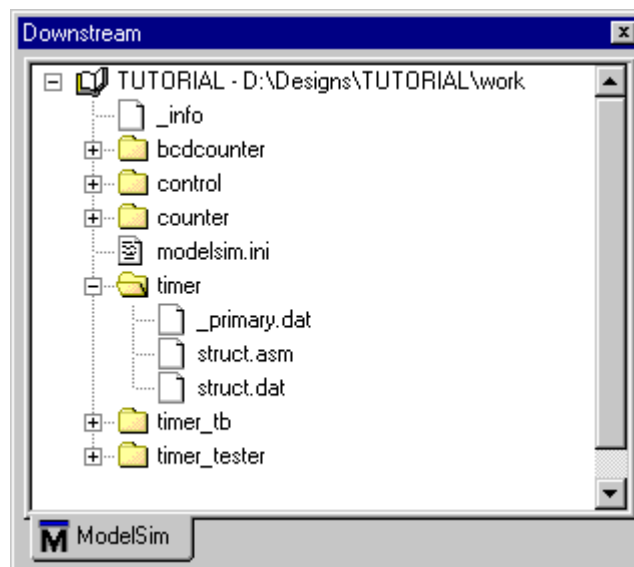
There should be no errors or warnings. If there are any errors, you can display the corresponding source diagram by double-clicking on the error message as described in the section “[Generate HDL for the Hierarchy](#)” on page 1-61.


After correcting any errors in the source diagrams, you can regenerate, recompile and start the simulator in a single step by choosing the  flow button to generate and compile all changed design unit views in the hierarchy beneath the active design unit. Notice that unmodified views are not re-generated unless you choose **Set Generate Always** from the HDL menu to force HDL generation.

 The flow buttons  and  are setup by default for a simulation flow (using ModelSim) and a synthesis flow (using LeonardoSpectrum). Refer to the “[Setting a Custom Flow](#)” online help topic if you want to modify the setup for either of the design flow buttons to use alternative tools.

Notice that downstream mapping is automatically created for your downstream data and the compiled objects are displayed in the [downstream browser](#).

Click on the  icon for the *TUTORIAL* library in the downstream browser and notice that the view is expanded to reveal a subdirectory for each design unit in your design. You can use the  icons to browse each of these folders.





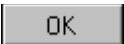
 Uppercase characters in VHDL design unit names are downcased by the ModelSim compiler.

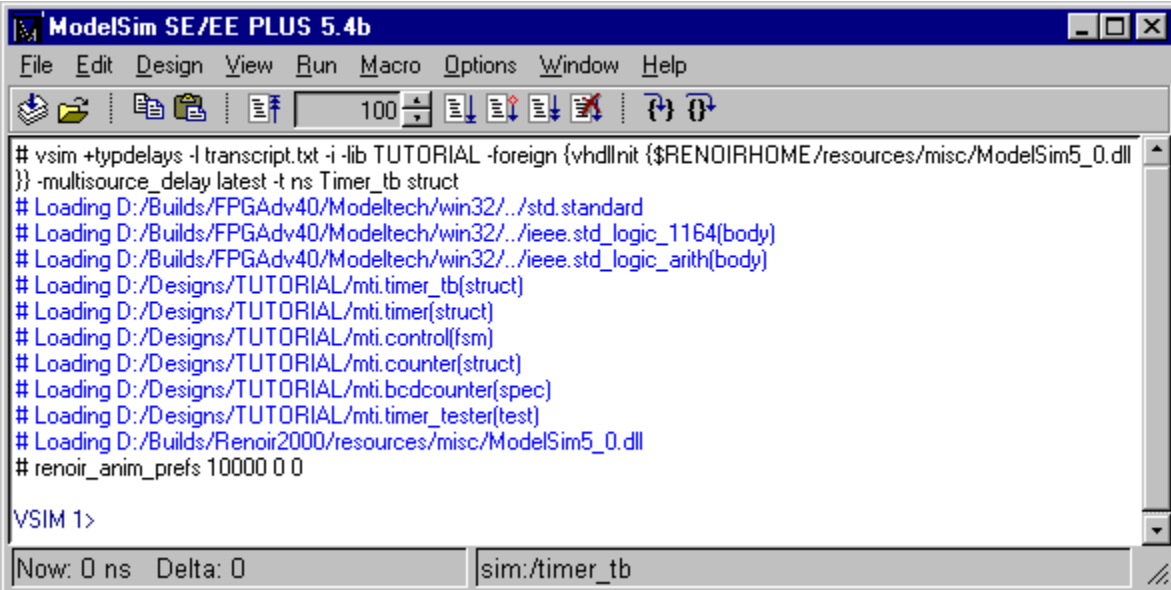
## Invoke the ModelSim Simulator

The simulator can be invoked from the design browser or from any block diagram, flow chart or state diagram in your design. However, it must be invoked at a level that includes the entire branch of your design that you want to test. This tutorial uses the test bench to verify your version of the *Timer* design.



You must have generated and compiled HDL for all design units in the hierarchy you want to simulate. If you did not set **Instrument HDL for animation** in the state machine and flow chart properties, refer to the topics “[Set Generation Properties](#)” on page 1-39 and “[Generate HDL for the Test Bench](#)” on page 1-68. If necessary, you can regenerate and recompile your design by choosing the  button to generate and compile all changed design unit views in the hierarchy beneath the active design unit. By default, this button will also start (or restart) the simulator if the generation and compilation are completed successfully.

Select the *Timer\_tb* design unit in the design browser and use the  button (or choose **Start Simulator** from the **HDL** menu) to display the Start dialog box for your simulator. Use the  button to confirm the invoke options.



```

ModelSim SE/EE PLUS 5.4b
File Edit Design View Run Macro Options Window Help
# vsim +typdelays -l transcript.txt -i -lib TUTORIAL -foreign {vhdllnit {$RENOIRHOME/resources/misc/ModelSim5_0.dll
}} -multisource_delay latest -t ns Timer_tb struct
# Loading D:/Builds/FPGAAdv40/Modeltech/win32/./std.standard
# Loading D:/Builds/FPGAAdv40/Modeltech/win32/./ieee.std_logic_1164(body)
# Loading D:/Builds/FPGAAdv40/Modeltech/win32/./ieee.std_logic_arith(body)
# Loading D:/Designs/TUTORIAL/mti.timer_tb(struct)
# Loading D:/Designs/TUTORIAL/mti.timer(struct)
# Loading D:/Designs/TUTORIAL/mti.control(fsm)
# Loading D:/Designs/TUTORIAL/mti.counter(struct)
# Loading D:/Designs/TUTORIAL/mti.bcdcounter(spec)
# Loading D:/Designs/TUTORIAL/mti.timer_tester(test)
# Loading D:/Builds/Renoir2000/resources/misc/ModelSim5_0.dll
# renoir_anim_prefs 10000 0 0
VSIM 1>
Now: 0 ns Delta: 0 sim:/timer_tb
  
```

The simulator is invoked and issues a number of loading messages ending with:  
 # hds\_anim\_prefs 10000 0 0



## Setup the Simulator Windows


An additional Simulation toolbar is available in the block diagram, flow chart and state diagram when a simulator is invoked to support cross-probing between the simulator and source design objects in the HDL Designer Series tool. This toolbar is normally displayed automatically at the bottom of the diagram window but can be undocked, moved, docked or hidden in the same way as the other toolbars.




Display the *Timer\_tb* block diagram and choose **View Panel** from the **View** menu to display the panel you added earlier in this tutorial.



If you added more than one panel, a dialog box is displayed for you to choose the panel to display.

Select the signals *alarm*, *clk*, *d*, *reset*, *start*, *stop*, *high* and *low*. Use the  button from the Simulation toolbar or choose **Add Wave** from the **Display** cascade in the **Simulation** menu and notice that the simulator Wave window is automatically opened with these signals loaded.

Use the  button from the Simulation toolbar or choose **Add List** from the **Display** cascade in the **Simulation** menu to add the selected signals to the simulator list window.




You can optionally add signals to the simulation log without displaying them in the Wave or List window by choosing **Add Log** from the **Display** cascade of the **Simulation** menu.

You can also open any other simulator window by choosing from the **View** cascade of the **Simulation** menu. For example, you may wish to use this menu to open the ModelSim **Source** and **Structure** windows.

## Enable Animation

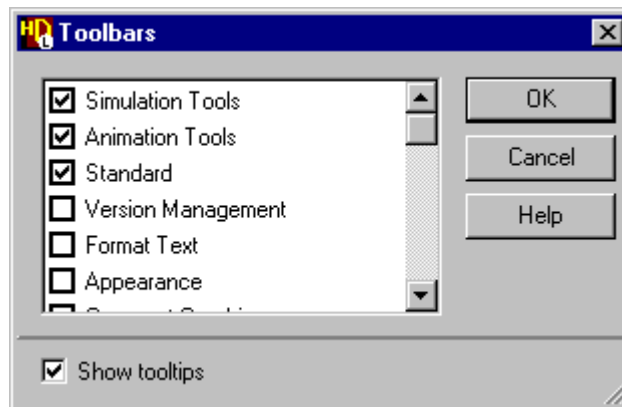
Display the *Control* state diagram and its child hierarchical state diagram. Use the **View Panel** command in each window to adjust the window view to the panel areas you defined earlier in this tutorial.

-  You can optionally also display the *Timer\_tester* flow chart. Flow charts (including hierarchical and concurrent flow charts) can be animated in a similar way to that described below for the state diagrams.


Notice that an additional Animation toolbar is available in the state diagram (and flow chart) windows when the simulator is invoked. This toolbar is normally docked at the bottom of the diagram window next to the Simulation toolbar but can be moved independently.




The editing toolbars are not usually required during simulation and animation. You can hide or show toolbars using the Toolbars dialog box which is displayed by choosing **Settings** from the **Toolbars** cascade of the **View** menu.




For example, use the dialog box to hide the Format Text, Appearance, Comment Graphics and Arrange Object toolbars.


-  Individual toolbars can also be hidden by unsetting the corresponding options in the **Toolbars** cascade of the **View** menu.

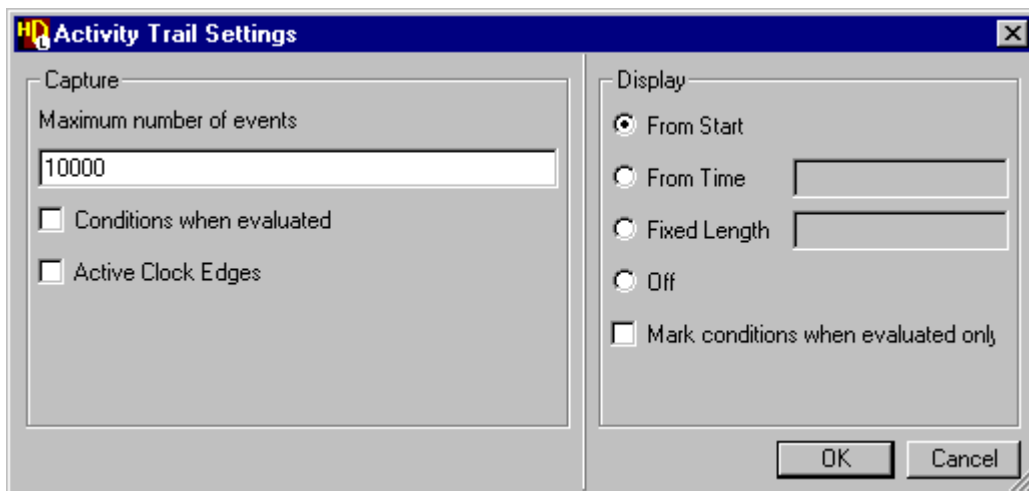
Use the  buttons from the Animation toolbars (or choose **Data Capture** from the **Animation** menus) in each state diagram or flow chart you want to animate. (It is not necessary to repeat this command for each hierarchical or concurrent view since these are considered part of the same diagram.)

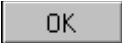
Notice that all objects (except the start state, exit and entry points in the state diagram and the start point in the flow chart) are redrawn with white fill. The start state and start point are drawn with red fill to indicate that they represent the current animation state.



This animation view is automatically displayed when you set data capture but can be toggled at any time by using the  button (or by toggling the **Show Animation** option in the **Animation** menu). You can also set data capture for all instances in the current simulation hierarchy by choosing **Global Capture On** from the **Animation** menu.

Use the  button from the Animation toolbar (or choose **Activity Trails** from the **Animation** menu) in the flow chart or state diagram window and choose the **From Start** option in the Activity Trail Settings dialog box.






Use the  button to confirm the Activity Trail Settings dialog box.





The activity trail is applied to all windows in the current simulation and any flow chart or state diagram window.

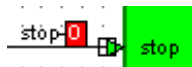
## Simulate the Design


Run the simulator for the default timestep (100 nano-seconds) by using the  button or by choosing **For Time** from the **Run** cascade of the **Simulation** menu in the block diagram, flow chart or state diagram window.


 The default timestep can be changed by using the  button to select from a list of alternative timesteps or choose a user specified timestep.

Notice that the *Initialization* action box is highlighted red in the animated flow chart for the *Timer\_tester* and the initial signal values are updated in the simulator Wave and List windows.


Select the *stop* signal in the *Timer\_tb* block diagram and set a breakpoint by using the  button or by choosing **Add** from the **Breakpoints** cascade of the **Simulation** menu. Set a simulation probe on the same signal by using the  button or by choosing **Add** from the **Probes** cascade of the **Simulation** menu. Notice that the signal value shown in the probe is '0'.




Run the simulator until there are no more events scheduled by using the  button from the Simulation toolbar or by choosing **Forever** from the **Run** cascade of the **Simulation** menu.

 Run commands can be issued from the toolbars or menus in any block diagram, flow chart or state diagram window in the current simulation environment.



The simulation should run until the breakpoint is encountered when the *stop* signal value (shown by the simulation probe) changes to '1'. Notice that the *Store* action box is highlighted in red as the current step in the animated flow chart and the *counting* state is entered in the child animated state machine for the *count* state.

Use the  button or choose **Continue** from the **Run** cascade of the **Simulation** menu to continue the simulation until the *stop* signal changes back to zero. Notice that the *suspended* state is entered in the animated state machine and the following note is issued in the main simulation window:

```
**Note: Count suspended correctly.
```

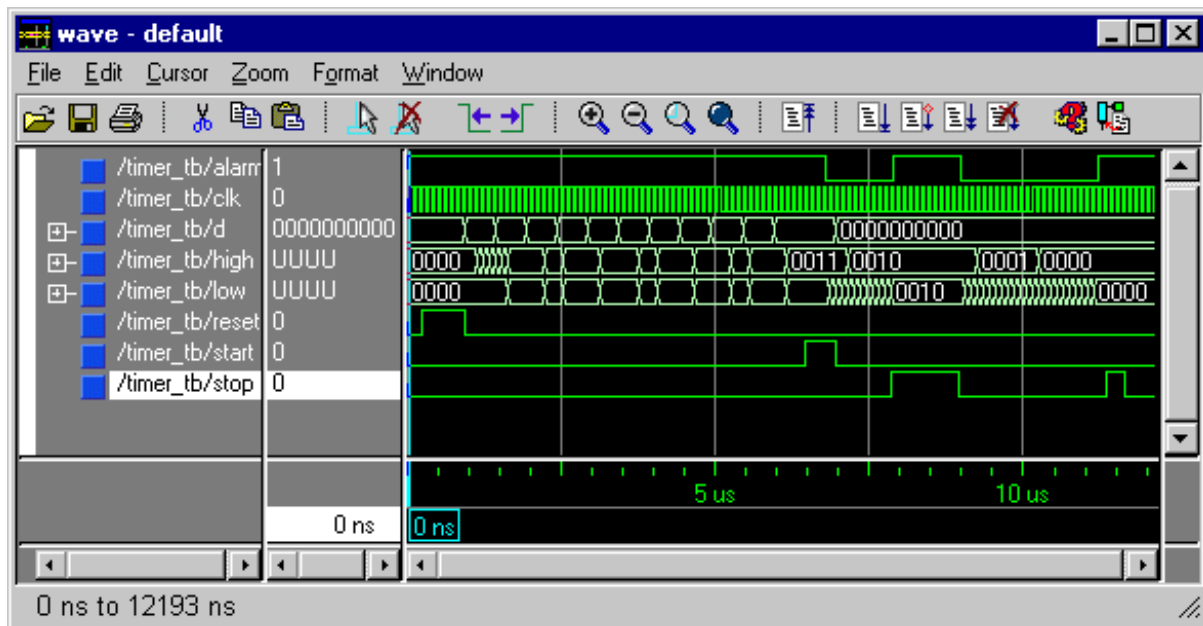
Run the simulator to the next change of the *stop* signal by using the  button and notice the message:

```
**Note: Alarm asserted correctly
```


Use the  button in the test bench block diagram to delete the breakpoint on the *stop* signal and complete the simulation by using the  button. The simulation should run to completion and issue the message:


```
**Failure: Timer test completed
```



Examine the results displayed in the Wave window and ensure that the simulation has performed correctly by comparison with the example waveforms below.



You can display the full simulation waveform by choosing **Zoom Full** from the Wave window **Zoom** menu.


If any VHDL errors are discovered, correct your design and use the  button to regenerate and recompile the modified diagram. If the diagram is successfully generated and compiled, the flow button automatically restarts the simulation for the current simulation hierarchy.






Alternatively, you can use the  button or choose **Restart Simulator** from the Simulation menu to repeat the simulation.


 You can use the  button or choose **Highlight Object** from the **Display** cascade of the simulation menu to highlight all occurrences in the simulation windows of a signal corresponding to any selected graphical object.




## Review the Animation

Notice how the animation activity trail in both the flow chart and state diagram is highlighted in blue and the exit break point for `Timer test completed` is indexed on the flow chart VHDL architecture displayed in the simulator Source window.

Use the  button or choose **Link Diagrams** from the **Animation** menu to link the animated diagrams. When the diagrams are linked, all animation review commands are applied to all animated diagrams in the simulation hierarchy.


You can review the animation by using the , , ,  or  buttons (or by choosing **Goto Next**, **Goto Previous**, **Goto Time**, **Goto Start** or **Goto Latest** from the **Animation** menu).


The  and  buttons step through each object in the flow chart.

However in the state diagram, you can set options in the **Animation** menu (or from a pulldown menu on the toolbar button) to move by change of state , simulation event  or change of clock edge . Notice how the toolbar button icon changes to indicate the current mode of movement.


You may want to change the activity trail to display the trail from a specified time or specify a fixed length. The current simulation step (or current state and last transition taken in an animated state diagram) is always shown in red and the previous step (or previous state and transition) in yellow. All other visited objects included in the activity trail are shown in blue. Remember that you can open down the hierarchical action box or hierarchical state to view the animation in the child diagrams.

Compare the animation with the results displayed in the Waveform window and ensure that the simulation has performed correctly by comparison with the example waveforms on the previous page.

A  button is available in the Animation toolbar and a **Cause** option from the **Animation** menu. These commands can be used to move the ModelSim Wave and List windows to the current animation time.

A corresponding  button and **Cause** option (in the **Cursor** menu) are available in the ModelSim Wave window. These commands can be used to update all currently open animation windows to the simulation step or event immediately preceding the time marked by the Wave window cursor.

A **Cause** command is also added to the **Markers** menu in the simulator List window which can be used to update all open animation windows to the simulation step or event corresponding to the selected line in the List window.

A  button is added to the toolbar (and a **Show HDS Source** option is added to the **Edit** menu) in the Source window and can be used to open the graphic window corresponding to the line of HDL code under the cursor.


See the HDL Designer Series help topics “Cross Probing from ModelSim” and “Using the ModelSim Source Window” for more information about the cross-probing commands added to ModelSim when it is used with a HDL Designer Series tool.

Exit the simulator (by choosing **Quit** from the **File** menu in the main ModelSim window). Any other simulator windows are automatically closed.

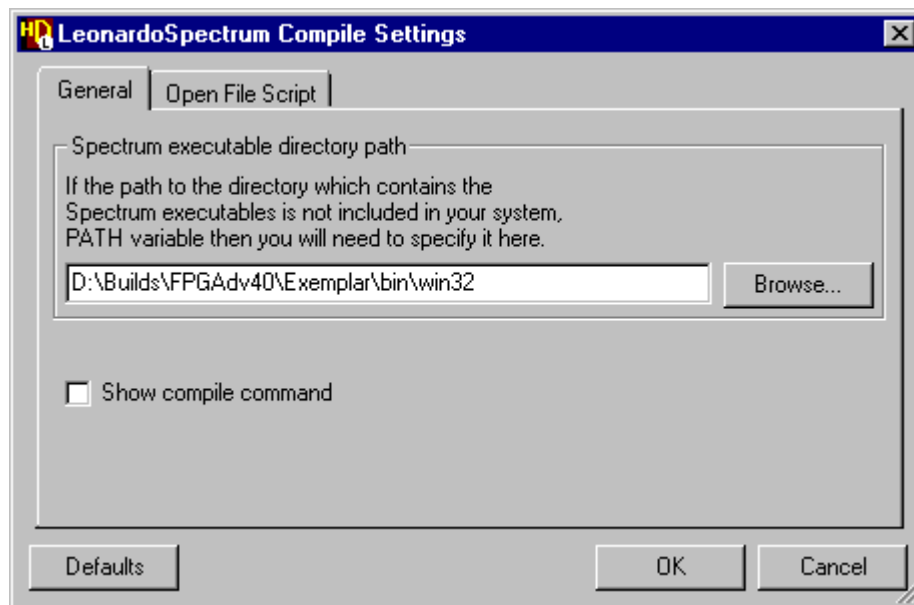
Close all graphic editor windows (except the design browser) by choosing **Close All Windows** from the **File** menu in the design browser.

## Setup the Synthesis Tool

If a synthesis tool is available on your workstation you can synthesize your completed design. The Exemplar LeonardoSpectrum or Synopsys Design Analyzer tools can be invoked directly from within a HDL Designer Series tool.

The synthesis tool can be setup using the synthesis compiler  button in the **Tool Settings** tab of the Compile Settings dialog box in a similar way to the simulation compiler which was described in “[Setup the Downstream Tools](#)” on [page 1-71](#).

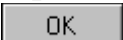
For example, the following picture shows the LeonardoSpectrum Compile Settings dialog box.



Enter the pathname to the synthesis compiler executable if it is not already accessible using your PATH environment variable.

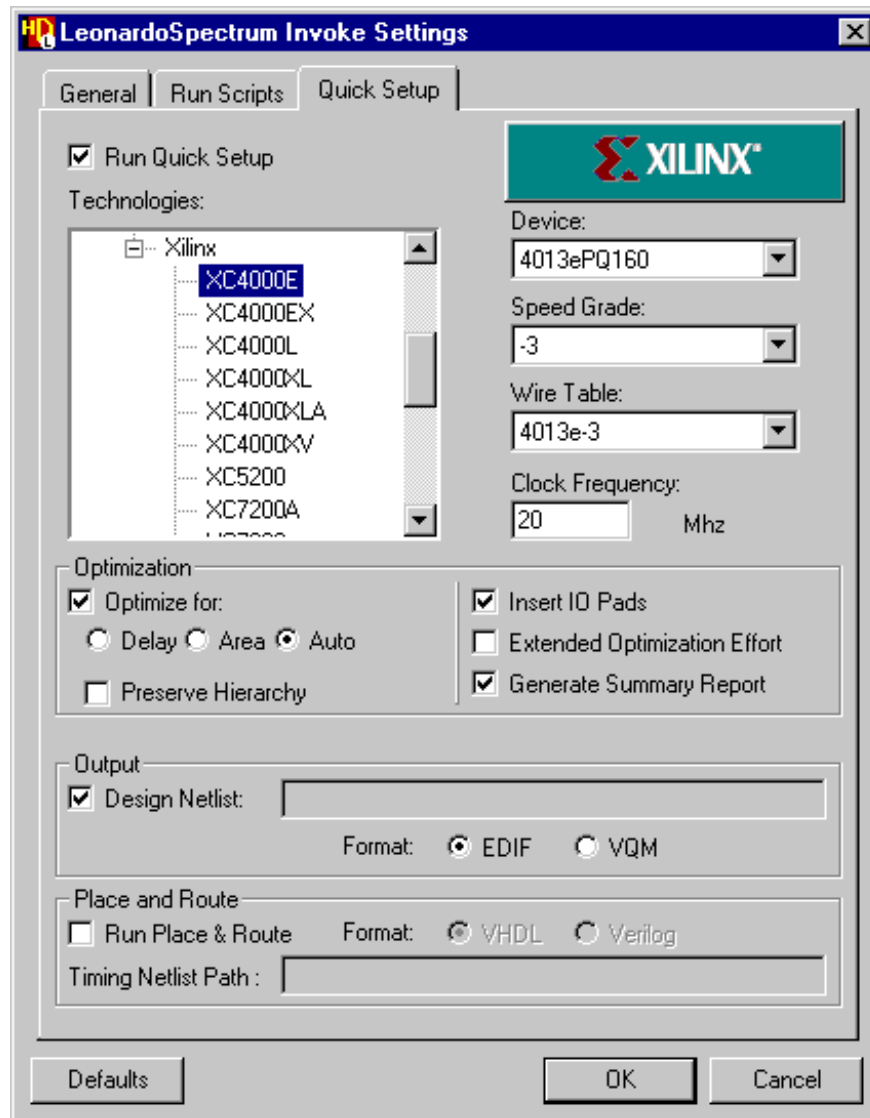


If you have installed the HDL Designer Series tool as part of FPGA Advantage, the ModelSim simulator and LeonardoSpectrum synthesis tools will already have been set up automatically.

The options in the **Open File Script** tab can be left with their default values. Use the  button to confirm and close the dialog box.




Use the synthesis tool  button to display the default invoke settings for your tool. If you are using LeonardoSpectrum 2000 (or later), the **Quick Setup** tab is displayed:




Use the  icons to expand the FPGA/CPLD technologies list and select the technology of your choice. For example, Xilinx XC4000E as shown above.





If you have an Exemplar level 3 license, ASIC and FPGA technology libraries are available. If you have a level 2 license, only the FPGA libraries are available.

The Device, Speed Grade and Wire Table fields are automatically selected when you have chosen a technology. Complete the dialog box by entering a clock frequency (for example 20 MHz) and use the  button to confirm the invoke settings dialog box. All other options can be left with their default values.

Use the  button to confirm and close the Compile Settings dialog box.

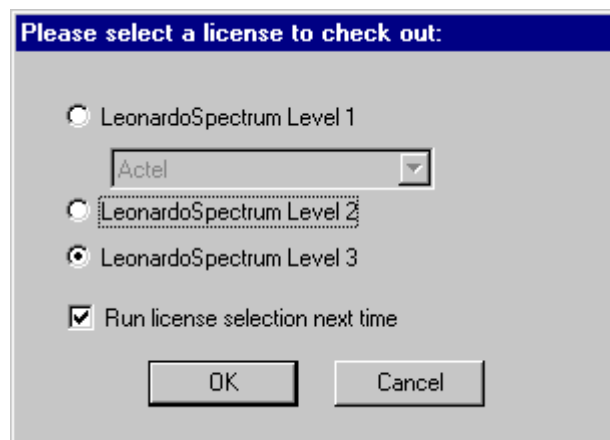
## Run the Synthesis Flow


Select the *Timer* design unit in the source browser and use the  button to invoke the synthesis flow.

-  You cannot synthesize the test bench or flow chart which contain unsynthesizable HDL. Ensure that you have selected the *Timer* design unit in the source browser.

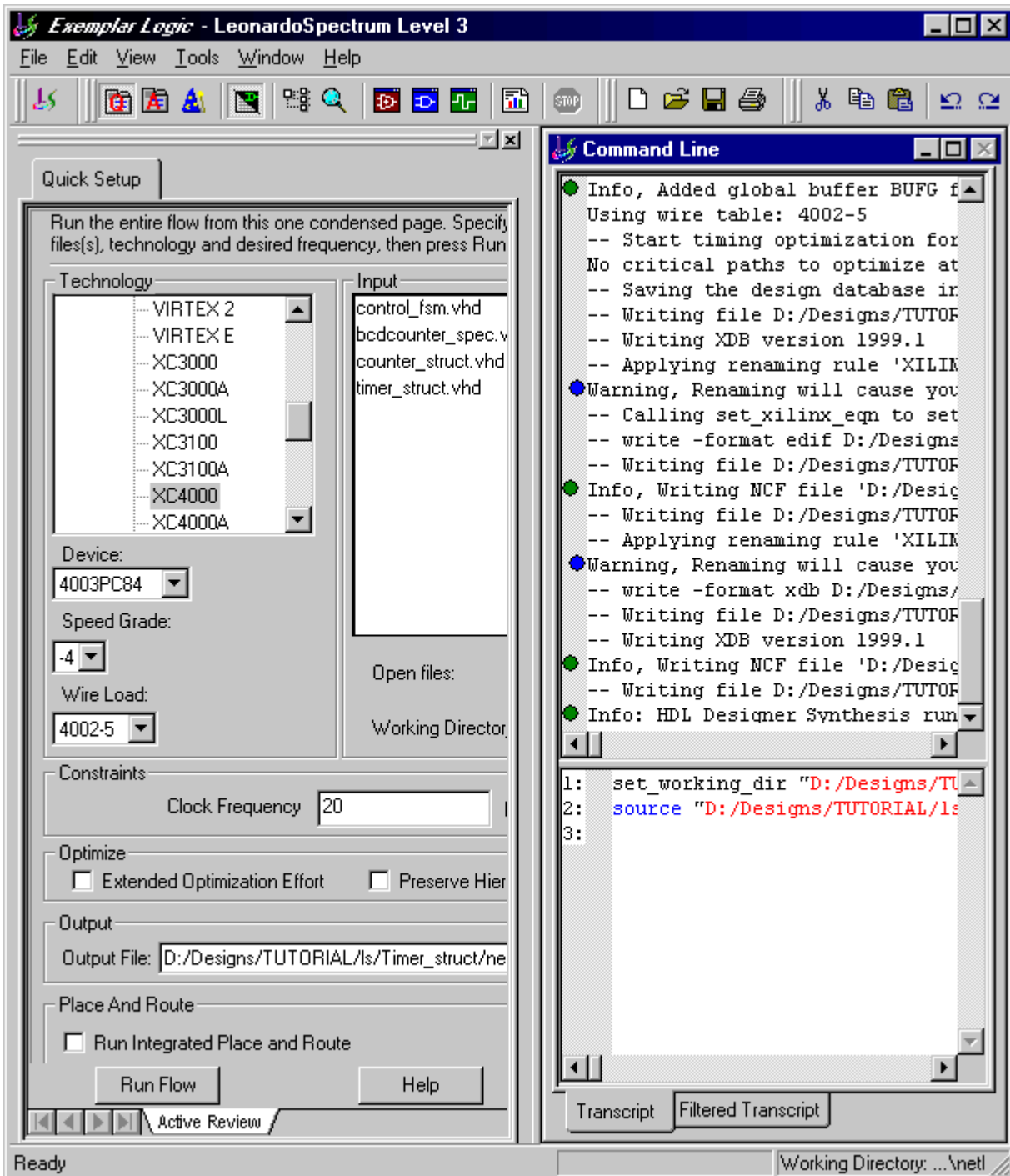
The design will be regenerated if necessary and compiled for synthesis. The Invoke Settings dialog box is displayed for you to confirm or modify the synthesis tool settings.



When you confirm the invoke settings, LeonardoSpectrum is invoked and you are prompted to select an Exemplar level 1, 2 or 3 license.

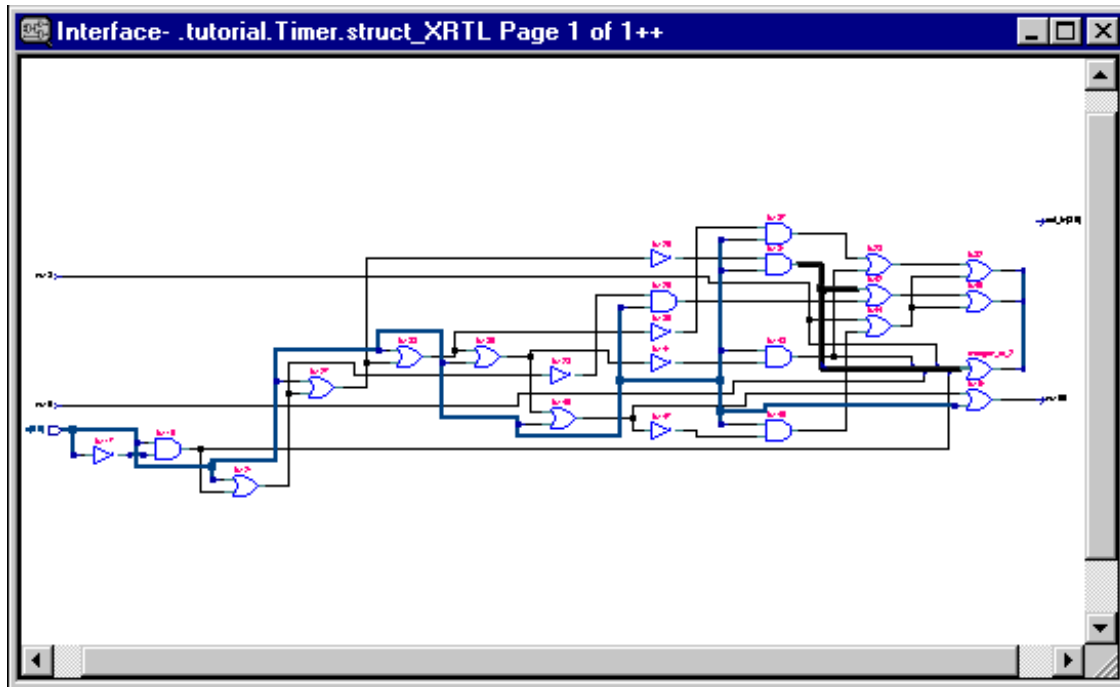


-  Higher level licenses enable more features but you can choose to run at a lower level using a higher level license.

When you confirm the license, your design is synthesized and optimized for the selected technology. Status messages are transcribed in the Command window ending with the message “HDL Designer Synthesis run finished”:



You can use the  button from the LeonardoSpectrum toolbar to display the RTL schematic or the  button to display the technology schematic for your design. For example, the following picture shows the RTL schematic:



You can move around the schematic using the scroll bars or the diagram can be enlarged or maximized in the schematic window by choosing the **Zoom In** or **Zoom Fit** options from the **Zoom** cascade of the popup menu.

You can crossprobe to the corresponding source design object by selecting an object in the schematic window and choosing **Trace to HDL Designer** from the popup menu.

Exit from LeonardoSpectrum by choosing **Exit** from the LeonardoSpectrum **File** menu and respond  when you are prompted whether to save your project settings.

You have now completed this tutorial. If you have not successfully verified your design, a completed reference example is available in the examples subdirectory of the HDL Designer Series installation. This example can be accessed by opening the *TIMER\_Vhdl* library in the design browser.

## Using the Example VHDL Design

A completed example design can be accessed from the *TIMER\_Vhdl* library in the design browser. The example design includes generated HDL but only dummy library mapping for downstream data. To use the example design, you should change the location of the generated and downstream data directories to locations for which you have write permissions.





Windows users typically have write access to the installation directories. However, it is recommended that you change the mapping to a suitable directory for user data rather than overwrite the installed examples.

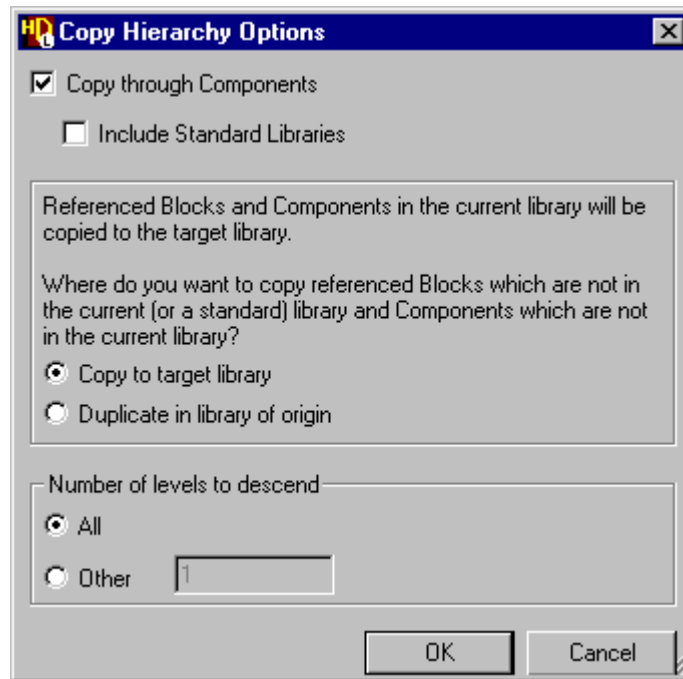
Once you have modified the library mapping (see [“Set Library Mapping” on page 1-2](#)), you can browse, compile, simulate and animate the example design by following the procedures from [“Browse the Completed Design” on page 1-70](#).

In order to animate the flow chart and state machine in the example design, you must regenerate HDL through components after setting **Instrument HDL for animation** in the state machine and flow chart properties. (Refer to the topics [“Set Generation Properties” on page 1-39](#) and [“Generate HDL for the Test Bench” on page 1-68](#).)

Alternatively, you can easily create your own copy of the example design by using the following procedure:

1. Set library mapping (including source, generated HDL and downstream directories to which you have write permissions) for a new empty library.
2. Use the  button to open the new library in the source browser.
3. Select the *Timer\_tb* design unit in the *TIMER\_Vhdl* library and drag it over the new library name with the  mouse button.
4. Choose **Hierarchical Copy Here** from the popup menu which is displayed when you release the mouse button to display the Copy Hierarchy Options dialog box.
5. Select **Copy Through Components, Copy to target library** and **All** levels in the dialog box.

A complete copy of the origin library will be made in the target library and can be browsed, edited, generated, compiled, simulated and animated without any impact on the original examples.



# Chapter 2

## Verilog Timer Exercise

This exercise creates a simple timer using [block diagrams](#) and a control [block](#) described as a hierarchical [state machine](#). A simple [truth table](#) is used to decode four-bit binary codes from the ten-bit input bus. The design is completed using a re-usable [component](#) described by a [HDL text](#) view.

A [test bench](#) is created using a [flow chart](#) which can be used as a test harness to simulate the generated Verilog for the timer design. The simulation results can be displayed as animation on the flow chart and state machine to assist in debugging the design. The verified timer design is then synthesized.

The tutorial instructions assume that a ModelSim or Verilog-XL simulator and the LeonardoSpectrum synthesis tools are available. However, the Verilog generated from the diagrams can also be used by other compatible downstream tools that are available on your system.

## Specification

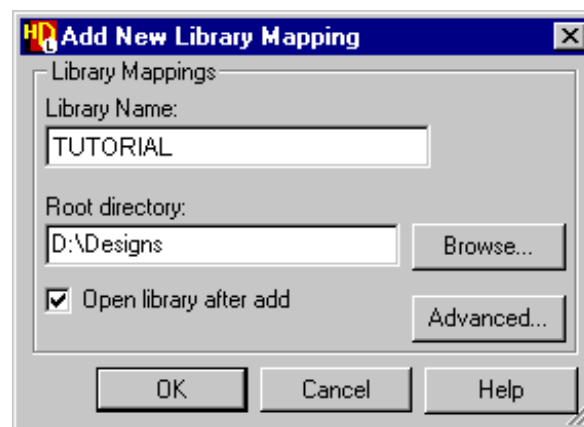
The timer outputs time data on two four-bit buses representing low and high values. There is also a logic output signal which triggers an audible alarm. The data input is provided on a ten-bit bus and control is provided by start, stop, reset and clock signals. These signals are summarized in the following table:

<b>Inputs</b>	<b>Outputs</b>
start (logic signal)	high (4-bit bus)
stop (logic signal)	low (4-bit bus)
reset (logic signal)	alarm (logic signal)
clk (logic signal)	
d (10-bit bus)	

## Set Library Mapping

A **library** is the logical location of the directories containing your design data. The source design, generated HDL and downstream data can be stored at any writable locations on your available file system but is typically saved below a common root directory.

To set library mapping, choose **New Library** from the **File** menu in the **design browser** window to display the Add New Library Mapping dialog box.

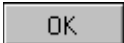


Enter a logical library name (for example, *TUTORIAL*) and specify the pathname for the root directory that will contain your library data (for example, *D:\Designs*).



By default, the root directory is set to `..\hds_scratch` which is created at `$HOME/hds_scratch` on UNIX or `<install_dir>\examples\hds_scratch` on Windows.

Library names and pathnames can be entered using upper, lower or mixed case but note that UNIX systems are case sensitive and the case used for pathnames should match the file structure. (On a PC, library names are case sensitive but pathnames are case insensitive.).

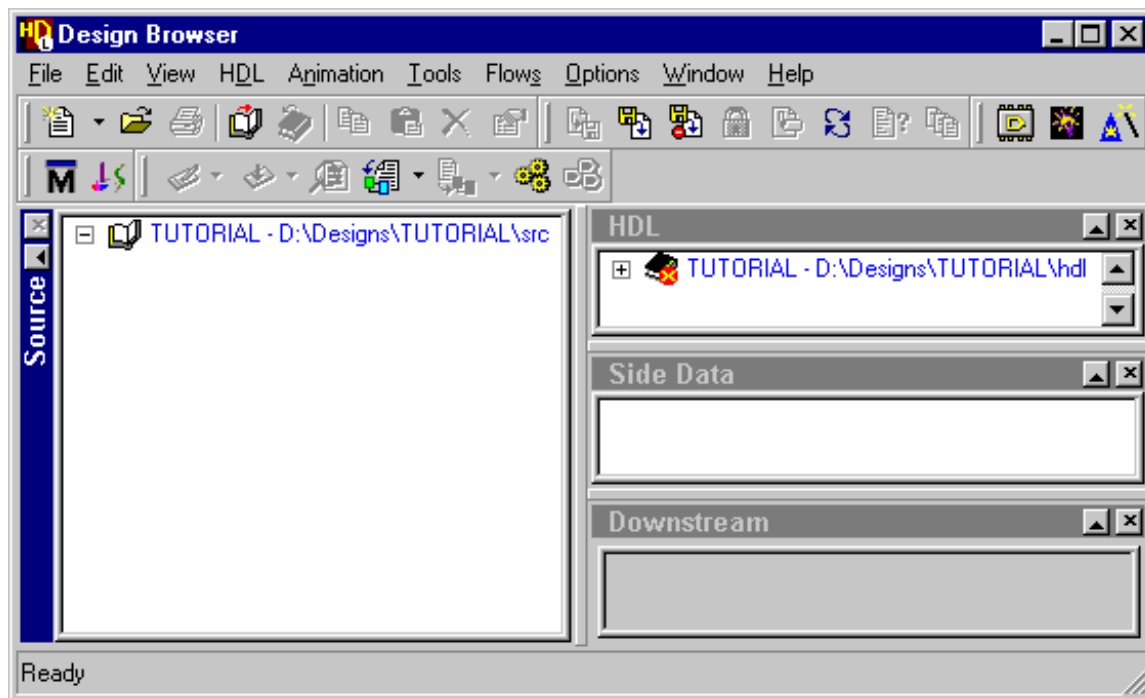
Check that the **Open library after add** option is set and use the  button to create and open the new library. Notice that the source design data directory is named `<root_directory>\TUTORIAL\src` and that the generated HDL directory is named `<root_directory>\TUTORIAL\hdl`.



The source design data directory is created (if it does not already exist) when you save a **design unit** to the *TUTORIAL* library (provided that the **parent** directory exists and you have write access to create a subdirectory at the specified location). The generated HDL directory is created when you generate HDL for the design. The mapping for the location of compiled data is defined automatically when you set up a downstream tool and the directories created when the design is compiled.

Your library definition is saved in an initialization file which is automatically saved in your **user directory** with the file name *hds.ini* and is read the next time that the HDL Designer Series tool is invoked.

The Source design data directory is displayed in the design browser as an open "book" and the HDL directory as a closed book similar to the following picture:

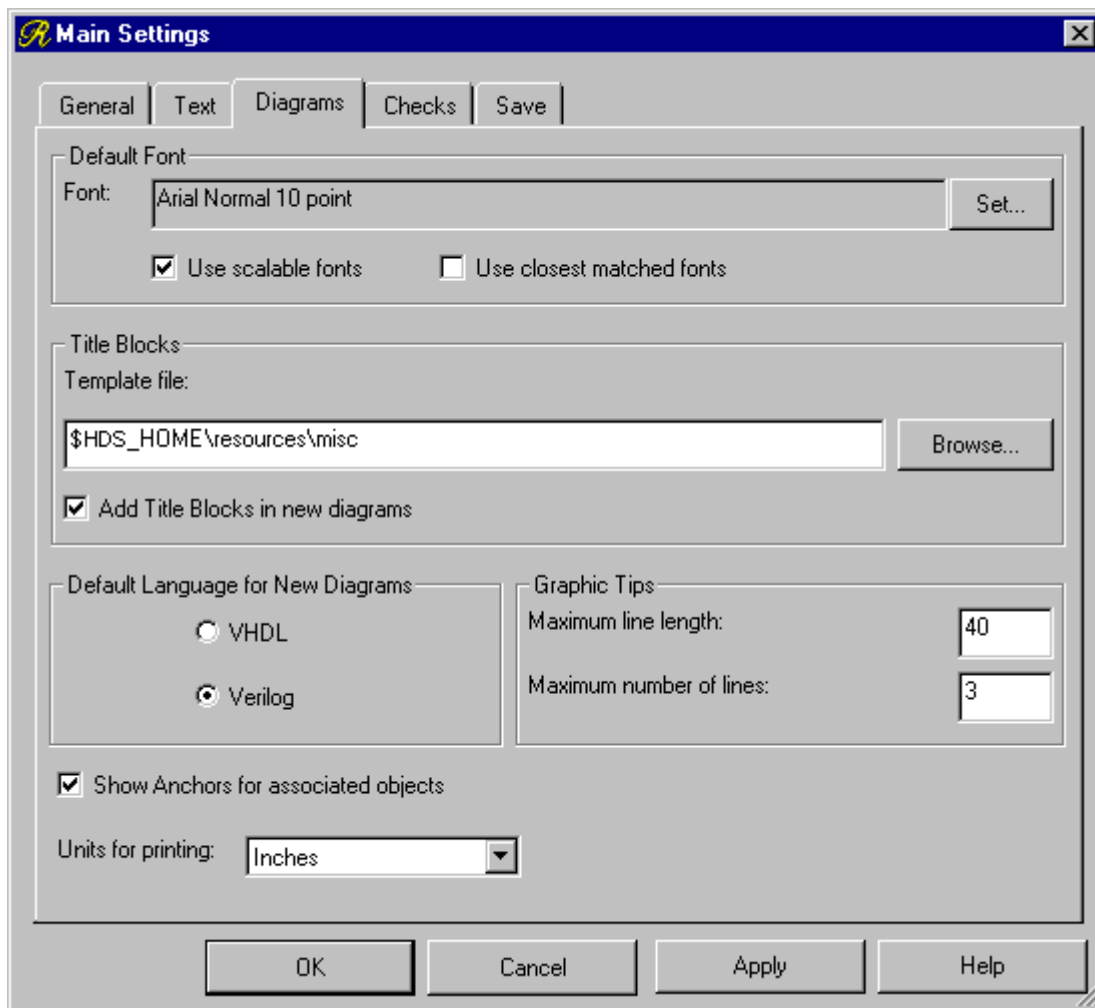


The Source and HDL pathnames are shown in blue text because the directories do not exist yet but will be updated in the browser when you create them by saving design units and generating HDL for the library. The Side Data and Downstream sub browsers are both empty at this stage unless any other libraries are open.

## Set the Default Language

A set of default preferences are loaded when you invoke a HDL Designer Series tool for the first time. There are separate tabbed dialog boxes for the main settings, VHDL and Verilog options, compile settings, HDL import options, version management settings and master preferences for each type of graphical diagram. The preference dialog boxes can be accessed from the **Options** menu.

Choose **Main** from the **Options** menu to display the Main Settings dialog box, select the Diagrams tab and ensure that **Verilog** is set as the default language to be used for new [graphic editor](#) views. Use the  button to confirm your choice.

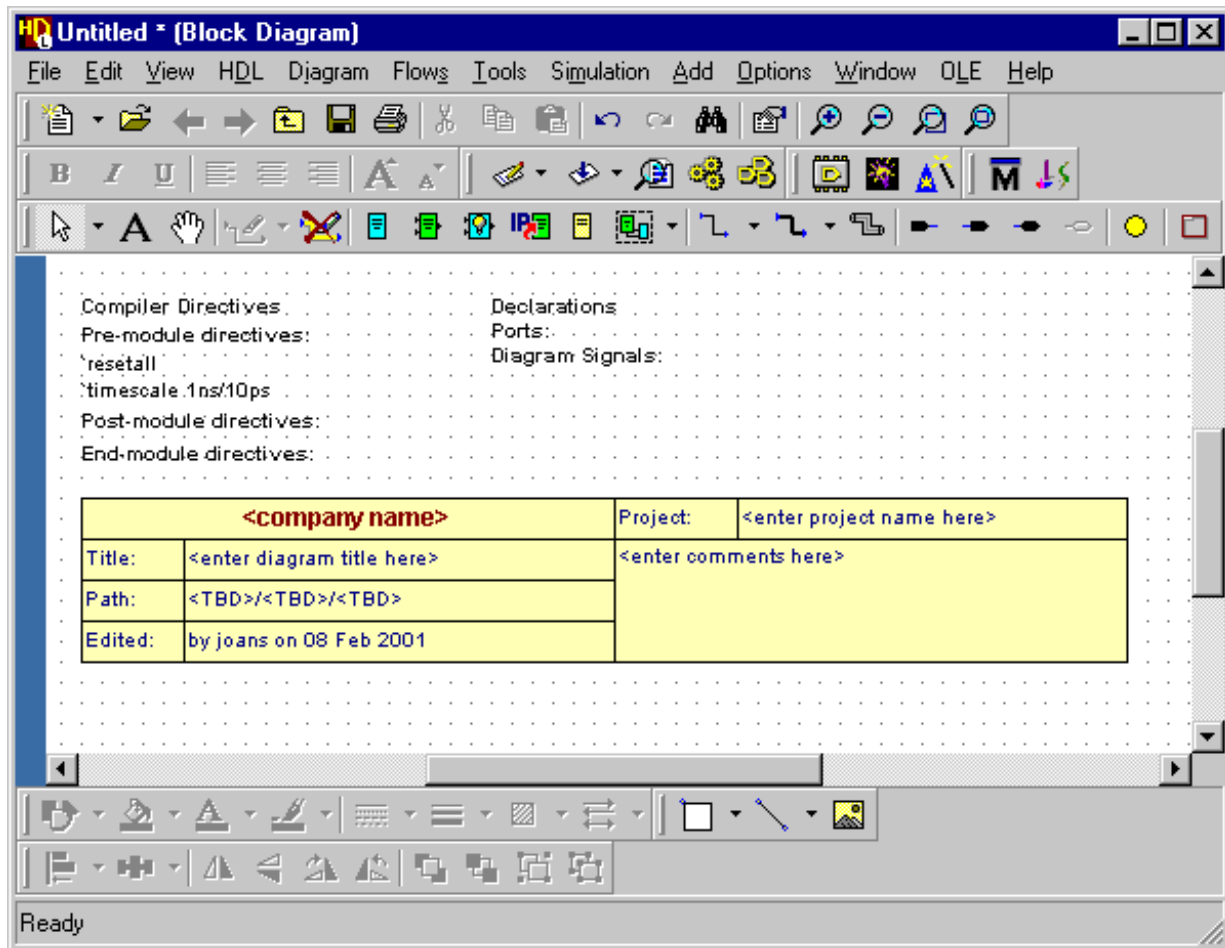


All other preferences can be left with their default values for this tutorial.

## Create a Block Diagram

Use the  button in the design browser window and select **Block Diagram**.

A new untitled [block diagram](#) is created.



The block diagram is a blank sheet except for a background grid, a list of compiler directives (with *'resetall* and *'timescale 1ns/10ps* set by default), a copy of the default title block, and empty text fields with labels for Declarations, Ports and Diagram Signals.

Notice the five toolbars at the top and three toolbars at the bottom of the diagram. The toolbar buttons provide quick access to many of the most frequently used editing and formatting commands.

## Edit the Title Block Template

A title block is automatically added to all new diagrams if the **Add Title Blocks in new diagrams** option is set in your diagram preferences. The default title block is a template with default text for your company name, project name, diagram title and comments. The default title block incorporates internal variables which automatically enter your login name and the current date. Internal variables are also used to enter the logical pathname for the design. This path is initially shown as <TBD>/<TBD>/<TBD> but the internal variables are converted to show the library, design unit and view name when you save the diagram.

Click twice on <company name> in the default title block to display a popup edit box and replace the default text by the name of your company. Click twice on <enter project name here> and enter a name for your project (for example, HDL Designer Tutorial).

Display the **Diagrams** tab of the Main Settings dialog box (as described in “[Set the Default Language](#)” on page 2-4) and change the **Template file** pathname to a write-able location such as your [working directory](#) or home directory.

Select the title block by clicking with the mouse so that the selection handles are displayed and choose **Save Title Block** from the **File** menu.


<b>Mentor Graphics</b>		Project:	HDL Designer Tutorial
Title:	<enter diagram title here>	<enter comments here>	
Path:	<TBD>/<TBD>/<TBD>		
Edited:	by joans on 08 Feb 2001		


A dialog box is displayed with a warning that saving the title block cannot be undone. Click the  button to proceed.

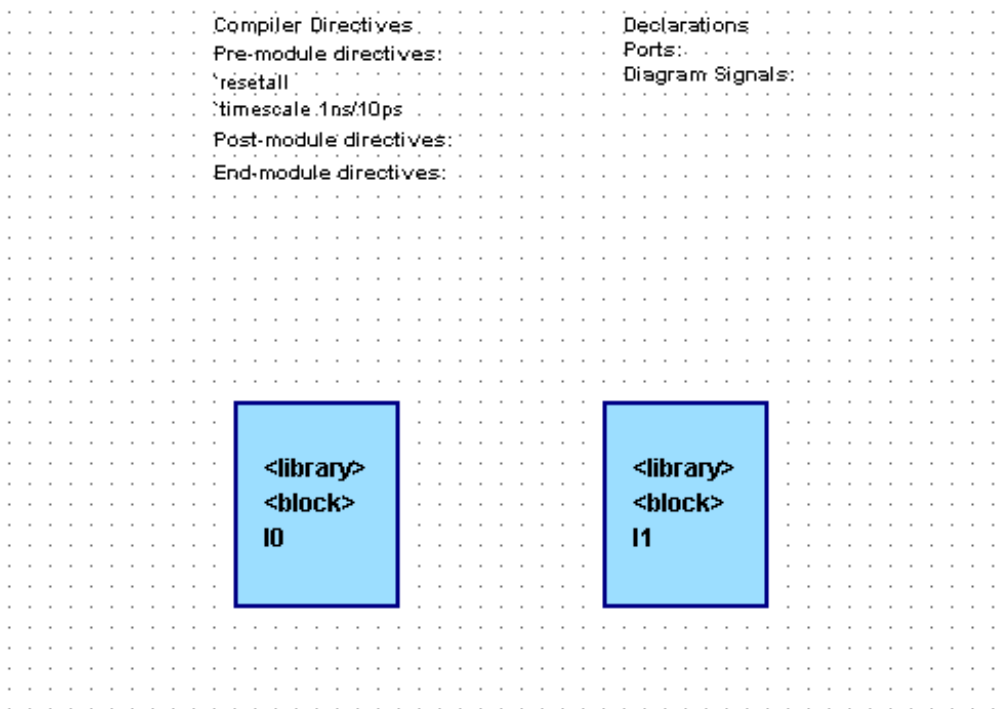
The title block is saved at the new location specified in your preferences and will be used as the default template in new diagrams.


Refer to the online help topics for information about how you can add and modify title blocks.




## Add Blocks

Move the title block to the bottom of your block diagram by dragging it with the  mouse button.


Use the  button to add two **blocks** as shown in the picture below. The blocks are added with the default library `<library>`, the default name `<block>` and unique instance names (*I0* and *I1*).

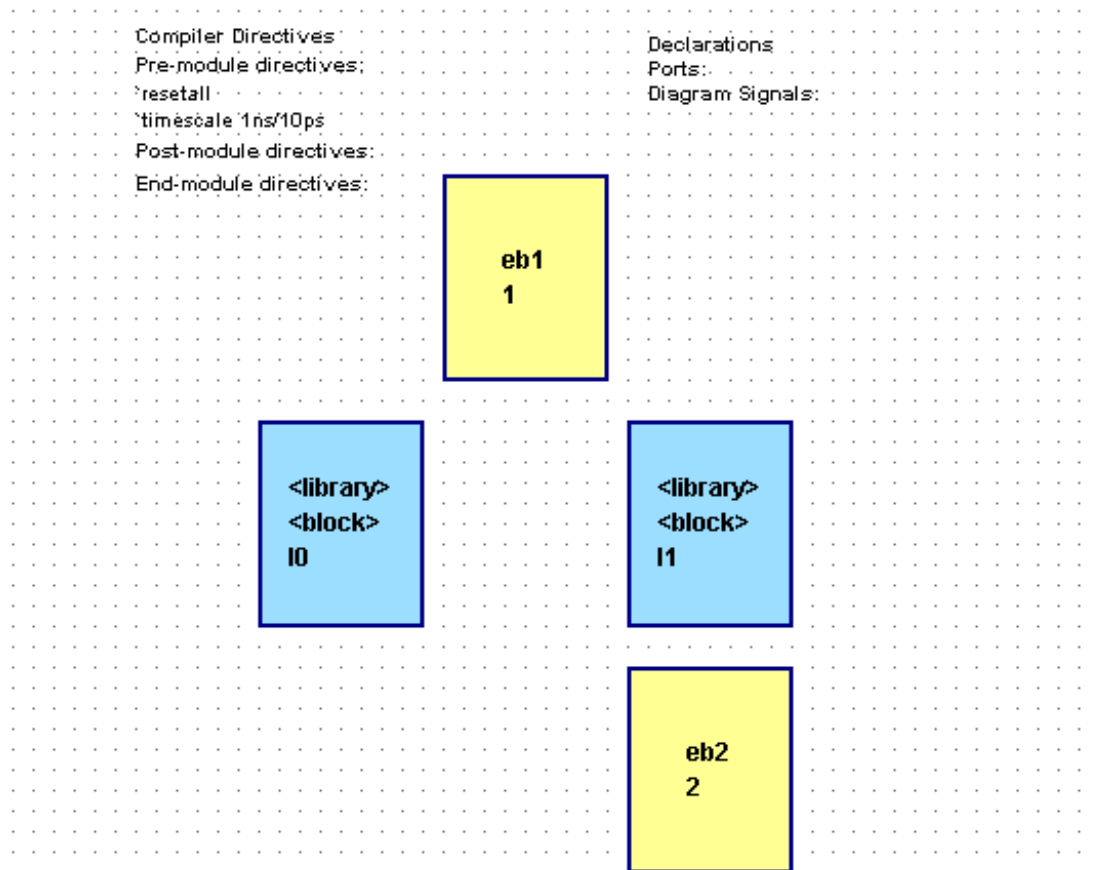


The command normally auto-repeats until you select another command or terminate the repeating command by using the right mouse button or  key. However, you can change the behavior of the toolbar buttons by setting the **Activate once only** preference in the **General** tab of the Main Settings dialog box as described on [page 2-4](#).

You can also use the  key with any toolbar button to toggle the repeat mode. For example, when **Remain active** is set,  +  adds a single block on a block diagram.

## Add Embedded Blocks

Use the  button to add two **embedded blocks** on your block diagram as shown in the picture below. The embedded blocks are added with unique default names (*eb1* and *eb2*) and numbers (*1* and *2*)


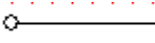

















The view describing a block must be saved as a uniquely named **design unit** in a library directory. However, the view describing an embedded block is saved as part of the parent block diagram and does not impose hierarchy when HDL is generated for your design. The name of an embedded block must be unique on the diagram and is used as a label in the generated HDL.


The blocks (*I0* and *I1*) will be used to define a child state machine and block diagram view. The embedded blocks (*eb1* and *eb2*) will be defined by concurrent assignment statements on the top level block diagram.

## Add Ports and Signals


You can use the following buttons to add [signals](#), [buses](#) and [ports](#) on a block diagram:

	Add a signal without a port	
	Add a signal with a port	
	Add a bus without a port	
	Add a bus with a port	
	Add an input port	
	Add an output port	
	Add a buffer port	
	Add a bidirectional port	

Signals or buses can be added between any existing [connectable items](#) on the diagram or left unconnected by double-clicking to terminate the [net](#) with a dangling [net connector](#). However, you can use the  pull-down on the buttons to change the default setting to terminate with a default port at unconnected end points. Notice that the toolbar button changes to show the current setting.


Choose **Signal with Port** and use the  button to connect three signals originating from the block on the left (instance *I0* in the picture) to the block on the right (instance *I1*) and one signal returning from *I1* to *I0*. The signals are added with unique names (*sig0*, *sig1*, *sig2* and *sig3*) and the default [type](#) *wire*. Notice how declarations are automatically added to the list of Diagram Signals.



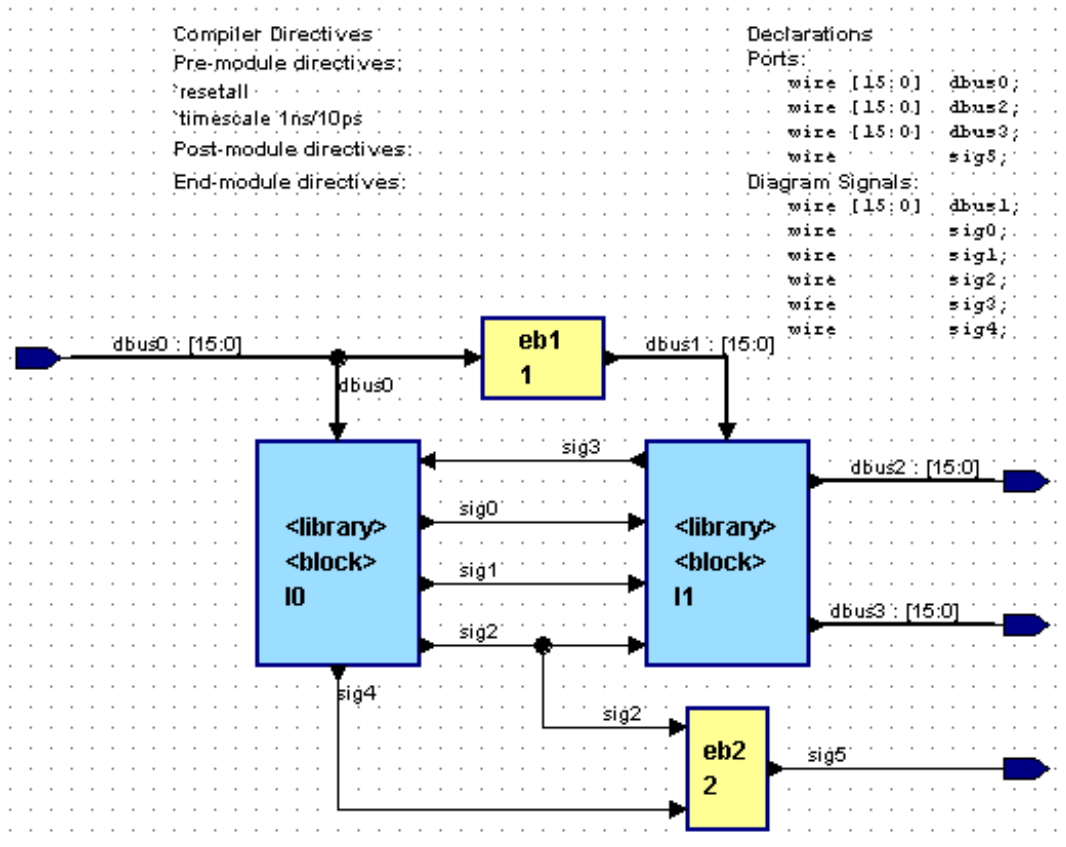
Allow two or more grid lines between each port or signal. You can resize objects by selecting a block or embedded block and dragging one of its resize handles. If necessary, you can drag text elements such as the signal name and type using the  mouse button.

Add a signal from *I0* to the embedded block *eb2* and another signal from a point on *sig2* terminating on the embedded block.

Add signals from *I0* and from a point on *sig2* terminating on the embedded block *eb2* and a signal from the embedded block terminating in space on the right side of your diagram. Notice that an output port is automatically added when you double-click at the end of the last signal and its declaration is added to the list of ports.


Choose **Bus with Port** and use the  button to add a bus from a **source** on the left side of your diagram with its **destination** on the upper embedded block *eb1*. A default input port is automatically created at the beginning of the bus. Add another bus starting from this bus and terminating on instance *I0*. Notice how both bus segments have the same default name *dbus0* but the default **bounds** *[15:0]* is shown only on the first bus segment. The full declaration showing the default bus type and bounds *wire [15:0]* is added to the list of ports. See the online help topic Changing the Display of Signal Properties for information about the format for displaying signals and buses.



Add a bus (*dbus1*) from *eb1* to *I1*. The add two buses (*dbus2* and *dbus3*) from *I1* terminated with default output ports by double-clicking on the right side of the diagram. Your diagram should now look similar to the picture below





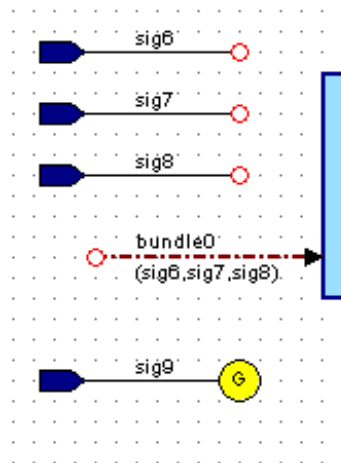




## Add a Bundle and Global Connector

Use the  button to add three signals on the left side of your diagram. Notice that a default input port is added at the source of each signal but a dangling [net connector](#) is drawn when you double-click at the end of each signal.

Select the three signals (by dragging a select rectangle with the  mouse button held down) and use the  button to connect a [bundle](#) containing these signals to block instance *I0* as shown in the picture below. Notice that the bundle has the default name *bundle0* and the three selected signals are automatically included in the bundle with their names listed under the bundle name.


Use the  button to add a [global connector](#) on your diagram below the bundle and use the  button to add a signal between the global connector and a default input port. (This will be a clock signal which is implicitly connected to every block on the diagram.)

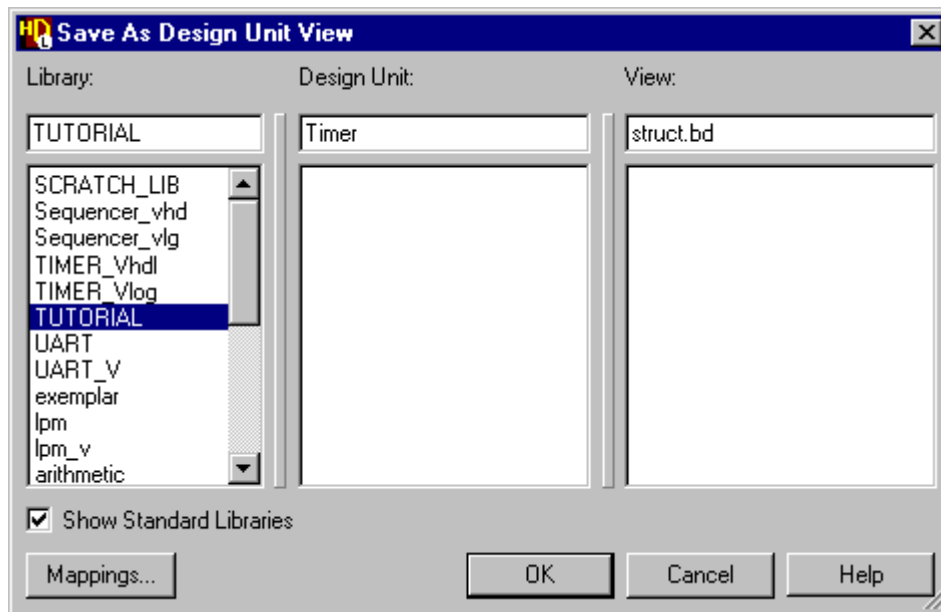


If you make a mistake when editing a diagram, you can use the  button to undo your last edit and the  button to redo an undo operation. You can also use commands from the **Align** cascade of the **Edit** menu to re-align and distribute objects on the diagram.

## Save the Block Diagram

Notice the asterisk (\*) character in the header of the block diagram editor window. This indicates that the diagram has been edited since it was last saved.


Use the  button to save the block diagram. The Save As dialog box is displayed which allows you to choose from the currently mapped libraries and specify the **design unit** and **design unit view** names. Choose the *TUTORIAL* library and enter design unit *Timer*. The dialog box should look similar to the example below:



The view name can be entered using any valid HDL identifier but normally defaults as follows:






struct.bd	block diagram
struct.ibd	interface-based design view
fsm.sm	state diagram
flow.fc	flow chart
tbl.tt	truth table
symbol.sb	symbol

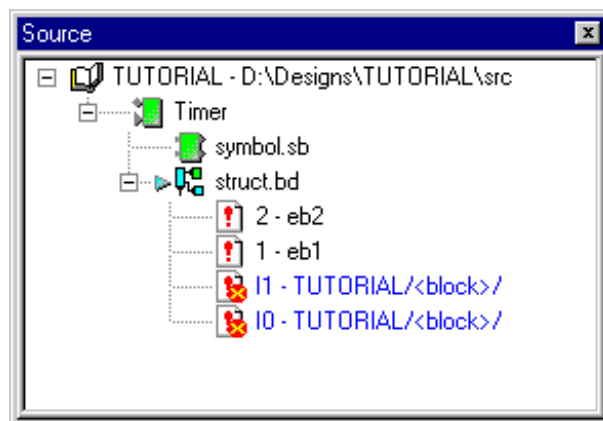
If you omit the two-character extension it is automatically added to identify the type of diagram you are saving. The default leaf names can be changed by setting preferences. However, you should not change the extension (*.bd*, *.ibd*, *.sm*, *.fc*, *.tt* or *.sb*) or the design data file will not be recognized and cannot be reopened.


When you click the  button, your diagram is saved and the window title bar is updated to show the diagram name *TUTORIAL\Timer\struct*. This path is also added in the title block replacing the <TBD> used when the diagram was created. Notice that the asterisk (\*) character has been cleared in the block diagram header and the library name used on the blocks in your diagram has been updated to *TUTORIAL*.



If the design browser window is obscured, you can pop it to the front by selecting the **Design Browser** window from the **Windows** menu list in the block diagram window.

Click on the  icon for the *TUTORIAL* library in the [source browser](#) and notice that the view is expanded to display the *Timer* design unit. Click on the  icon for the *Timer* design unit to reveal that it contains a [symbol](#) and block diagram view. Click on the  icon for the *struct.bd* view to display the hierarchy of views instantiated as blocks and embedded blocks on the block diagram. The embedded blocks (*eb1* and *eb2*) are shown using the  icon to indicate that no views have been defined. The blocks (*I0* and *I1*) are also shown as undefined but with blue text and an  overlay indicating that no design units exist for their child views.

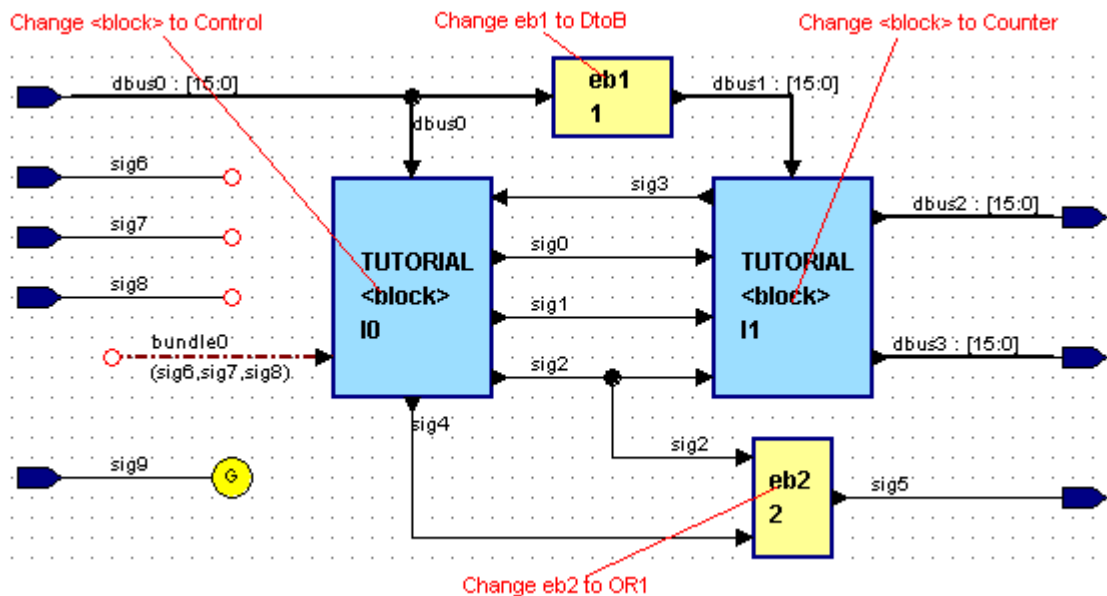


You can change the design browser layout and undock the source browser (shown above), [HDL browser](#), [side data browser](#) or [downstream browser](#) from the main window. Blue text and an  overlay in the source browser indicates that a view is not write-able. (In this case, because design units have not been created for the blocks. This convention is also used to show when you have read-only access.)

## Edit Block and Signal Names

You now have a completed top-level block diagram for the *Timer* design. However, the blocks and signals have default names.

Click on the text `<block>` in the lower block on the left (instance *I0* in the picture) and notice the small handles which indicate that the text object is selected. Click again and notice that the text is now highlighted and can be directly overwritten. Type the new name *Control* and click outside the text to complete the edit. Repeat this procedure to change the name of block instance *I1* to *Counter*, embedded block *eb1* to *DtoB* and embedded block *eb2* to *OR1*.




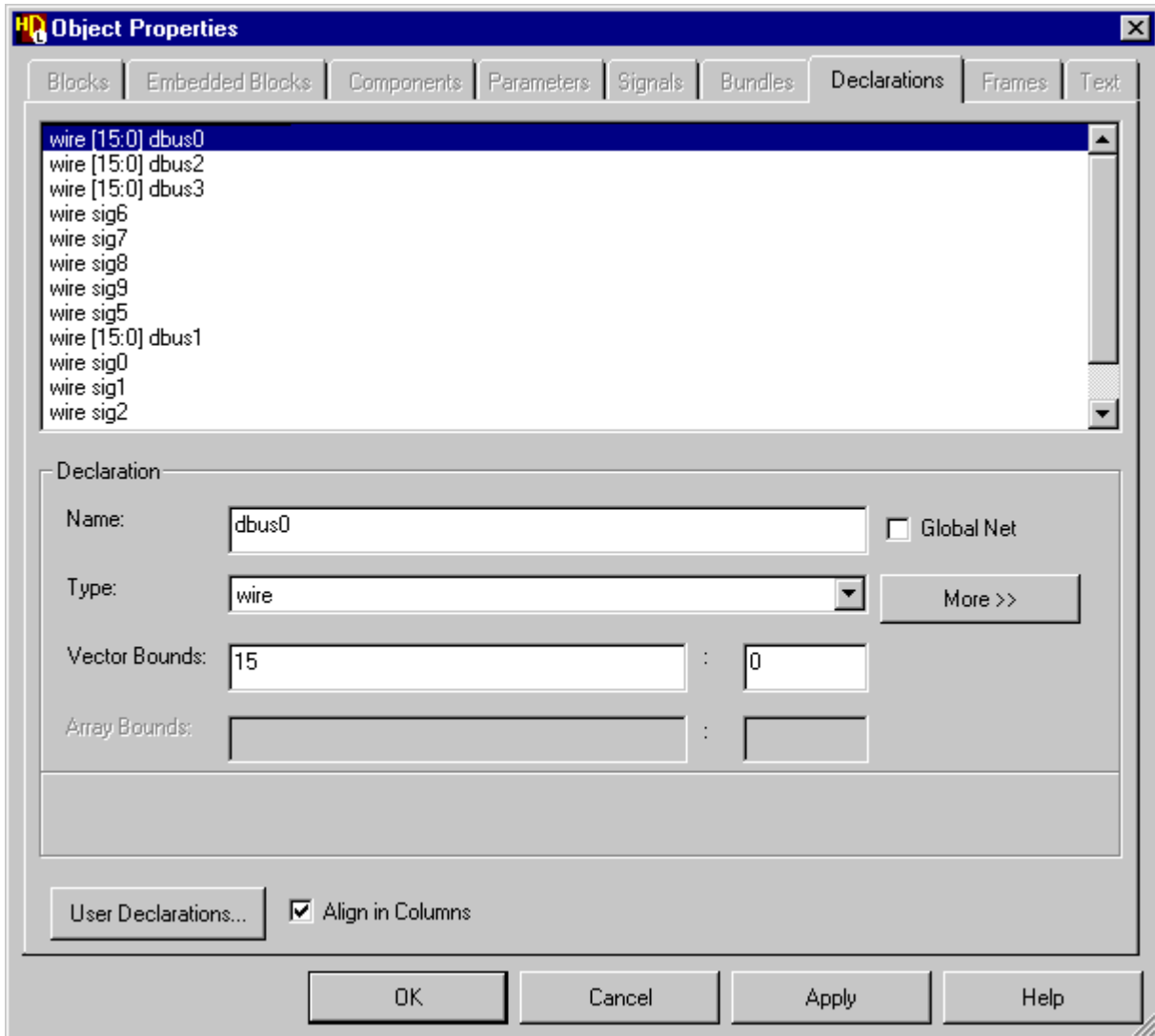
The text can be overwritten when it is highlighted. If you click again, the cursor changes to an I-beam which allows you to move the cursor in the text and edit individual characters.

While a text element is selected, an [anchor](#) that attaches the text to its associated object is visible and you can move the text independently to improve diagram clarity.

This text editing technique can also be used to edit the signal and bus names.

Alternatively, you can use a dialog box which allows you to edit the properties for a selected object.

Double-click on the existing declarations, use the  button or choose **Object Properties** from the **Edit** menu to display the Block Diagram Object Properties dialog box and choose the **Declarations** tab.



Notice that the port declarations are listed at the top of the dialog box and the other internal diagram signals at the bottom. Input ports are listed before the output ports, otherwise the declarations are listed in alphanumeric order.

You can choose one or more existing declarations in the dialog box, enter new values for any of the declaration fields. For example, choose *dbus1*, *dbus2* and *dbus3*, then enter a new **bounds** 3:0 to update all three buses while all other fields remain AS IS.

The changes are applied to the diagram when you click the  or  button. Notice that all occurrences on the diagram are updated including the declarations list, signals, buses and bundle contents and that the lists of port and signal declarations are sorted alphanumerically when the dialog box is applied to the diagram.

Use the dialog box to update the port and signal declarations as shown in the following tables.

Ports:

Old Name	New Name	Type	Bounds
dbus0	d	wire	9 : 0
dbus2	low	wire	3 : 0
dbus3	high	wire	3 : 0
sig6	start	wire	none
sig7	stop	wire	none
sig8	reset	wire	none
sig9	clk	wire	none
sig5	alarm	wire	none

Diagram Signals:



Old Name	New Name	Type	Bounds
dbus1	dat_in	wire	3 : 0
sig0	clear	wire	none
sig1	load	wire	none
sig2	hold	wire	none
sig3	zero	wire	none
sig4	beep	wire	none



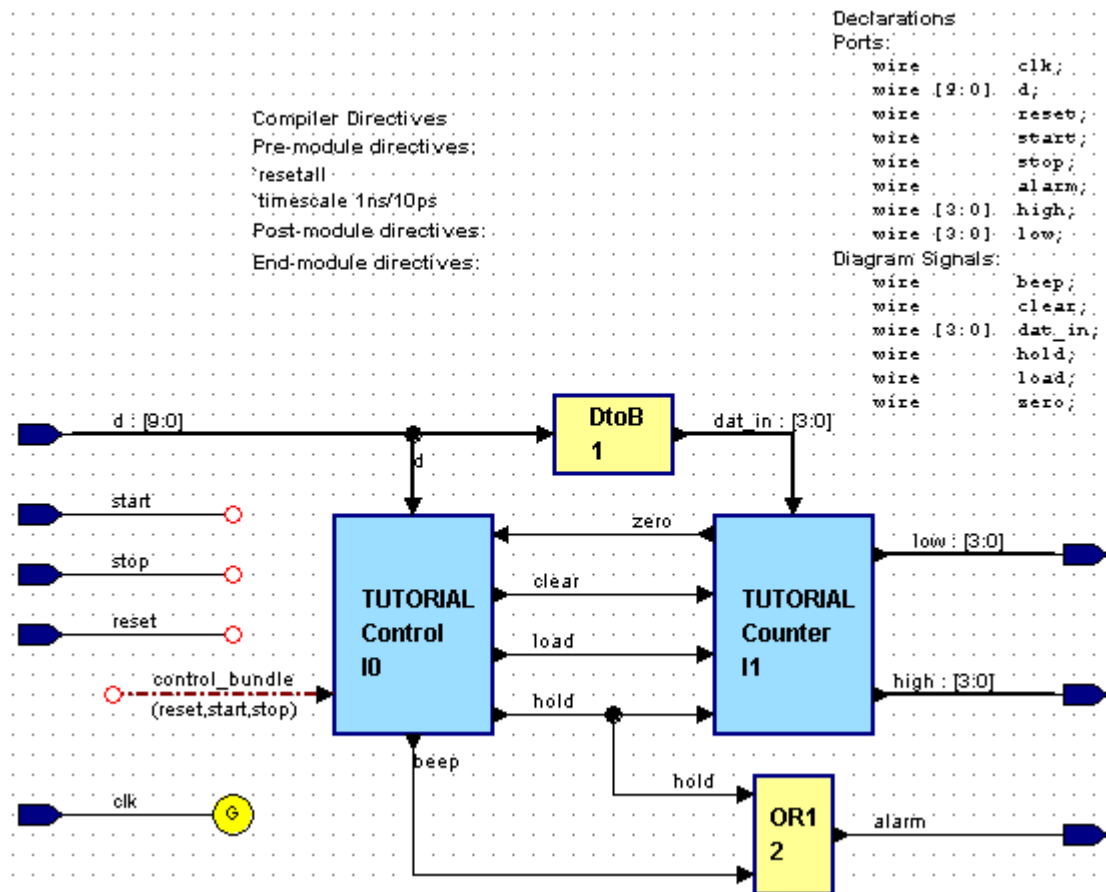
All occurrences of each signal name (including the bundle contents) are automatically updated when you use the **Declarations** tab. However, if you use, direct text editing (or the **Signals** tab) to change signal names, it may be necessary to check that all occurrences have been updated.

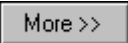
Select the bundle name and use direct text editing or the **Bundles** tab of the Object Properties dialog box to change the default bundle name to *control\_bundle*.




You can change the selection mode to select text or object shapes only by using the  pull-down menu on the  button.


Your block diagram should now look similar to the following picture:

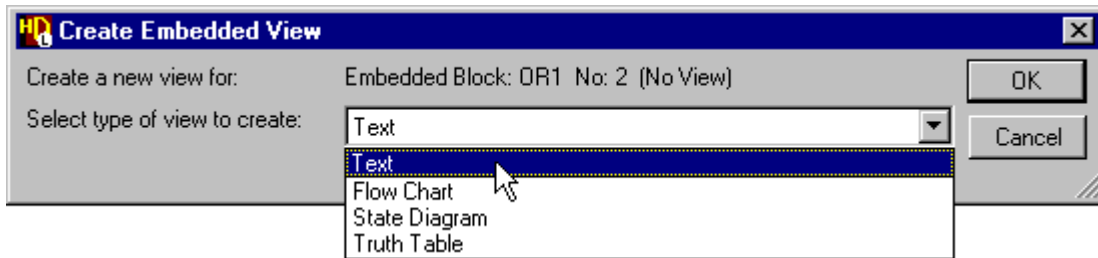



The  button on the dialog box allows you to disclose additional fields which allow you to modify other signal properties including delay, expansion, charge strength and attributes for embedded synthesis constraints.

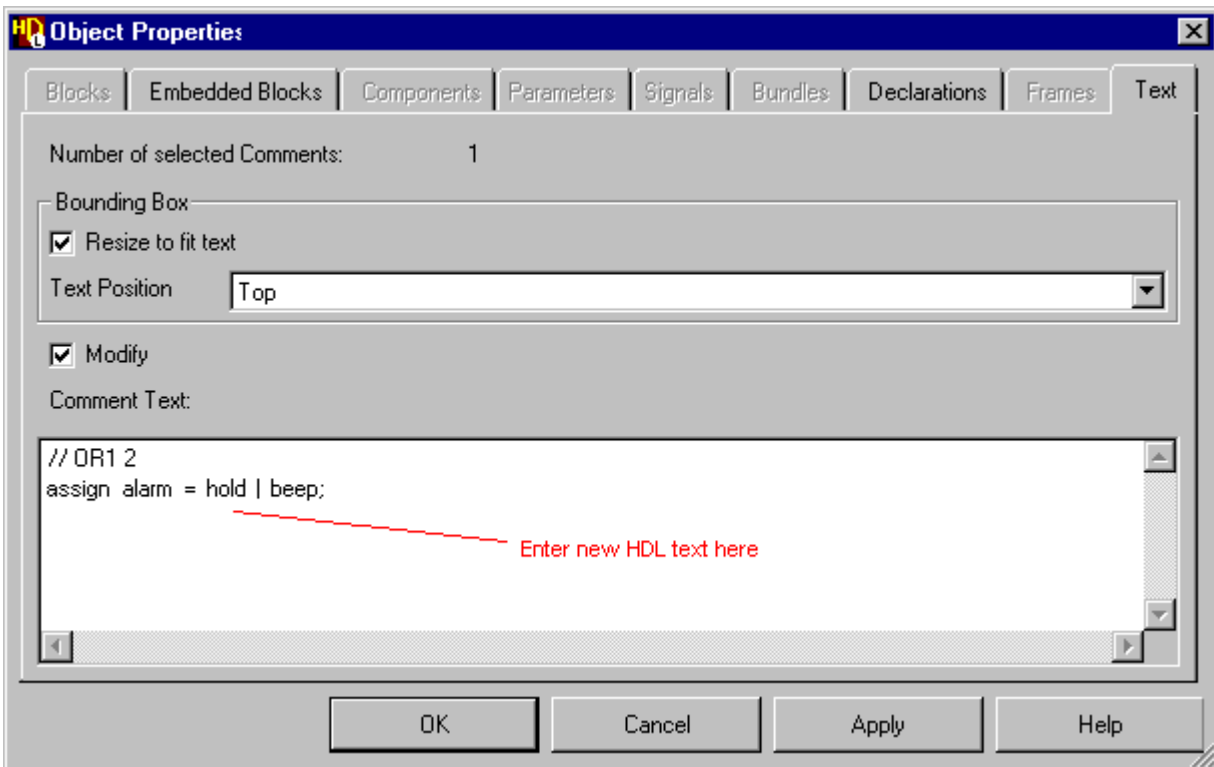
The  button allows you to add additional user-entered [module declarations](#) to the structural Verilog. Refer to the “Editing Verilog Signal Declarations” help topic for more information about these features which are not used in this tutorial.

## Add an Embedded HDL Text View

Select the *OR1* embedded block and display the popup menu by clicking the  mouse button. Choose **New View** from the **Open** cascade in the popup menu to display the Create Embedded View dialog box. Choose Text from the pulldown list of views in the dialog box.



An embedded **HDL text** view containing default text is displayed on the block diagram adjacent to the embedded block. Select the text, re-display the Object Properties dialog box if necessary (using the  button) and choose the **Text** tab.







Check the **Resize to fit text** option and enter the following Verilog statement under the default // OR1 2 comment in the dialog box:

```
assign alarm = hold | beep;
```

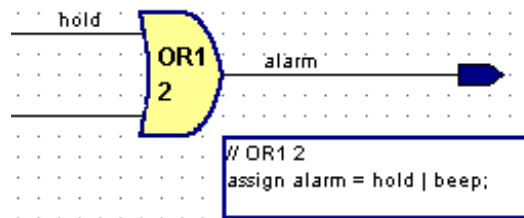
The modified HDL text is checked for syntax errors and applied to the diagram when you click the  or  button on the dialog box.

The functional blocks on the diagram are shown by default as simple rectangular shapes. However, it is sometimes useful to use logic notation when a block has a specific logical function. For example, in this block diagram, the *ORI* embedded block represents a logical OR function.

Select the embedded block and choose the pulldown  on the  button to display a palette of alternative shapes. Select  from the palette to apply a logical OR shape to the embedded block on the diagram.

You can also hide the port arrow heads by clearing the **Show Ports when connected** check box in the **Embedded Blocks** tab of the Object Properties dialog box.

The *ORI* embedded block should now look similar to the following picture:





It is also possible to indicate an active low (Not) or edge triggered clock signal. This feature can be used with the alternative shapes to represent extra functions such as an inverter, NAND, NOR or flip-flop. If required, you can rotate any block or component by 90 degree steps.

Refer to the “Logic Shape Notation” help topic for more information about these features..



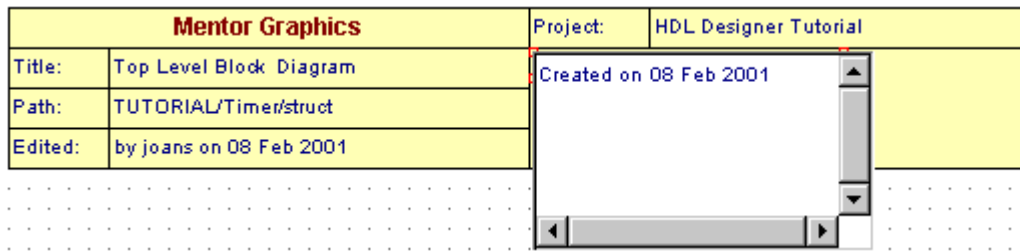
The logical OR function could also be implemented by a ModuleWare component similar to that used in [“Add ModuleWare Components”](#) on [page 2-51](#).

## Add a Panel and Edit the Title Block

Use the  button to add a panel and hold down the  mouse button to drag the panel and enclose the graphical objects on your diagram. The panel is added with the default name `Panel0` and can be useful to outline areas of a diagram. For example, you can divide a large diagram into separate printable page-sized areas or use a panel to outline a view used for simulation or animation.

Complete the block diagram by editing the title and comments in the title block on the diagram. For example, enter the title `Top Level Timer Block Diagram` and a comment of the form: `Created by <your name> on <date>`.

The title block comprises a number of grouped [comment text](#) objects. Each comment text object can be edited directly by clicking twice on the text to display a text entry box.




You can enter free-format text including line breaks and spaces which will be preserved on the diagram.

Click the  mouse button outside the entry box to complete the text entry.

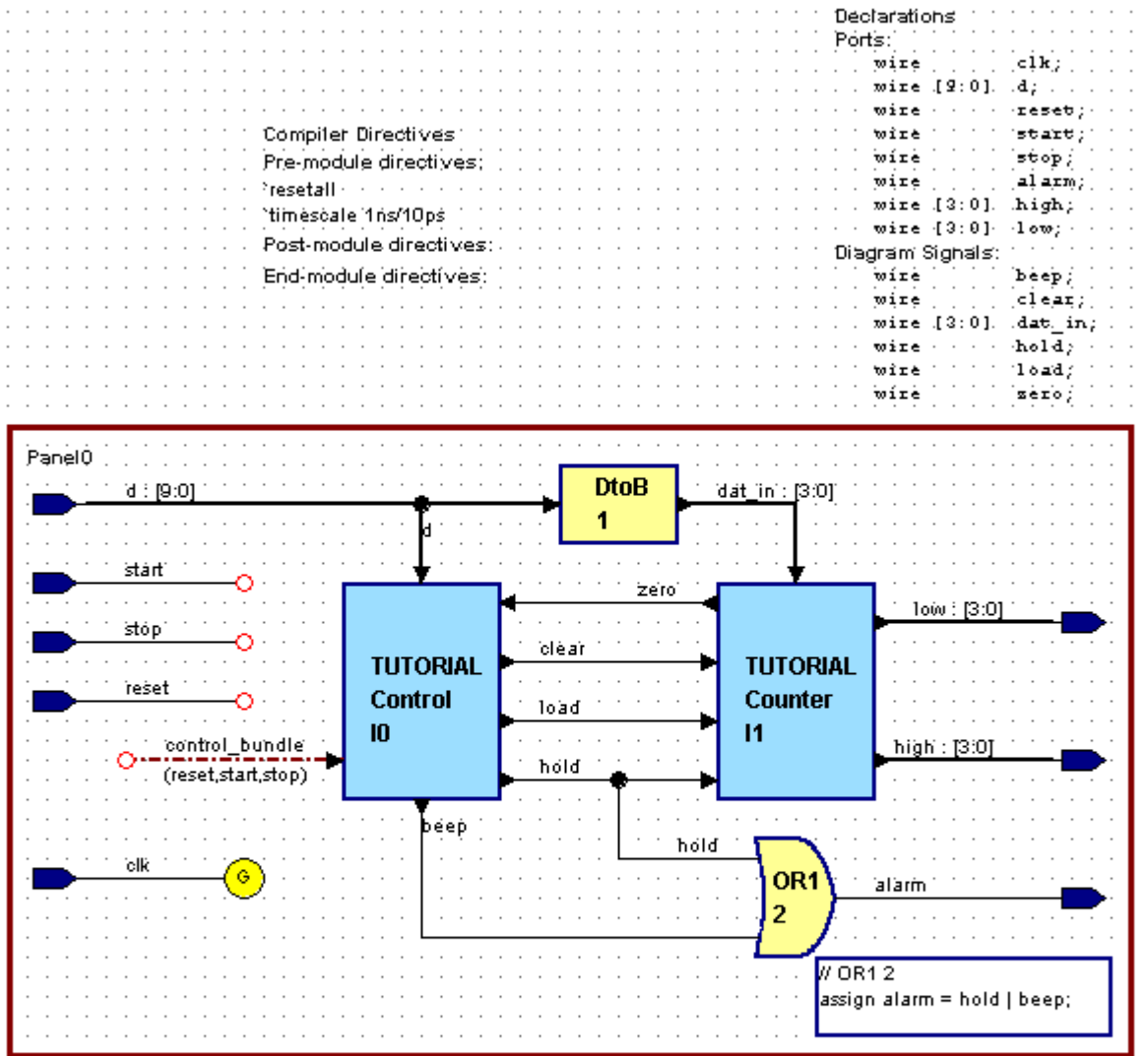


You can also edit an existing comment text object by double-clicking to display the **Text** tab in the Object Properties dialog box. When comments are edited in the dialog box, it is possible to enter any special characters (for example accents or Kanji characters) which are supported on your system.

Use the  button to save the block diagram.

You have previously saved the diagram so you are not prompted for library and design unit names. However, you have changed the names of signals connected to input and output ports and the block diagram will be inconsistent with the symbol that was automatically created by the previous save. You are prompted whether to update the symbol. Click the  button to confirm.

The completed block diagram should look similar to the following picture:



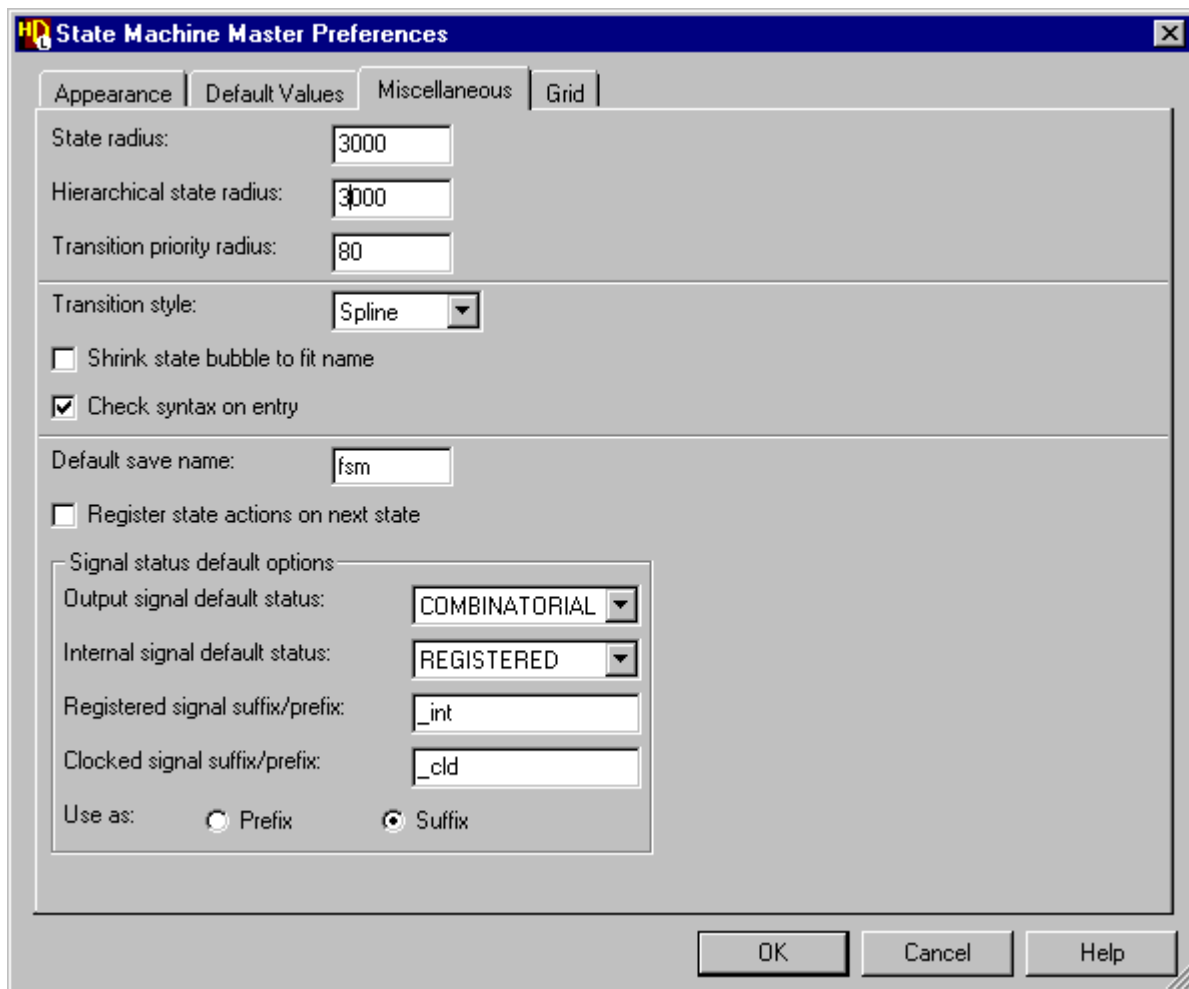
<b>Mentor Graphics</b>		Project:	HDL Designer Tutorial
Title:	Top Level Block Diagram	Created on	08 Feb 2001
Path:	TUTORIAL/Timer/struct		
Edited:	by joans on 08 Feb 2001		

The procedures in the following sections create a graphical state machine to describe the *Control* block. If you are using one of the HDL text design tools, refer to [Appendix A](#) for an alternative HDL text view of the *Control* block.

## Set State Machine Preferences

You will create a [state machine](#) view in the next procedure. You can set master preferences which modify the way new diagrams are drawn. As an example, for this tutorial it is suggested that you reduce the default size used for a [state](#).

Choose **State Machine** from the **Master Preferences** cascade of the **Options** menu in the design browser to display the State Machine Master Preferences dialog box and select the **Miscellaneous** tab:



You can set the minimum radius for states, [hierarchical states](#) and the [transition priority](#) object. The states will auto-size if the state name is larger than can be enclosed by the minimum radius. However, the minimum state size is overridden if you check **Shrink state bubble to fit name** which allows the states to shrink below this size if the state names are short.

Transitions on a [state diagram](#) are normally drawn with curved [splines](#) instead of the orthogonal [polylines](#) used for signals on a block diagram. However, the [transition](#) style can be changed to straight polylines.

Syntax checking on entry can be enabled or disabled (for example, if you want to enter non-HDL identifiers or comments while drafting a diagram). You can choose to register state actions on the next state instead of the current state.

You can also specify the default leaf save name for state diagrams, the default status for output and internal signals and the suffixes used for the internal names of registered or clocked signals.

Change the [state](#) radius and [hierarchical state](#) radius values to 3000. This radius should be sufficient to enclose the state names used in this tutorial.

Examine the other preferences available in each tab of the State Machine Master Preferences dialog box.

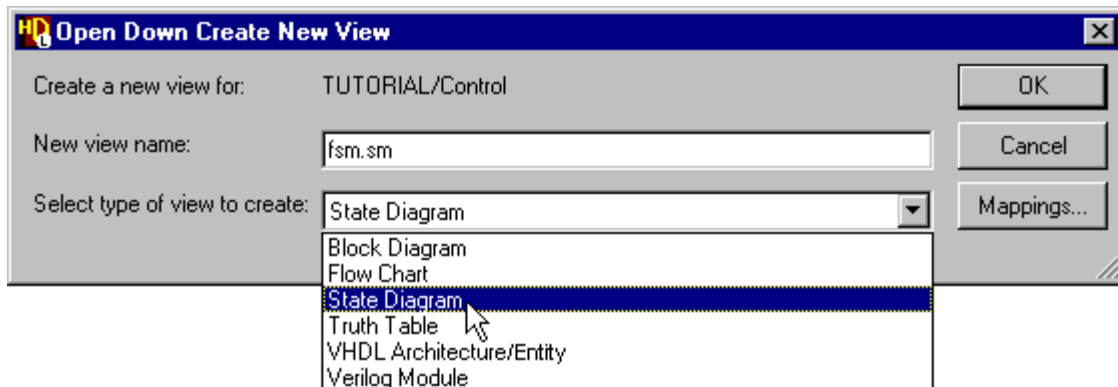
Use the  button to set the changed preference. You are prompted to confirm the change which will be used in the master preference next time you open a state diagram. Click the  button to confirm.



Notice that you can also display dialog boxes which allow you to set the master preferences for the block diagram, symbol, flow chart and truth table editors. You can use the  buttons in each dialog box for more information about these preferences. However, you are advised not to change any of the other preferences before you have completed this tutorial.

## Create a Child State Diagram

Move the cursor over the body of the *Control* block on the *Timer* block diagram, then press and release the **Right** mouse button to select the block and display the popup menu. Choose the **Open** cascade menu option **New View**. The Open Down Create New View dialog box is displayed:



Use the pulldown list to select the type of view you want to create. The view name defaults to *struct.bd* for a **block diagram**, *struct.ibd* for an **Interface-Based Design (IBD)** view, *flow.fc* for a **flow chart**, *fsm.sm* for a **state machine**, *tbl.tt* for a **truth table** or *untitled* for a **VHDL architecture** or **Verilog module** view. Alternatively, you can enter any other name of your choice or use the **Mappings...** button to modify the mapping for the current library.



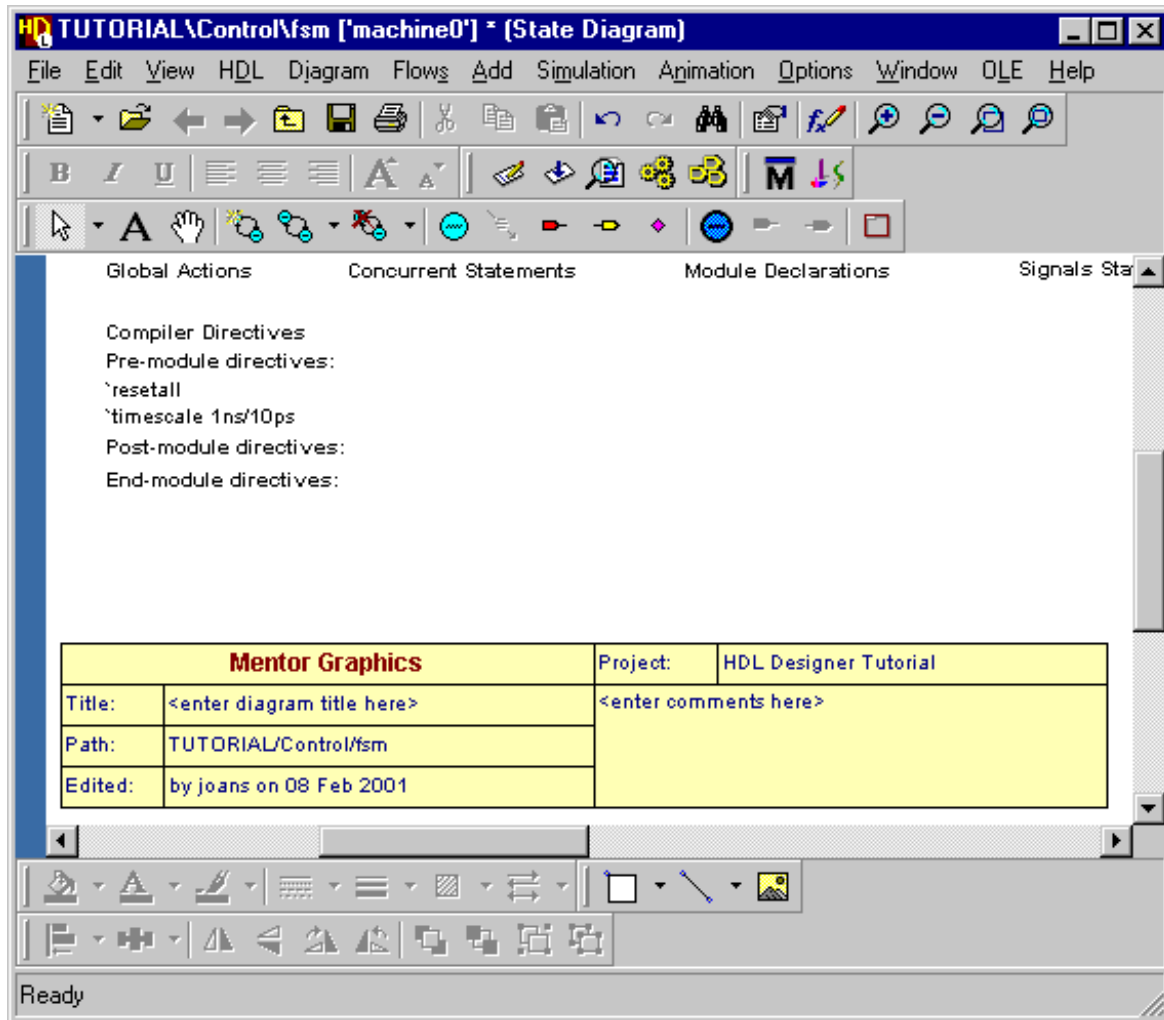
You need not enter the two character extension (*.bd*, *.fc*, *.sm* or *.tt*) for graphical views as the correct extension is automatically added. However, if you do enter any other extension you are warned that the file will not be recognized.

Select **State Diagram** from the pulldown list of views you can create and use the default view name *fsm.sm*.



Solid handles are displayed when the body of a block (or other re-sizable object) is selected. You can display the Open Down Create New View dialog box directly by double-clicking on the body of a block which has no views defined.


A new [state diagram](#) (*TUTORIAL\Control\fsm ['machine0']*) is created as a [child](#) view of the *Control* block:




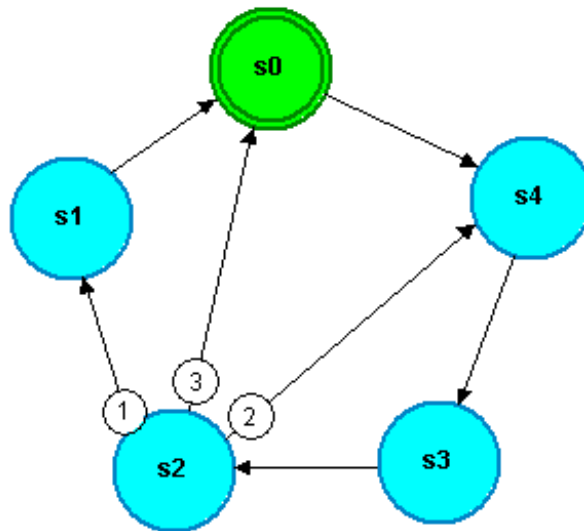
A default state machine name (*machine0*) is appended to the design unit and view names but can be changed by choosing **Rename Concurrent State Machine** from the **Diagram** menu.

The state diagram is a blank sheet except for the default compiler directives list and labels for [global actions](#), [concurrent statements](#), [module declarations](#), [signals status](#) and [state register statements](#). The state diagram also includes the default title block which you saved as a template in an earlier topic.

## Add States and Transitions

Use the  button to add five [states](#) on your state diagram. The states are added with default names *s0*, *s1*, *s2*, *s3* and *s4*. Notice that the first state you add is assumed to be the [start state](#) and is drawn in green with a double outline. The other states are drawn in cyan with a single outline.

Use the  button to add [transitions](#) between the states as shown in the picture below. Notice that when you add more than one transition leaving a state, the [transition priority](#) is indicated by a number associated with the [transition arc](#).




The priorities are initially assigned in the order that you add the transitions but will be re-assigned in a later topic if necessary.




If you add a transition in the wrong direction, you can easily change its direction by choosing **Reverse Direction** from the popup or **Diagram** menu. Note that a popup description (known as a [graphic tip](#)) is displayed when the cursor is stationary over an object. In particular, when the cursor is over a transition, the source state and the destination state are named even if the states are outside the current window.

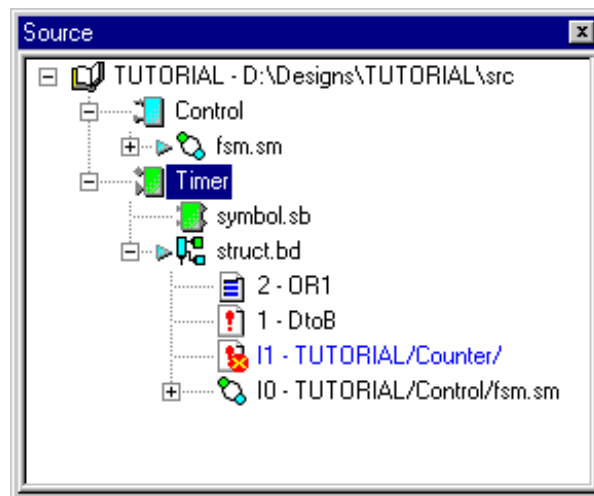




## Save the State Diagram

Use the  button to save the state diagram. The state diagram was created as a **child** view from its **parent** block diagram and is saved using the library, design unit and view names specified when it was created. The design browser view is updated to display the *Control* design unit.

 You can pop the design browser window to the front by selecting it from the **Windows** menu list in an editor window.


The *Control* design unit is shown as a block in the browser because its interface is defined by the connections on its parent block diagram. The *Timer* design unit is shown as a component because it has no parent block diagram and its interface is defined by a symbol. Click on the  icon for the *Control* design unit to expand the design unit revealing that it contains a state diagram view.

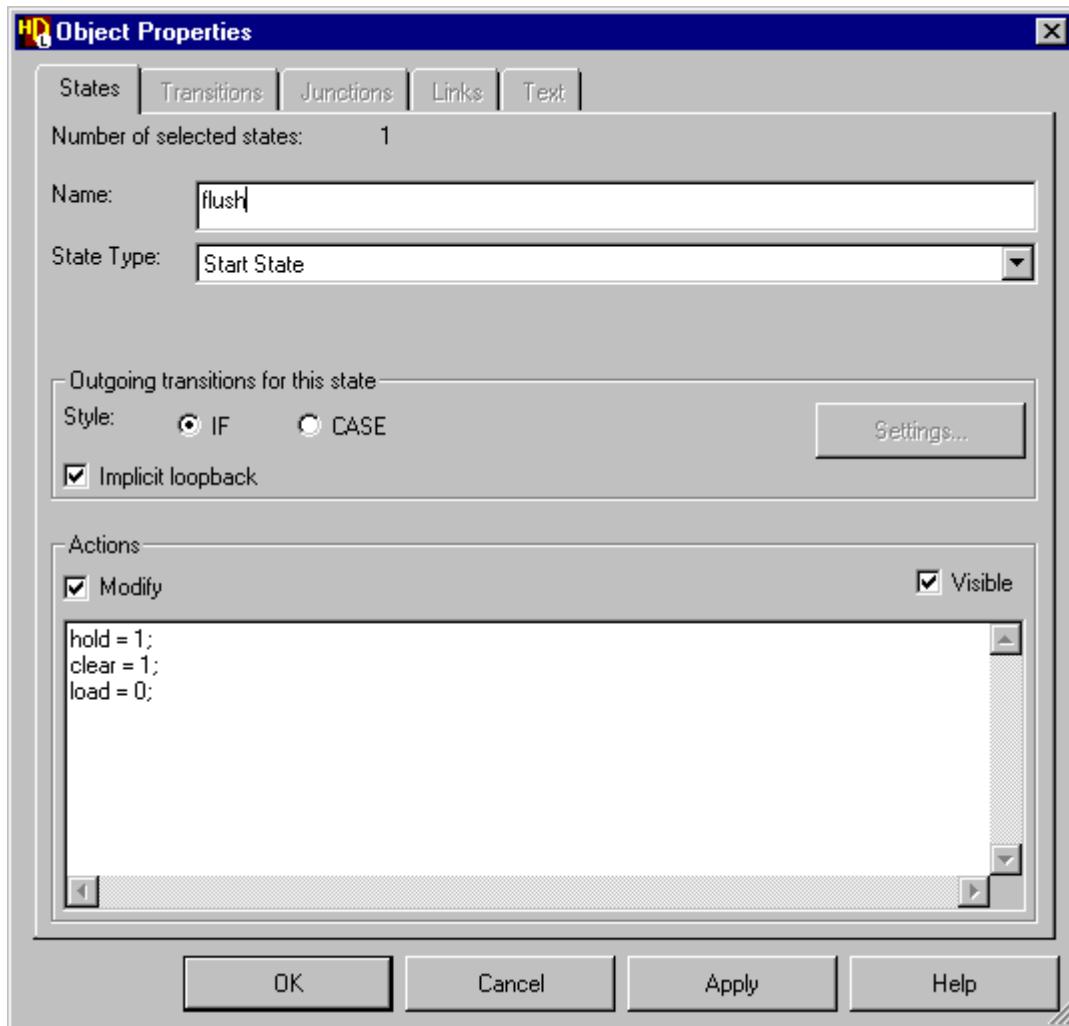


Notice also that the icon used for instance *I0* in the hierarchy for the *struct.bd* view has changed to  indicating that it is now described by a state diagram. (If the hierarchy is not already displayed, click on the  icon for the *struct.bd* view.)

The *ORI* embedded block is shown as a text view but the *DtoB* embedded block and the *Counter* block are still undefined.

## Edit the States

Select the start state (*s0* in the picture on [page 2-26](#)) and use the  button to display the **States** tab of the State Machine Object Properties dialog box. (Alternatively, you can display the dialog box by choosing **Object Properties** from the **Edit** menu or double-clicking on a state.)



The **States** tab allows you to enter a name and actions text for one or more selected states on a state diagram. You can also change the visibility of state actions and (when a single state is selected) change the state to a [start state](#) or a [hierarchical state](#).

Use the dialog box to enter the following state names and **actions** replacing the default state names *s0* to *s4* in the picture on [page 2-26](#):

Old Name	New Name	Style	Actions
s0	flush	IF	hold = 1; clear = 1; load = 0;
s1	count	IF	(no actions)
s2	getkey	IF	hold = 1; clear = 0; load = 0;
s3	load_t	IF	hold = 1; clear = 0; load = 0;
s4	load_u	IF	hold = 1 ; clear = 0; load = 1 ;

Notice that the Verilog Expression Builder dialog box is automatically displayed when you start to enter the actions. The dialog box can be used to choose from lists of the available port or local signal names, operators and example values. For example, choose the *hold* signal,  button, value '1' and  button to enter the action *hold = 1;*.

The syntax for state actions is automatically checked and any errors reported on entry. Verilog statements must be terminated by a semicolon (;) character. Line breaks and spaces can be used for clarity and will be preserved on the diagram.

If the name is larger than the state, it auto-resizes to fit the new name. You can also resize a state by selecting the state and dragging one of its resize handles but you cannot make it smaller than the enclosed name.




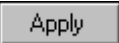



You can also edit the state name or actions by direct text editing on the diagram or copy a state and paste its state actions into another state by choosing the **Paste State Actions** option from the **Paste Special** cascade in the popup menu.

## Edit the Transitions

Add **conditions** to the state diagram transitions as shown in the following table:

Origin	Destination	Priority	Condition
count	flush	1	stop
getkey	flush	3	stop
getkey	count	1	start
getkey	load_u	2	d!=ZEROS
flush	load_u	1	d!=ZEROS
load_u	load_t	1	(none)
load_t	getkey	1	d!=ZEROS



Hold down the  mouse button and select the two transitions entering the state **flush** by dragging the cursor across them (or by using  mouse button). Use the  button to display the **Transitions** tab of the State Machine Object Properties dialog box. Enter the condition text *stop* in the dialog box using the expression builder and click the  button to add this condition to both of the selected transitions.

A confirmation dialog box is displayed warning that the transition from state *getkey* to state *count* has no condition and is not the lowest priority. Click the  button to acknowledge the warning.

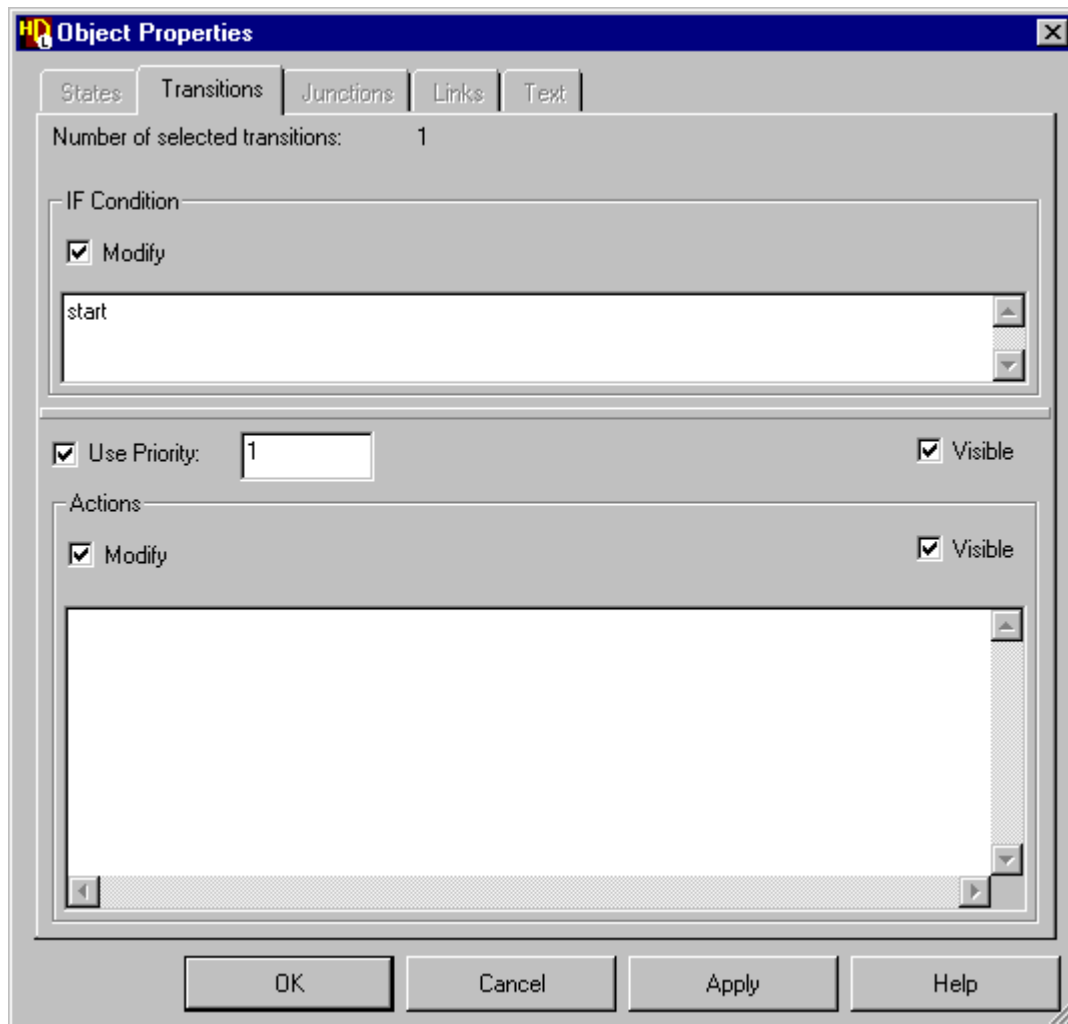
Repeat this procedure for the *start* and *d!=ZEROS* conditions.




The condition syntax is checked on entry. Any valid Verilog fragment can be entered using line breaks and indentation to improve the legibility on the diagram if required.

Ensure that the transition priorities leaving the state *getkey* are the same as those shown in the table. You can change a transition priority in the **Transitions** tab of the State Machine Object Properties dialog box. Alternatively, you can use the  button to zoom in or the  button to view an area and use direct text editing to change a transition priority.

When you change a transition priority, the priority of the other transitions leaving the same state are automatically adjusted.

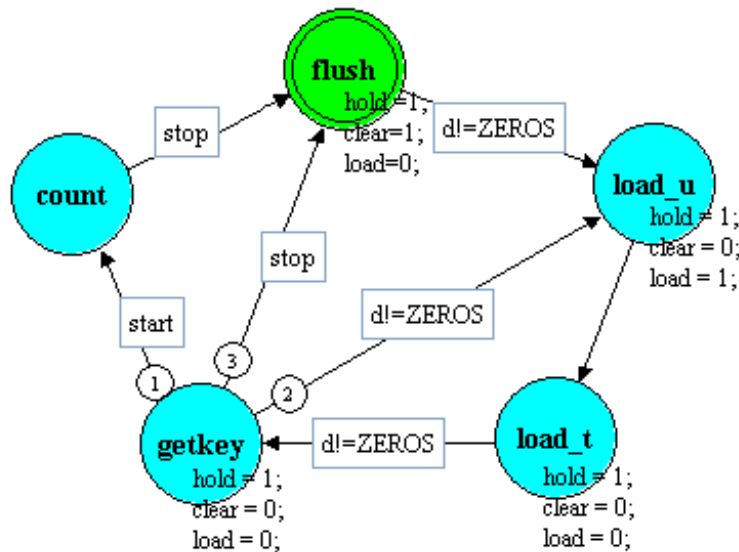


If you have zoomed in, use the  button to view all of the state diagram.

Use the  button to save your changes to the state diagram.

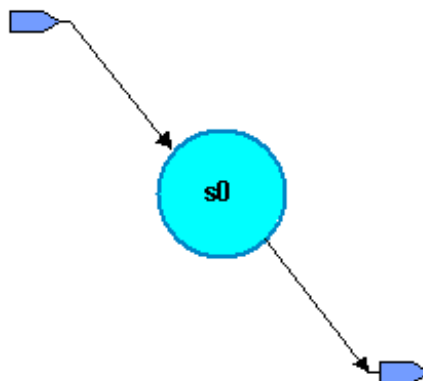
# Create a Hierarchical State Diagram

The state diagram should look similar to the following picture.




Select the *count* state and choose **Hierarchical State** in the **Change To** cascade from the **Diagram** menu. Alternatively, select the *count* state and choose the Hierarchical State pulldown option for the State Type in the **States** tab of the State Machine Object Properties dialog box. The *count* state is redrawn as a **hierarchical state** with a triple outline and darker fill color.



Double-click on the hierarchical state (or use the **Open Down** option from the popup menu) to create the new hierarchical child state diagram. A new child state diagram window is initialized with a default state connected to an **entry point** and **exit point**.

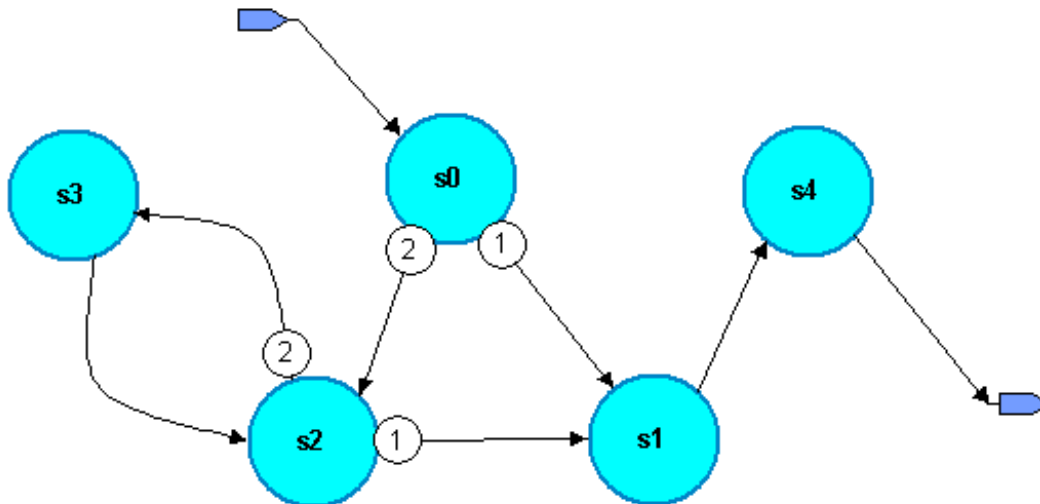


Notice that the name of the hierarchical state (*count*) is included in the window title: *TUTORIAL\Control\ fsm[:count 'machine0']*. This naming convention shows that the child diagram is a partial view of the parent diagram.

Select the exit point and choose **Change to State** from the **Diagram** menu. While the new state is selected, drag with the  mouse button and release the mouse button with the ghosted state to the left and below the first state. Use the **Copy Here** option from the popup menu to make a copy of the state at the cursor position.

Repeat this procedure to add two more states on the diagram. This is an alternative method for adding objects which can be useful when you want to add an object with the same or similar properties and attributes to an existing object.

Use the  button to add a new exit point and the  button to connect transitions between the states as shown in the following picture:



You can add route points while routing a transition by clicking at several points between states to create a smooth arc as shown in the picture between states *s2* and *s3*

## Complete the Hierarchical State Diagram

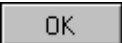
Use the **States** tab of the State Machine Object Properties dialog box to rename the states and add state actions as shown below:


Old Name	New Name	Style	Actions
s0	standby	IF	hold = 1; clear = 0; load = 0;
s1	alarm	IF	hold = 1; clear = 1; load = 0; beep = 1;
s2	counting	IF	hold = 0; clear = 0; load = 0;
s3	suspended	IF	hold = 1; clear = 0; load = 0;
s4	end_count	IF	hold = 1; clear = 1; load = 0;

Use the **Transitions** tab to add the following conditions:

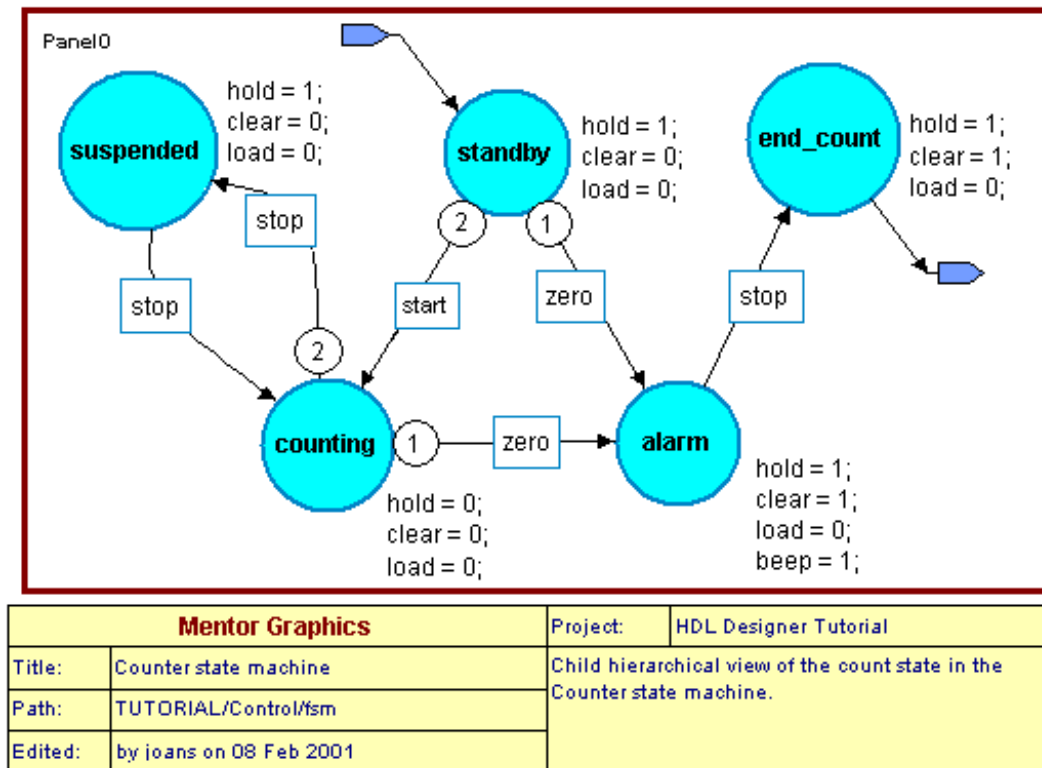
Origin State	Destination State	Priority	Condition	Actions
suspended	counting	1	~stop	none
counting	suspended	2	stop	none
counting	alarm	1	zero	none
standby	counting	2	start	none
standby	alarm	1	zero	none
alarm	end_count	1	stop	none




A confirmation dialog box may be displayed warning that a transition you have not yet edited has no condition and is not the lowest priority. Click the  button to acknowledge the warning.

Complete the hierarchical state diagram by editing the title and comments in the title block and using the  button to add a panel around the graphical objects on your diagram. This panel can be used to set the diagram view when you animate the state diagram later in this tutorial.

The state diagram should look similar to the picture below:



Ensure that the transition priorities are as shown and that you include the terminating semi-colon when you enter the state actions.


Use the  button to save the state diagram.

Although it is displayed as a separate window, a child hierarchical diagram is a partial view of a state machine and the parent and child diagrams are saved as a single design unit view (*TUTORIAL\Control\fsm.sm*) although the name in the title bar is *TUTORIAL\Control\fsm[:count 'machine0']*.

## Editing State Machine Properties

There are a number of properties associated with a state machine which can be edited using the State Machine Properties dialog box. The dialog box can be accessed by choosing **State Machine Properties** from the **Diagram** or popup menu in the parent or child state diagram. It can also be accessed by double-clicking over one of the labels which are displayed in the parent diagram of a hierarchical state machine.

Use the  button (or choose the **Open Up** option from the **File** menu) to re-display the parent state diagram if it is not already displayed.

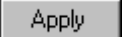
Double-click the  mouse button over the Global Actions or Concurrent Statements or State Registers label to display the **Statement Blocks** tab of the State Machine Properties dialog box.

There are no [global actions](#), [concurrent statements](#) or [state register statements](#) required in this design. Hide the labels for these object on the state diagram by clearing the **Visible** check box for Global Actions, Concurrent Statements and State Register Statements in the **Statement Blocks** tab.

Choose the **Module Declarations** tab (by double-clicking over the Declarations label on the state diagram).

Enter the following constant declaration in the free-format entry box:

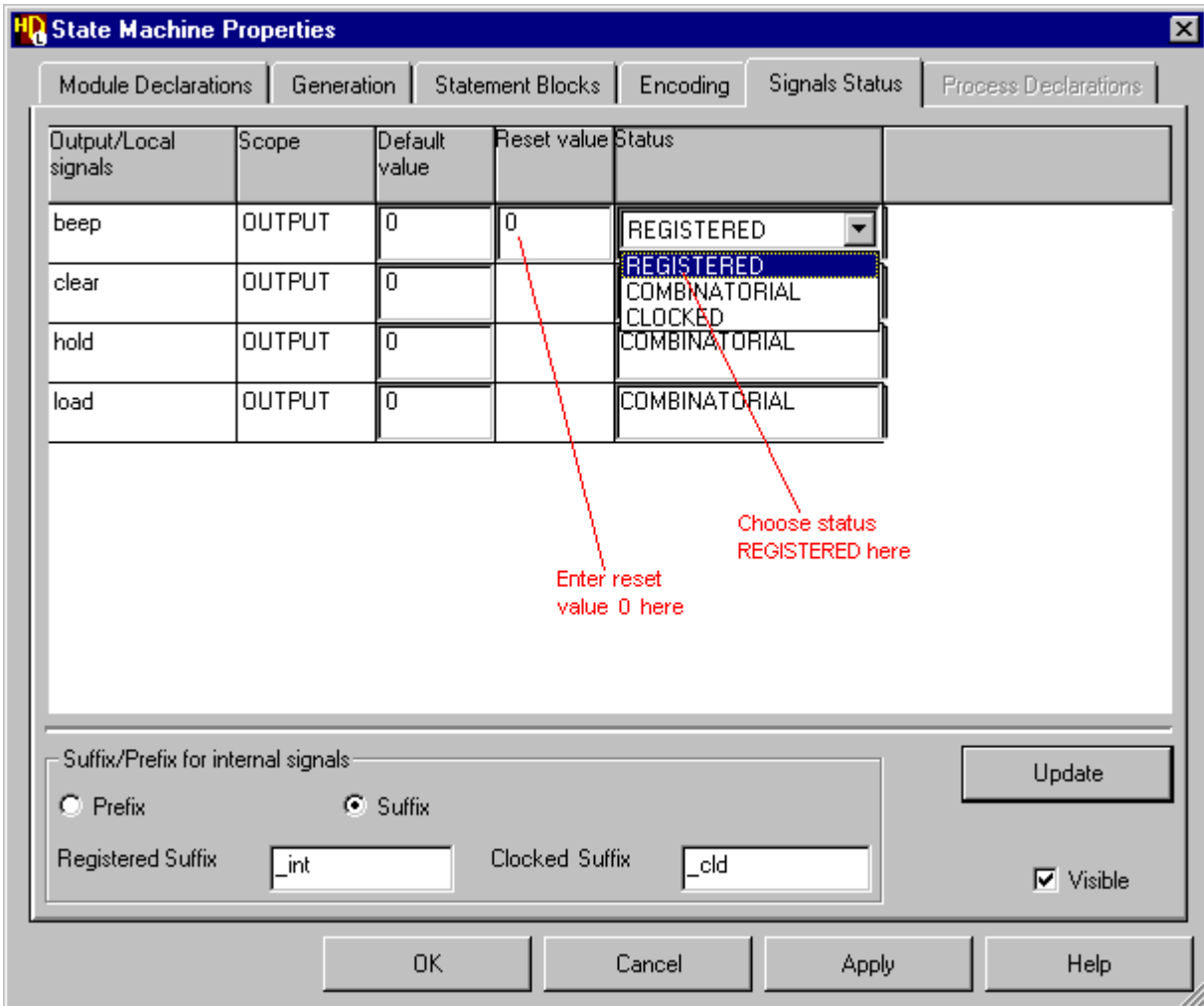
```
parameter ZEROS = 10'b0;
```

Click the  button to update the diagram. The constant declaration is added below the Module Declarations label on the state diagram and will be included as [module declarations](#) when HDL is generated for the state machine.

Re-display the State Machine Properties dialog box if necessary and choose the **Signals Status** tab (by double-clicking over the Signals Status label on the state diagram).

Notice how the output signals are listed in the dialog box with the default value set to 0 and the status *COMBINATORIAL*.


Click on the Status field for the *beep* signal and choose *REGISTERED* from the drop down box. Click in the Reset Value field and enter the value 0.

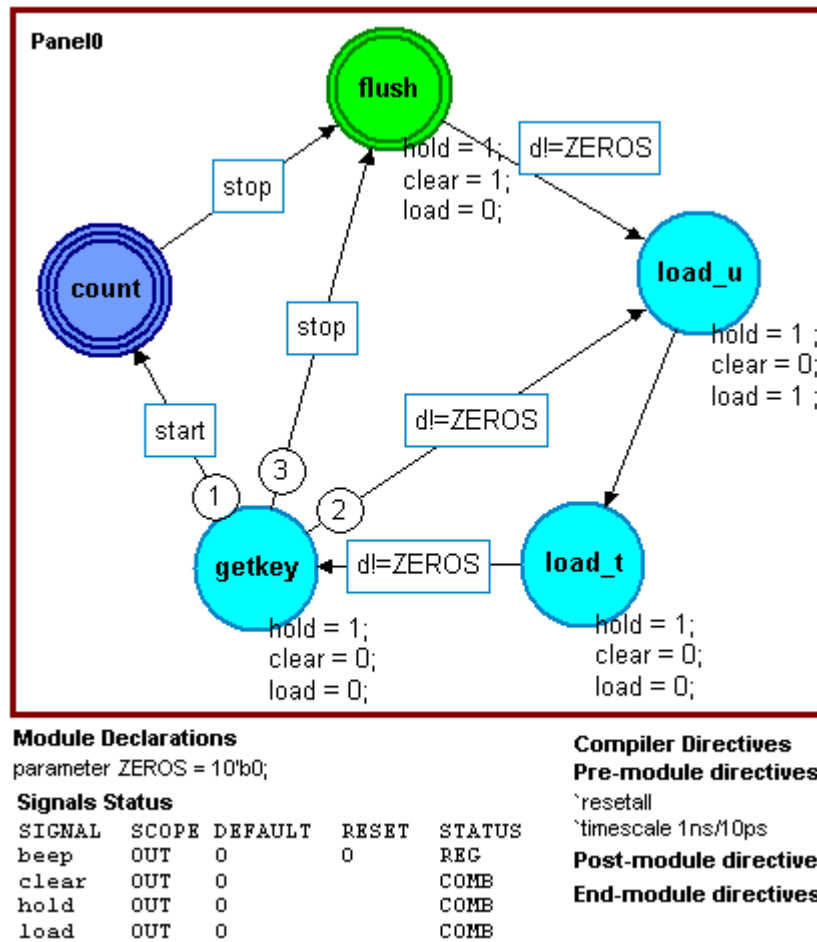


Any locally declared internal signals would be shown with the status REGISTERED. Bidirectional or buffer signals are treated as output signals.


The dialog box also allows you to change the suffix or prefix used for internal registered or clocked signal names. For this tutorial, suffixes are used with the default values *\_int* and *\_cld*.

Confirm the dialog box by clicking the **OK** button.

Complete the state diagram by editing the title and comments in the title block and using the  button to add a panel around the graphical objects on your diagram. This panel can be used to set the diagram view when you animate the state diagram later in this tutorial. The final state diagram should look similar to the picture below:

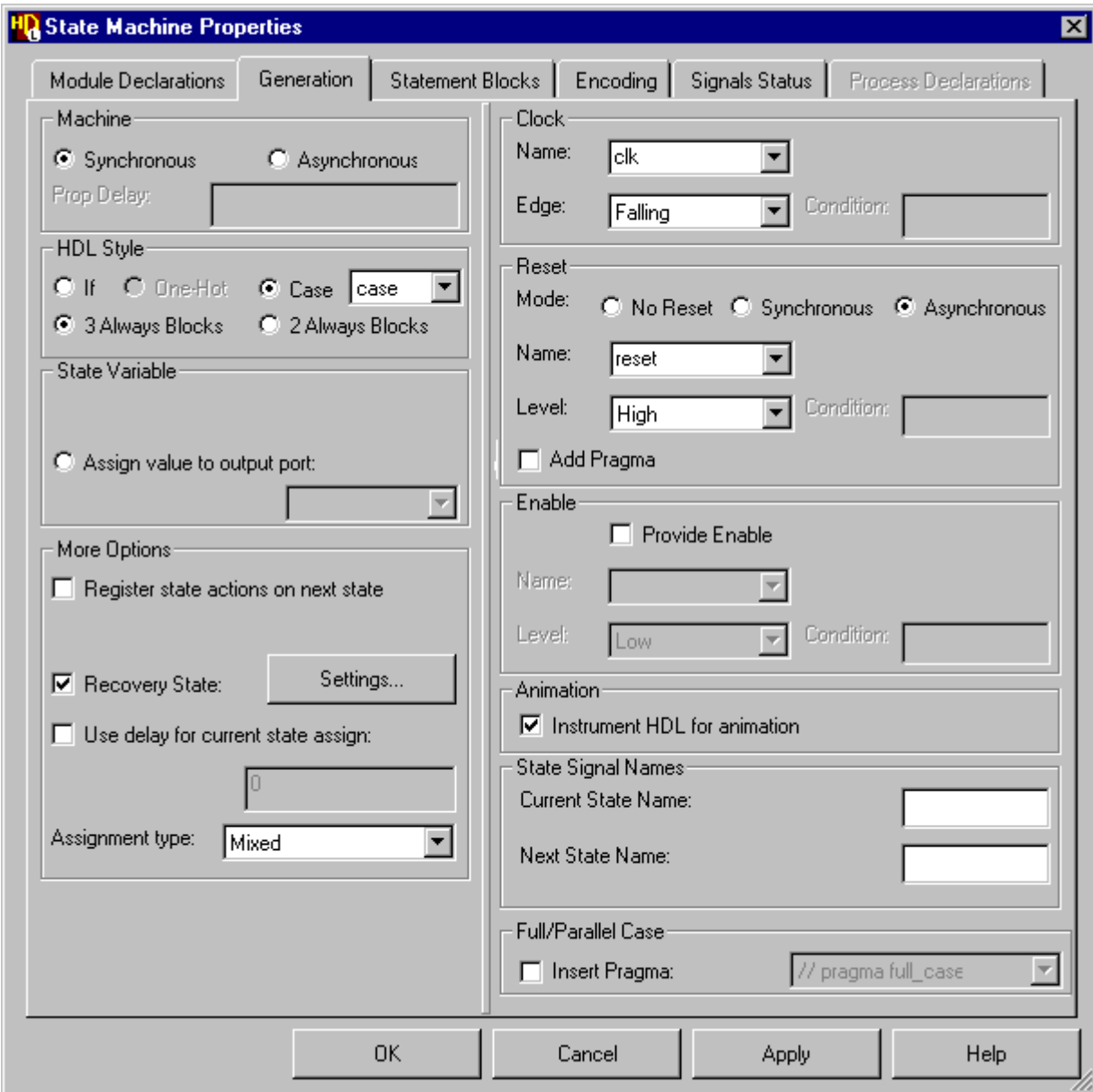


Notice that the *beep* signal (in the *alarm* state on the child hierarchical state diagram) has been replaced by the internal signal name *beep\_int*.


The State machine Properties dialog box also provides tabs for setting HDL generation characteristics and state machine encoding. State machine encoding is not used in this tutorial and the encoding mode can be left as **Auto**. You can use the  buttons for more information about each tab of the State Machine Properties dialog box.

## Set Generation Properties

Re-display the State Machine Properties dialog box if necessary and choose the **Generation** tab. This tab sets generation characteristics used for HDL generation.



Choose the **Synchronous** option in the Machine box. Use the default options **Case** and **3 Always Blocks** for HDL Style and leave the **State Variable** option unassigned.

Check that the **Recovery State** is set to `<start_state>` by using the  button to display the Recovery State Settings dialog box.



This entry automatically assigns the start state or you can choose from a pulldown list of states and specify recovery state actions.

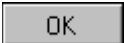
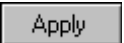
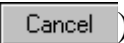
Leave the **Register state actions on next state** and **Use delay for current state assign** options unset, and leave the **Assignment type** set to Mixed.


Choose *clk* from the pulldown list of clock signal names and *Falling* from the Edge pulldown list. Set an **Asynchronous** reset and choose the signal name *reset* with a *High* level from the pulldown lists. Leave the **Provide Enable** options unset.

If a ModelSim or Verilog-XL simulator is available, set the **Instrument HDL for animation** option. This option adds additional code to the generated HDL which is used for state machine animation later in this tutorial.



This additional code is surrounded by translation control pragmas which ensure that it is ignored by downstream synthesis tools.

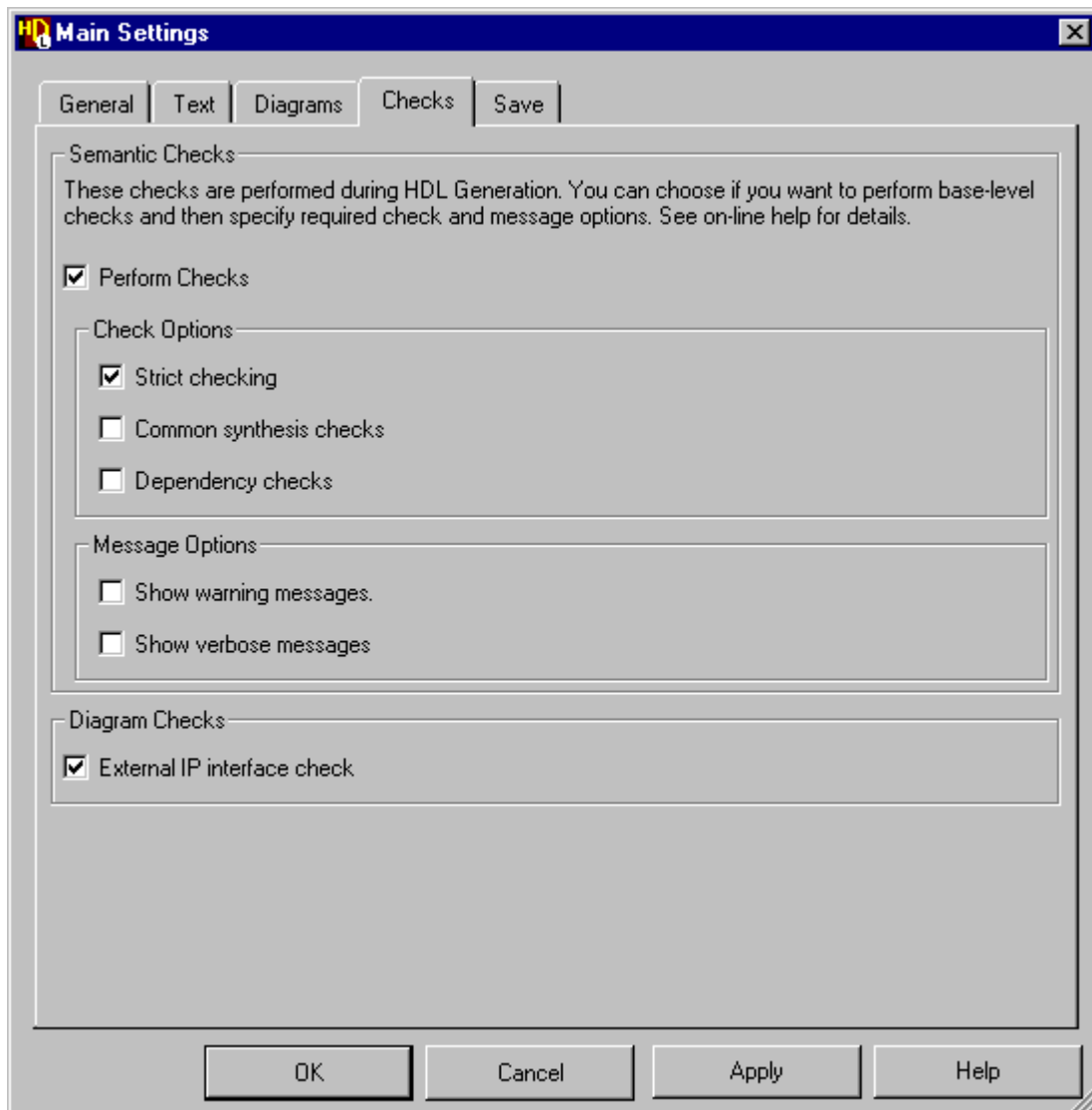
Use the  button (or  followed by ) to apply these generation characteristics to your state machine and dismiss the dialog box.

Use the  button to save the state diagram.

## Set Checks for HDL Generation

Although the *Timer* design is not yet complete, you can now generate HDL for the *Control* state machine.

You can set the level of checks performed when HDL is generated by using the **Checks** tab in the Main Settings dialog box which can be accessed by choosing **Main** from the **Options** menu.

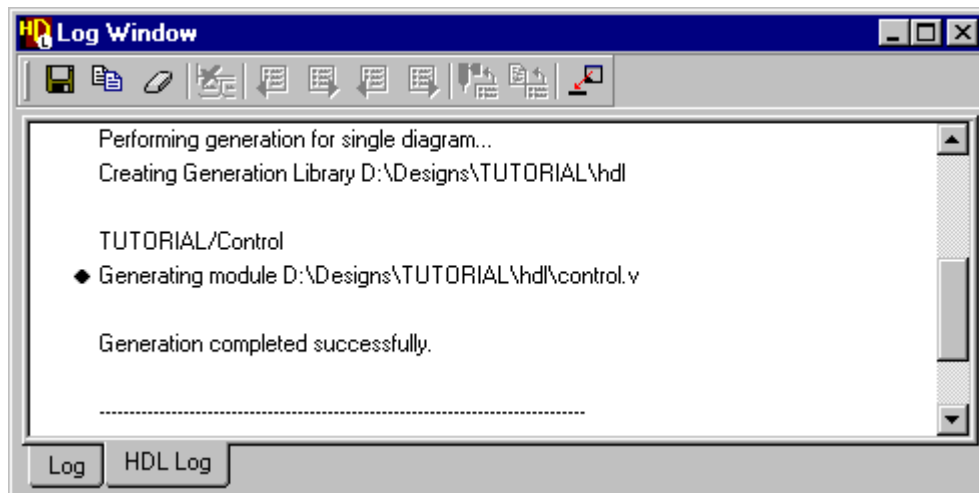





For this tutorial, the **Strict checking** option should be selected.


## Generate HDL for the State Machine

Choose the **Generate** option from the **HDL** menu (or use the  button) in the state diagram window.

The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation.




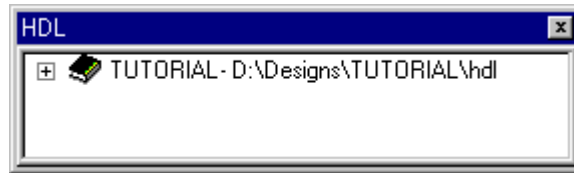
If there are any errors, you can move to the next error message using the  button or the previous error using the  button. You can display the source graphics corresponding to the error by double-clicking on the error message (or using the  button when the error is selected).

Alternatively, you can use the  button when the error is selected to view the generated HDL. If your text editor supports a line number argument, the HDL line corresponding to the error is selected automatically.

If there are no errors, you can use the toolbar buttons to view the source graphics or generated HDL corresponding to any message which is marked by the ◆ icon in the log window. For example, the “Generating module ...” message shown above.





Any files in the generated HDL directory can also be opened from the [HDL browser](#). Click on the  icon for the *TUTORIAL* library in the HDL browser and notice that the view is expanded to reveal the generated [Verilog module](#) for the *Control* design unit.



You can open this file by double-clicking on the generated file name or by choosing **Open** from the popup menu.



You can also view the generated [Verilog module](#) file for the active diagram by using the  button in any editor window or when the diagram view is selected in the design browser.

If you are using the default text editor (ESview on Windows or NEdit on UNIX), you can cross-reference from an error message to the corresponding line in the generated HDL file. You can also use the  button or the **Show HDS Source** menu command in the text editor to cross-reference from any line in the generated HDL to the corresponding graphics object.




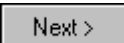
These options can also be set up for other user customizable external text editors. Refer to the “Setting the Text Editor” help topic for more information.

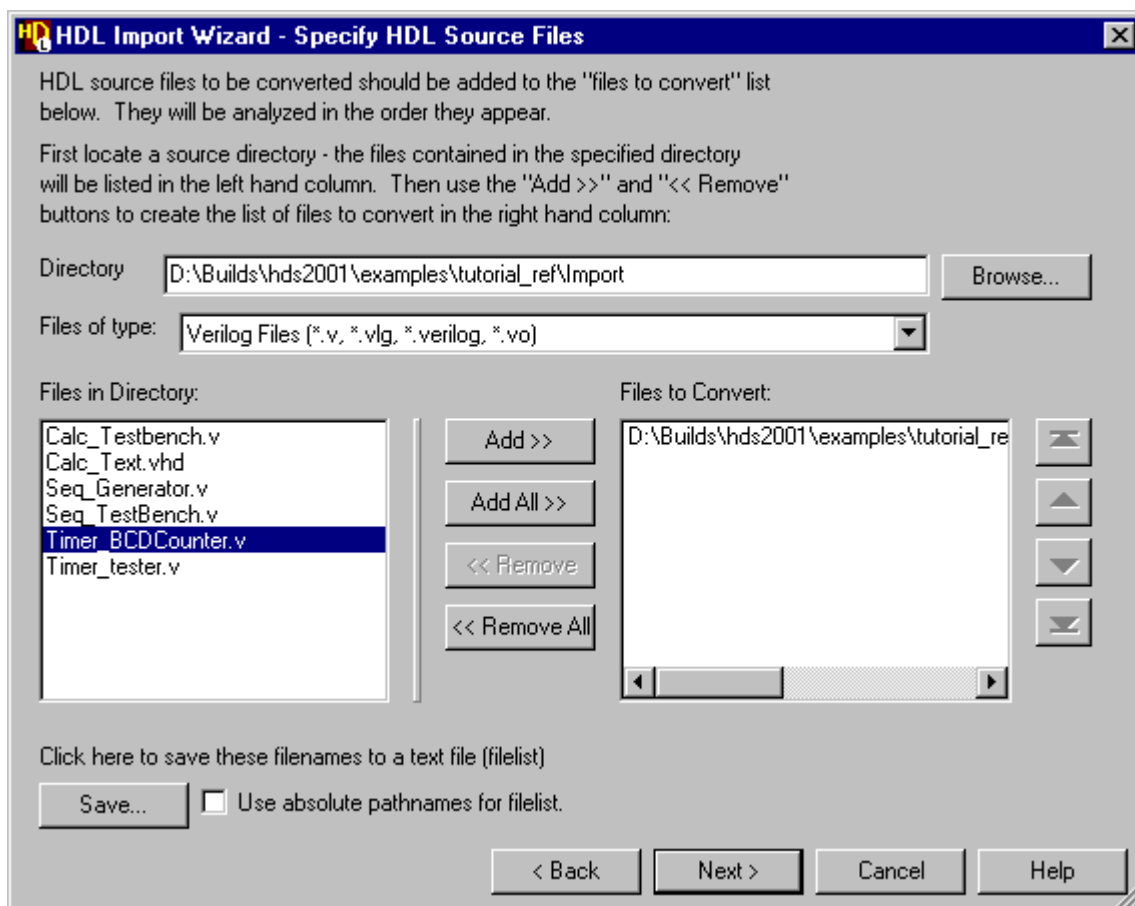
Close the text editor after you have viewed the generated HDL.

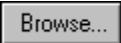

Use the **Close Window** option from the **File** menu to close the parent (*fsm [‘machine0’]*) and child (*fsm [:count ‘machine0’]*) views of the *Control* state machine.

## Import the BCDCounter Design Unit

You have now defined the *Control* block by drawing a graphical state machine. The *Counter* block will be defined by instantiating two instances of a HDL text component in a child block diagram.

Choose the pulldown  on the  button and select the  option from the palette or choose **Text HDL Import** from the **HDL Import** cascade of the **HDL** menu to display the HDL Import wizard. Choose **Specify HDL files** in the first page of the wizard and click the  button to display the Specify HDL Source Files page.

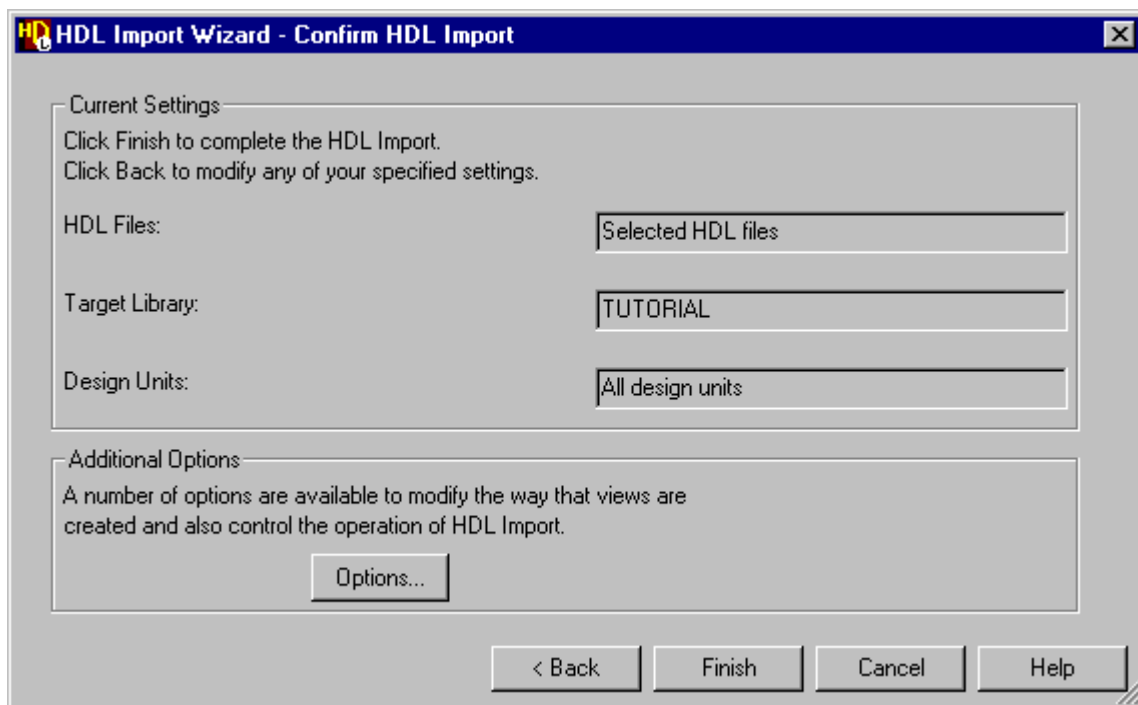


Choose Verilog files from the pulldown filter for Files of type and use the  button to locate the `..\examples\tutorial_ref\Import` installation subdirectory. Select the `Timer_BCDCounter.v` file and add it to the list of files for conversion by using the  button.

Click the  button on the wizard to display the Design Units page. This page allows you to select from a list of design units found in the source HDL files. In this case, there is only the *BCDCounter* and the **All** option should be selected.

Click the  button on the wizard to display the Target Libraries page. This page allows you to select the default target library and add libraries if there are any instantiations for black box components in the source HDL code.

Ensure that your *TUTORIAL* library is selected as the target library and click the  button to display the Confirm HDL Import page.



Click the  button in this page to complete the HDL import. The following completion message should be displayed in the HDL log window:

```
** Creating component BCDCounter in library TUTORIAL
** Creating HDL view spec.v of component BCDCounter in
                                                                    library TUTORIAL

HDL Import complete
-----
1 HDS design unit saved,
1 component
  1 HDL view
-----
```

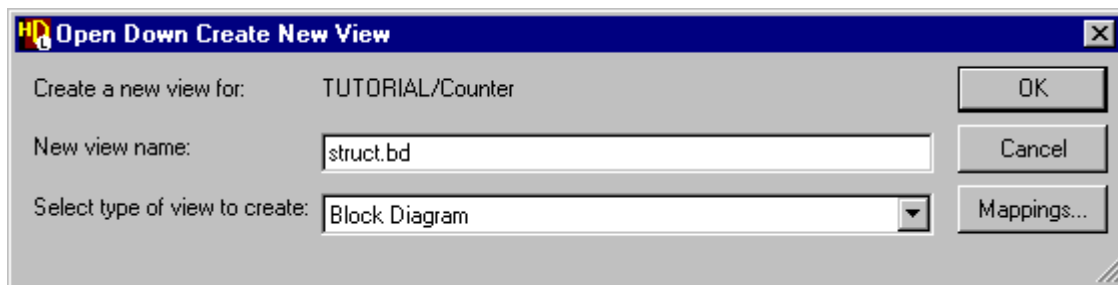
## Create a Child Block Diagram

Display the *Timer* block diagram.



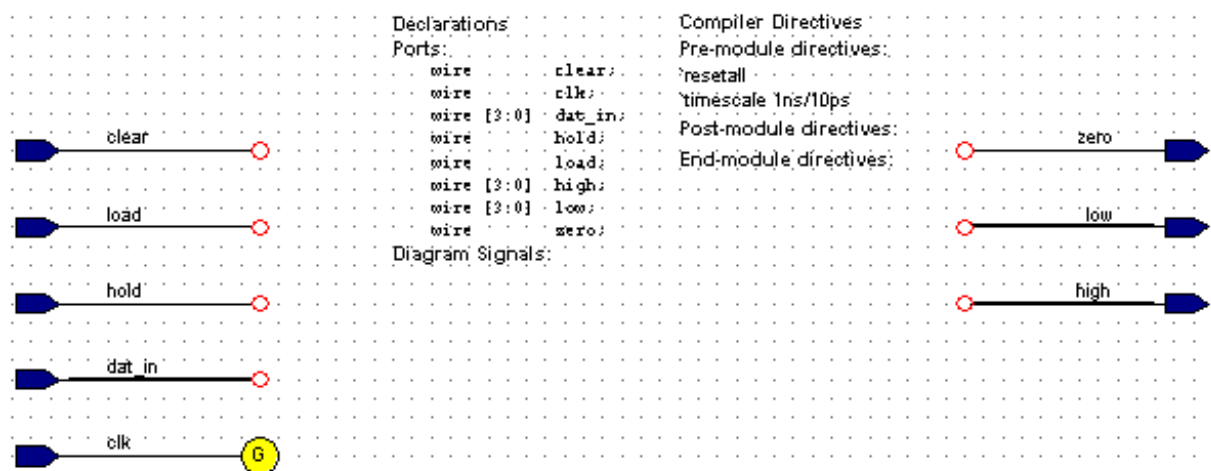
If the window is obscured, you can pop it to the front by selecting the window name from the **Windows** menu list in any other window.


Double-click (or choose **Open** from the popup menu) on the body of the *Counter* block to display the Open Down Create New View dialog box.

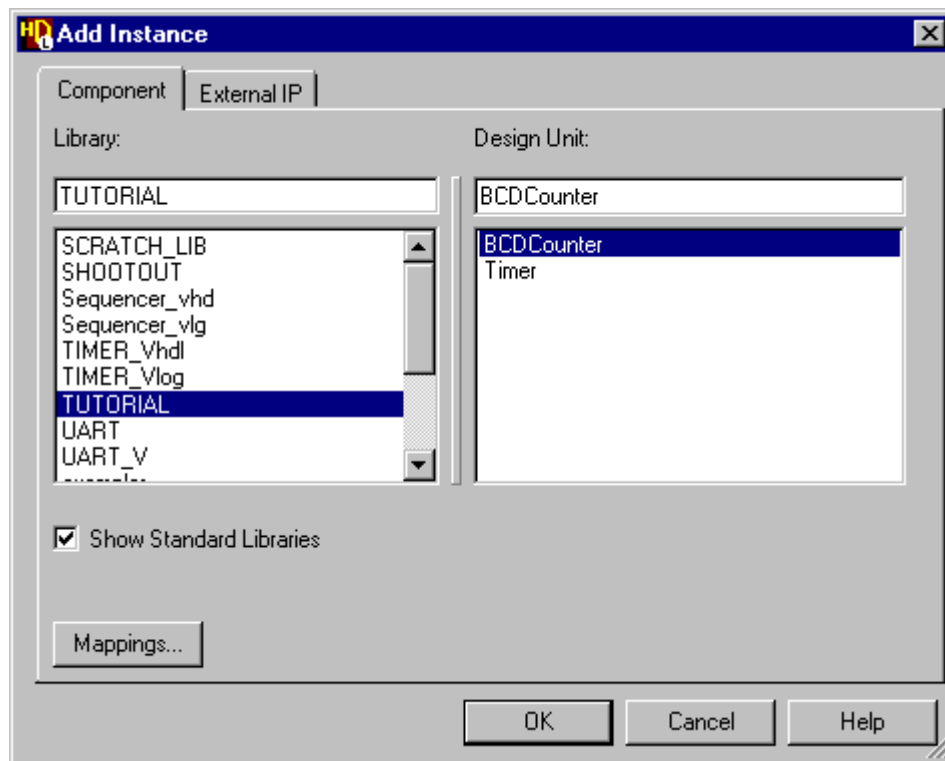


Select **Block Diagram** from the pulldown list of views you can create and accept the default view name *struct.bd*.

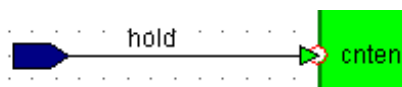
A new block diagram (*TUTORIAL\Counter\struct*) is created as a child view of the *Counter* block. The child block diagram is initialized with declarations and ports corresponding to the connected signals and buses on the parent diagram (including a global connector for the *clk* signal). Notice how inputs are initialized on the left and outputs on the right.




Use the  button to display the Add Instance dialog box. Use the dialog box to add two instantiations (*I0* and *I1*) of the *BCDCounter* component placing one below the other on the diagram but allowing several grid lines for connections between them.





Move the *hold* signal so that it overlaps the *cnten* port on instance *I0*.

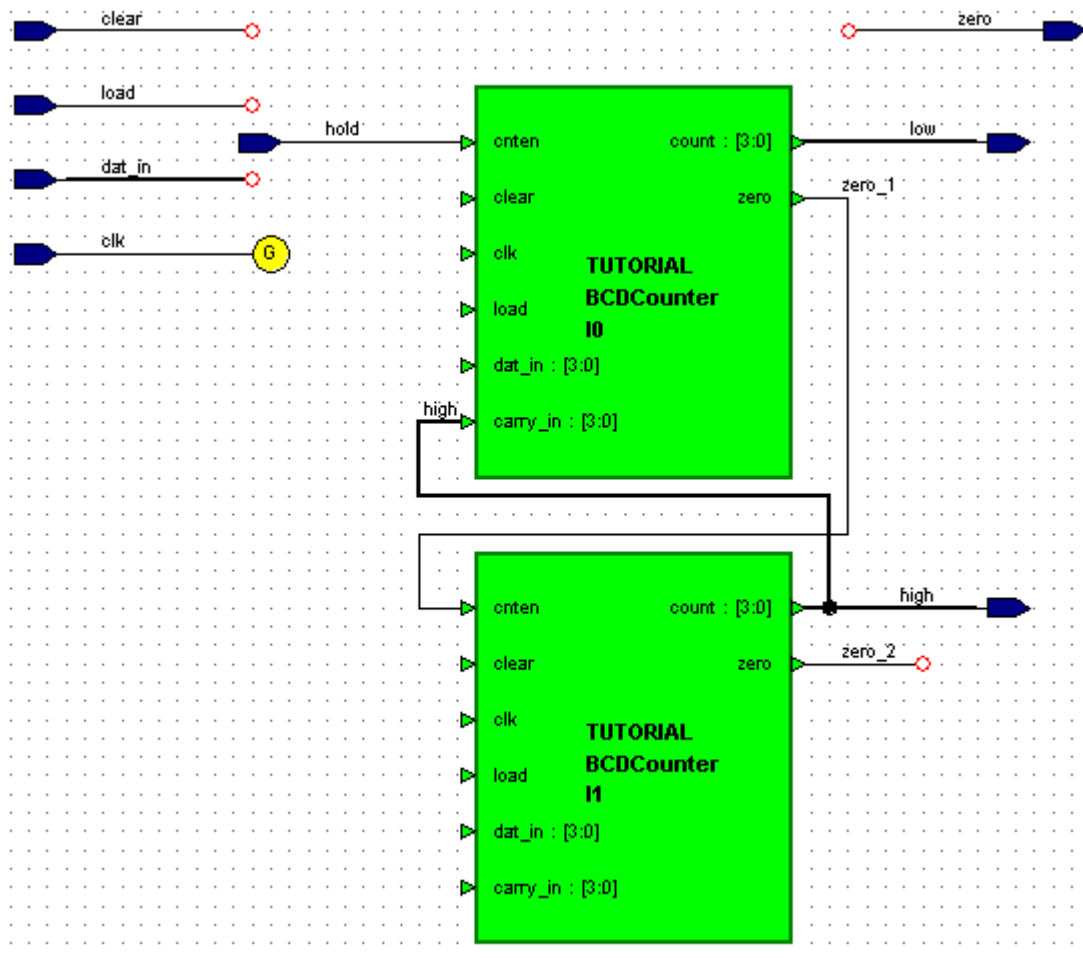


Similarly, move the *low* signal over the *count* output port on instance *I0* and the *high* signal over the *count* output port on instance *I1*. Select both instances and choose **Connect** from the popup menu to make the connections.

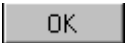
Use the  button to connect a signal between the *zero* output port on *I0* and the *cnten* port on *I1*. This signal is automatically named *zero\_1* since the output signal *zero* already exists on the diagram.

Use the  button to connect a bus between a point on the *high* output bus and the *carry\_in* port on *I0*.

Use the  button to connect a stub signal to the zero output port on I1. This signal is automatically named *zero\_2* since the signals *zero* and *zero\_1* already exist on the diagram. The block diagram should now look similar to the following picture:



Select the global connector and choose **Delete** from the popup menu taking care not to delete the stub *clk* signal and input port.

Select instance *I0* and choose the **Add Signal Stubs** command from the popup menu. You are warned that the nets *clear*, *clk*, *dat\_in* and *load* already exist on the diagram. Click the  button to acknowledge the warning.

The signal stubs are added when you accept the warning and the matching signals are implicitly connected by name.

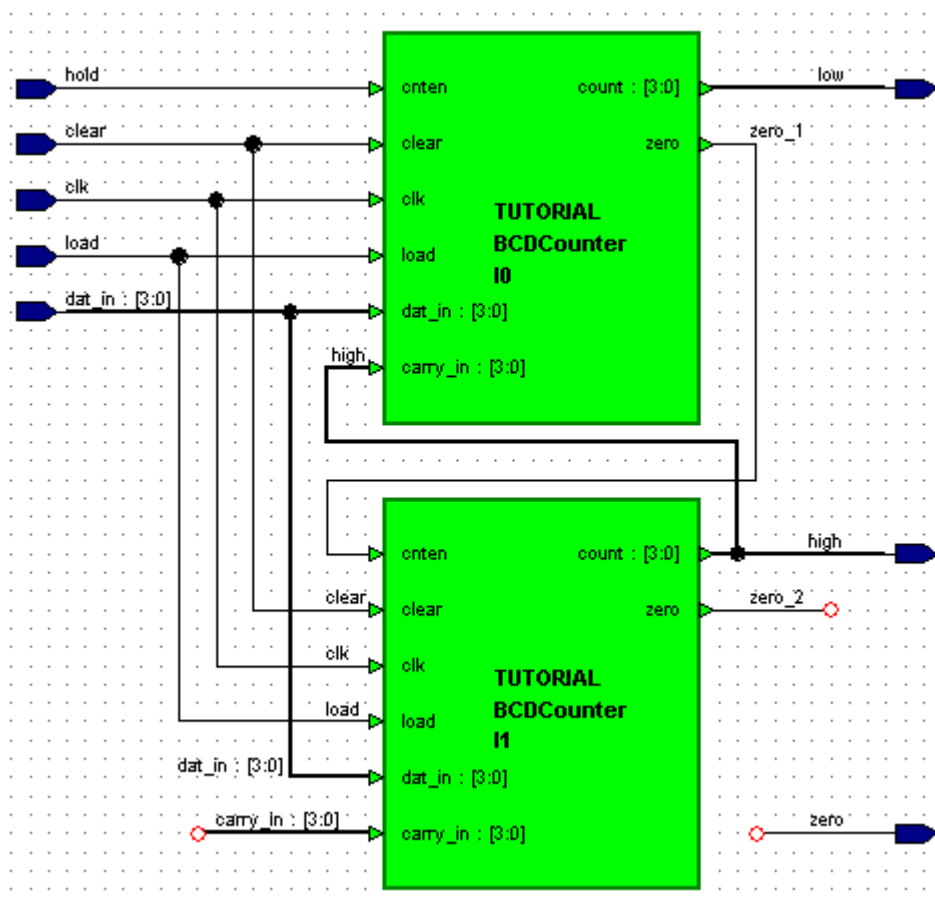
Select the input port signals *clear*, *load*, *dat\_in* and *clk* then choose **Autoconnect** from the **Autoroute** cascade of the popup menu. The signals are automatically connected to the matching stub signals on instance *I0*.



If nothing is selected on the diagram, autoconnect attempts to re-route all connections on the diagram. Ensure that only the required signals are selected if you do not want to change all the connections on a diagram.


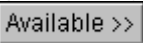

Select instance *I1* and choose the **Add Signal Stubs** command from the popup menu to add stub signals for the remaining ports on this instance. Select these signals and choose **Autoconnect**. Click the  button and acknowledge the warning that the nets already exist on the diagram to complete the connections.

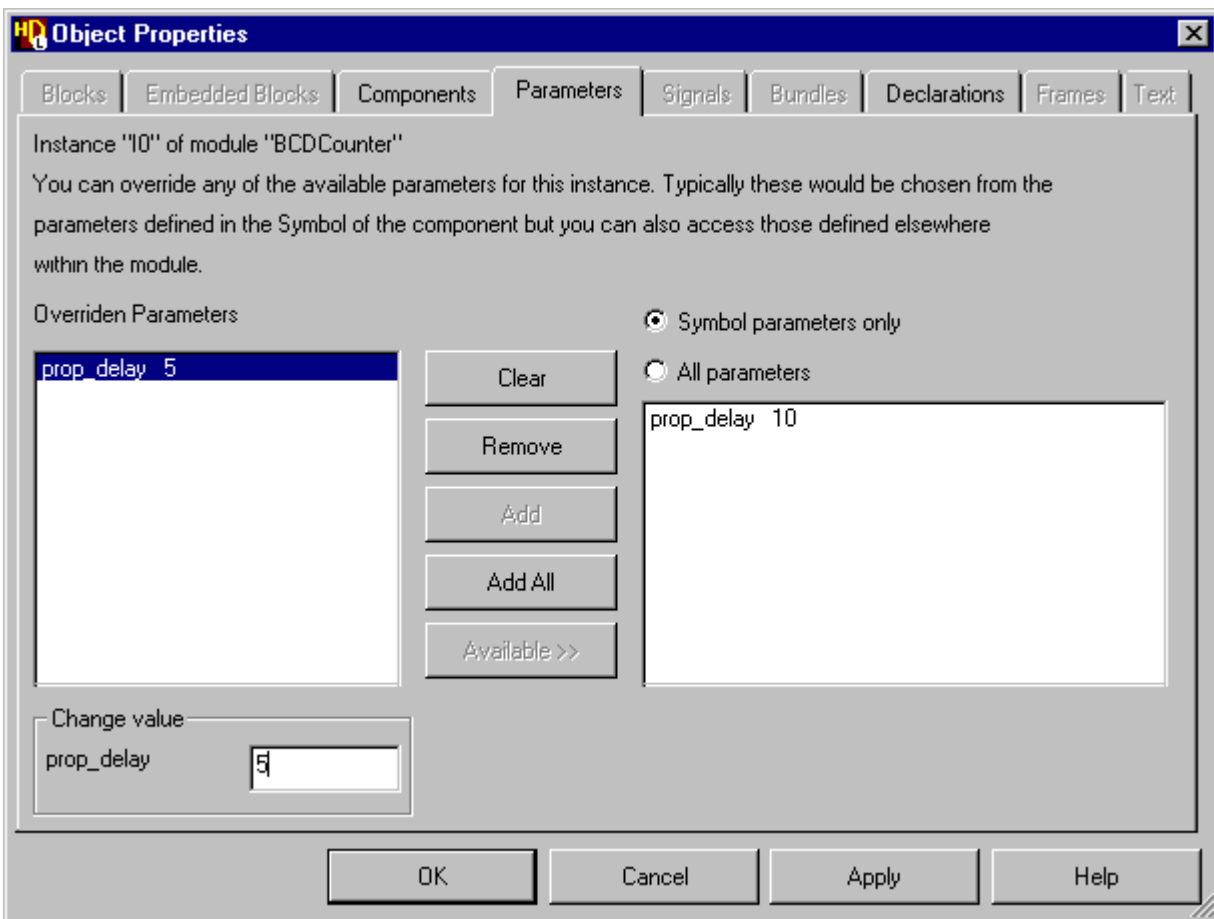
The block diagram should now look similar to the following picture:





## Edit the Parameter Mapping

Parameter mapping allows you to modify the value of a [parent](#) which is declared in the [symbol](#) defining the component which can have different values for each instance of the component.

Select instance I0 of the *BCDCounter* component, use the  button to display the Block Diagram Object Properties dialog box and choose the **Parameters** tab. Click the  button to display a list of the parameters defined on the symbol and the  button to move the *prop\_delay* parameter into the list of overridden parameters.




Edit the value in the Change value box to be 5 and use the  button to update the block diagram. The parameter mapping is added above the component on the diagram as a text object which can be moved independently by dragging it with the  mouse button.

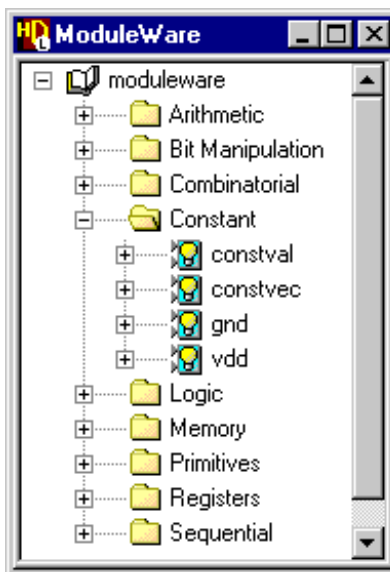



Repeat this procedure to set the parameter mapping for the second instance of *BCDCounter*. The parameter mapping is placed above the component (and may overlap the other instance). Use the mouse to drag the parameter declaration below the component so that it is clearly associated with the second instance.

## Add ModuleWare Components

Although you have routed the *Counter* block diagram, several signals have been left unconnected. These will be connected by using [ModuleWare](#) components.

Display the ModuleWare browser by choosing **ModuleWare Browser** from the **Window** menu. The browser displays the contents of the *moduleware* library which is divided into a number of folders. Click on the  icon to expand the folder for the *Constant* category::



Use the  mouse button to drag the *gnd* component over the *Counter* block diagram and release the button to place it near the *carry\_in* input to instance *I1*. The ModuleWare instance is added with a default instance name (*I2*).



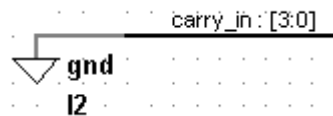
You can also add a ModuleWare component by selecting it in the Add Instance dialog box or by dragging it from the *moduleware* library when it is displayed in the source browser.

Use the mouse to position the stub signal on the *gnd* component over the dangling connector on the *carry\_in* net and choose **Connect** from the popup menu.



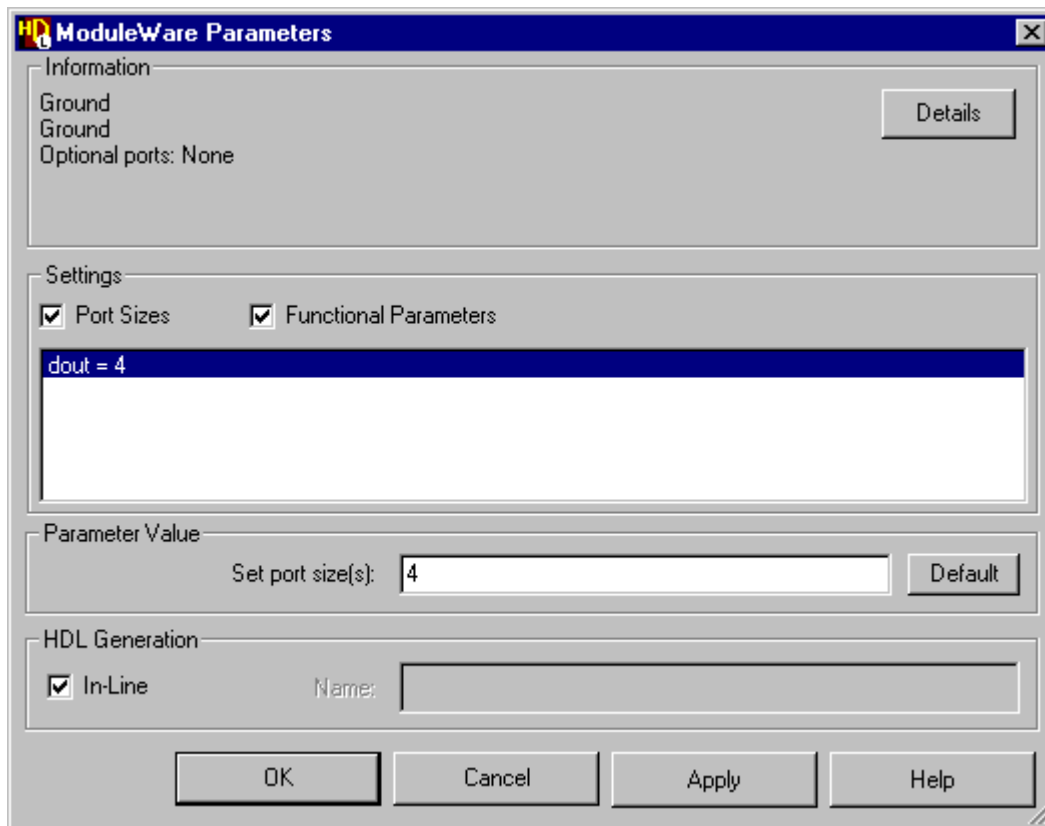
The port arrow head on the stub signal for the ModuleWare component instance automatically disappears when you make the connection.

The *I2* ModuleWare instance should now look similar to the following picture.



Notice that the *carry\_in* net is a 4-bit bus. The stub signal on the ModuleWare instance has a default width of 1 and must be set to match the width of the connected net.

Double-click over the *gnd* instance (or choose **ModuleWare Parameters** from the popup menu) to display the ModuleWare Parameters dialog box:



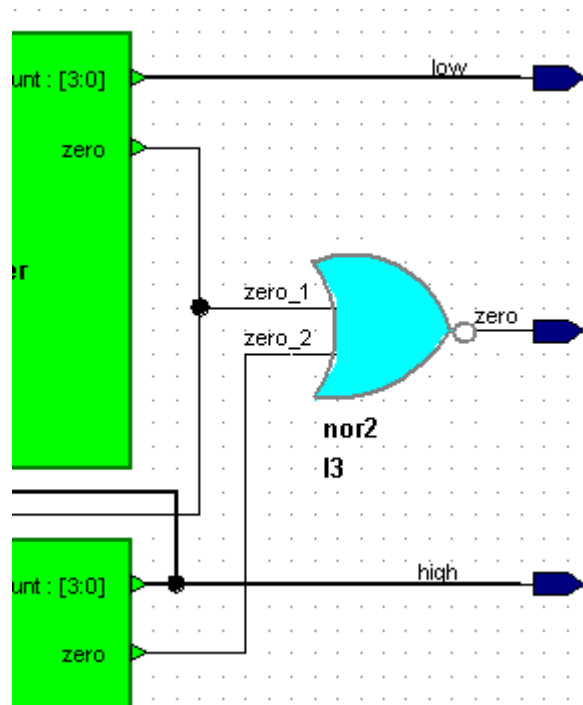
Select the *dout* signal and set its port size to 4. Check that the **In-Line** option is set for HDL Generation and confirm the dialog box by clicking the  button.



HDL can be generated in-line (within the same file as the view containing the ModuleWare instance) or as a separate file when this option is unset.

Repeat this procedure to add an instance of the *nor2* component from the *Logic* folder of the *moduleware* library near the *zero\_1* output from instance *I0* on your block diagram.

Connect this instance to the *zero\_1*, *zero\_2* and *zero* nets as shown in the following picture:





Notice that this instance is connected to 1-bit signals and you do not need to modify its port sizes.

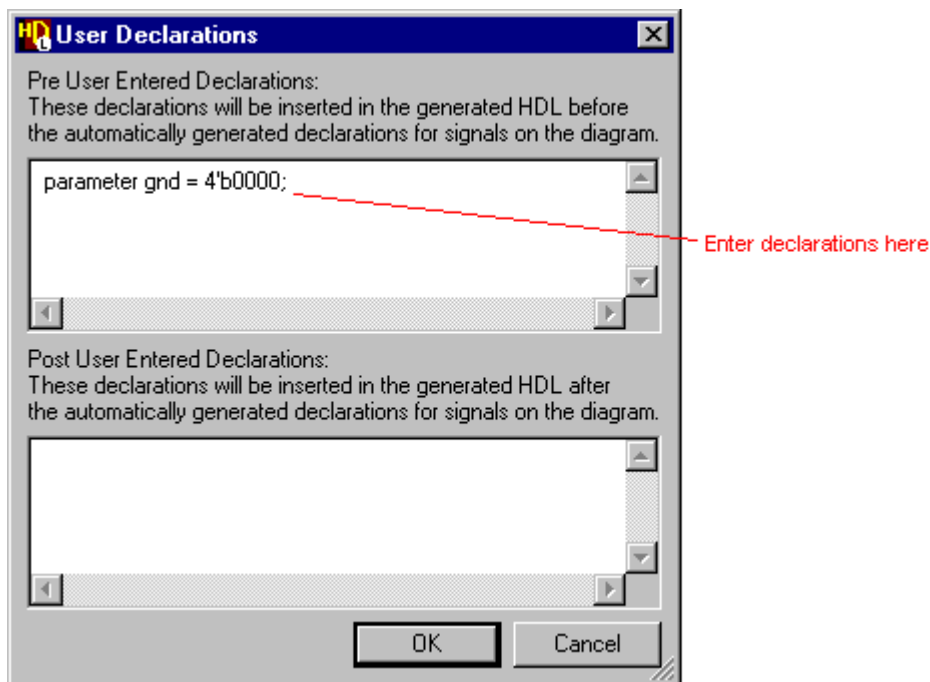


This function could also be implemented by instantiating a 2-input OR gate and using the ModuleWare Parameters dialog box to set the output port with the enumerated type *ActiveLow*. The logical NOR function could also be implemented by an embedded block similar to that used in [“Add an Embedded HDL Text View”](#) on page 2-18.

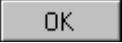

## Add a User Declaration

Double-click over the Declarations label on the diagram (or use the  button) to display the **Declarations** tab of the block diagram Object Properties dialog box. Click the  button to display a free format entry dialog box. Define the *gnd* signal by entering the following constant declaration in the Pre User Entered Declarations section of the dialog box:

```
parameter gnd = 4'b0000;
```




The syntax is automatically checked and any errors reported on entry. You can enter any valid Verilog statements which must be terminated by a semicolon (;) character. Line breaks and spaces can be used for clarity and will be preserved on the diagram.

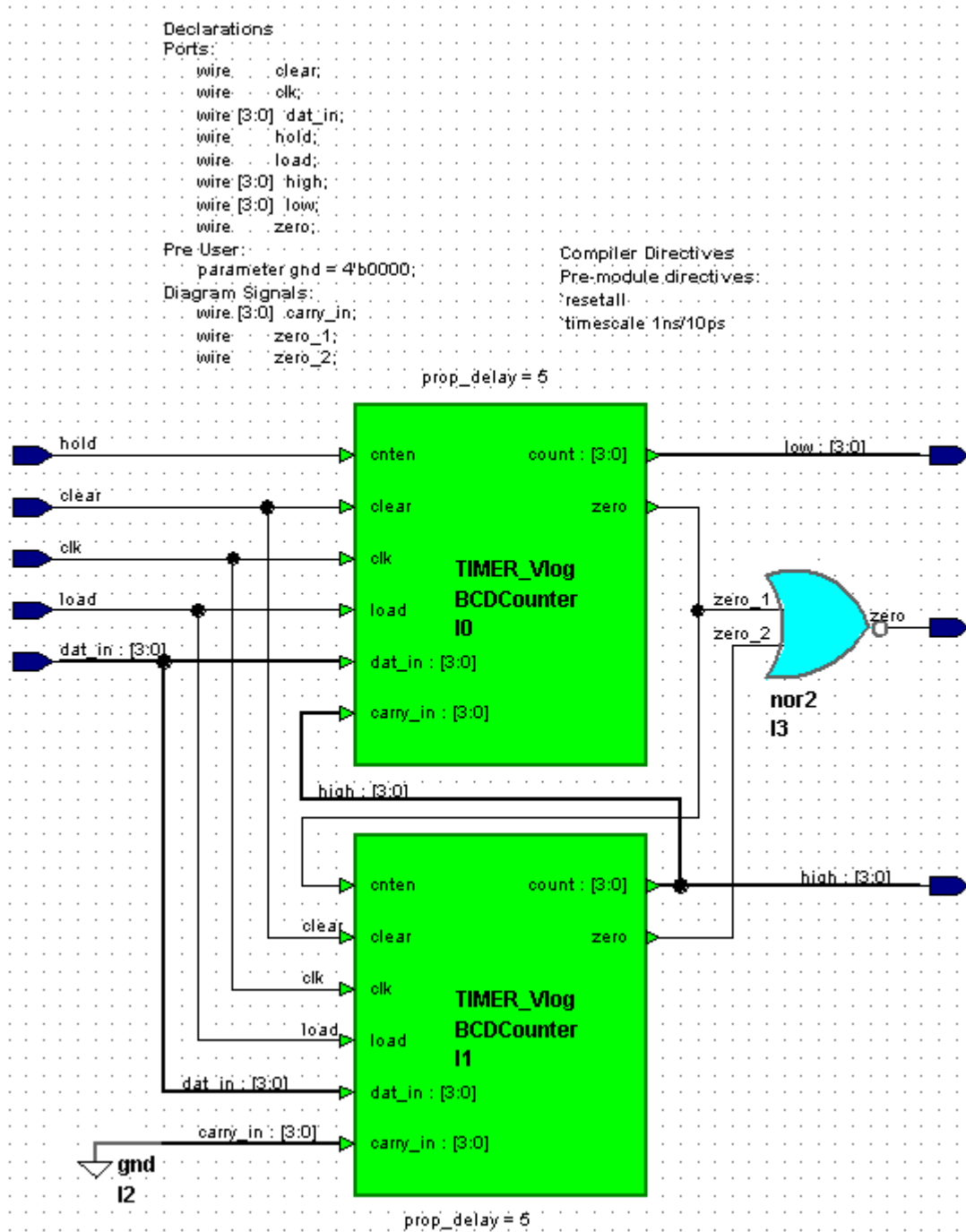
Click on the  button to confirm the user declaration and click the  button in the Block Diagram Object Properties dialog box to add the new user declaration on the diagram.



The pre user entered declarations are added to the declarations list on the diagram between the port and signal declarations.

Use the  button to save the block diagram.

The *Counter* block diagram should look similar to the following picture:



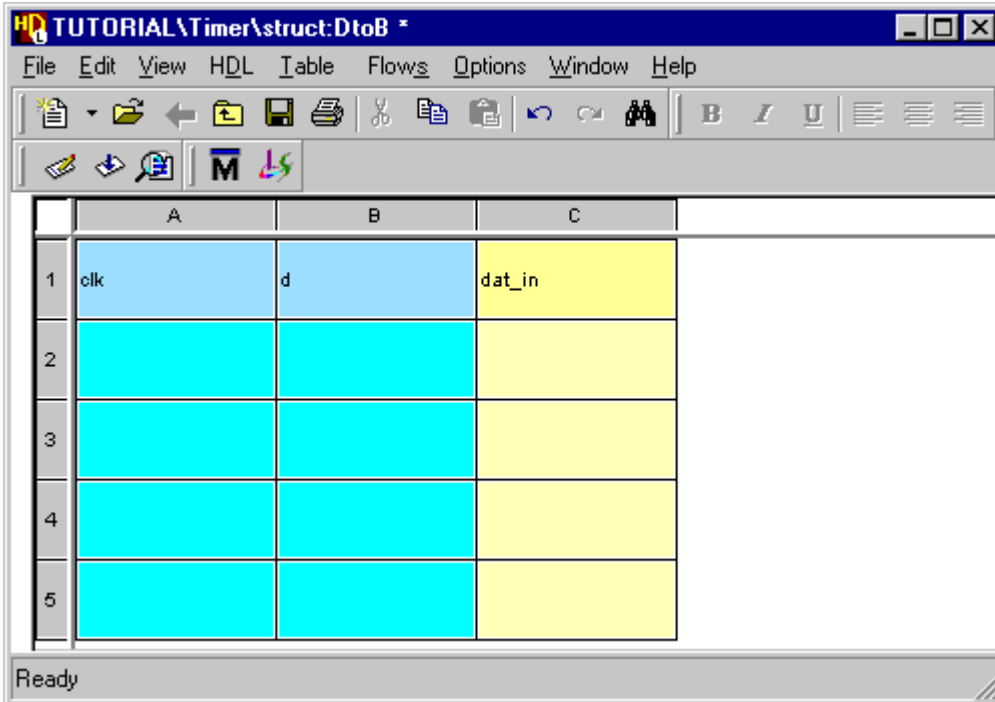
## Create a Truth Table

Use the  button (or choose the **Open Up** option from the **File** menu) in the *Counter* block diagram to display its parent diagram (the *Timer* block diagram).

You have now defined a state diagram view for the *Control* block and a block diagram view for the *Counter* block. The *Timer* design is completed by using a [truth table](#) to describe the *DtoB* embedded block.

Truth table views are not available if you are using the HDL text design tools. However, alternative procedures for using a HDL text view of the *DtoB* embedded block are given in [Appendix A](#).

Double-click over the *DtoB* embedded block to display the Create Embedded View dialog box and select **Truth Table**. A new truth table is created as an embedded view (*TUTORIAL\Timer\struct:DtoB*). The table is initialized as a matrix of four rows with two blue input columns (for the *clk* and *d* inputs) and one yellow output column (for the *dat\_in* output).



	A	B	C
1	clk	d	dat_in
2			
3			
4			
5			

Ready

## Edit the Truth Table

Click the mouse in either of the blue input columns (A or B) and use the **Add Column** option from the popup menu to add another eight input columns (C to J).

Click the mouse in one of the blue input rows (rows 2, 3, 4 or 5) and use the **Add Row** option to add another seven rows (6 to 12) to the truth table.





You can add multiple rows or columns by selecting more than one cell to add the corresponding number of rows or columns.







Rename the input columns to represent each bit ( $d[9]$  down to  $d[0]$ ) of the  $d$  input bus and enter the value  $1$  in one row for each column. (The  $clk$  signal is not used and can be renamed.)

Rename the output column  $d\_out$  and enter the following values:

```
4'b0000
4'b0001
4'b0010
4'b0011
4'b0100
4'b0101
4'b0110
4'b0111
4'b1000
4'b1001
4'b0000
```

To add text to a cell, click in the cell, enter the text and click in any other cell to confirm the entry. You can edit the text in an existing cell by double-clicking the cell and using the edit cursor to change individual text characters.

You can also use  button to copy and  button to paste text between cells and re-size the rows or columns by dragging the control sash in the non-scrolling area.

You can format text in single or multiple selected cells by using the  (bold),  (italic) or  (underline) buttons and align text using the  (align center),  (align left) or  (align right) buttons.

The completed truth table should look similar to the picture below.

	A	B	C	D	E	F	G	H	I	J	K
1	d[9]	d[8]	d[7]	d[6]	d[5]	d[4]	d[3]	d[2]	d[1]	d[0]	d_out
2										1	4'b0000
3									1		4'b0001
4								1			4'b0010
5							1				4'b0011
6						1					4'b0100
7					1						4'b0101
8				1							4'b0110
9			1								4'b0111
10		1									4'b1000
11	1										4'b1001
12											4'b0000



The last (empty) row in a truth table represents an ELSE condition and assigns the output expression a known value when none of the input expressions are true.

## Set Truth Table Properties

Choose **Truth Table Properties** from the **Table** menu to display the Truth Table Properties dialog box.

Select the **Concurrent Statements** tab and enter the following assignment:

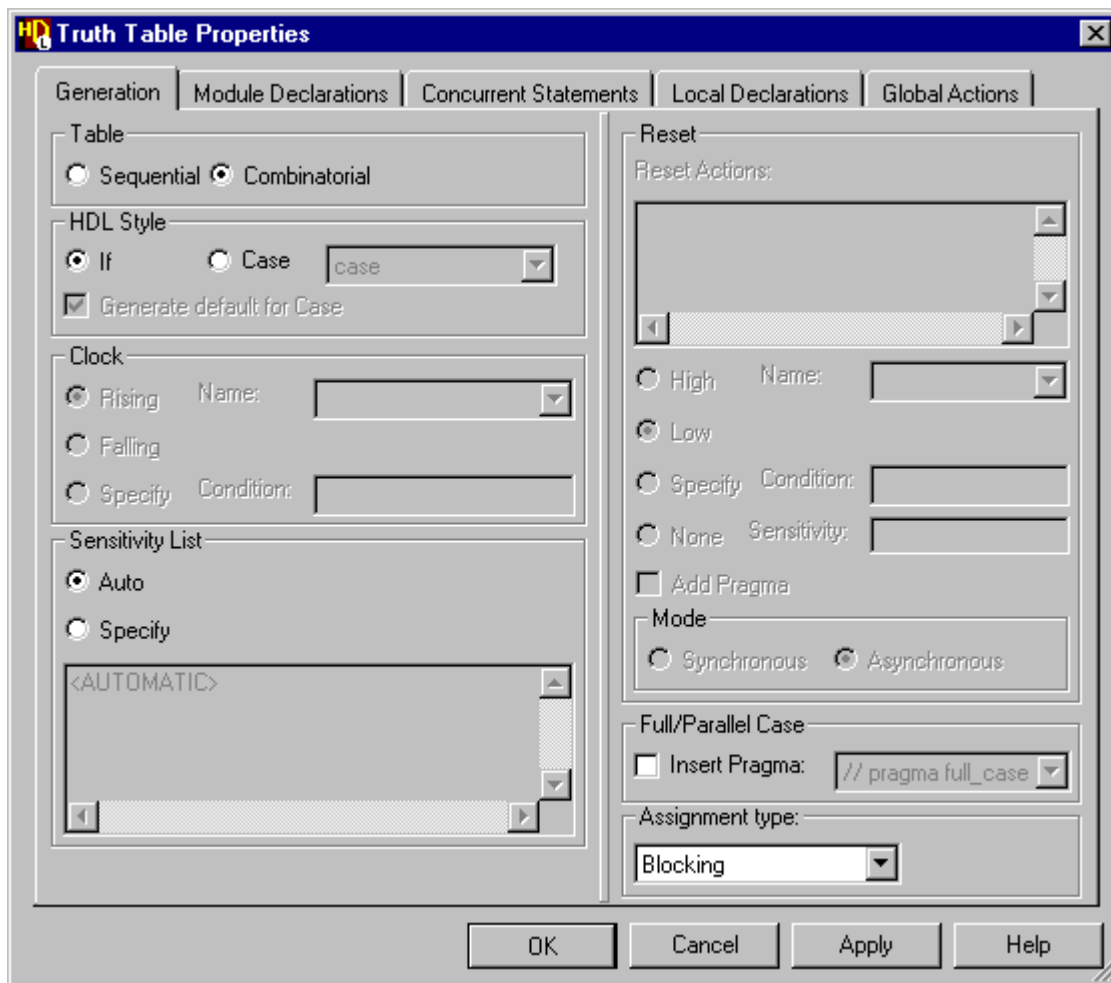
```
assign dat_in = d_out;
```


This statement assigns the value of the truth table output (*d\_out*) to the *dat\_in* bus which is connected to the *Counter* block. This is necessary because Verilog compilation requires the truth table output to have type *reg* but the *dat\_in* bus has the type *wire*.



The *d\_out* signal must also be declared in the module for the design unit. Since the truth table is an embedded view of the *Timer* design unit, this must be done from the parent diagram and is described in the next procedure.

Select the **Generation** tab to display the HDL generation characteristics for the truth table. Notice that the Sensitivity List is set to **Auto**. When this option is set, the sensitivity list is automatically created by HDL generation.






Check that the default options for **Combinatorial** and **If** style are set and use the  button to confirm the truth table properties.

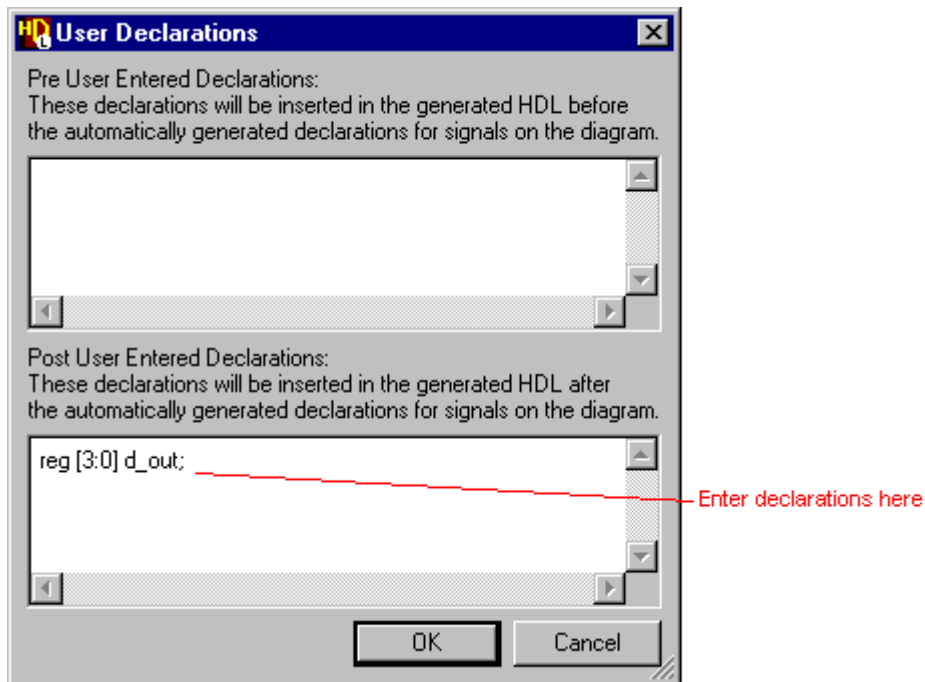


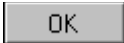
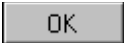
The tabs for Module Declarations, Local Declarations and Global Actions can be left empty for this tutorial.

## Add a Module Declaration

Use the  button (or choose the **Open Up** option from the **File** menu) in the *DtoB* truth table to display its parent diagram (the *Timer* block diagram). Double-click over the **Declarations** label on the diagram (or use the  button) to display the **Declarations** tab of the block diagram Object Properties dialog box. Click the  button to display a free format entry box. Enter the following signal declaration in the Post User Entered Declarations section of the dialog box:

```
reg [3:0] d_out;
```




Click on the  button to confirm the user declaration and click the  button in the Block Diagram Object Properties dialog box to add the new user declaration on the diagram.

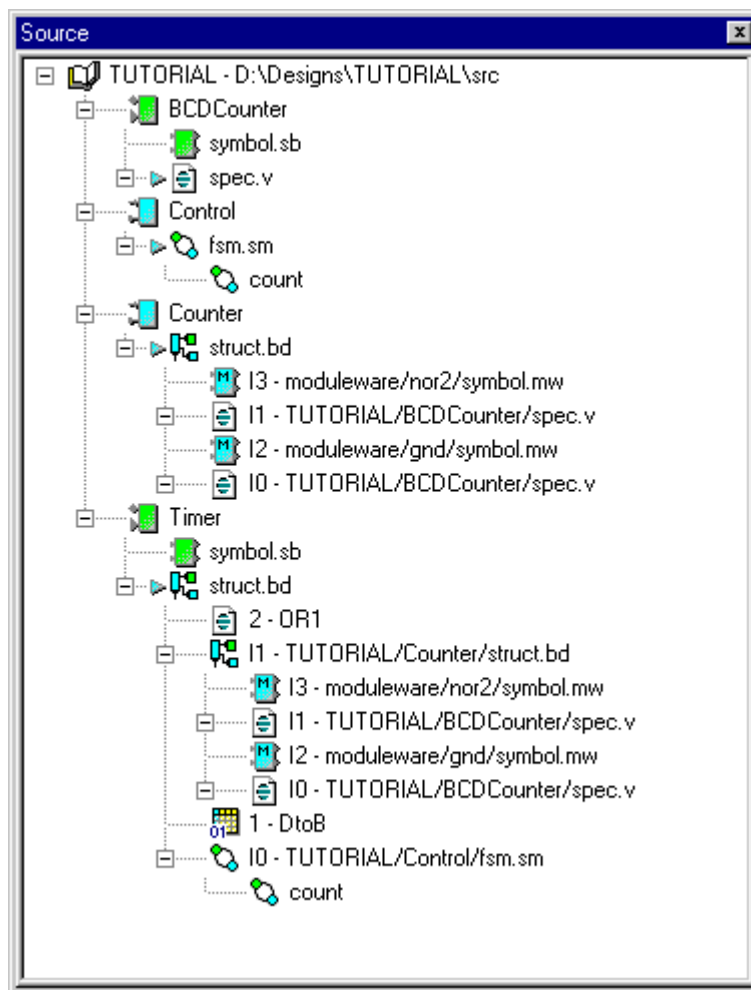


The post user entered declarations are added to the declarations list on the diagram below the port and signal declarations.

Save the block diagram. This also saves the truth table since it is an embedded view within the same design unit as the *Timer* block diagram.




## Browse the Timer Design

Examine the completed design in the source browser. Use the  icons to expand each design unit. You can expand the *BCDCounter* component to show it contains a Verilog module (*spec.v*) and a symbol (*symbol.sb*). You can expand the *Control* block to show it contains a state machine (*fsm.sm*) and also expand the state machine to show the child hierarchical state machine (*count*). You can expand the *struct.bd* view below the *Counter* block to show it contains two instantiations of the *BCDCounter*.

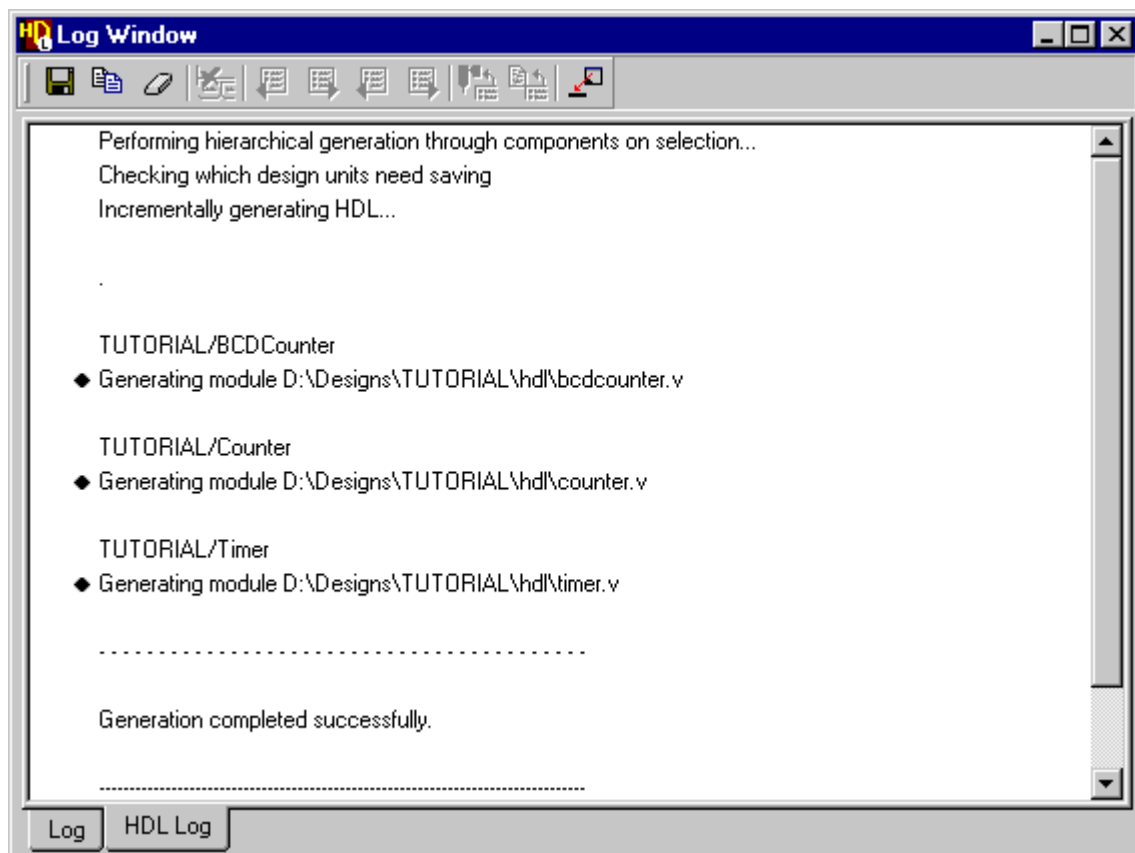


Notice that you can expand the *Timer* component to display the complete hierarchy including the HDL text views describing the *BCDCounter* and the *control* state machine.




## Generate HDL for the Hierarchy


Select the *Timer* design unit in the design browser window and choose the pulldown  on the  button. Select the  option from the palette (or choose the **Generate Through Components** option from the **HDL** menu).

The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation.





The *Control* design unit was generated earlier in this tutorial and is not generated unless it has been changed. HDL is generated for all other design units. You are prompted to save any design units which have not been saved since they were edited.


If there are any errors, you can move to the next error message using the  button or the previous error using the  button. You can display the source graphics corresponding to the error by double-clicking on the error message (or by using the  button when the error is selected).

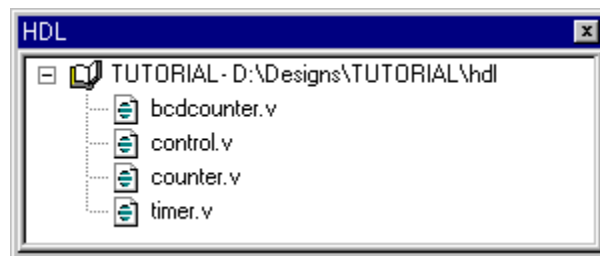
Alternatively, you can use the  button when the error is selected to view the generated HDL. If your text editor supports a line number argument, the HDL line corresponding to the error is selected automatically.

The embedded truth table view is generated as part of the *Timer* design unit. If any syntax errors are issued for the truth table, the corresponding cells are highlighted. For example, if HDL generation encounters an expression which is not defined in the diagram interface, the corresponding input cells are highlighted.



If there are no errors, you can use the toolbar buttons to view the Graphics or HDL corresponding to the “Generating...” message in the HDL Log window.

 You can cross-reference to the graphics or HDL from any message which is marked by the  icon in the log window.


Any files in the generated HDL directory can also be opened from the HDL browser. Click on the  icon for the *TUTORIAL* library in the HDL browser and notice that the view is expanded to reveal the generated files for the each design unit.



You can open any of these files by double-clicking on the generated file name or by choosing **Open** from the popup menu.


 You can also view the generated HDL files for the active diagram by using the  button in any editor window or when the diagram view is selected in the source browser.

## Edit the Timer Symbol

Use the  button (or choose the **Interface** option from the **Open** cascade of the **File** menu) in the *Timer* block diagram. A symbol editor window is opened which shows the external interface for the *Timer* design as a default symbol.

This symbol is used when the *Timer* design is instantiated as a component in a higher level design.

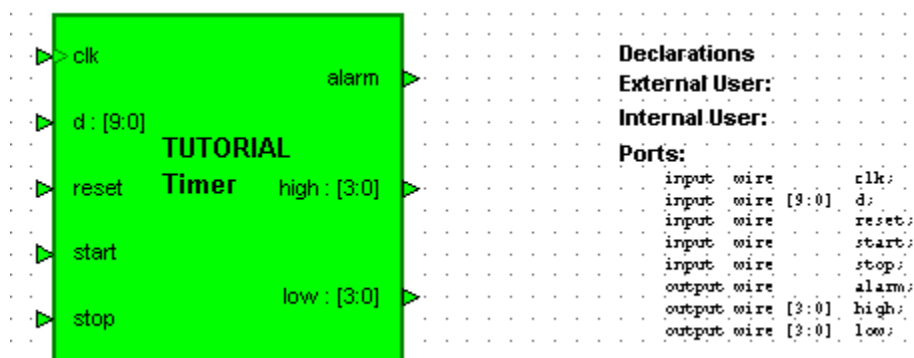
Ports representing the interface connections are shown in default locations but you can select and drag the ports to any location around the body of the symbol. You may also want to resize the symbol body.

 You can redistribute the ports evenly by selecting the symbol body and choosing **Equidistant Ports** from the popup menu.

The full port declarations are shown as a list in the symbol editor and are not shown on the individual ports. However, you can set preferences to control whether type and bounds information is displayed or set the properties visible for an individual port. See the “Changing the Visibility of Symbol Port Properties” help topic for more information.

Select the *clk* port and choose **On** from the **Clock** cascade to apply an edge triggered clock indicator to the port.

The edited symbol should look similar to the picture below:

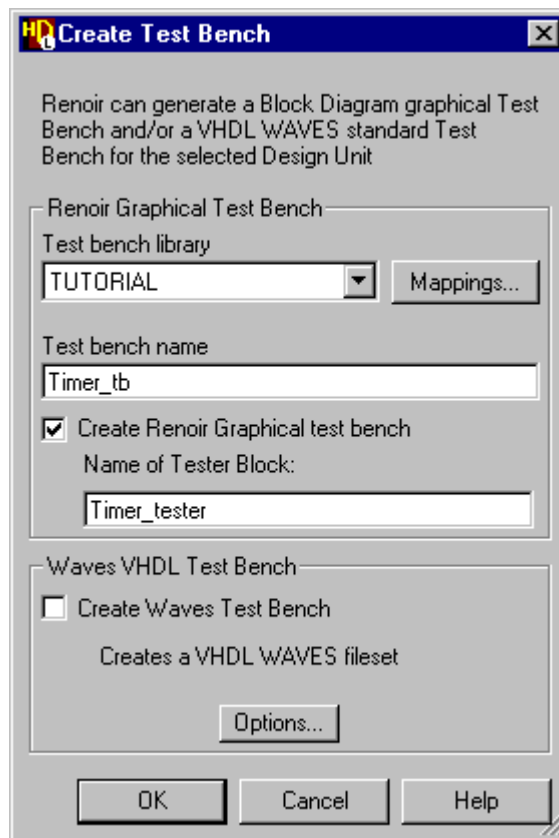


Close and save the symbol.


## Create a Test Bench

Select the *Timer* design unit in the design browser and choose **Create Test Bench** from the **New** cascade in the **File** menu to display the Create Test Bench dialog box. Notice that the dialog box defaults to the *TUTORIAL* library, with the test bench name derived by adding *\_tb* to the *Timer* design unit and the default tester block by adding *\_tester*.

Check that the **Create Graphical test bench** check box is set and confirm the dialog box by clicking the  button.



Notice that a *Timer\_tb* design unit is added to the source browser window.

Click on the  icon for the *Timer\_tb* design unit in the source browser to expand the view and reveal that a symbol and block diagram view (with the default name *struct.bd*) have been created. Double click on the *struct.bd* view to open the block diagram.

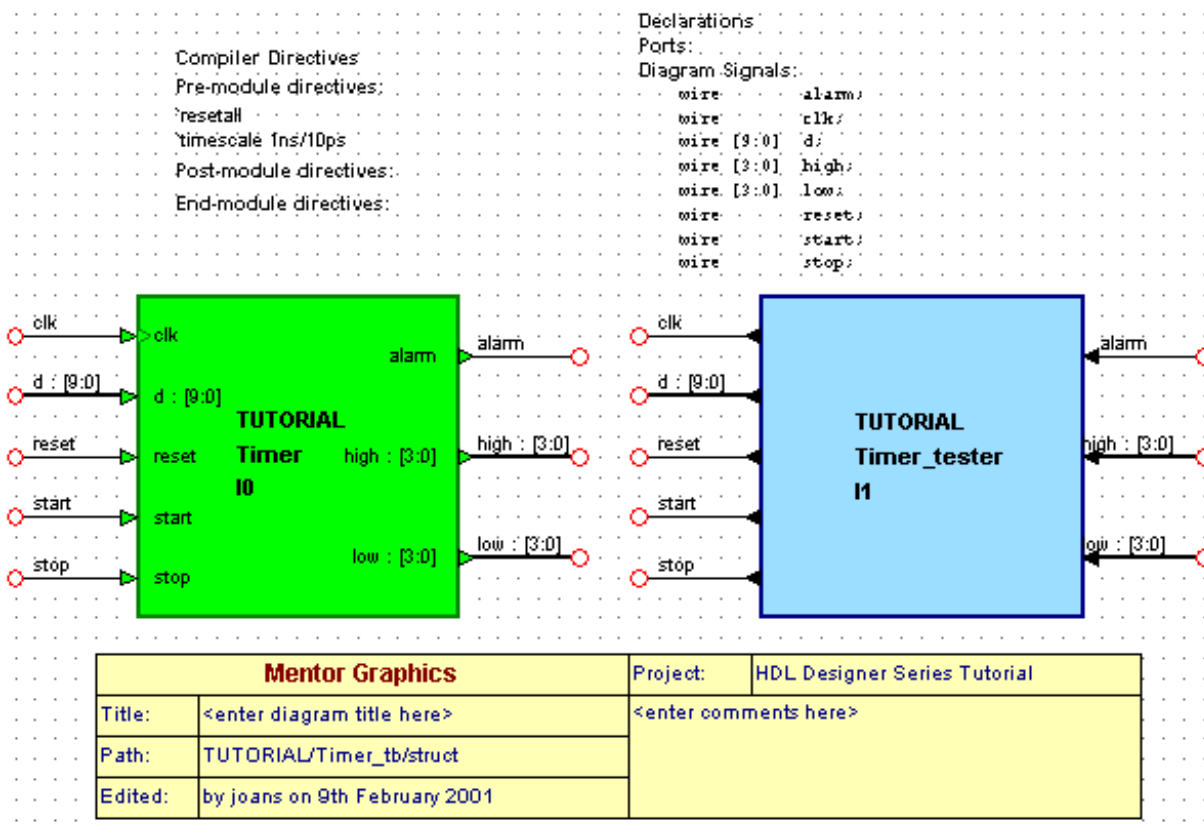
Notice that the *Timer* design has been instantiated as a component with stub signals connected to the input and output ports. The adjacent *Timer\_tester* block has corresponding output and input ports which are implicitly connected by name to the stub signals on the component.



The port locations are identical on the *Timer* component and *Timer\_tester* block but their directions are reversed. Hence the output ports are on the left and input ports on the right of the block.

All the signals and buses are declared as diagram signals since a test bench typically has no external ports.

The test bench block diagram should look similar to the following picture.




It is not necessary to explicitly connect signals and buses between each port on the component and the tester block as these are implicitly connected by name.



## Import the Tester Design Unit

The *Timer\_tester* block should provide test stimulus to the *Timer* design unit and monitor its output. This can be achieved by using a [flow chart](#) or HDL text view.

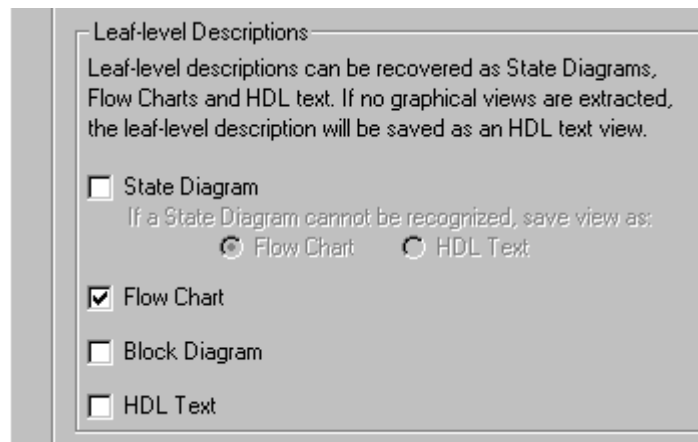
For, this tutorial, the *Timer\_tester* design unit can be imported from the *examples* installation subdirectory using a similar procedure to that used to “[Import the BCDCounter Design Unit](#)” on page 2-44. However, alternative procedures for [Creating a Verilog Flow Chart](#) are provided in [Appendix D](#).

Use the  button (or choose **Full HDL Import** from the **HDL Import** cascade of the **HDL** menu) in the design browser to display the full HDL Import wizard, choose **Specify HDL files** in the first page and select the *Timer\_tester.v* file in the Specify HDL Source Files page of the wizard.



The previous HDL import directory and files are saved as preferences. You should only need to remove the *Timer\_BCDCounter.v* file and add the *Timer\_tester.v* file.

Set the **Flow Chart** option in the Leaf-level Descriptions section of the Choose View Styles page.




This page is only displayed if you are using the graphical design tools and have selected the **Full HDL Import** option. If you are using the text design tools, the tester design unit is imported as a HDL text view.


Use the  button on the Confirm HDL Import page to import the flow chart.

# Instantiate the Imported Tester

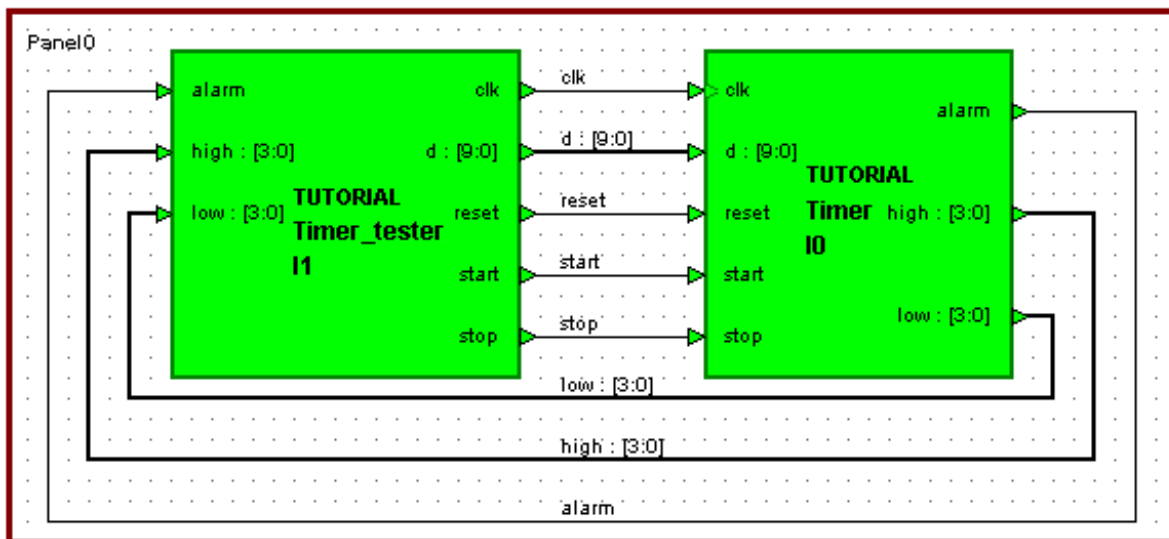
Re-open the *Timer\_tb* block diagram if it is not still open. Select the *Timer\_tester* block including all the connected signal stubs and delete them using the **Del** key.

Use the  key to display the Add Component dialog box and instantiate the imported *Timer\_tester* component in the block diagram to the left of the *Timer* instance. Notice that the *Timer\_tester* and *Timer* components have matching ports but they are located on opposite sides because they have the inverse direction.

Choose **Add Signal Stubs** from the popup menu and add stub signals to the *Timer\_tester* component. The matching nets on the two components are now implicitly connected by name. However, you may like to connect them explicitly by choosing **Autoconnect** from the **Autoroute** cascade of the **Diagram** menu.

Complete the test bench by editing the title and comments in the title block and using the  button to add a panel around the components. This panel can be used to set the diagram view when you simulate your test bench later in this tutorial.

The final test bench should look similar to the picture below:



<b>Mentor Graphics</b>		Project:	HDL Designer Series Tutorial
Title:	<enter diagram title here>	<enter comments here>	
Path:	TUTORIAL/Timer_tb/struct		
Edited:	by joans on 9th February 2001		

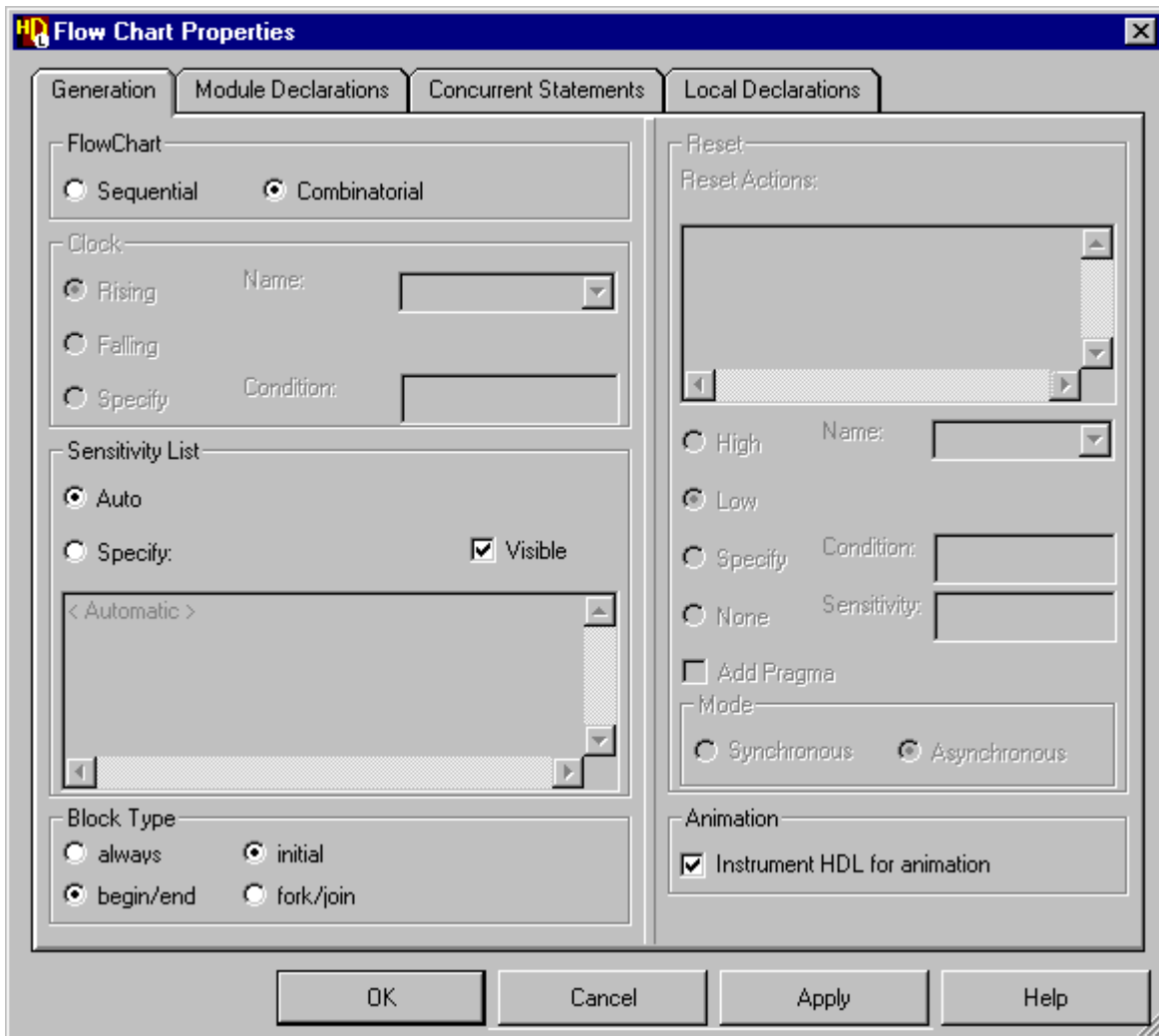
## Generate HDL for the Test Bench

If the *Timer\_tester* component is described by a flow chart, double-click on its instance in the test bench to open the flow chart view and choose **Flow Chart Properties** from the **Edit** menu to display the Flow Chart Properties dialog box.



If you have recovered the *Timer\_tester* component as a HDL text view, all properties for the view are defined in the HDL text and do not need to be set using a dialog box.

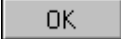
Choose the **Generation** tab and check that the **Combinatorial** option is set. Set the Block Type options to **initial** and **begin/end**. This flow chart represents a test bench and the HDL code must be generated using initial style code.


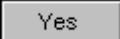


If a ModelSim or Verilog-XL simulator is available, set the **Instrument HDL for animation** option. This option adds additional code to the generated HDL which is used for flow chart animation later in this tutorial.

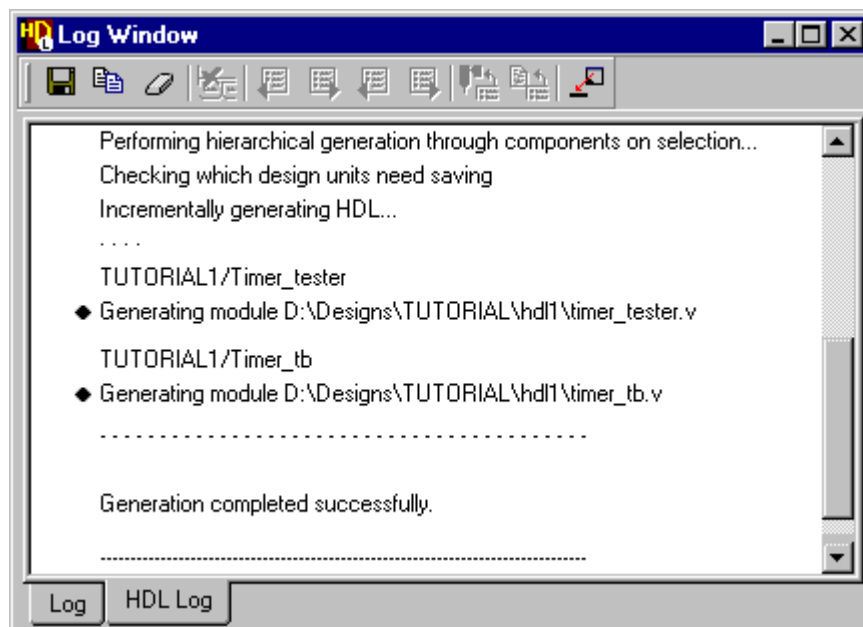


The additional animation code is surrounded by translation control pragmas which ensure that it is ignored by downstream synthesis tools.

Use the  button to confirm the dialog box.

Select the *Timer\_tb* design unit in the design browser and choose the  button (or choose the **Generate Through Components** option from the **HDL** menu). All views to be generated must have been saved and if you have not saved the test bench you are prompted whether to continue. Confirm the dialog box by clicking the  button.

Verilog is generated for the test bench and tester design units (but not the *Timer* design hierarchy since these design units have been generated earlier in this tutorial and do not need to be regenerated unless they have been modified).



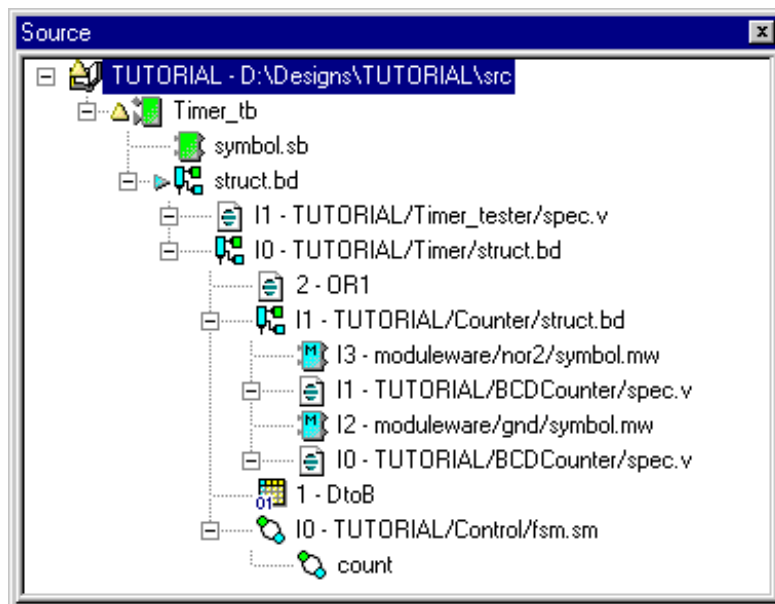
The progress of the HDL generation is monitored in a log window which includes any errors and warnings issued during generation. If there are any errors, you can display the corresponding graphics by double-clicking on the error message as described in the section [“Generate HDL for the Hierarchy” on page 2-62](#).

## Browse the Completed Design

View the hierarchy of the *Timer\_tb* design unit in the design browser by choosing **Expand All** from the design browser **View** menu or by using the  $\oplus$  icons to expand each design unit and display the full hierarchy which also includes all views of the *Timer* design.

Select the *Timer\_tb* design unit in the source browser and choose **Toggle Top Marker** from the **Edit** menu. Notice that the  $\blacktriangle$  icon is displayed adjacent to the design unit. Notice that the  $\blacktriangle$  icon is also shown against the *BCDCounter* and *Timer\_tester* design units. This is because they were automatically marked as top level design units when they were imported. Use **Toggle Top Marker** to unset the marker for these design units.

Select the *TUTORIAL* library and choose **Show Marked Design Units** from the **View** menu. Notice that all the unmarked design units are hidden in the design browser. However, all the design unit views are accessible from the *Timer\_tb* hierarchy.



You can restore the hidden design units by choosing **Show All Design Units** from the **View** menu.

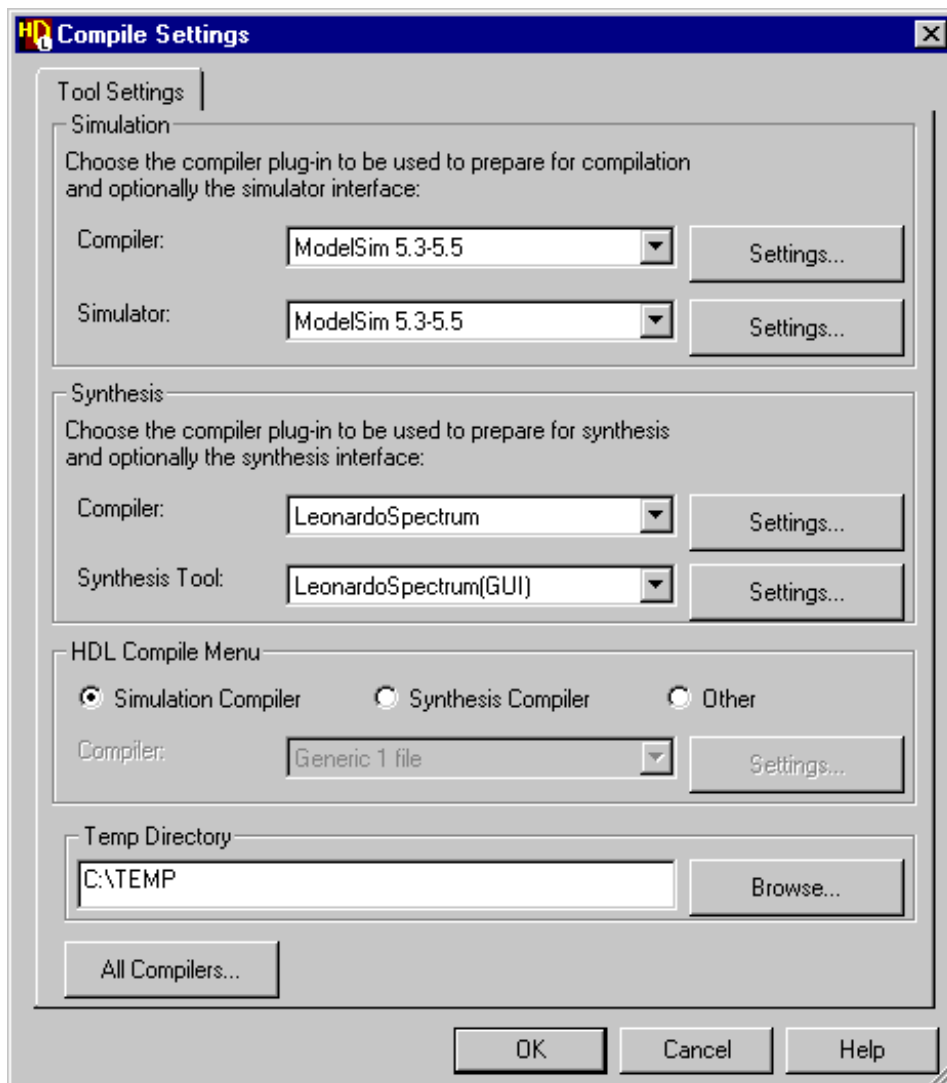
## Setup the Downstream Tools

For this tutorial, it is assumed that a ModelSim or Verilog-XL compiler is available. You can alternatively prepare data for use with other compatible downstream tools.



If you have installed the HDL Designer Series tool as part of FPGA Advantage, the ModelSim simulator and LeonardoSpectrum synthesis tools will already have been set up automatically.

Choose **Compile** from the **Options** menu to display the **Tool Settings** tab of the Compile Settings dialog box.

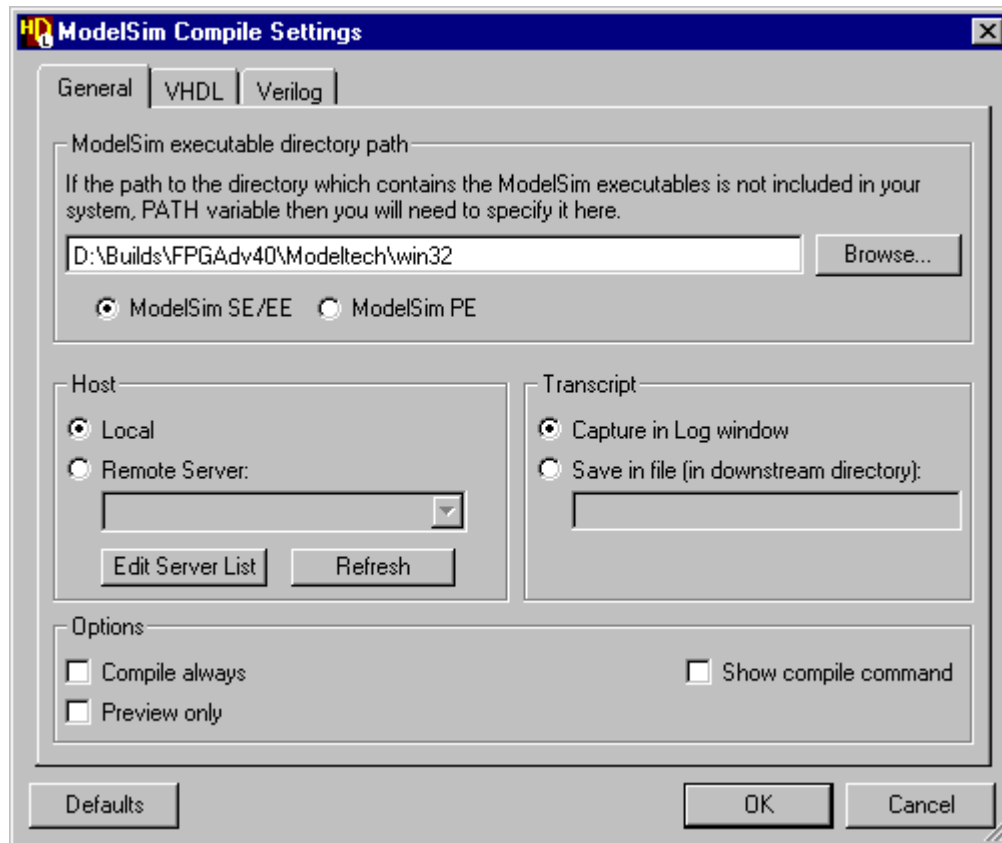


Choose a simulation compiler and simulator that is available on your workstation.



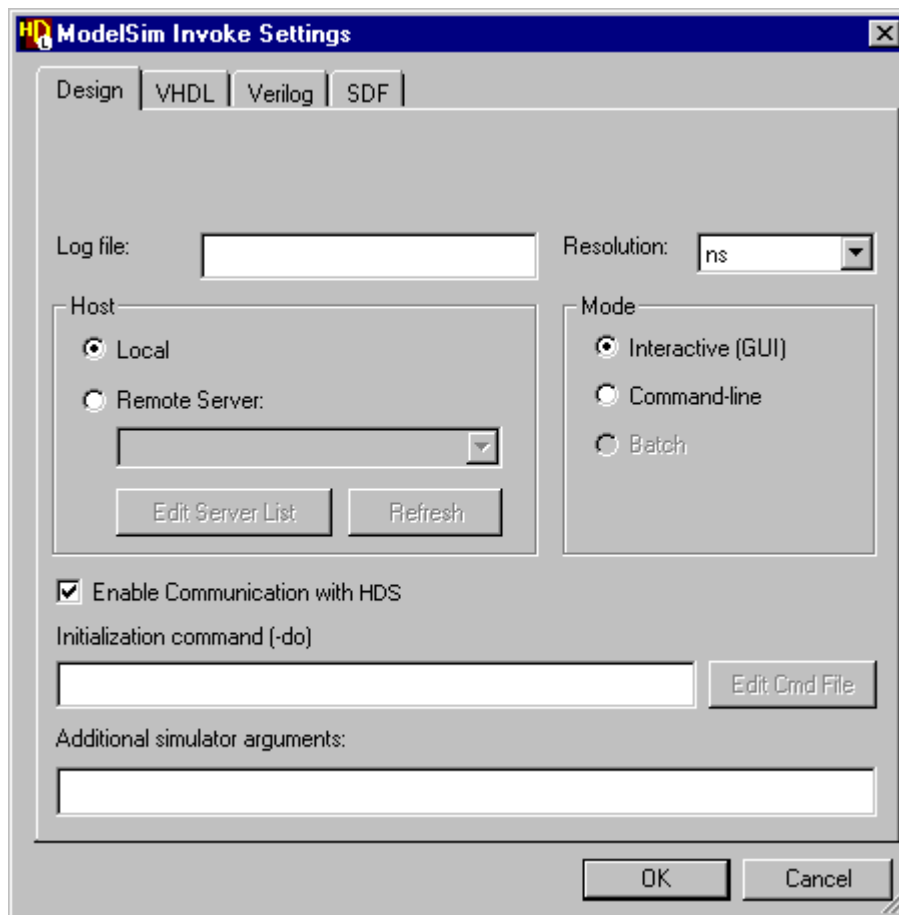
Select the ModelSim 5.3-5.5 compiler and ModelSim 5.3-5.5 simulator if version 5.3 or later of the ModelSim SE/EE or PE simulator is available.

Use the simulation compiler **Settings...** button to display the tool settings for your compiler. For example, the ModelSim Compile Settings dialog box:



Enter the pathname to the simulation compiler executable if it is not already accessible using your PATH environment variable. If you are using ModelSim, ensure that the correct edition (Special or Elite Edition: SE/EE; Personal Edition: PE) is selected. All other options can normally be left in their default state. Use the **OK** button to confirm and close the dialog box.

Use the Simulator  button to display the default invoke settings for your simulator. For example, the ModelSim Invoke Settings dialog box:



Accept the default resolution and check the **Enable Communication with HDS** and **Interactive** check boxes are selected. The other entry fields can be left in their default state. Use the  button to confirm the dialog box.

**i** Refer to [Using Verilog-XL](#) in Appendix B for information about setting up Verilog-XL or the online help topics for information about setting options for other supported downstream tools.




On both PC or UNIX systems, ensure that the temporary directory specified in the Compile Settings dialog box (which normally defaults to `/tmp` on UNIX or to `C:\TEMP` on PC systems) exists.

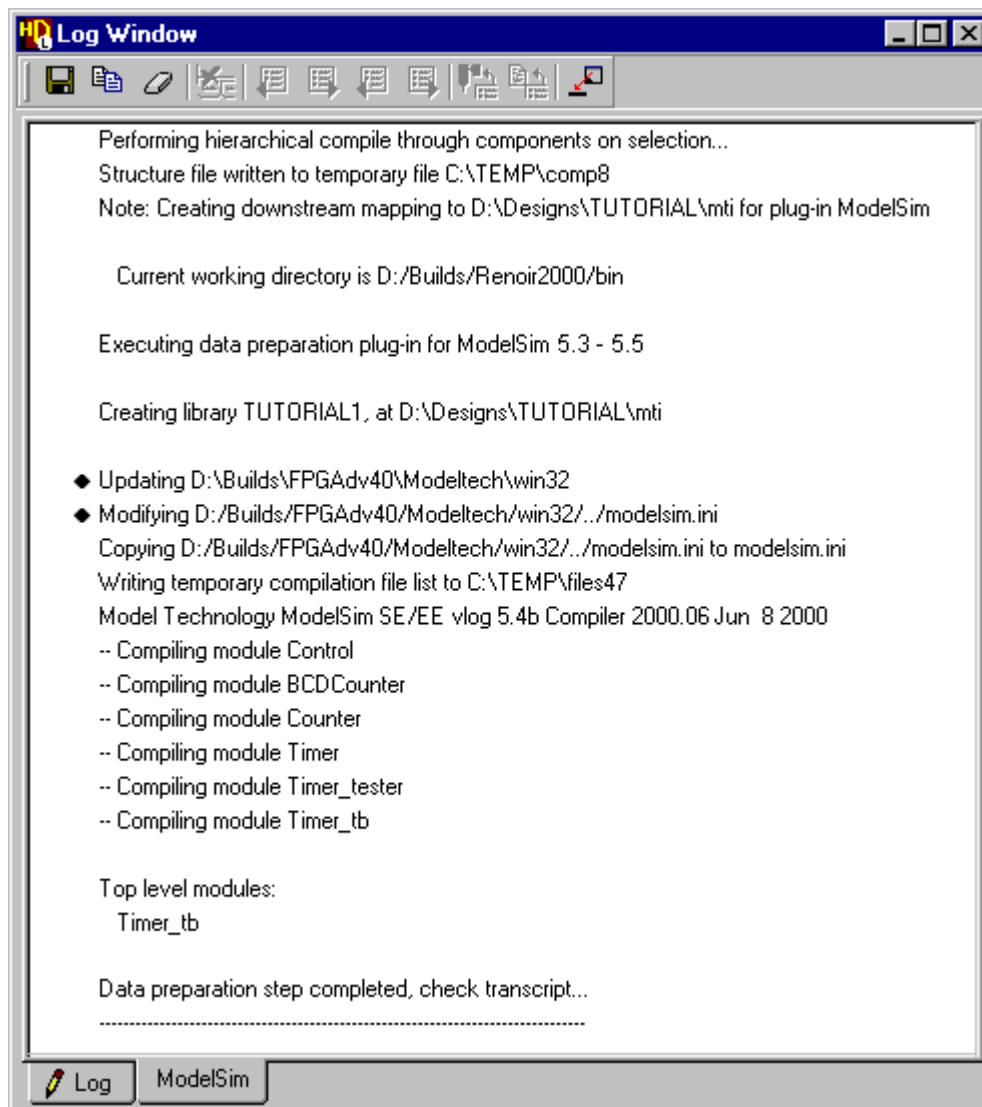
Use the  button to confirm the Compile Settings dialog box.




## Compile the Design




You do not need to explicitly compile your design if a Verilog-XL simulator is available on your workstation. Proceed instead directly to the [Invoke the Verilog-XL Simulator](#) procedure in Appendix B.

Select the *Timer\_tb* design unit in the design browser and choose the pulldown  on the  button. Select the  option from the palette to **Compile Through Components**. The progress of the compilation is shown in the HDL Log Window.





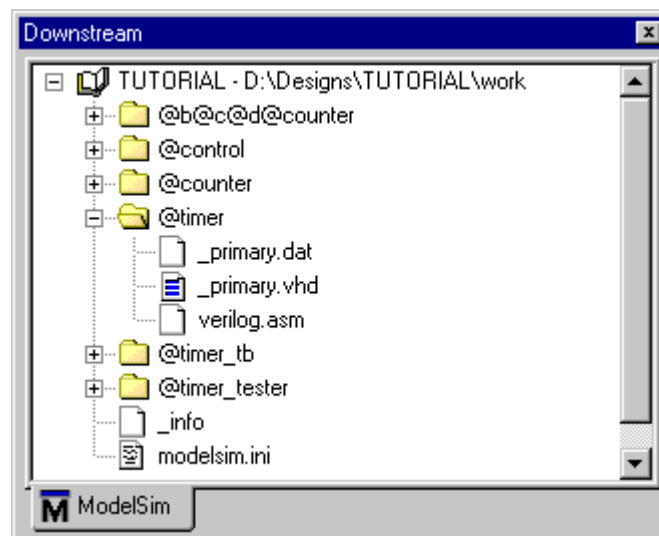
There should be no errors or warnings. If there are any errors, you can display the corresponding graphics by double-clicking on the error message as described in the section “[Generate HDL for the Hierarchy](#)” on page 2-62.


After correcting any errors in the source diagrams, you can regenerate, recompile and start the simulator in a single step by choosing the  flow button to generate and compile all changed design unit views in the hierarchy beneath the active design unit. Notice that unmodified views are not re-generated unless you choose **Set Generate Always** from the HDL menu to force HDL generation.

 The flow buttons  and  are setup by default for a simulation flow (using ModelSim) and a synthesis flow (using LeonardoSpectrum). Refer to the “[Setting a Custom Flow](#)” online help topic if you want to modify the setup for either of the design flow buttons to use alternative tools.

Notice that downstream mapping is automatically created for your downstream data and the compiled objects are displayed in the [downstream browser](#).

Click on the  icon for the *TUTORIAL* library in the downstream browser and notice that the view is expanded to reveal a subdirectory for each design unit in your design. You can use the  icons to browse each of these folders.





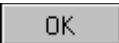
 Uppercase characters in Verilog design unit names are preserved by the ModelSim compiler.

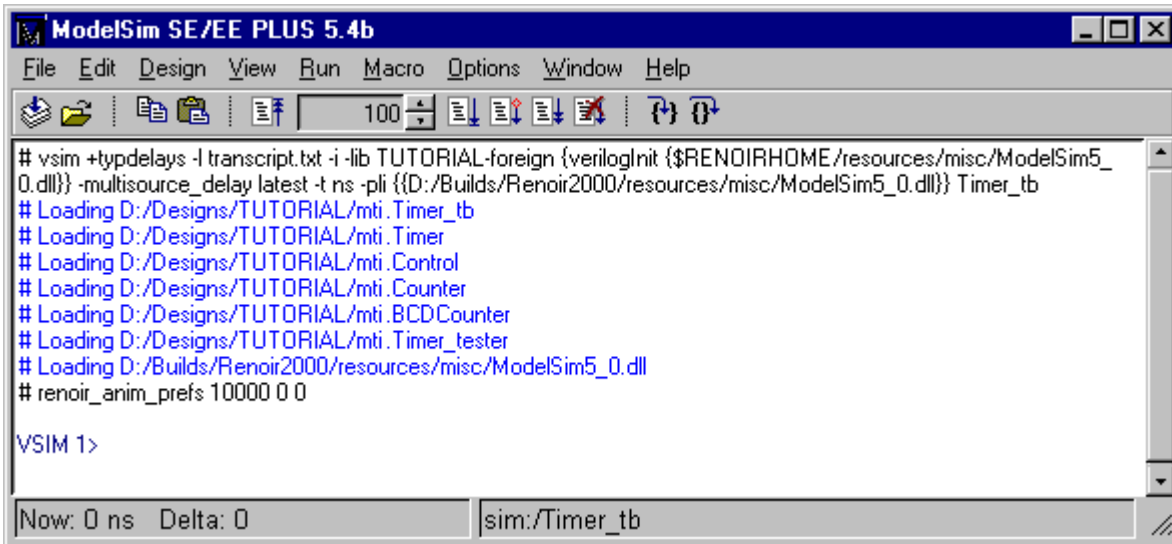
## Invoke the ModelSim Simulator

The simulator can be invoked from the design browser or any block diagram, flow chart or state diagram in your design. However, it must be invoked at a level that includes the entire branch of your design that you want to test. This tutorial uses the test bench to verify your version of the *Timer* design.



You must have generated HDL (and compiled HDL if you are using a ModelSim simulator) for all design units in the hierarchy you want to simulate. If you did not set **Instrument HDL for animation** in the state machine and flow chart properties, refer to the topics “[Set Generation Properties](#)” on page 2-39 and “[Generate HDL for the Test Bench](#)” on page 2-69. If necessary, you can regenerate and recompile your design by choosing the  button to generate and compile all changed design unit views in the hierarchy beneath the active design unit. By default, this button will also start (or restart) the simulator if the generation and compilation are completed successfully.

Select the *Timer\_tb* design unit in the design browser and use the  button (or choose **Start Simulator** from the **HDL** menu) to display the Start dialog box for your simulator. Use the  button to confirm the invoke options.



```
# vsim +typdelays -l transcript.txt -i -lib TUTORIAL-foreign {verilognit {$RENOIRHOME/resources/misc/ModelSim5_0.dll}} -multisource_delay latest -t ns -pli {{D:/Builds/Renoir2000/resources/misc/ModelSim5_0.dll}} Timer_tb
# Loading D:/Designs/TUTORIAL/mti.Timer_tb
# Loading D:/Designs/TUTORIAL/mti.Timer
# Loading D:/Designs/TUTORIAL/mti.Control
# Loading D:/Designs/TUTORIAL/mti.Counter
# Loading D:/Designs/TUTORIAL/mti.BCDCounter
# Loading D:/Designs/TUTORIAL/mti.Timer_tester
# Loading D:/Builds/Renoir2000/resources/misc/ModelSim5_0.dll
# renoir_anim_prefs 10000 0 0

VSIM 1>
```

The simulator is invoked and issues a number of loading messages ending with:  
# hds\_anim\_prefs 10000 0 0

## Setup the Simulator Windows


An additional Simulation toolbar is available in the block diagram, flow chart and state diagram when a simulator is invoked to support cross-probing between the simulator and source design objects in the HDL Designer Series tool. This toolbar is normally displayed automatically at the bottom of the diagram window but can be undocked, moved, docked or hidden in the same way as the other toolbars.




Display the *Timer\_tb* block diagram and choose **View Panel** from the **View** menu to display the panel you added earlier in this tutorial.



If you added more than one panel, a dialog box is displayed for you to choose the panel to display.

Select the signals *alarm*, *clk*, *d*, *reset*, *start*, *stop*, *high* and *low*. Use the  button from the Simulation toolbar or choose **Add Wave** from the **Display** cascade in the **Simulation** menu and notice that the simulator Wave window is automatically opened with these signals loaded.

Use the  button from the Simulation toolbar or choose **Add List** from the **Display** cascade in the **Simulation** menu to add the selected signals to the simulator list window.




You can optionally add signals to the simulation log without displaying them in the Wave or List window by choosing **Add Log** from the **Display** cascade of the **Simulation** menu.

You can also open any other simulator window by choosing from the **View** cascade of the **Simulation** menu. For example you may wish to use this menu to open the ModelSim **Source** and **Structure** windows.

## Enable Animation

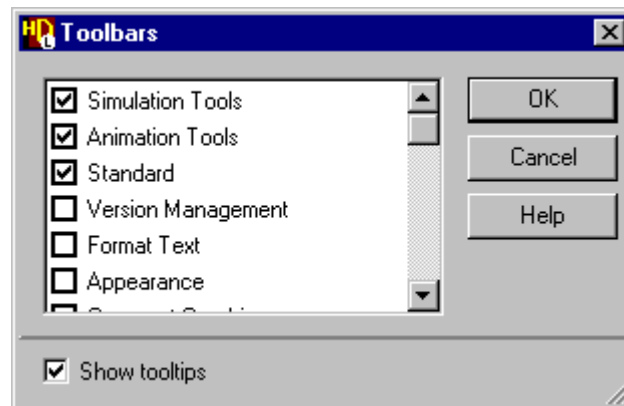
Display the *Control* state diagram and its child hierarchical state diagram. Use the **View Panel** command in each window to adjust the window view to the panel areas you defined earlier in this tutorial.

-  You can optionally also display the *Timer\_tester* flow chart. Flow charts (including hierarchical and concurrent flow charts) can be animated in a similar way to that described below for the state diagrams.


Notice that an additional Animation toolbar is available in the state diagram (and flow chart) windows when the simulator is invoked. This toolbar is normally docked at the bottom of the diagram window next to the Simulation toolbar but can be moved independently.




The editing toolbars are not usually required during simulation and animation. You can hide or show toolbars using the Toolbars dialog box which is displayed by choosing **Settings** from the **Toolbars** cascade of the **View** menu.




For example, use the dialog box to hide the Format Text, Appearance, Comment Graphics and Arrange Object toolbars.


-  Individual toolbars can also be hidden by unsetting the corresponding options in the **Toolbars** cascade of the **View** menu.

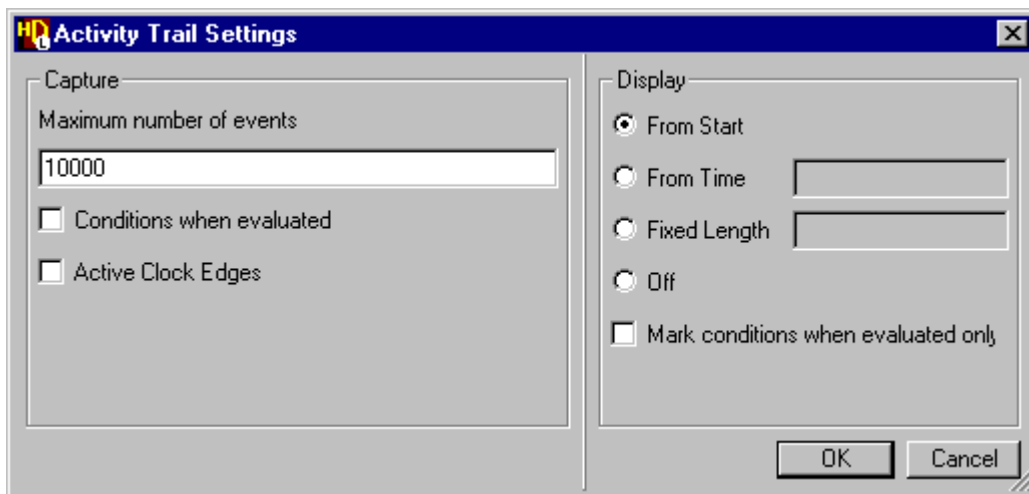
Use the  buttons from the Animation toolbars (or choose **Data Capture** from the **Animation** menus) in each state diagram or flow chart you want to animate. (It is not necessary to repeat this command for each hierarchical or concurrent view since these are considered part of the same diagram.)

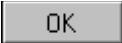
Notice that all objects (except the start state, exit and entry points in the state diagram are redrawn with white fill. The start state is drawn with red fill to indicate that it is the current animation state.



This animation view is automatically displayed when you set data capture but can be toggled at any time by using the  button (or by toggling the **Show Animation** option in the **Animation** menu). You can also set data capture for all instances in the current simulation hierarchy by choosing **Global Capture On** from the **Animation** menu.

Use the  button from the Animation toolbar (or choose **Activity Trails** from the **Animation** menu) in the flow chart or state diagram window and choose the **From Start** option in the Activity Trail Settings dialog box.






Use the  button to confirm the Activity Trail Settings dialog box.





The activity trail is applied to all windows in the current simulation and need only be set once in any flow chart or state diagram window.

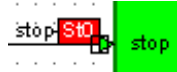
## Simulate the Design


Run the simulator for the default timestep (100 nano-seconds) by using the  button or by choosing **For Time** from the **Run** cascade of the **Simulation** menu in the block diagram, flow chart or state diagram window.


 The default timestep can be changed by using the  button to select from a list of alternative timesteps or choose a user specified timestep.

Notice that the *Initialization* action box is highlighted red in the animated flow chart for the *Timer\_tester* and the initial signal values are updated in the simulator Wave and List windows.


Select the *stop* signal in the *Timer\_tb* block diagram and set a breakpoint by using the  button or by choosing **Add** from the **Breakpoints** cascade of the **Simulation** menu. Set a simulation probe on the same signal by using the  button or by choosing **Add** from the **Probes** cascade of the **Simulation** menu. Notice that the signal value shown in the probe is '0'.




Run the simulator until there are no more events scheduled by using the  button from the Simulation toolbar or by choosing **Forever** from the **Run** cascade of the **Simulation** menu.

 Run commands can be issued from the toolbars or menus in any block diagram, flow chart or state diagram window in the current simulation environment.



The simulation should run until the breakpoint is encountered when the *stop* signal value (shown by the simulation probe) changes to 1. Notice that the *Store* action box is highlighted in red as the current step in the animated flow chart and the *counting* state is entered in the child animated state machine for the *count* state.

Use the  button or choose **Continue** from the **Run** cascade of the **Simulation** menu to continue the simulation until the *stop* signal changes back to zero. Notice that the *suspended* state is entered in the animated state machine and the following note is issued in the main simulation window:

```
# Count suspended correctly
```

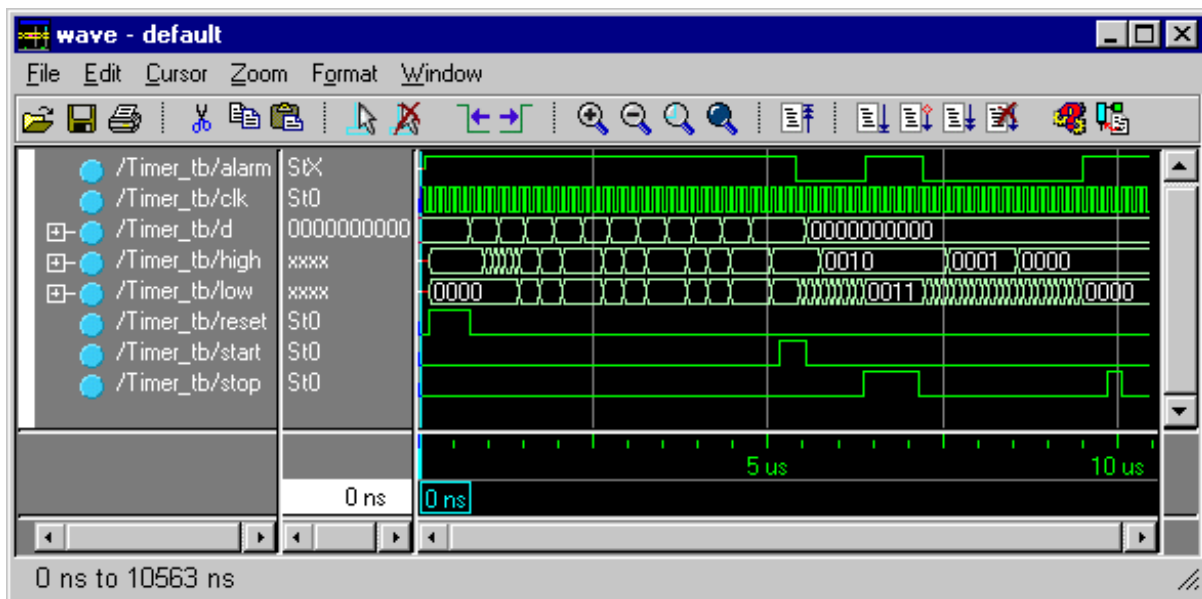
Run the simulator to the next change of the *stop* signal by using the  button and notice the message:

```
# Alarm asserted correctly
```

Use the  button in the test bench block diagram to delete the breakpoint on the *stop* signal and complete the simulation by using the  button. The simulation should run to completion and issue the message:


```
# Timer test completed
```


Examine the results displayed in the Wave window and ensure that the simulation has performed correctly by comparison with the example waveforms below:




You can display the full simulation waveform by choosing **Zoom Full** from the Wave window **Zoom** menu.



If any Verilog errors are discovered, correct your design and use the  button to regenerate and recompile the modified diagram. If the diagram is successfully generated and compiled, the flow button automatically restarts the simulation for the current simulation hierarchy.


Alternatively, you can use the  button or choose **Restart Simulator** from the Simulation menu to repeat the simulation.








You can use the  button or choose **Highlight Object** from the **Display** cascade of the simulation menu to highlight all occurrences in the simulation windows of a signal corresponding to any selected graphical object.




## Review the Animation

Notice how the animation activity trail in both the flow chart and state diagram is highlighted in blue and the exit break point for `Timer test completed` is indexed on the flow chart Verilog module displayed in the simulator Source window.

Use the  button or choose **Link Diagrams** from the **Animation** menu to link the animated diagrams. When the diagrams are linked, all animation review commands are applied to all animated diagrams in the simulation hierarchy.


You can review the animation by using the , , ,  or  buttons (or by choosing **Goto Next**, **Goto Previous**, **Goto Time**, **Goto Start** or **Goto Latest** from the **Animation** menu).


The  and  buttons step through each object in the flow chart.

However in the state diagram, you can set options in the **Animation** menu (or from a pulldown menu on the toolbar button) to move by change of state , simulation event  or change of clock edge . Notice how the toolbar button icon changes to indicate the current mode of movement.


You may want to change the activity trail to display the trail from a specified time or specify a fixed length. The current simulation step (or current state and last transition taken in an animated state diagram) is always shown in red and the previous step (or previous state and transition) in yellow. All other visited objects included in the activity trail are shown in blue. Remember that you can open down the hierarchical action box or hierarchical state to view the animation in the child diagrams.

Compare the animation with the results displayed in the Waveform window and ensure that the simulation has performed correctly by comparison with the example waveforms on the previous page.

A  button is available in the Animation toolbar and a **Cause** option from the **Animation** menu. These commands can be used to move the ModelSim Wave and List windows to the current animation time.

A corresponding  button and **Cause** option (in the **Cursor** menu) are available in the ModelSim Wave window. These commands can be used to update all currently open animation windows to the simulation step or event immediately preceding the time marked by the Wave window cursor.

A **Cause** command is also added to the **Markers** menu in the simulator List window which can be used to update all open animation windows to the simulation step or event corresponding to the selected line in the List window.

A  button is added to the toolbar (and a **Show HDS Source** option is added to the **Edit** menu) in the Source window and can be used to open the graphic window corresponding to the line of HDL code under the cursor.


See the HDL Designer Series help topics “Cross Probing from ModelSim” and “Using the ModelSim Source Window” for more information about the cross-probing commands added to ModelSim when it is used with a HDL Designer Series tool.

Exit the simulator (by choosing **Quit** from the **File** menu in the main ModelSim window). Any other simulator windows are automatically closed.

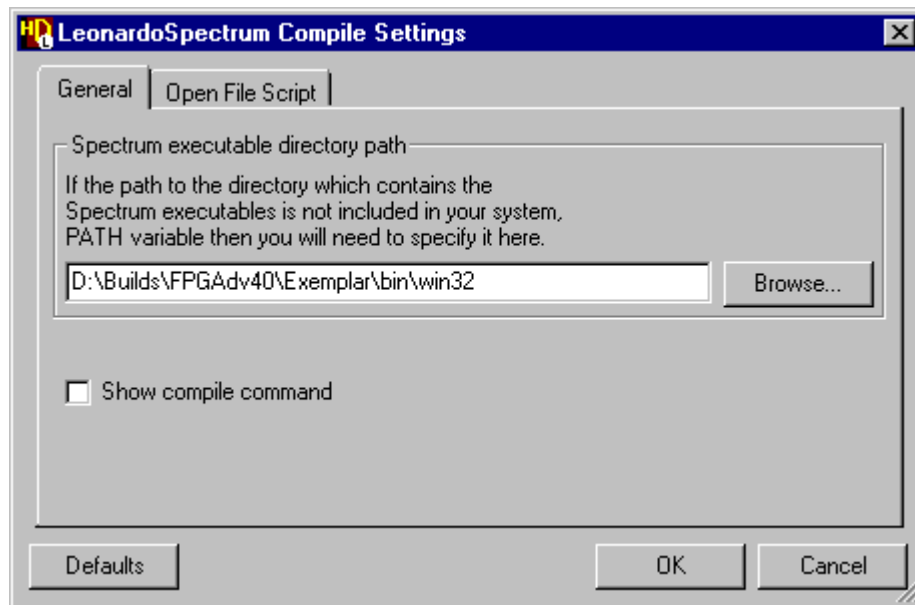
Close all graphic editor windows (except the design browser) by choosing **Close All Windows** from the **File** menu in the design browser.

## Setup the Synthesis Tool


If a synthesis tool is available on your workstation you can synthesize your completed design. The Exemplar LeonardoSpectrum or Synopsys Design Analyzer tools can be invoked directly from within a HDL Designer Series tool.

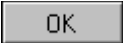
The synthesis tool can be setup using the synthesis compiler  button in the **Tool Settings** tab of the Compile Settings dialog box in a similar way to the simulation compiler which was described in “[Setup the Downstream Tools](#)” on [page 2-72](#).

For example, the following picture shows the LeonardoSpectrum Compile Settings dialog box.

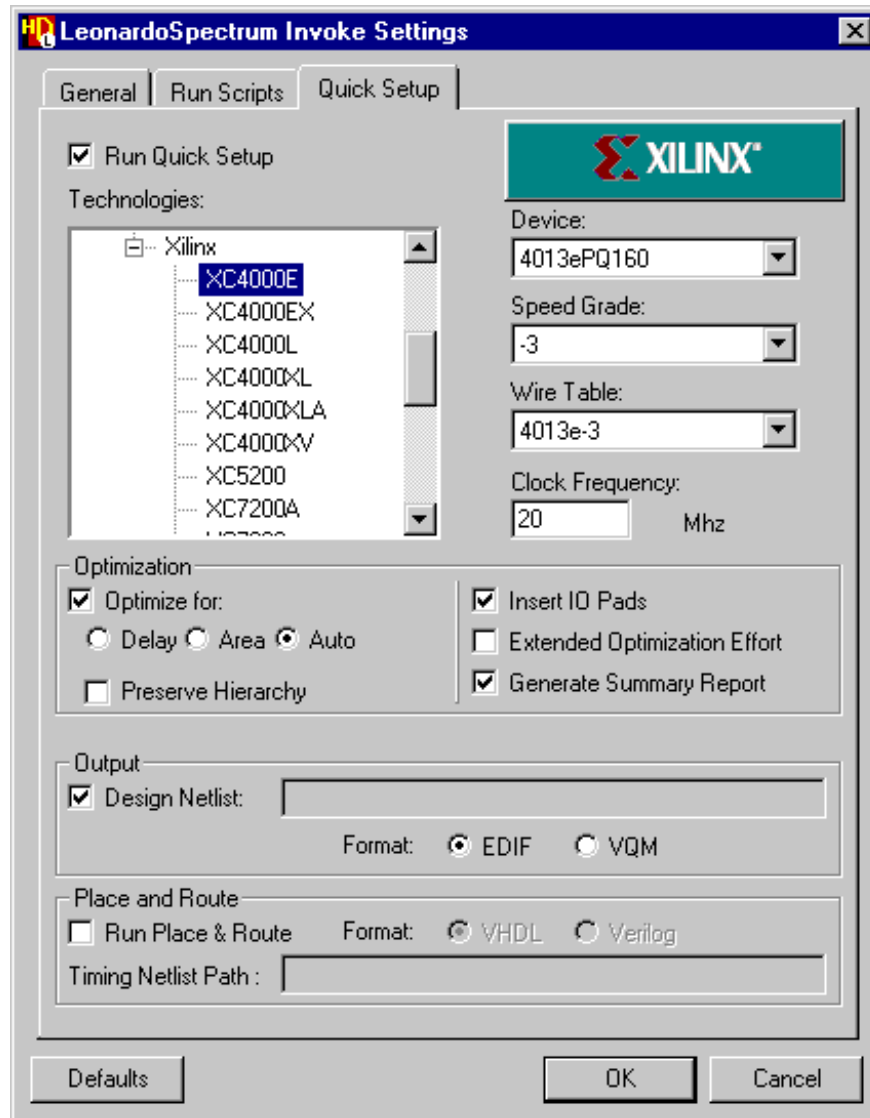


Enter the pathname to the synthesis compiler executable if it is not already accessible using your PATH environment variable.

 If you have installed the HDL Designer Series tool as part of FPGA Advantage, the ModelSim simulator and LeonardoSpectrum synthesis tools will already have been set up automatically.

The options in the **Open File Script** tab can be left with their default values. Use the  button to confirm and close the dialog box.


Use the synthesis tool  button to display the default invoke settings for your tool. If you are using LeonardoSpectrum 2000 (or later), the **Quick Setup** tab is displayed:

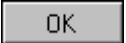


Use the  icons to expand the FPGA/CPLD technologies list and select the technology of your choice. For example, Xilinx XC4000E as shown above.





If you have an Exemplar level 3 license, ASIC and FPGA technology libraries are available. If you have a level 2 license, only the FPGA libraries are available.

The Device, Speed Grade and Wire Table fields are automatically selected when you have chosen a technology. Complete the dialog box by entering a clock frequency (for example 20 MHz) and use the  button to confirm the invoke settings dialog box. All other options can be left with their default values.

Use the  button to confirm and close the Compile Settings dialog box.

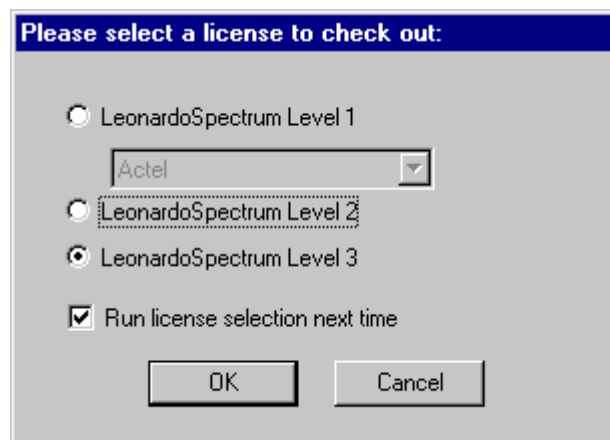
## Run the Synthesis Flow


Select the *Timer* design unit in the source browser and use the  button to invoke the synthesis flow.

-  You cannot synthesize the test bench or flow chart which contain unsynthesizable HDL. Ensure that you have selected the *Timer* design unit in the source browser.

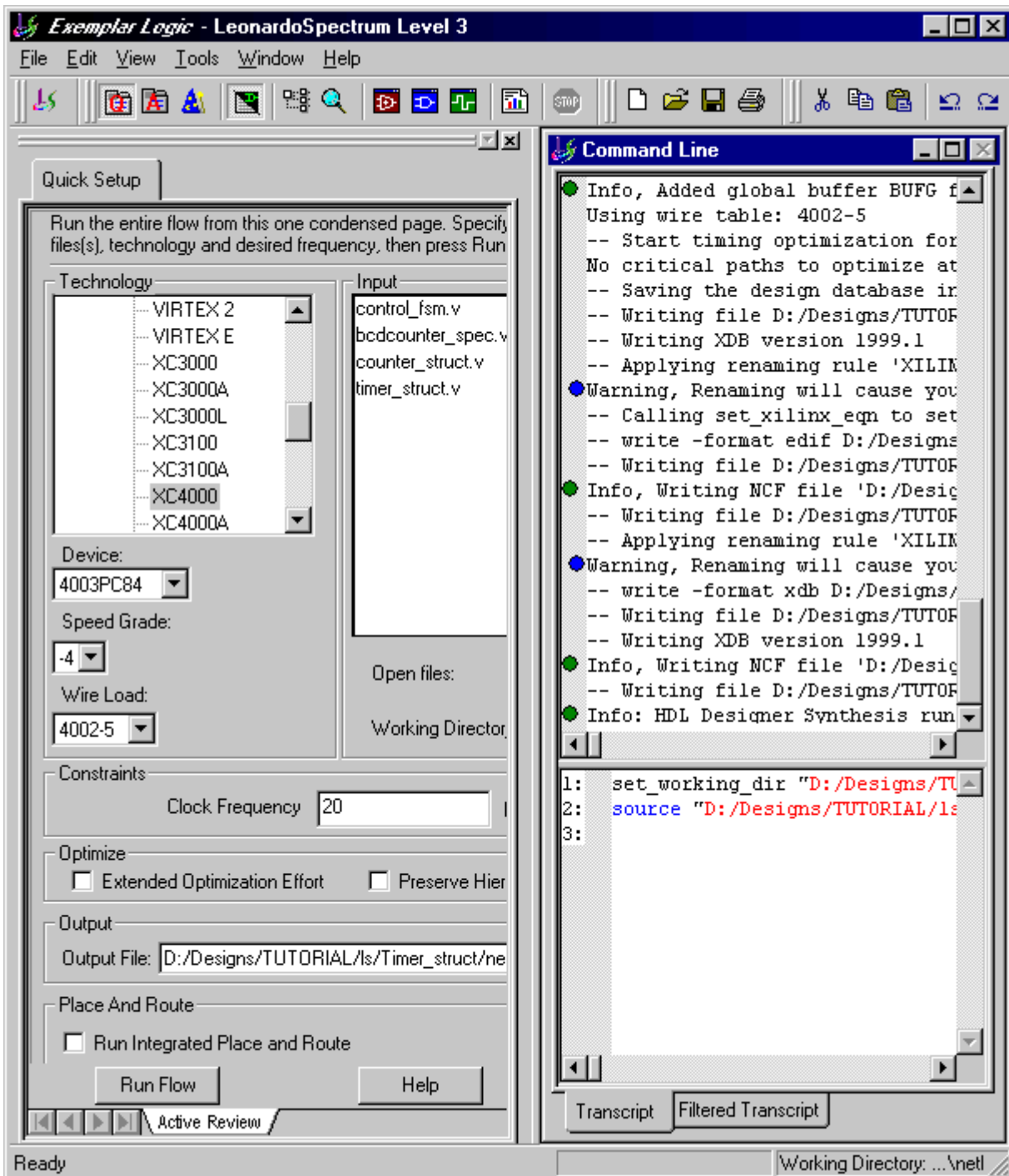
The design will be regenerated if necessary and compiled for synthesis. The Invoke Settings dialog box is displayed for you to confirm or modify the synthesis tool settings.



When you confirm the invoke settings, LeonardoSpectrum is invoked and you are prompted to select an Exemplar level 1, 2 or 3 license.

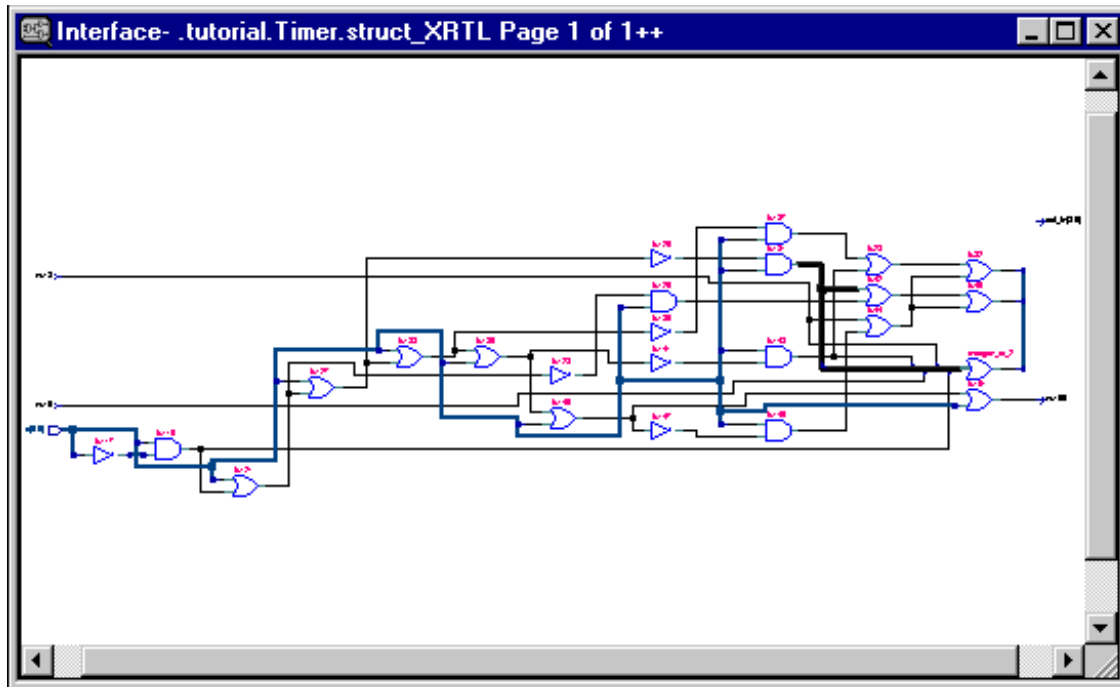


-  Higher level licenses enable more features but you can choose to run at a lower level using a higher level license.

When you confirm the license, your design is synthesized and optimized for the selected technology. Status messages are transcribed in the Command window ending with the message “HDL Designer Synthesis run finished”:



You can use the  button from the LeonardoSpectrum toolbar to display the RTL schematic or the  button to display the technology schematic for your design. For example, the following picture shows the RTL schematic:



You can move around the schematic using the scroll bars or the diagram can be enlarged or maximized in the schematic window by choosing the **Zoom In** or **Zoom Fit** options from the **Zoom** cascade of the popup menu.

You can crossprobe to the corresponding source design object by selecting an object in the schematic window and choosing **Trace to HDL Designer** from the popup menu.

Exit from LeonardoSpectrum by choosing **Exit** from the LeonardoSpectrum **File** menu and respond  when you are prompted whether to save your project settings.

You have now completed this tutorial. If you have not successfully verified your design, a completed reference example is available in the examples subdirectory of the HDL Designer Series installation. This example can be accessed by opening the *TIMER\_vlog* library in the design browser.

## Using the Example Verilog Design

A completed example design can be accessed from the *TIMER\_Vlog* library in the design browser. The example design includes generated HDL but only dummy library mapping for downstream data. To use the example design, you should change the location of the generated and downstream data directories to locations for which you have write permissions.





Windows users typically have write access to the installation directories. However, it is recommended that you change the mapping to a suitable directory for user data rather than overwrite the installed examples.

Once you have modified the library mapping (see [“Set Library Mapping” on page 2-2](#)), you can browse, compile and simulate the example design by following the procedures from [“Browse the Completed Design” on page 2-71](#).

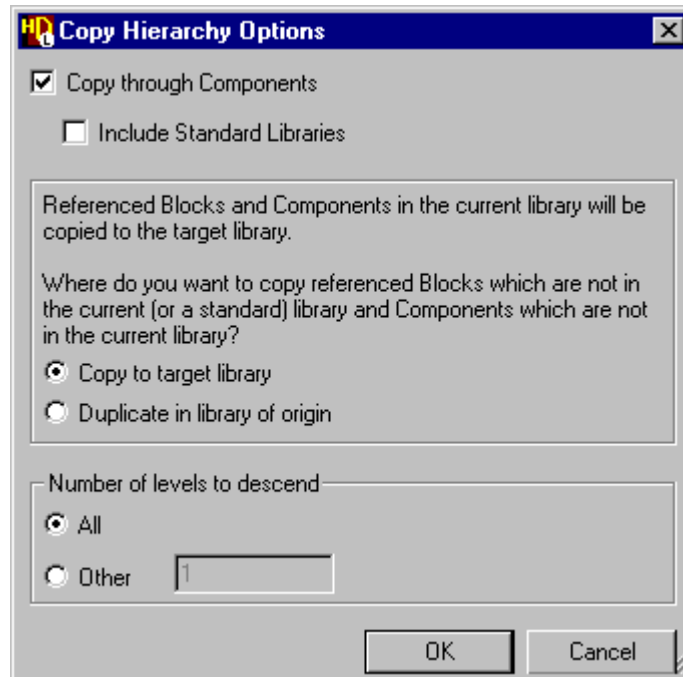
In order to animate the flow chart and state machine in the example design, you must regenerate HDL through components after setting **Instrument HDL for animation** in the state machine and flow chart properties. (Refer to the topics [“Set Generation Properties” on page 2-39](#) and [“Generate HDL for the Test Bench” on page 2-69](#).)

Alternatively, you can easily create your own copy of the example design by using the following procedure:

1. Set library mapping (including source, generated HDL and downstream directories to which you have write permissions) for a new empty library.
2. Use the  button to open the new library in the source browser.
3. Select the *Timer\_tb* design unit in the *TIMER\_Vlog* library and drag it over the new library name with the  mouse button.
4. Choose **Hierarchical Copy Here** from the popup menu which is displayed when you release the mouse button to display the Copy Hierarchy Options dialog box.
5. Select **Copy Through Components, Copy to target library** and **All** levels in the dialog box.



A complete copy of the origin library will be made in the target library and can be browsed, edited, generated, compiled, simulated and animated without any impact on the original examples.





---

# Appendix A

## Using Text Design Tools

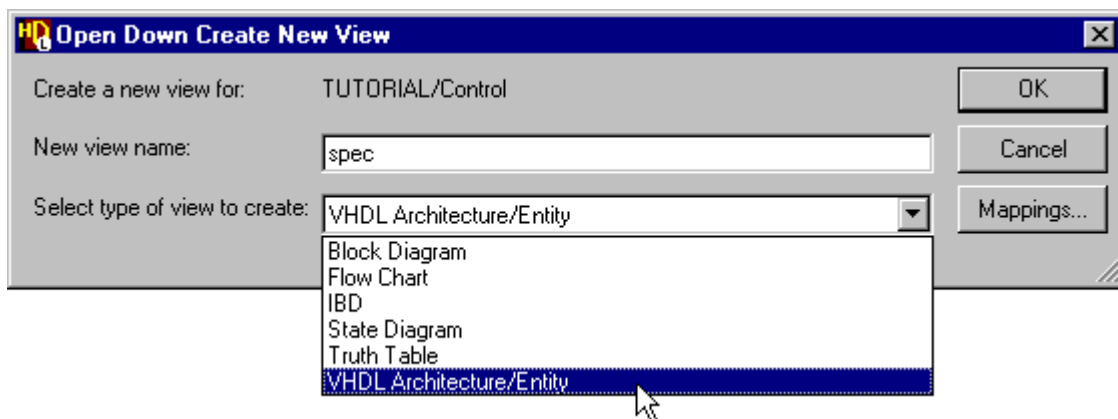
### Introduction

The HDL Designer Series text design tools include the core facilities for creating and editing block diagrams and HDL text views but do not include the flow chart, state machine or truth table editors.

This appendix describes alternative VHDL and Verilog text views which can be used instead of the state machines, truth tables and flow charts described in the main tutorial procedures.

### Create HDL Text for the Control Block

Move the cursor over the *body* of the *Control* block on the *Timer* block diagram, then press and release the **Right** mouse button to select the block and display the popup menu. Choose the **Open** cascade menu option **New View**. The Open Down Create New View dialog box is displayed:



Select **VHDL Architecture/Entity** (if you are using VHDL) or **Verilog Module** (if you are using Verilog) in the Open Down Create New View dialog box using the new view name *spec*.

A template **HDL text** file is opened using the default text editor. Notice that the file contains a default header enclosed by the pragmas `hds header_start` and `hds header_end`. Take care not to modify the text between these pragmas but enter the VHDL or Verilog shown in the following sections after the pragma `hds interface_end`.

If you are using VHDL, modify the architecture as follows:

```
-- hds interface_end
ARCHITECTURE spec OF Control IS
  -- Architecture Declarations
  Constant ZEROS : std_logic_vector := "0000000000";
  TYPE STATE_TYPE IS (flush, getkey, load_t, load_u,
    standby, alarm, counting, suspended, end_count);

  -- State vector declaration
  ATTRIBUTE state_vector : string;
  ATTRIBUTE state_vector OF spec : ARCHITECTURE IS
    "current_state" ;

  -- Declare current and next state signals
  SIGNAL current_state : STATE_TYPE ;
  SIGNAL next_state : STATE_TYPE ;
  -- Declare any pre-registered internal signals
  SIGNAL beep_int : std_logic ;

BEGIN
  clocked : PROCESS(clk, reset)
  BEGIN
    IF (reset = '1') THEN
      current_state <= flush;
      -- Reset Values
      beep <= '0';
    ELSIF (clk'EVENT AND clk = '0') THEN
      current_state <= next_state;
      -- Registered output assignments
      beep <= beep_int;
    END IF;
  END PROCESS clocked;
```

```
nextstate : PROCESS (current_state, d, start, stop, zero)
BEGIN
    CASE current_state IS
    WHEN flush =>
        IF (d/=ZEROS) THEN
            next_state <= load_u;
        ELSE
            next_state <= flush;
        END IF;
    WHEN getkey =>
        IF (start='1') THEN
            next_state <= standby;
        ELSIF (d/=ZEROS) THEN
            next_state <= load_u;
        ELSIF (stop='1') THEN
            next_state <= flush;
        ELSE
            next_state <= getkey;
        END IF;
    WHEN load_t =>
        IF (d/=ZEROS) THEN
            next_state <= getkey;
        ELSE
            next_state <= load_t;
        END IF;
    WHEN load_u =>
        next_state <= load_t;
    WHEN standby =>
        IF (zero='1') THEN
            next_state <= alarm;
        ELSIF (start='1') THEN
            next_state <= counting;
        ELSE
            next_state <= standby;
        END IF;
    WHEN alarm =>
        IF (stop='1') THEN
            next_state <= end_count;
        ELSE
            next_state <= alarm;
        END IF;
    WHEN counting =>
        IF (zero='1') THEN
```

```
        next_state <= alarm;
    ELSIF (stop='1') THEN
        next_state <= suspended;
    ELSE
        next_state <= counting;
    END IF;
WHEN suspended =>
    IF (stop='0') THEN
        next_state <= counting;
    ELSE
        next_state <= suspended;
    END IF;
WHEN end_count =>
    IF (stop='1') THEN
        next_state <= flush;
    ELSE
        next_state <= end_count;
    END IF;
WHEN OTHERS =>
    next_state <= flush;
END CASE;
END PROCESS nextstate;

output : PROCESS (current_state)
BEGIN
    -- Default Assignment
    beep_int <= '0';
    clear <= '0';
    hold <= '0';
    load <= '0';
    -- State Actions
    CASE current_state IS
    WHEN flush =>
        hold<='1';
        clear<='1';
        beep_int<='0';
    WHEN getkey =>
        hold<='1';
    WHEN load_t =>
        hold<='1';
    WHEN load_u =>
        hold<='1';
        load<='1';
```

```
        WHEN standby =>
            hold<='1';
        WHEN alarm =>
            hold<='1';
            clear<='1';
            beep_int<='1';
        WHEN suspended =>
            hold<='1';
        WHEN end_count =>
            hold<='1';
            clear<='1';
        WHEN OTHERS =>
            NULL;
    END CASE;
END PROCESS output;
END spec;
```

If you are using Verilog, modify the module as follows:

```
//hds interface_end
wire clk;
wire [9:0] d;
wire reset;
wire start;
wire stop;
wire zero;
reg beep;
reg clear;
reg hold;
reg load;
// Declare any pre-registered internal signals
reg beep_int ;

// Module Declarations
parameter ZEROS = 10'b0;

// State encoding
parameter [3:0] // pragma enum current_state_code
    getkey = 4'd0 ,
    flush = 4'd1 ,
    load_t = 4'd2 ,
    load_u = 4'd3 ,
    standby = 4'd4 ,
    alarm = 4'd5 ,
```

```
    counting = 4'd6 ,
    suspended = 4'd7 ,
    end_count = 4'd8 ;

reg [3:0] /* pragma enum current_state_code */ current_state,
                                                next_state ;
// pragma state_vector current_state

// Next state code for machine machine0
always @(current_state or d or start or stop or zero)
begin
    case (current_state)
        getkey:
            if (start==1)
                next_state = standby;
            else if (d!=ZEROS)
                next_state = load_u;
            else if (stop==1)
                next_state = flush;
            else
                next_state = getkey;
        flush:
            if (d!=ZEROS)
                next_state = load_u;
            else
                next_state = flush;
        load_t:
            if (d!=ZEROS)
                next_state = getkey;
            else
                next_state = load_t;
        load_u:
            next_state = load_t;
        standby:
            if (zero==1)
                next_state = alarm;
            else if (start == 1)
                next_state = counting;
            else
                next_state = standby;
        alarm:
            if (stop==1)
                next_state = end_count;
```



```
        else
            next_state = alarm;
    counting:
        if (zero==1)
            next_state = alarm;
        else if (stop==1)
            next_state = suspended;
        else
            next_state = counting;
    suspended:
        if (stop==0)
            next_state = counting;
        else
            next_state = suspended;
    end_count:
        if (stop==1)
            next_state = flush;
        else
            next_state = end_count;
    default: begin
        next_state = flush;
    end
    endcase
end // Next state code

// Output code for machine machine0
always @(current_state)
begin
    // Default Assignment
    beep_int = 0;
    clear = 0;
    hold = 0;
    load = 0;

    // State Actions
    case (current_state)
        getkey: begin
            hold = 1;
            clear = 0;
            load = 0;
        end
        flush: begin
            hold = 1;
        end
    endcase
end
```

```
        clear = 1;
        load = 0;
    end
    load_t: begin
        hold = 1;
        clear = 0;
        load = 0;
    end
    load_u: begin
        hold = 1 ;
        clear = 0;
        load = 1 ;
    end
    standby: begin
        hold = 1;
        clear = 0;
        load = 0;
    end
    alarm: begin
        hold = 1;
        clear = 1;
        load = 0;
        beep_int = 1;
    end
    counting: begin
        hold = 0;
        clear = 0;
        load = 0;
    end
    suspended: begin
        hold = 1;
        clear = 0;
        load = 0;
    end
    end_count: begin
        hold = 1;
        clear = 1;
        load = 0;
    end
endcase
end // Output code
```

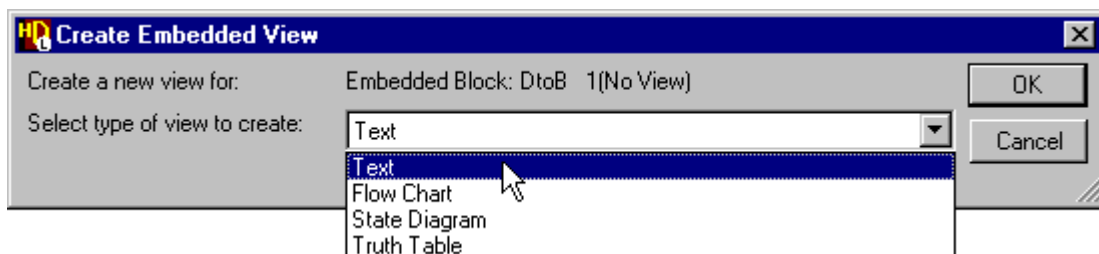
```
// Clocked code for machine machine0
always @(negedge clk or posedge reset)
begin
  if (reset) begin
    current_state <= flush;
    // Reset Values
    beep <= 0;
  end
  else begin
    current_state <= next_state;
    // Registered output assignments
    beep <= beep_int;
  end
end // Clocked code

endmodule // Control
```

Continue the VHDL tutorial from [“Import the BCDCounter Design Unit”](#) on page 1-44 or the Verilog tutorial from [“Import the BCDCounter Design Unit”](#) on page 2-44.

## Create HDL Text for the DtoB Block


Double-click over the *DtoB* embedded block in the *Timer* block diagram to display the Create Embedded View dialog box. Select a **Text** view in the dialog box and use the  button.



An embedded **HDL text** view containing default text is displayed on the block diagram adjacent to the embedded block.

Select the default text and choose **Send to Editor** from the popup menu to open the default text editor. Notice that the default text is dimmed and cannot be edited on the diagram



You can also edit the embedded text directly on the diagram or by using the  button to display the **Text** tab of the Object Properties dialog box.

If you are using VHDL, enter the following HDL code in the text editor:

```
truth_process: PROCESS(d)
BEGIN
  IF (d(0) = '1') THEN
    dat_in <= "0000";
  ELSIF (d(1) = '1') THEN
    dat_in <= "0001";
  ELSIF (d(2) = '1') THEN
    dat_in <= "0010";
  ELSIF (d(3) = '1') THEN
    dat_in <= "0011";
  ELSIF (d(4) = '1') THEN
    dat_in <= "0100";
  ELSIF (d(5) = '1') THEN
    dat_in <= "0101";
  ELSIF (d(6) = '1') THEN
    dat_in <= "0110";
  ELSIF (d(7) = '1') THEN
    dat_in <= "0111";
  ELSIF (d(8) = '1') THEN
    dat_in <= "1000";
  ELSIF (d(9) = '1') THEN
    dat_in <= "1001";
  ELSE
    dat_in <= "0000";
  END IF;
END PROCESS truth_process;
```



It is possible to copy the HDL code from this document window after choosing **Select Text** from the **Tools** menu in the Acrobat viewer. However, when copying HDL code, check that punctuation characters are copied correctly. In particular, line feed characters may not be translated on UNIX systems and may need to be re-entered.

If you are using Verilog, enter the following HDL code in the text editor:

```
//Internal signal declaration
reg [3:0] d_out;

always @ (d)
begin
    if ((d[0] == 1))
        d_out = 4'b0000;
    else if ((d[1] == 1))
        d_out = 4'b0001;
    else if ((d[2] == 1))
        d_out = 4'b0010;
    else if ((d[3] == 1))
        d_out = 4'b0011;
    else if ((d[4] == 1))
        d_out = 4'b0100;
    else if ((d[5] == 1))
        d_out = 4'b0101;
    else if ((d[6] == 1))
        d_out = 4'b0110;
    else if ((d[7] == 1))
        d_out = 4'b0111;
    else if ((d[8] == 1))
        d_out = 4'b1000;
    else if ((d[9] == 1))
        d_out = 4'b1001;
    else
        d_out = 4'b0000;
end

// Concurrent statement
assign dat_in = d_out;
```

Save the text view and exit from the text editor. Choose **Finish Edit** from the popup menu to update the embedded HDL text view on the block diagram.

Select the HDL text and choose **Hide Text** from the popup menu. The text is hidden but can be re-displayed by double-clicking on the embedded block or choosing **Show Text** from the popup menu.

Continue the VHDL tutorial from [“Browse the Timer Design” on page 1-60](#) or the Verilog tutorial from [“Browse the Timer Design” on page 2-61](#).


## Importing the Tester Design Unit

You can import the tester design unit as described on [page 1-66](#) (for VHDL) or [page 2-67](#) (for Verilog). However, ensure that the **Flow Chart** option is not selected in the Choose View Styles page of the HDL Import wizard.

Flow charts are not supported by the text design tools and you must import a HDL text view for the tester design unit.

## Generating and Compiling the Design

It was not necessary to set generation characteristics for the HDL text views but these views must be copied to the generated library before your design can be compiled.

This is done automatically if you select the *Timer\_tb* design unit in the design browser and choose the  flow button. Any changed views are regenerated and the entire design is compiled ready for simulation.

---

# Appendix B

## Using Verilog-XL

### Introduction

The HDL Designer Series tools support compilation, simulation and animation using dynamically linked versions of Verilog-XL (2.6 or later) on UNIX or Windows NT.

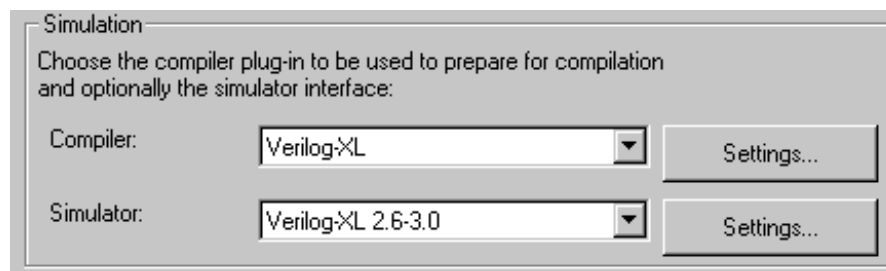
The tutorial design can be created using the procedures described in Chapter 2 for the [Verilog Timer Exercise](#) until you reach “[Setup the Downstream Tools](#)” on [page 2-72](#). You should then follow the alternative procedures in this appendix.

The Verilog-XL simulator is accessed using the `CDS_INST_DIR` environment variable which must be included in your search path before you invoke the HDL Designer Series tool.

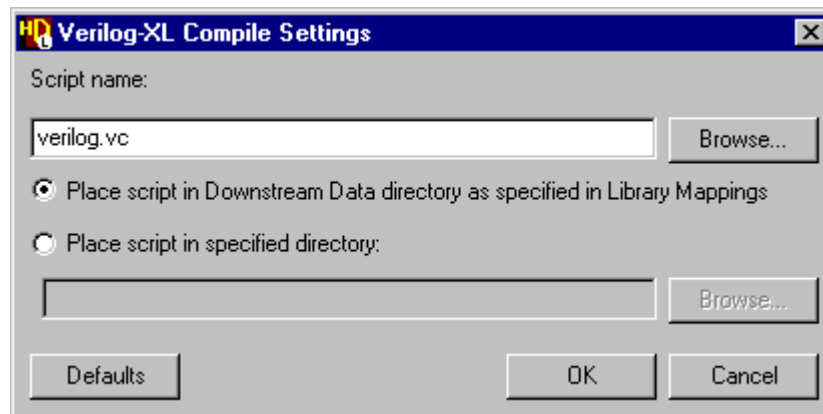
### Setup Verilog-XL

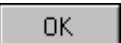
Choose **Compile** from the **Options** menu to display the **Tool Settings** tab of the Compile Settings dialog box.


Choose Verilog-XL from the Compiler pulldown list and Verilog-XL 2.6-3.0 from the Simulator pulldown list:

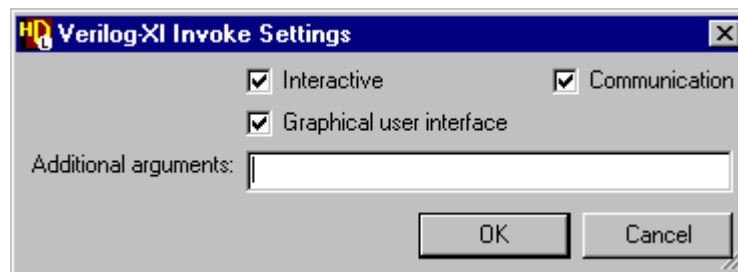


Use the Compiler  button to display the default downstream data settings dialog box for Verilog-XL.



Use the  button to confirm the default settings and close the dialog box.

Use the Simulator  button to display the default invoke settings for the Verilog-XL simulator:



Ensure that the **Interactive**, **Communication** and **Graphical user interface** check boxes are selected. Use the  button to confirm you simulator settings and close the dialog box.

On both PC or UNIX systems, ensure that the temporary directory specified in the Compile Settings dialog box (which normally defaults to `/tmp` on UNIX or `C:\TEMP` on PC systems) exists.


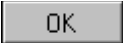


## Invoke the Verilog-XL Simulator

The simulator can be invoked from the design browser or any block diagram, flow chart or state diagram in your design. However, it must be invoked at a level that includes the entire branch of your design that you want to test. This tutorial uses the test bench to verify your version of the *Timer* design.



You must have generated HDL for all design units in the hierarchy you want to simulate. If you did not set **Instrument HDL for animation** in the state machine and flow chart properties, refer to the topics “[Set Generation Properties](#)” on page 2-39 and “[Generate HDL for the Test Bench](#)” on page 2-69.

Select the *Timer\_tb* design unit in the design browser and use the  button (or choose **Start Simulator** from the **HDL** menu) to display the Start Verilog-XL dialog box. Use the  button to confirm the invoke options.


The HDL for your design is compiled and completion messages are issued in the Verilog-XL window. For example:

```
Compiling source file "..... \Timer_tester.v"
Now connected to HDS
Compiling source file ".....\Control.v"
Compiling source file ".....\spec.v"
Compiling source file ".....\Counter.v"
Compiling source file ".....\Timer.v"
Compiling source file ".....\Timer_tb.v"
Highest level modules:
Timer_tb
```

There should be no errors or warnings. If there are any errors, correct your design and regenerate HDL.

## Setup the SimWave Window

Choose **Nets** from the Verilog-XL **Select** menu to select all nets in the generated HDL for the highest level design unit (*Timer\_tb*) which will be displayed in the Verilog-XL window.

Use the  button in the Verilog-XL window to open the SimWave window with these signals loaded.

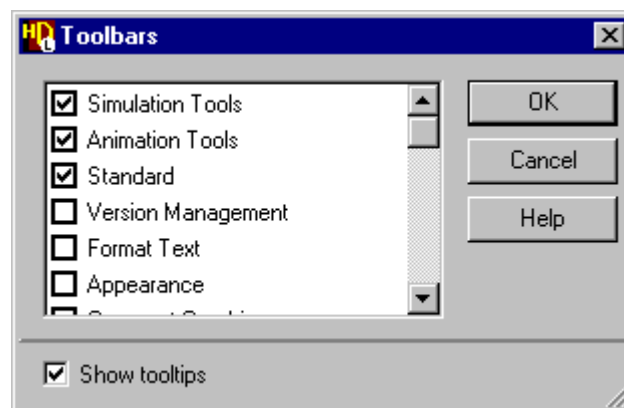
## Enable Animation

Display the *Monitor* concurrent view of the *Timer\_tester* flow chart and both hierarchical views of the *Control* state diagram. Use the **View Panel** command in each window to adjust the window view to the panel areas you defined earlier in this tutorial.

Notice that an additional Animation toolbar is available in these windows when the simulator is invoked. This toolbar is normally docked at the bottom of the diagram window but can be moved independently.




The editing toolbars are not usually required during simulation and animation. You can hide or show toolbars using the Toolbars dialog box which is displayed by choosing **Settings** from the **Toolbars** cascade of the **View** menu.



For example, use the dialog box to hide the Format Text, Appearance, Comment Graphics and Arrange Object toolbars.





Individual toolbars can also be hidden by unsetting the corresponding options in the **Toolbars** cascade of the **View** menu.

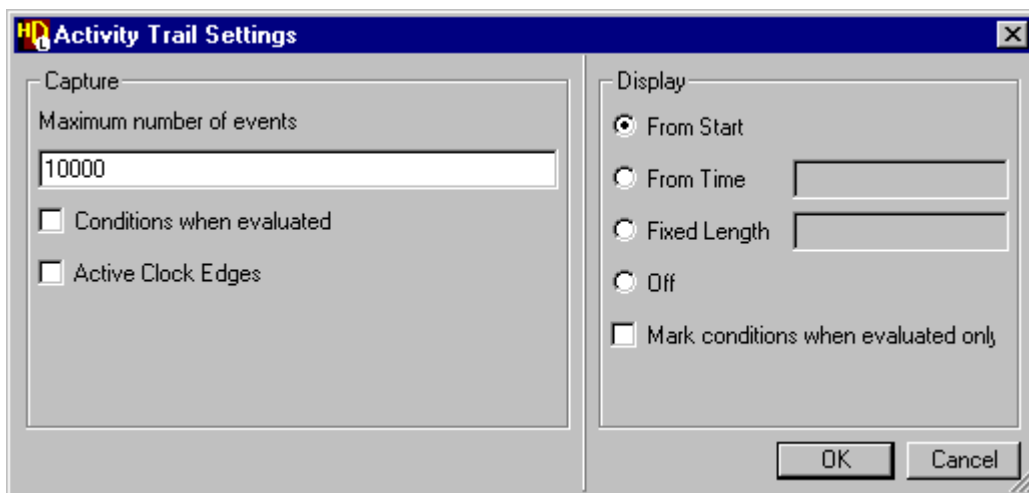
Use the  buttons from the Animation toolbars (or choose **Data Capture** from the **Animation** menus) in the state diagram and flow chart windows. (It is not necessary to repeat this command for each hierarchical or concurrent view since these are considered part of the same diagram.)


Notice that all objects (except the exit and entry points in the state diagram) are redrawn with white fill.



This animation view is automatically displayed when you set data capture but can be toggled at any time by using the  button (or by toggling the **Show Animation** option in the **Animation** menu). You can also set data capture for all instances in the current simulation hierarchy by choosing **Global Capture On** from the **Animation** menu.

Use the  button from the Animation toolbar (or choose **Activity Trails** from the **Animation** menu) in the flow chart or state diagram window and choose the **From Start** option in the Activity Trail Settings dialog box.



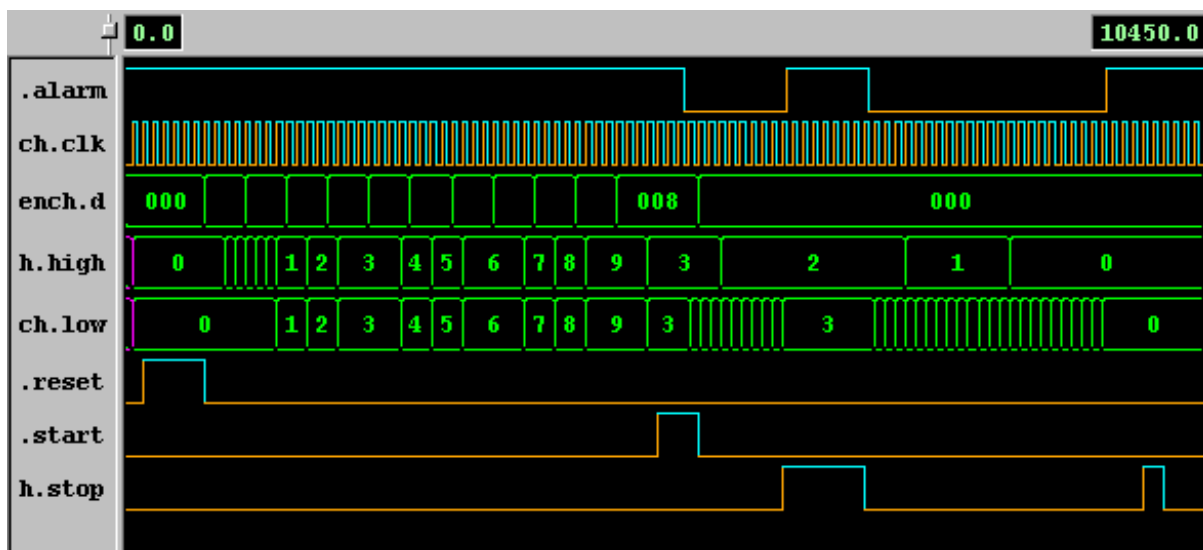
Use the  button to confirm the Activity Trail Settings dialog box. The activity trail is applied to all windows in the current simulation.

## Running the Verilog-XL Simulator

Choose **Run** from the Verilog-XL **Control** menu. The simulation should run to completion and issue the messages:

```
Count suspended correctly
Alarm asserted correctly
Timer test completed
L181 ".....\Timer_tester.v": $stop at simulation time 10450
C1 > $scope(Timer_tb.I1.process0);
C2 >
```

Examine the results displayed in the SimWave window and ensure that the simulation has performed correctly by comparison with the example waveforms shown below.




If any Verilog errors are discovered, correct your design, regenerate and recompile the modified diagram and use the **Reset Simulation** and **Run** commands from the Verilog-XL **Control** menu to repeat the simulation.








Simulation cross probing is not supported for Verilog-XL. However, all normal simulation controls are available from the simulation windows.




## Review the Animation

Notice how the animation activity trail in both the flow chart and state diagram is highlighted in blue and the end of the flow chart process for the `TimerTester` module is displayed in the main Verilog-XL simulator window.

Use the  button or choose **Link Diagrams** from the **Animation** menu to link the animated diagrams. When the diagrams are linked, all animation review commands are applied to all animated diagrams in the simulation hierarchy.

You can review the animation by using the , , ,  or  buttons (or by choosing **Goto Next**, **Goto Previous**, **Goto Time**, **Goto Start** or **Goto Latest** from the **Animation** menu).

The  and  buttons step through each object in the flow chart.

However in the state diagram, you can set options in the **Animation** menu (or from a pulldown menu on the toolbar button) to move by change of state , simulation event  or change of clock edge . Notice how the toolbar button icon changes to indicate the current mode of movement.

You may want to change the activity trail to display the trail from a specified time or specify a fixed length. The current simulation step (or current state and last transition taken in an animated state diagram) is always shown in red and the previous step (or previous state and transition) in yellow. All other visited objects included in the activity trail are shown in blue.

Remember that you can open down the hierarchical action box or hierarchical state to view the animation in the child diagrams.

Compare the animation with the results displayed in the SimWave window and ensure that the simulation has performed correctly by comparison with the example waveforms on the previous page.

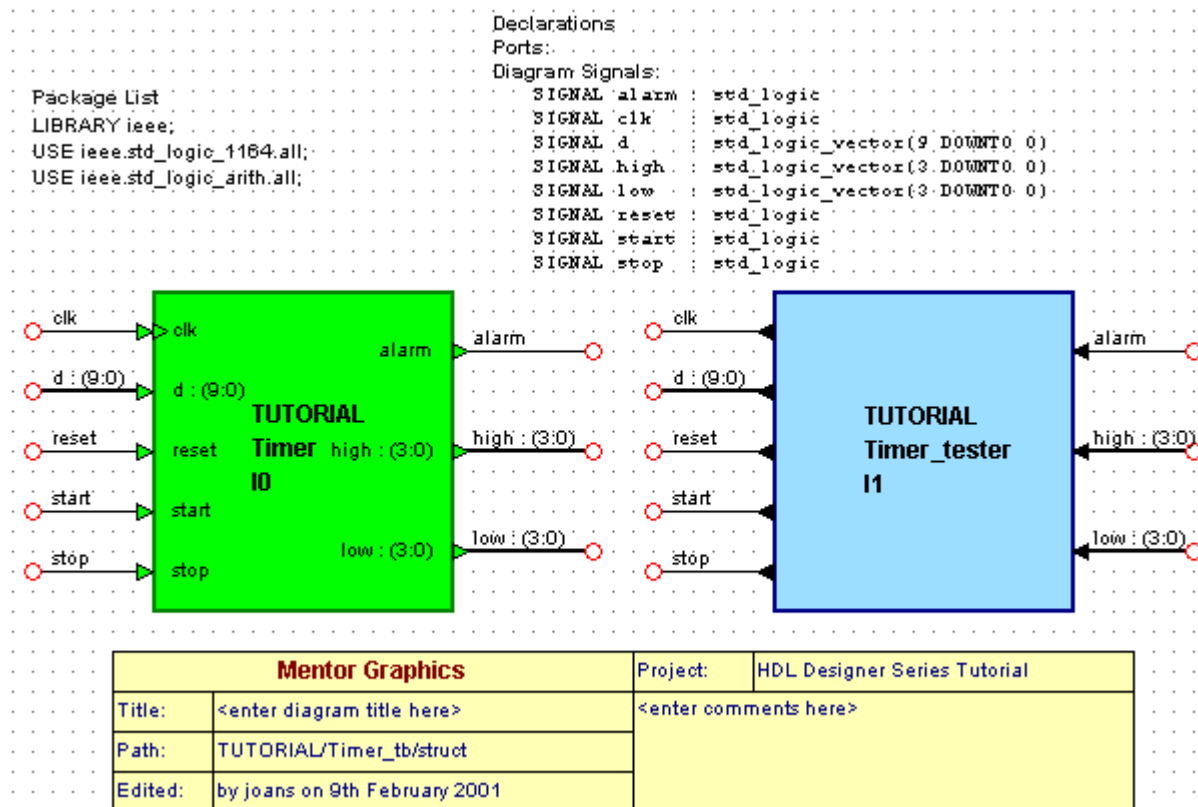


# Appendix C

## Creating a VHDL Flow Chart

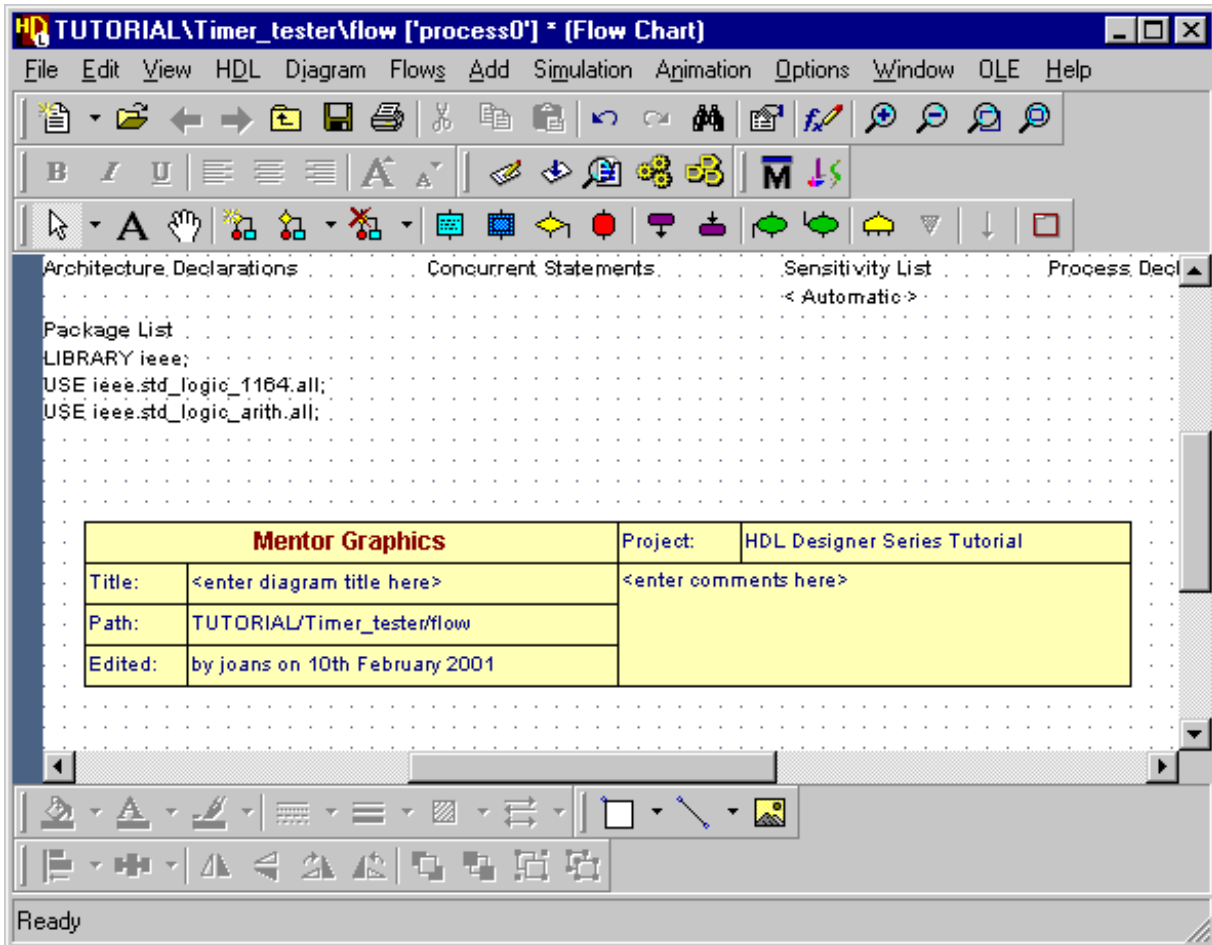
### Introduction

This appendix describes procedures for creating a VHDL flow chart as an alternative to importing the example flow chart. After you “[Create a Test Bench](#)” on page 1-64 the test bench block diagram should look similar to the following picture:



## Create the Tester Flow Chart

Double-click over the *Timer\_tester* block in the test bench block diagram and select a **Flow Chart** view in the Open Down Create New View dialog box using the default view name *flow.fc*. A new flow chart (*TUTORIAL\Timer\_tester\flow ['process0']*) is created as a child view of the *Timer\_tester* block:



The new flow chart is initialized with the default VHDL package list, default title block and labels for architecture declarations, concurrent statements, sensitivity list and process declarations.

Choose **Rename Flow Chart** from the **Diagram** menu and enter the new name *Monitor*. When you confirm the dialog box, this name replaces the default process name (*process0*) appended to the design unit and view names.



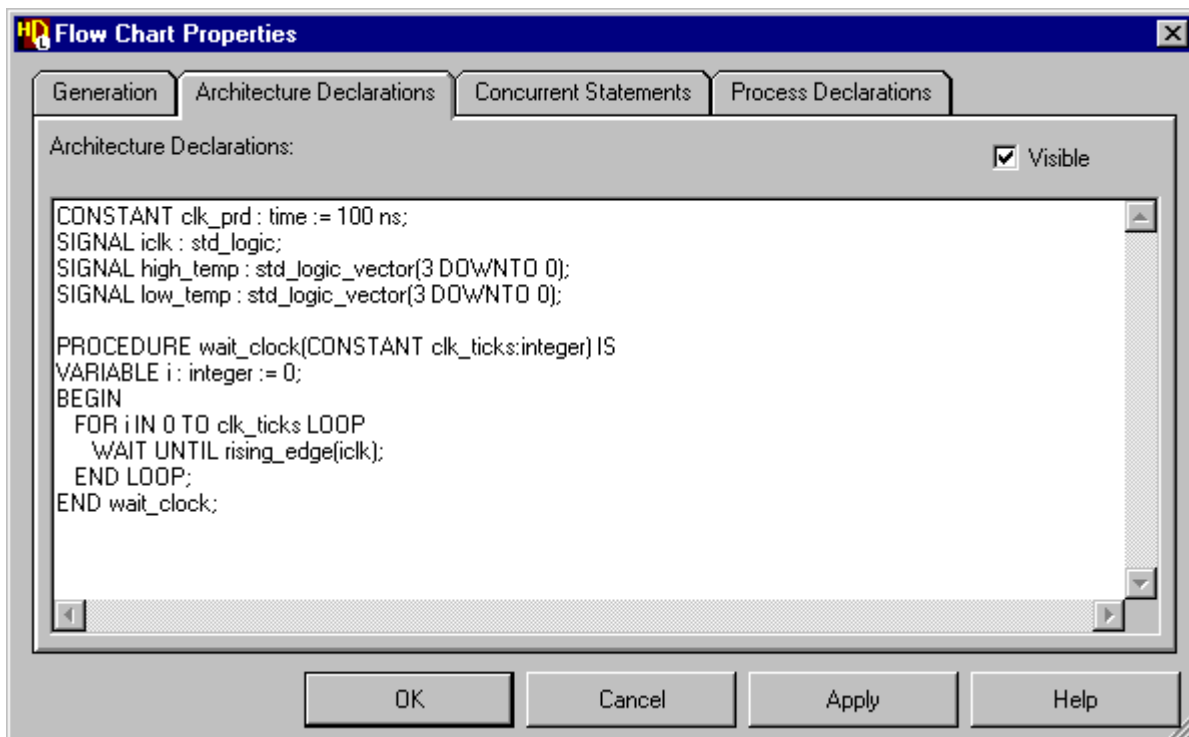
## Set Flow Chart Properties

Double-click with the mouse over the Architecture Declarations label in the flow chart to display the **Architecture Declarations** tab of the Flow Chart Properties dialog box.


Enter the following declarations which set up some internal variables and a simple wait procedure:

```
CONSTANT clk_prd : time := 100 ns;
SIGNAL iclk : std_logic;
SIGNAL high_temp : std_logic_vector(3 DOWNTO 0);
SIGNAL low_temp : std_logic_vector(3 DOWNTO 0);

PROCEDURE wait_clock(CONSTANT clk_ticks:integer) IS
VARIABLE i : integer := 0;
BEGIN
    FOR i IN 0 TO clk_ticks LOOP
        WAIT UNTIL rising_edge(icmp);
    END LOOP;
END wait_clock;
```

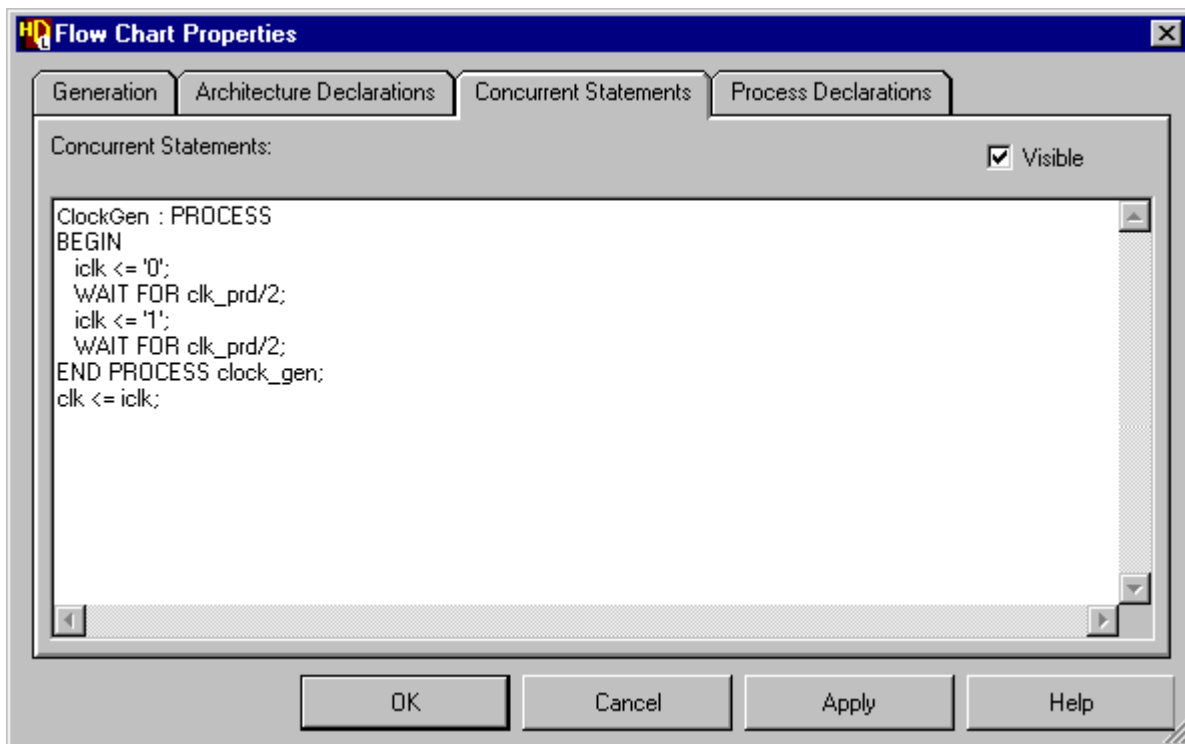


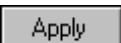
The architecture declarations are included at the top of the architecture description in the generated HDL and must be valid VHDL statements terminated by a semi-colon. Comment text can be included provided each comment is preceded by the VHDL comment characters (--).

Use the  button to display these properties on the flow chart. The HDL syntax is checked on entry and you are warned if any errors are encountered.

Select the **Concurrent Statements** tab and enter the following statements which generate a clock signal for the test bench:

```
ClockGen : PROCESS
BEGIN
    iclk <= '0';
    WAIT FOR clk_prd/2;
    iclk <= '1';
    WAIT FOR clk_prd/2;
END PROCESS clock_gen;
clk <= iclk;
```



Use the  button to display these properties on the flow chart.

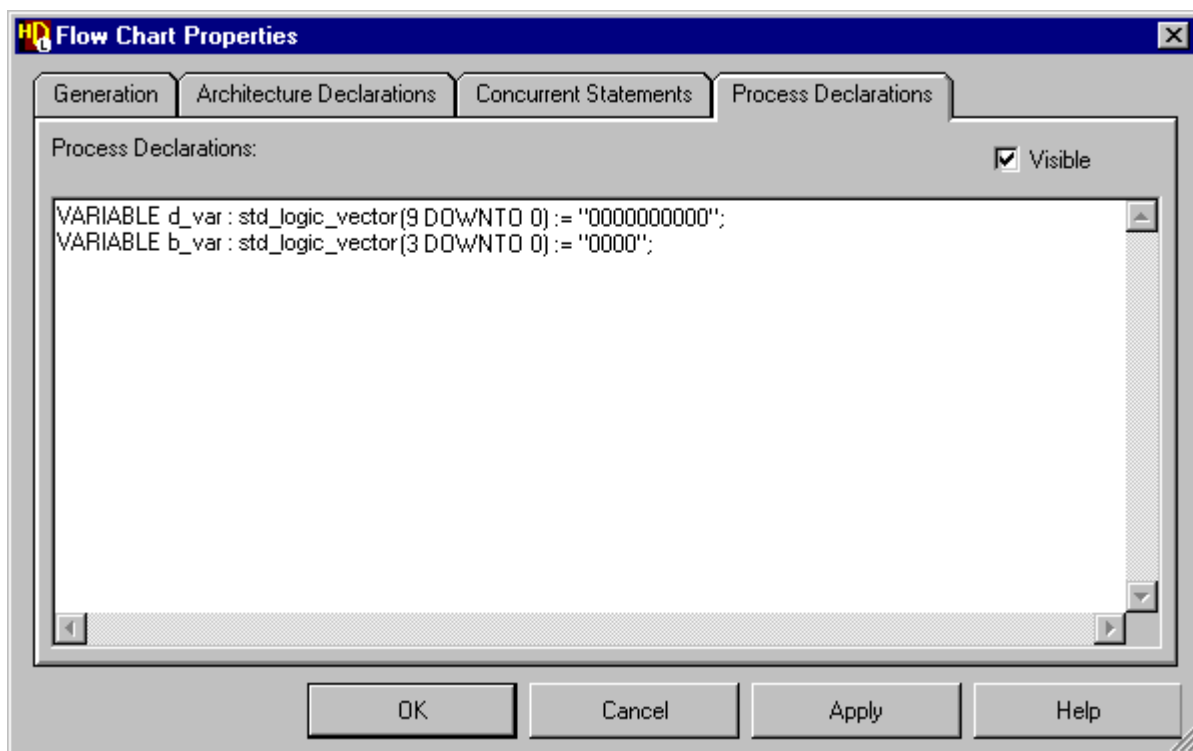
Concurrent statements are included after the BEGIN statement in the generated HDL for the flow chart and must be valid HDL statements and be terminated by a semi-colon. They are typically used for a concurrent process (such as the clock generator defined by this example).



If you use HDL import to recover the *Timer\_tester* flow chart, the *ClkGen* process is recovered as a concurrent flow chart view.

Select the **Process Declarations** tab and enter the following variable declarations:

```
VARIABLE d_var : std_logic_vector(9 DOWNTO 0) := "0000000000";  
VARIABLE b_var : std_logic_vector(3 DOWNTO 0) := "0000";
```







Process declarations are included at the top of the main process in the generated HDL for the flow chart and must be valid HDL statements terminated by a semi-colon.

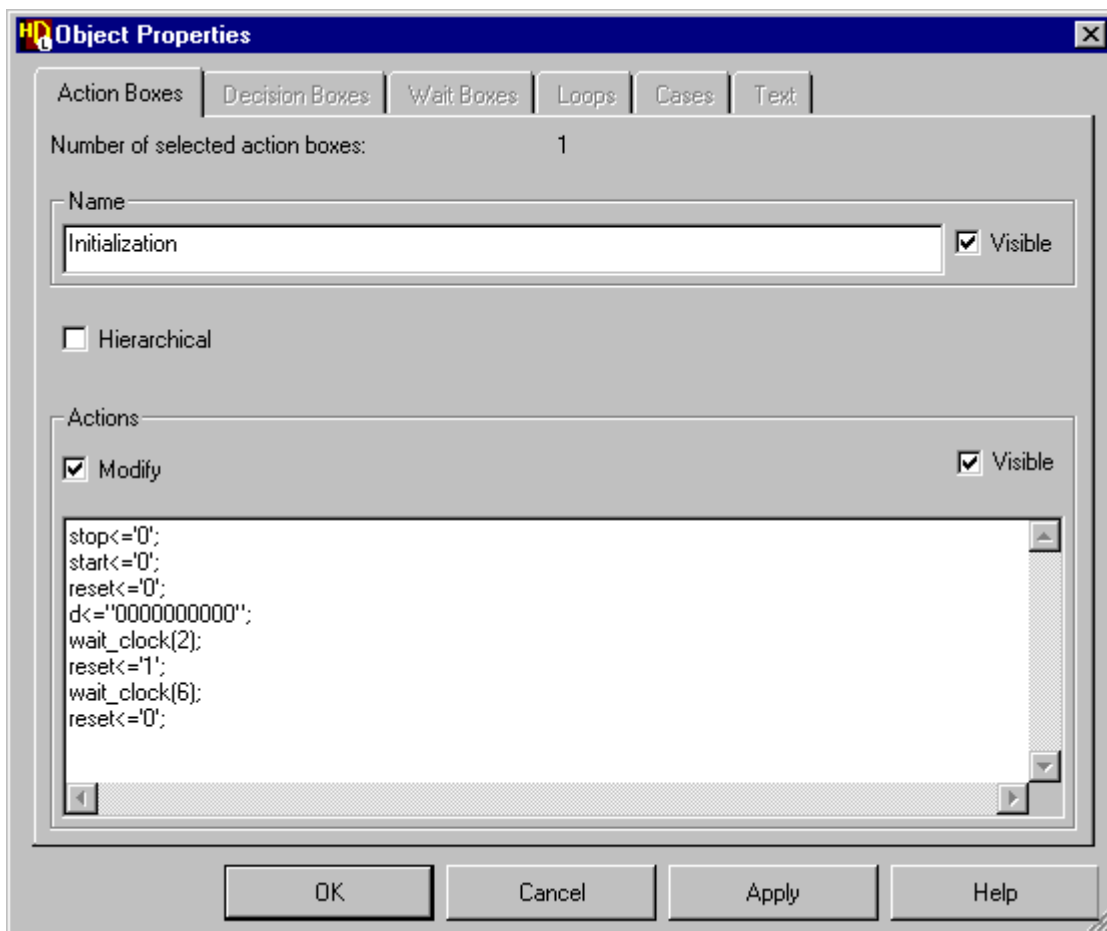
Use the  button to apply these properties to the flow chart and dismiss the dialog box.

## Add a Start Point and Action Box

Use the  button to add a **start point** near the top of the flow chart.

Use the  button to add an **action box** below the start point and use the  button to add a **flow** connecting it to the start point. To add a flow, click with the mouse over the body of the start point and the action box close to the  markers.


Double-click with the mouse over the action box (or use the  button) to display the **Action Boxes** tab of the Flow Chart Object Properties dialog box.

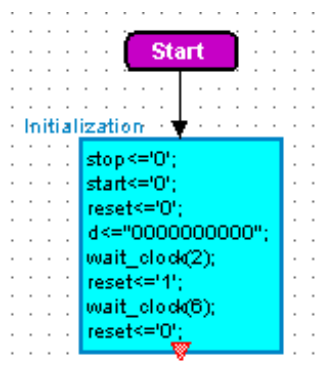


You can change the action box name and enter action statements. However, the name must be unique on the flow chart and cannot be changed when more than one action box is selected.

Change the default name in the dialog box to *Initialization* and enter the following actions:

```
stop<='0';
start<='0';
reset<='0';
d<="0000000000";
wait_clock(2);
reset<='1';
wait_clock(6);
reset<='0';
```

Use the  button to update the flow chart. You may want to resize the action box on the chart (by dragging the handles on the lower edge with the mouse) so that it encloses the actions text.



If you drag one side of an action box, the mid-point is preserved.

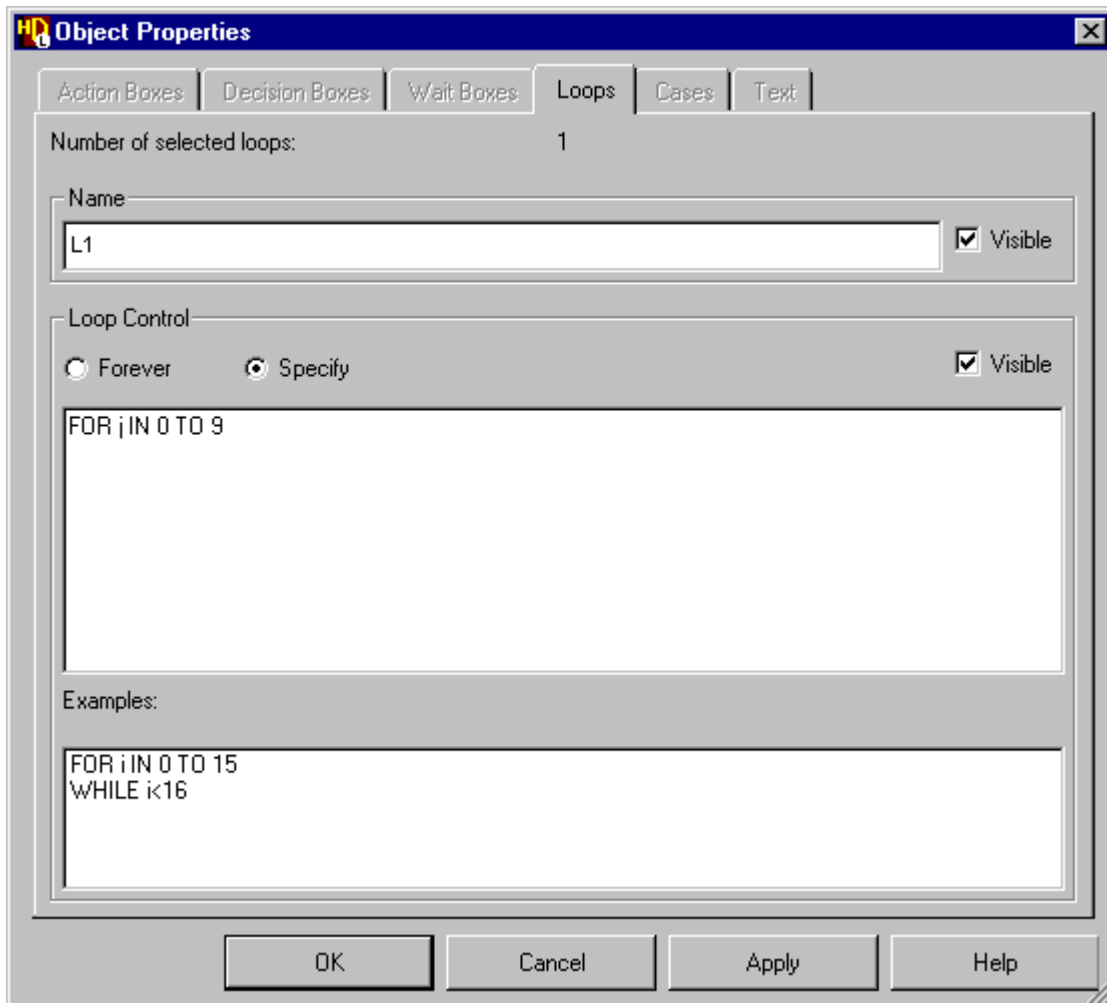
## Add a Loop and an Associated Comment

Use the  button to add a start loop below the *Initialization* action box.


Select the start loop to display the **Loops** tab of the Flow Chart Object Properties dialog box. Choose the **Specify** button and enter the following loop control statement:

```
FOR j IN 0 TO 9
```

Rename the loop *L1* and use the  button to update the flow chart.






By default, a loop will be repeated forever. However, if you choose the **Specify** option, the dialog box discloses an entry box for loop control statements. Template examples are provided which you can copy into the entry box by clicking on one of the examples with the mouse.

Use the  button to add an action box below the start loop.

Click with the mouse to select the new action box and use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to enter the following actions:

```
d_var:="0000000000";
d_var(j):='1';
d<=d_var;
wait_clock(4);
ASSERT ((high=b_var)AND(low=b_var))
REPORT "Decoder or Load failure."
SEVERITY failure;
b_var:=unsigned(b_var)+1;
```

Change the name of the action box to *Decoder* and use the  button to update the flow chart. Resize the action box so that it encloses the actions text.

Use the  button to add an end loop below the *Decoder* action box and use the  button to connect flows for the loop.

Use the  button to enter text mode. Click in a free space near the loop and enter the following text in the [comment text](#) entry box:

```
Loop through all
possible decoder
values and check
output values
```



You can enter free-format text including line breaks and spaces which will be preserved on the diagram.

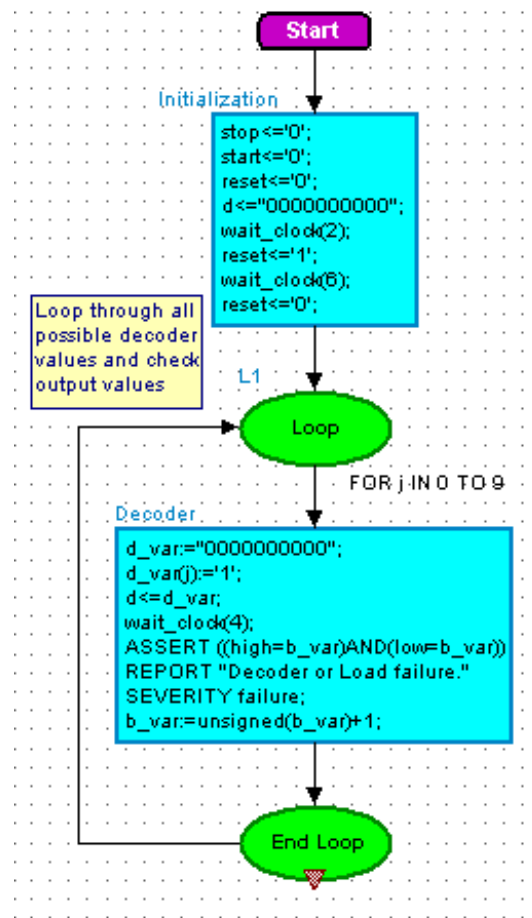
Click the left mouse button outside the text entry box to complete the text. Select the comment text and choose **Include in HDL** from the popup menu. Select **Before Object** from the popup menu. An anchor is attached to the cursor. Click the left mouse button with the cursor over the start loop object to terminate the anchor and associate the notes with the loop.

The comment text will be included immediately before the loop statements in the generated HDL for the flow chart.




If you use HDL import to recover the *Timer\_tester* flow chart, the associated comment is recovered as an actions box above the loop statement.

The flow chart should look similar to the following picture:





## Add an Action Box

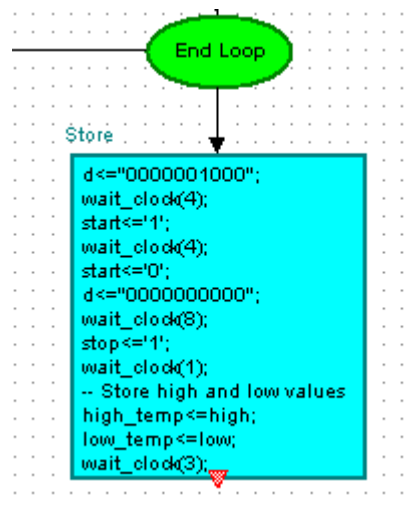
Use the  button to add another action box below the end loop and use the Flow Chart Object Properties dialog box to rename the action box *Store* and apply the following actions:

```
d<="0000001000";
wait_clock(4);
start<='1';
wait_clock(4);
start<='0';
d<="0000000000";
wait_clock(8);
stop<='1';
wait_clock(1);
--Store high and low values
high_temp<=high;
low_temp<=low;
wait_clock(3);
```



You can include comment text directly in an actions box by using HDL comment characters. For example the comment text `--Store high and low values` in the example above.

Use the  button to connect flows between the end loop and action boxes.





## Add a Hierarchical Action Box

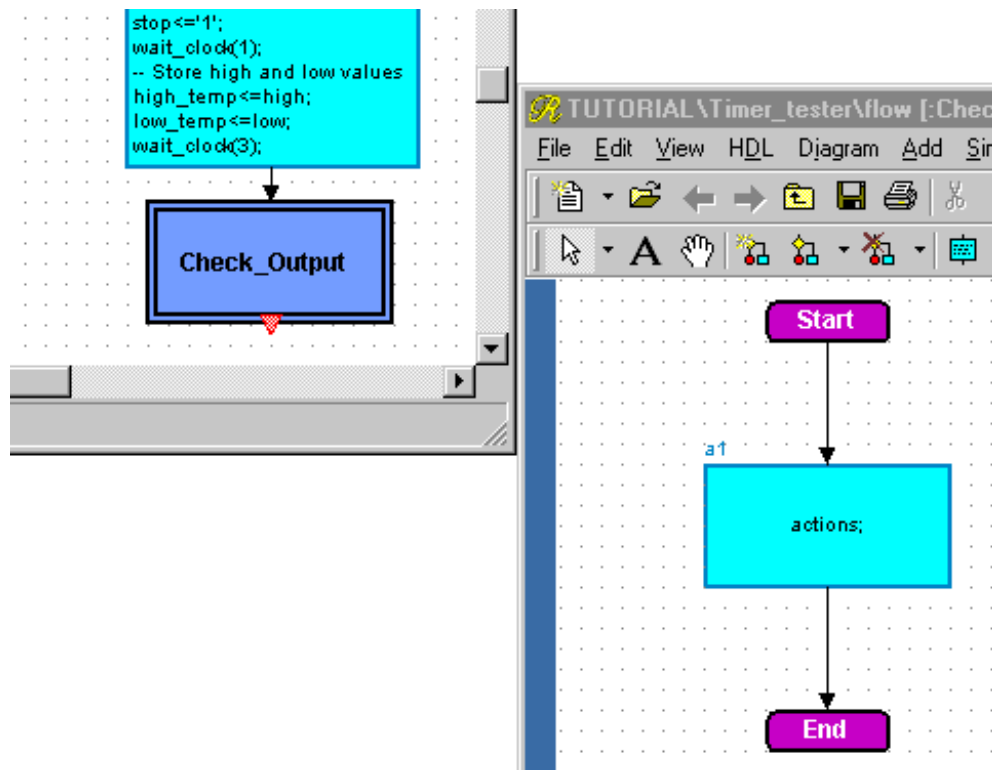
When a flow chart contains a large number of procedural statements it can rapidly become a very large diagram which can be difficult to read or print. However, such a chart can be represented by a number of separate hierarchical charts by using hierarchical action boxes.




Flow chart hierarchy does not effect the generated HDL and is not used if you use HDL import to recover the *Timer\_tester* flow chart.

Use the  button to add a **hierarchical action box** below the *Store* action box and use direct text editing to change the name of the hierarchical action box to *Check\_Output*. Use the  button to connect a flow between the action box and hierarchical action box. Double-click on the *Check\_Output* hierarchical action box to open a new child flow chart (or select the hierarchical action box and choose **Open Down** from the **File** menu).


The child flow chart is initialized as an action box connected by flows to a start point and **end point**.




## Add a Decision Box

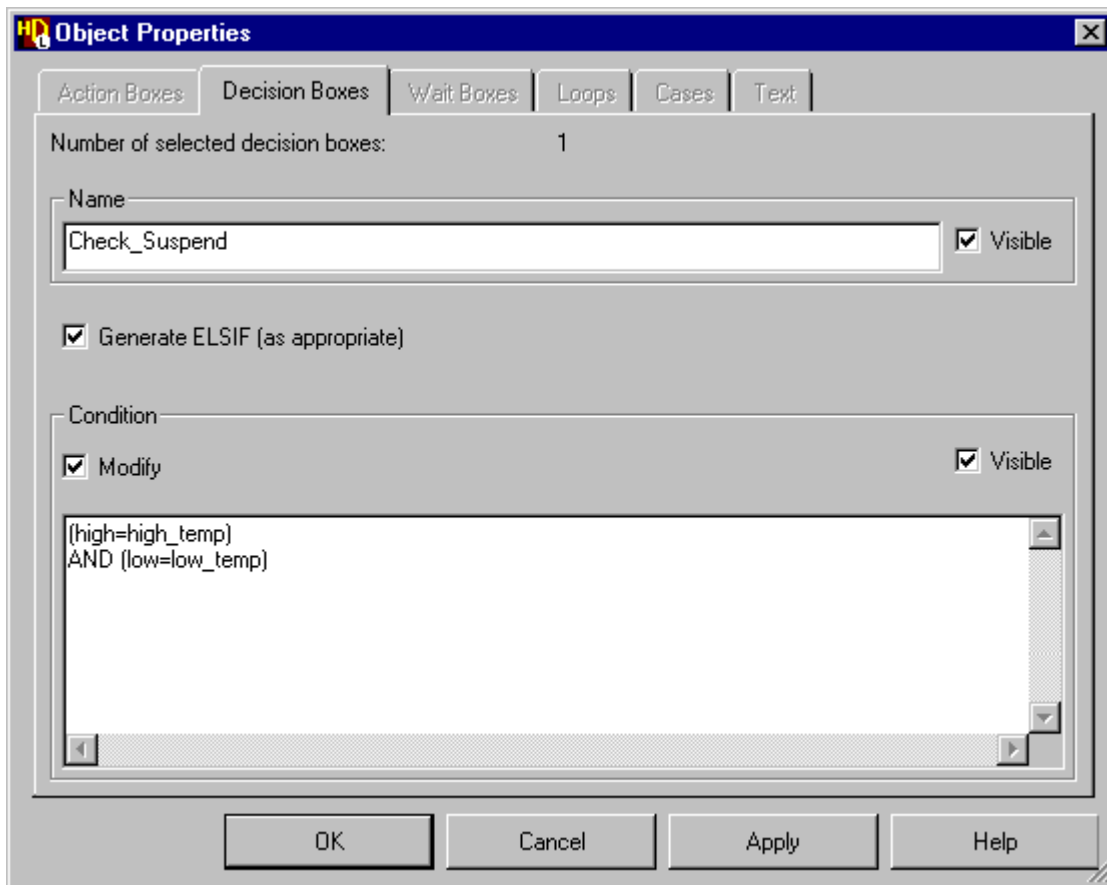
Select the flows connected to the start point and end point in the child flow chart and use the  key to delete them.



You can use  key with the left mouse button to select both flows.

Use the  button to add a **decision box** below the start point in the *Check\_Output* flow chart and use the **Decision Boxes** tab of the Flow Chart Object Properties dialog box to change the name of the decision box to *Check\_Suspend* and enter the following condition:


```
(high=high_temp)
AND (low=low_temp)
```



Select the existing action box. Use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to change its name to *Pass* and enter the following actions:


### Pass

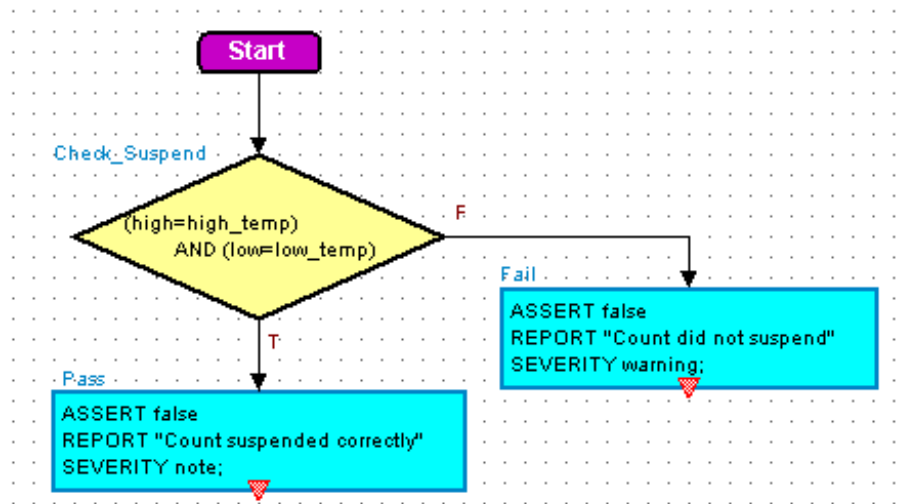
```
ASSERT false
REPORT "Count suspended correctly"
SEVERITY note;
```

Use the  button to add an action box for the False (F) condition. Use the **Action Boxes** tab to change its name to *Fail* and enter the following actions:

### Fail


```
ASSERT false
REPORT "Count did not suspend"
SEVERITY warning;
```

Reposition the action boxes by dragging with the mouse and use the  button to connect flows between the start point, decision box and action boxes.





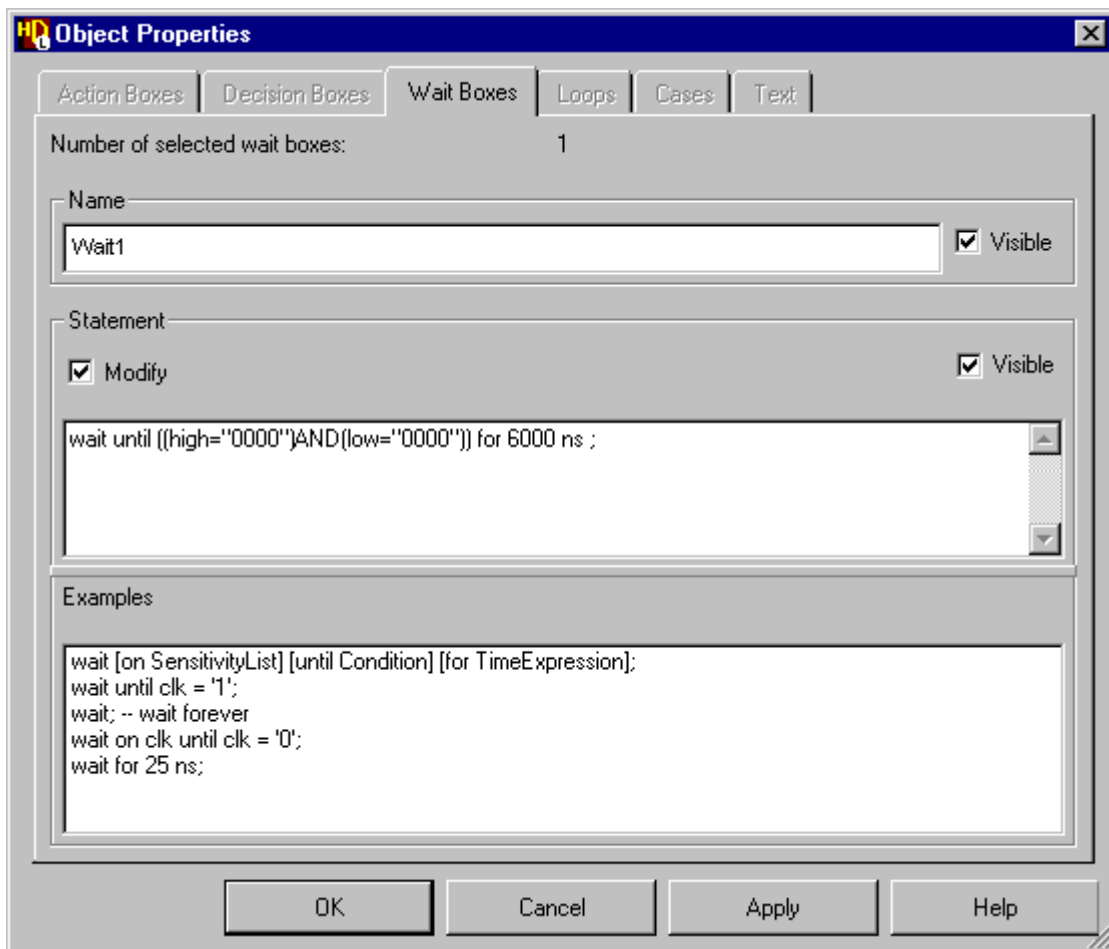
When a decision box object is selected, the popup menu includes an option to swap the True and False outputs.

## Add Wait Boxes

Use the  button to add an action box below the *Pass* action box and use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to enter the following actions:

```
wait_clock(4);
stop<='0';
```

Change the name of this action box to *Continue* and use the  button to add two **wait boxes** below it. Click the mouse over the first wait box (or use the  button) to display the **Wait Boxes** tab of the Flow Chart Object Properties dialog box:



Use the **Wait Box** tab of the Flow Chart Object Properties dialog box to rename the wait boxes and specify the following wait conditions:

### Wait1

```
wait until ((high="0000")AND(low="0000")) for 6000 ns ;
```

### Wait2

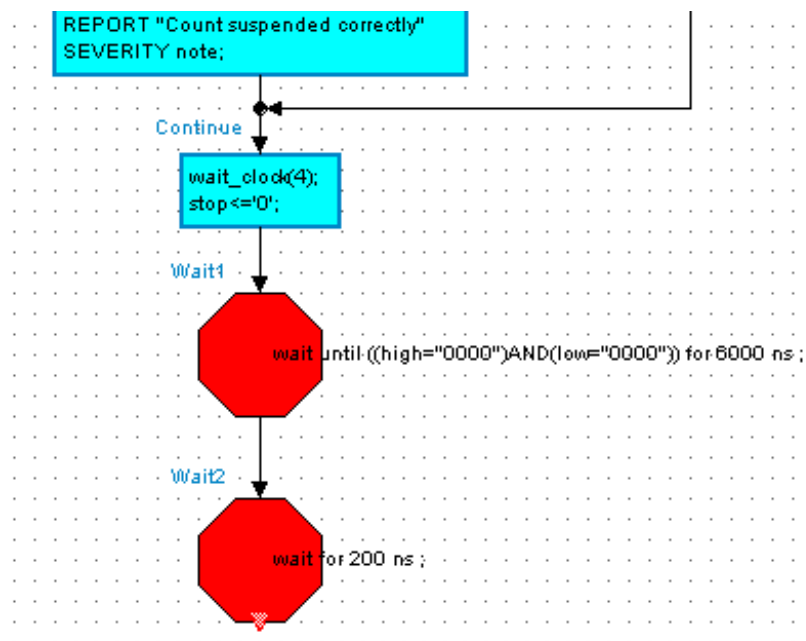
```
wait for 200 ns ;
```



You can click on one of the examples shown in the dialog box to use it as a template. Any valid combination of **on**, **until** and **for** can be used in the wait statement which must be terminated by a semi-colon.


Use the  button to connect flows between the action boxes and wait boxes.



Notice that a flow join is automatically created when you connect the two flows from the *Pass* and *Fail* action boxes together.



## Copy the Decision Tree

The child hierarchical flow chart can be completed by using similar procedures to those described in the last two topics. However, the next section can be done more quickly by copying the decision tree objects which have already been created.

Drag select the *Check\_Suspend* decision box, plus the *Pass* and *Fail* action boxes and use the  button (or choose **Copy** from the popup or **Edit** menu).

Scroll the window down and click the  mouse button below the last wait box to de-select the objects. Use the  button (or choose **Paste** from the popup or **Edit** menu) to paste a copy of the objects from the clipboard in the middle of the window. Use the mouse to drag the objects to the required position. Notice that each of the pasted objects is automatically given a unique name.

Use the Flow Chart Object Properties dialog box to change the names, conditions and actions where required.

### Check\_Alarm


```
alarm='1'
```

### Alarm\_Pass

```
ASSERT false  
REPORT "Alarm asserted correctly"  
SEVERITY note;
```

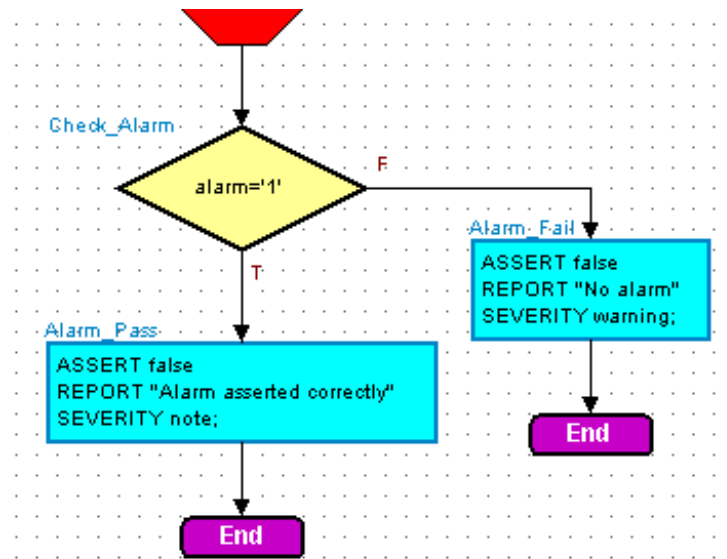
### Alarm\_Fail

```
ASSERT false  
REPORT "No alarm"  
SEVERITY warning;
```



Drag the end point beneath the other objects and use the  button to connect the remaining flows on the child flow chart.



Multiple end points can be used. For example, in the following picture, a separate end point has been attached to the *Alarm\_Pass* and *Alarm\_Fail* action boxes.




## Completing the Flow Chart

Use the  button (or choose **Open Up** from the **File** menu) to display the parent flow chart and the  button to add an action box below the hierarchical action box. Use the Flow Chart Object Properties dialog box to change the name of this action box to *Clear* and enter the following actions:



```
stop<='1';
wait_clock(2);
stop<='0';
wait_clock(4);
ASSERT false
REPORT "Timer test completed"
SEVERITY Failure;
```

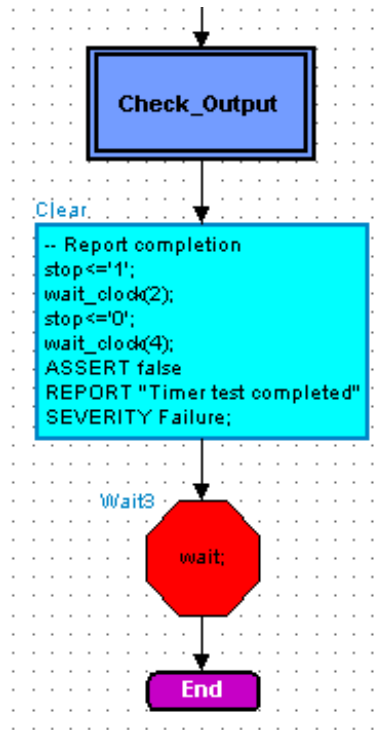


These actions are executed when the timer test has completed successfully but are given *Failure* severity to stop the simulator execution after the completion message is issued.


Use the  button to add a wait box below the action box. Change the name of this wait box to *Wait3* with the default wait condition (*wait;*). This condition corresponds to a wait for interaction at the completion of the test sequence.



Use the  button to add an end point and use the  button to connect any remaining flows.



Complete the flow chart by editing the title, project and comments fields in the title block.

Use the  button to save the flow chart.

Return to [“Generate HDL for the Test Bench”](#) on page 1-68 in the main VHDL tutorial.

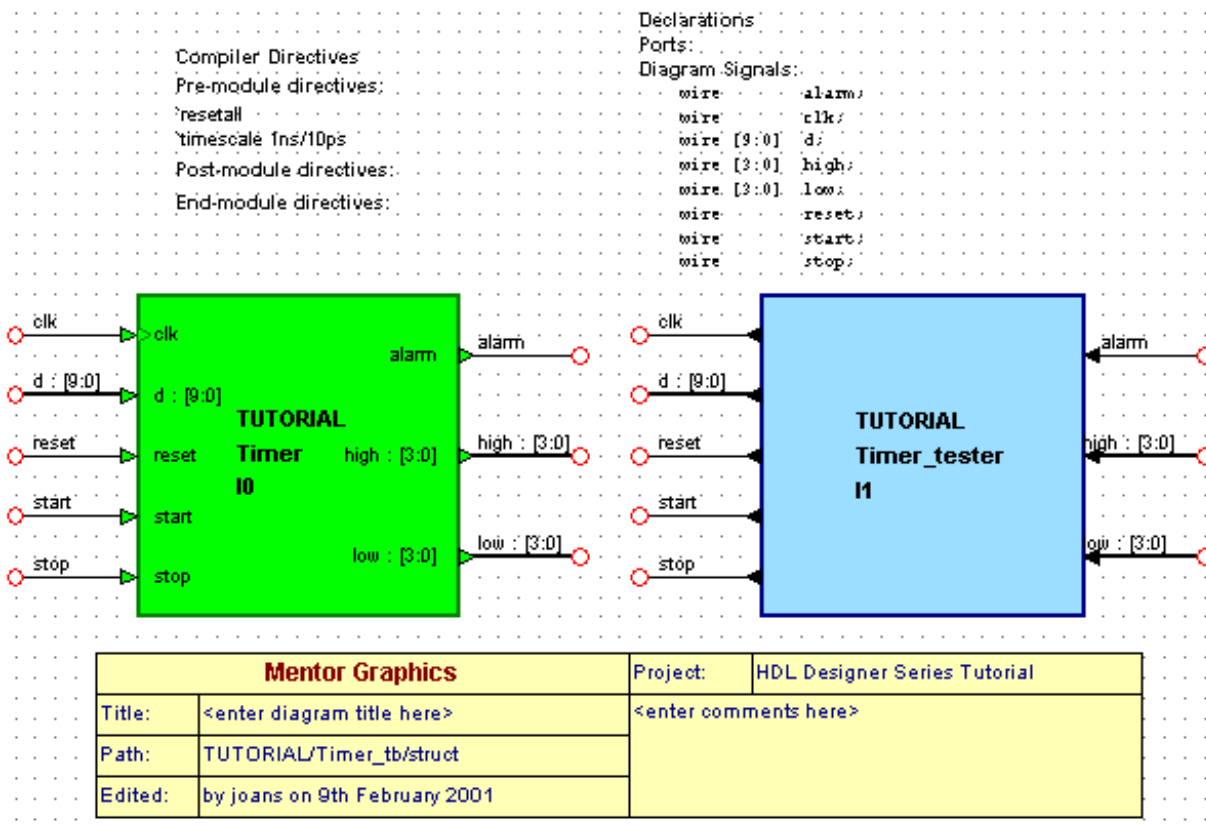


# Appendix D

## Creating a Verilog Flow Chart

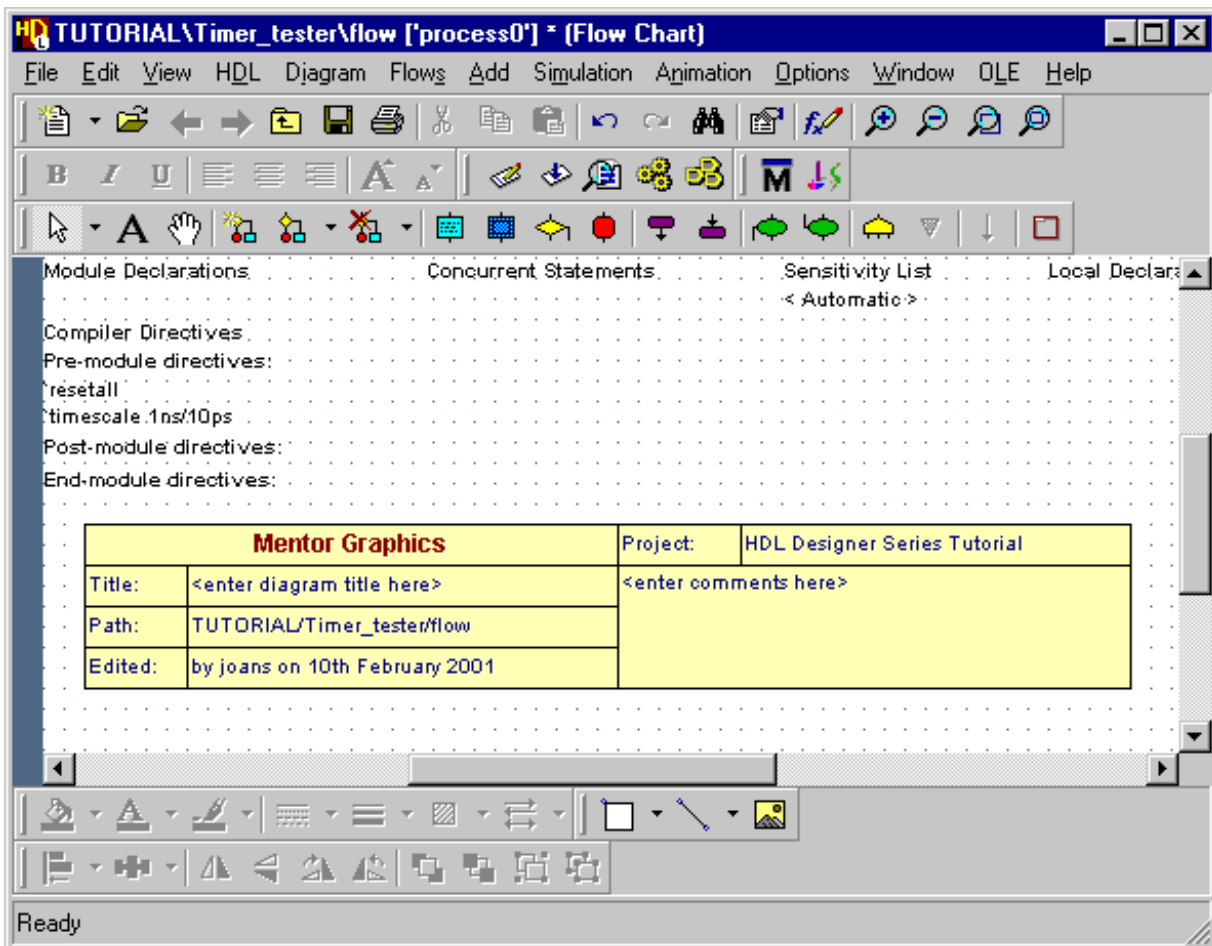
### Introduction

This appendix describes procedures for creating a Verilog flow chart as an alternative to importing the example flow chart. After you [“Create a Test Bench”](#) on page 2-65 the test bench block diagram should look similar to the following picture:



## Create the Tester Flow Chart

Double-click over the *Timer\_tester* block in the test bench block diagram and select **Flow Chart** view in the Open Down Create New View dialog box using the default view name *flow.fc*. A new flow chart (*TUTORIAL\Timer\_tester\flow ['process0']*) is created as a child view of the *TimerTester* block:

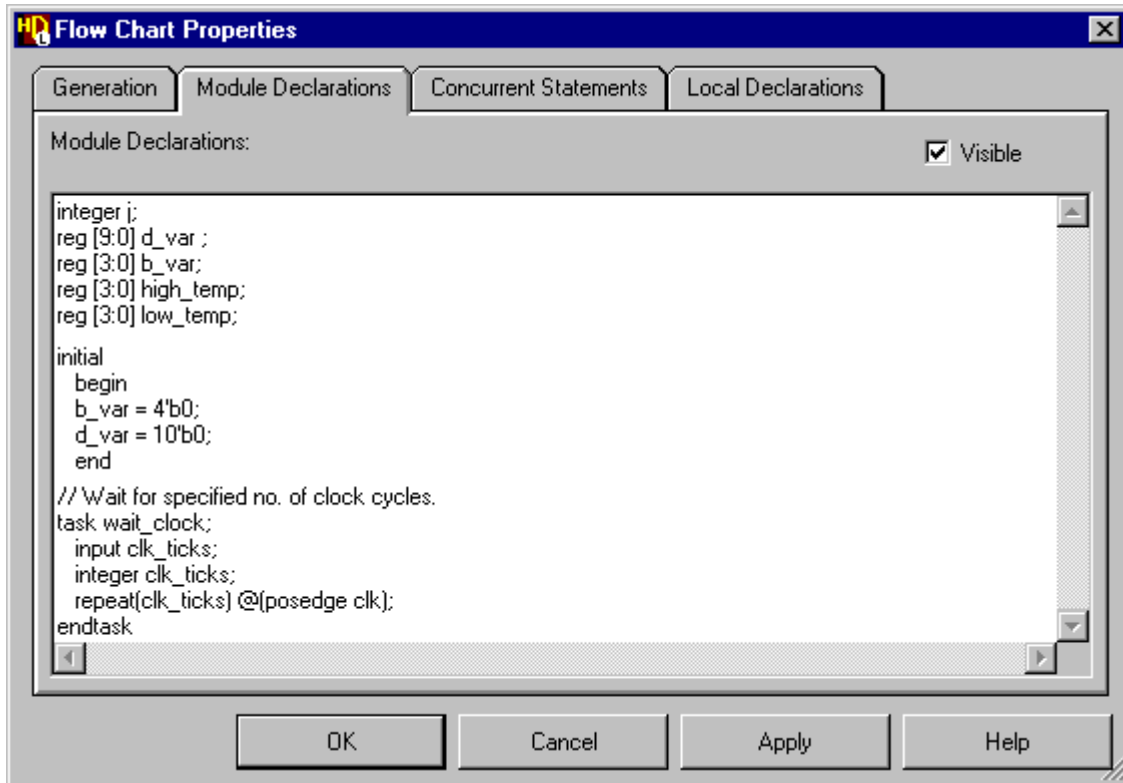


The new flow chart is initialized with the default compiler directives, default title block and labels for module declarations, concurrent statements, sensitivity list and local declarations.

Choose **Rename Flow Chart** from the **Diagram** menu and enter the new name *Monitor*. When you confirm the dialog box, this name replaces the default process name (*process0*) appended to the design unit and view names.

## Set Flow Chart Properties

Double-click with the mouse over the Module Declarations label in the flow chart to display the **Module Declarations** tab of the Flow Chart Properties dialog box.




Enter the following declarations which set up some internal variables and a simple wait procedure:

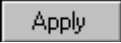
```
integer j;
reg [9:0] d_var ;
reg [3:0] b_var;
reg [3:0] high_temp;
reg [3:0] low_temp;

initial
begin : SetVar
b_var = 4'b0;
d_var = 10'b0;
end
```

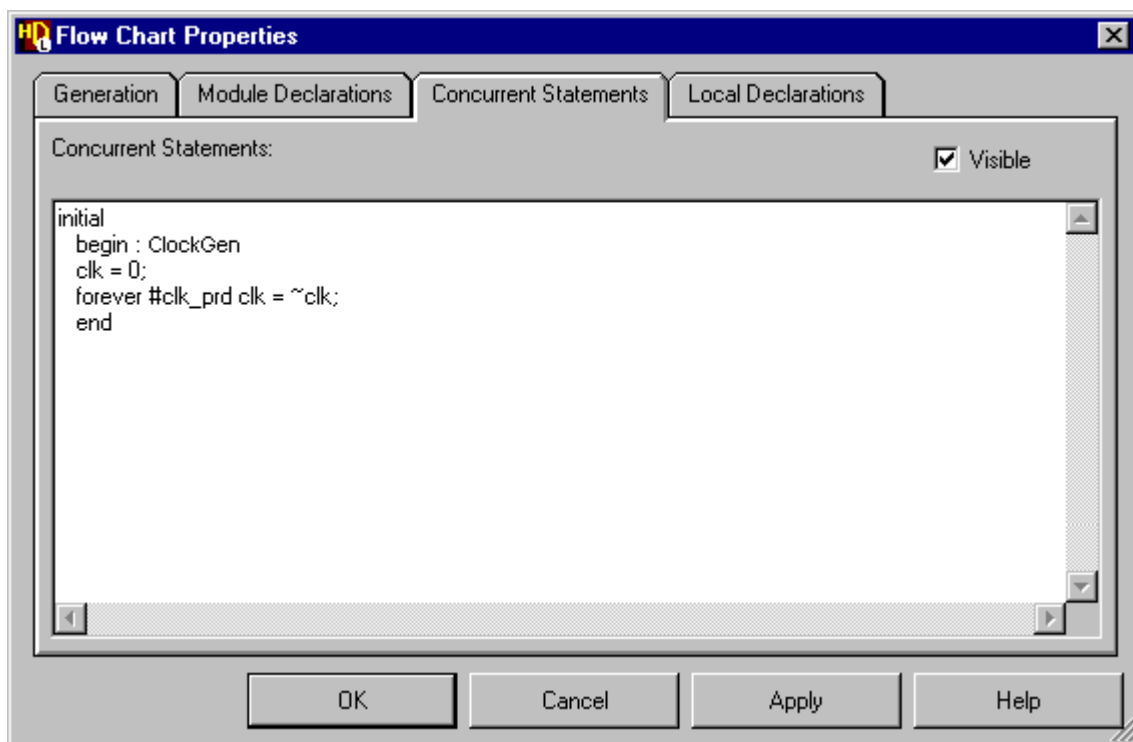
```
// Wait for specified no. of clock cycles.  
task wait_clock;  
    input clk_ticks;  
    integer clk_ticks;  
    repeat(clk_ticks) @(posedge clk);  
endtask
```

The module declarations are included at the top of the module description in the generated HDL and must be valid HDL statements terminated by a semi-colon. Comments can also be included provided each comment is preceded by the appropriate Verilog comment characters (*//*).

 If you use HDL import to recover the *Timer\_tester* flow chart, the *SetVar* initial statements are recovered as a concurrent flow chart view.


Use the  button to display these properties on the flow chart. The HDL syntax is checked on entry and you are warned if any errors are encountered.

Select the **Concurrent Statements** tab.



Enter the following statements which generate a clock signal for the test bench:

```
initial
  begin : ClockGen
    parameter clk_prd = 50;
    clk = 0;
    forever #clk_prd clk = ~clk;
  end
```

Use the  button to display these properties on the flow chart.

Concurrent statements are included between the *end* and *endmodule* statements in the generated HDL for the flow chart and must be valid HDL statements and be terminated by a semi-colon. They are typically used for a concurrent process (such as the clock generator defined by this example).

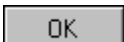


If you use HDL import to recover the *Timer\_tester* flow chart, the *ClkGen* process is recovered as a concurrent flow chart view.

There are no local declarations used in this design. Select the **Local Declarations** tab and uncheck the **Visible** option in order to hide this label on the chart.







Local declarations are included at the top of the initial or always code in the generated HDL for the flow chart and if entered must be valid HDL statements terminated by a semi-colon.

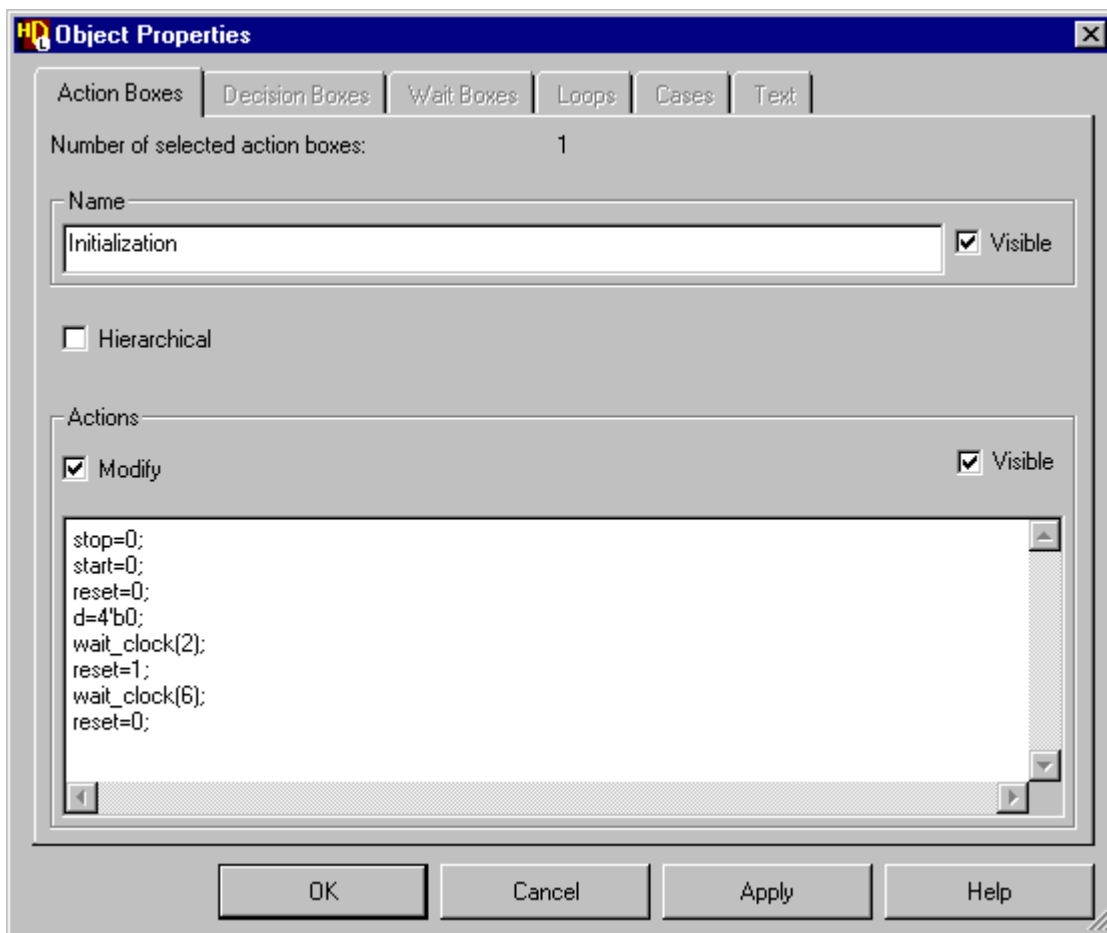
Use the  button to confirm and dismiss the Flow Chart Properties dialog box.

## Add a Start Point and Action Box

Use the  button to add a **start point** near the top of the flow chart.

Use the  button to add an **action box** below the start point and use the  button to add a **flow** connecting it to the start point. To add a flow, click with the mouse over the body of the start point and the action box close to the  markers.

Double-click with the mouse over the action box (or use the  button) to display the **Action Boxes** tab of the Flow Chart Object Properties dialog box.

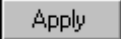


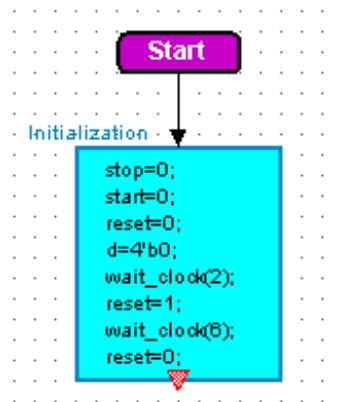
You can change the action box name and enter action statements. However, the name must be unique on the flow chart and cannot be changed when more than one action box is selected.



Change the default name to *Initialization* and enter the following actions:

```
stop=0;
start=0;
reset=0;
d=4'b0;
wait_clock(2);
reset=1;
wait_clock(6);
reset=0;
```

Use the  button to update the flow chart. You may want to resize the action box on the chart (by dragging the handles on the lower edge with the mouse) so that it encloses the actions text.



If you drag one side of an action box, the mid-point is preserved.

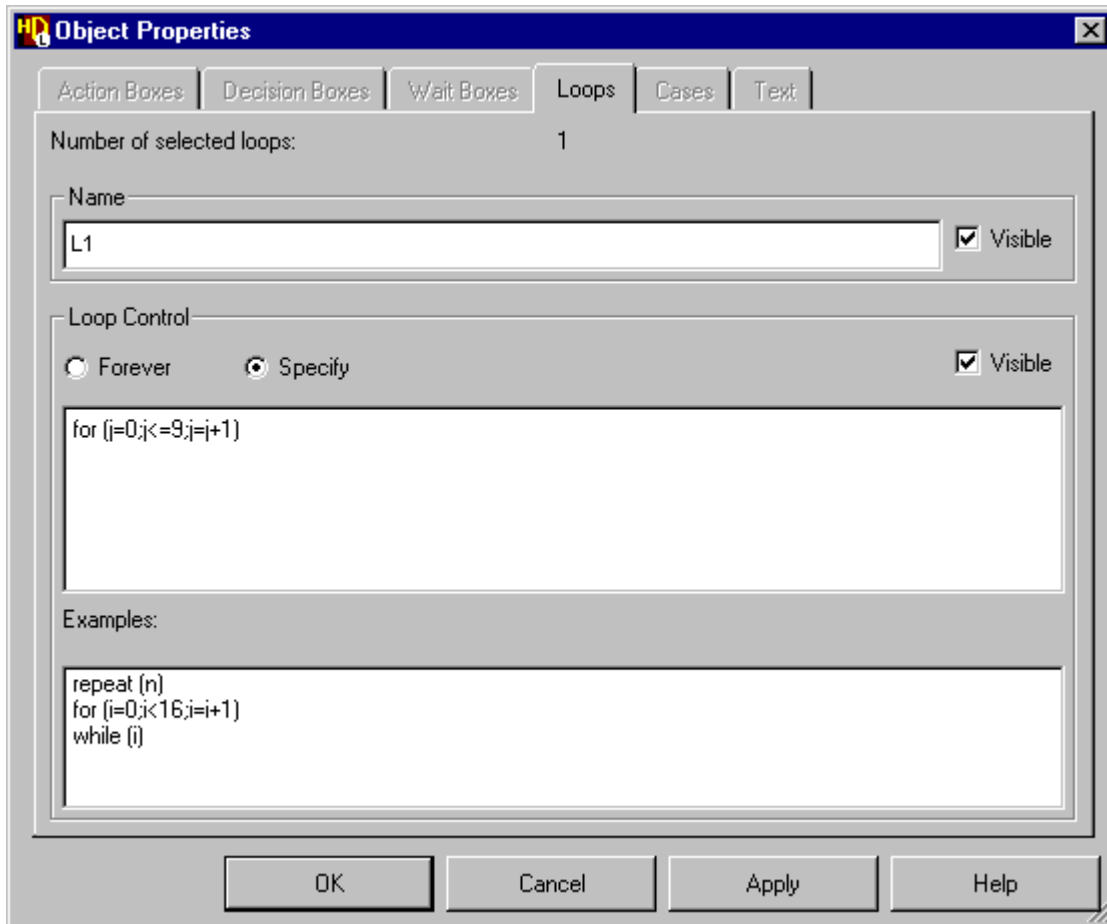
## Add a Loop and an Associated Comment

Use the  button to add a start loop below the *Initialization* action box.


Select the start loop to display the **Loops** tab of the Flow Chart Object Properties dialog box. Choose the **Specify** button and enter the following loop control statement:

```
for (j=0; j<=9; j=j+1)
```

Rename the loop *L1* and use the  button to update the flow chart.

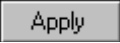




By default, a loop will be repeated forever. However, if you choose the **Specify** option, the dialog box discloses an entry box for loop control statements. Template examples are provided which you can copy into the entry box by clicking on one of the examples with the mouse.

Use the  button to add an action box below the start loop.

Click with the mouse to select the new action box and use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to enter the following actions:

```
d_var=10'b0;
d_var[j]=1;
d=d_var;
wait_clock(4);
if ((high!=b_var)&&(low!=b_var))
$display ("Decoder or Load failure.");
b_var=b_var+1'b1;
```

Change the name of the action box to *Decoder* and use the  button to update the flow chart. Resize the action box so that it encloses the actions text.

Use the  button to add an end loop below the *Decoder* action box and use the  button to connect flows for the loop.

Use the  button to enter text mode. Click in a free space near the loop and enter the following text in the [comment text](#) entry box:

```
Loop through all
possible decoder
values and check
output values
```



You can enter free-format text including line breaks and spaces which will be preserved on the diagram.

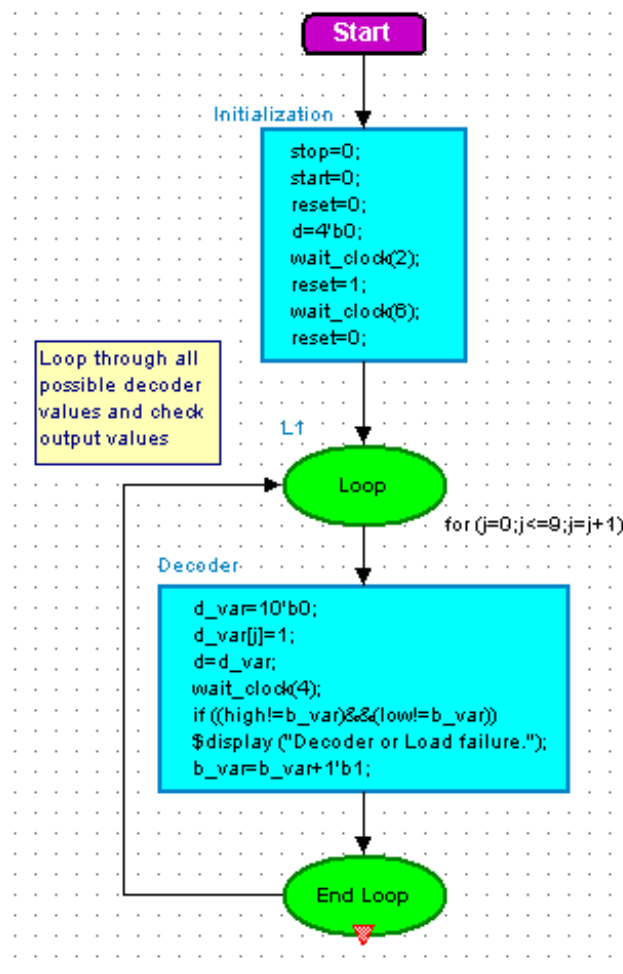
Click the left mouse button outside the text entry box to complete the text. Select the comment text and choose **Include in HDL** from the popup menu. Select **Before Object** from the popup menu. An anchor is attached to the cursor. Click the left mouse button with the cursor over the start loop object to terminate the anchor and associate the notes with the loop.

The comment text will be included immediately before the loop statements in the generated HDL for the flow chart.




If you use HDL import to recover the *Timer\_tester* flow chart, the associated comment is recovered as an actions box above the loop statement.

The flow chart should look similar to the following picture:



If you use HDL import to recover the *Timer\_tester* flow chart, the if statement is recovered as a separate decision box inside the loop statement.

## Add an Action Box

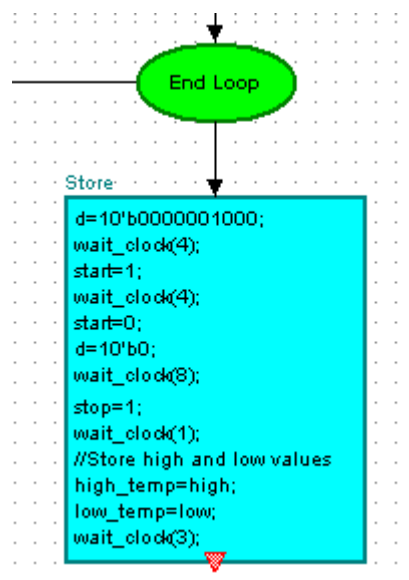
Use the  button to add another action box below the end loop and use the Flow Chart Object Properties dialog box to rename the action box *Store* and apply the following actions:

```
d=10'b0000001000;
wait_clock(4);
start=1;
wait_clock(4);
start=0;
d=10'b0;
wait_clock(8);
stop=1;
wait_clock(1);
//Store high and low values
high_temp=high;
low_temp=low;
wait_clock(3);
```



You can include comment text directly in an actions box by using HDL comment characters. For example the comment text: `//Store high and low values` in the example above.

Use the  button to connect flows between the end loop and action boxes.





## Add a Hierarchical Action Box

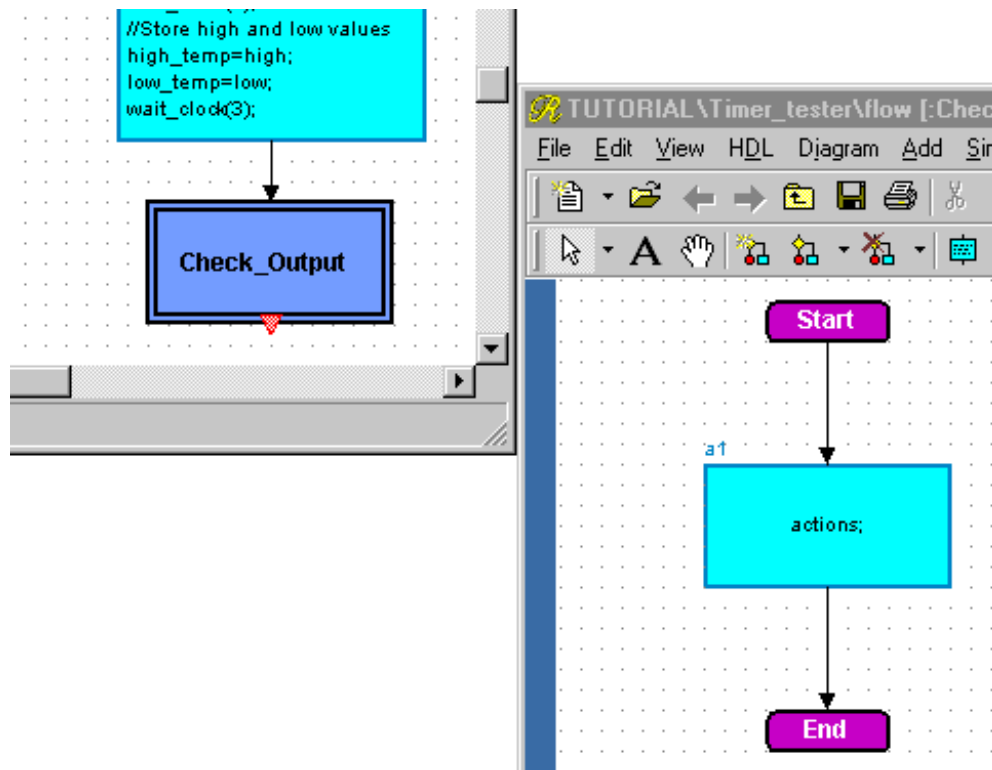
When a flow chart contains a large number of procedural statements it can rapidly become a very large diagram which can be difficult to read or print. However, such a chart can be represented by a number of separate hierarchical charts by using hierarchical action boxes.




Flow chart hierarchy does not effect the generated HDL and is not used if you use HDL import to recover the *Timer\_tester* flow chart.

Use the  button to add a **hierarchical action box** below the *Store* action box and use direct text editing to change the name of the hierarchical action box to *Check\_Output*. Use the  button to connect a flow between the action box and hierarchical action box. Double-click on the *Check\_Output* hierarchical action box to open a new child flow chart (or select the hierarchical action box and choose **Open Down** from the **File** menu).

The child flow chart is initialized as an action box connected by flows to a start point and **end point**.




## Add a Decision Box

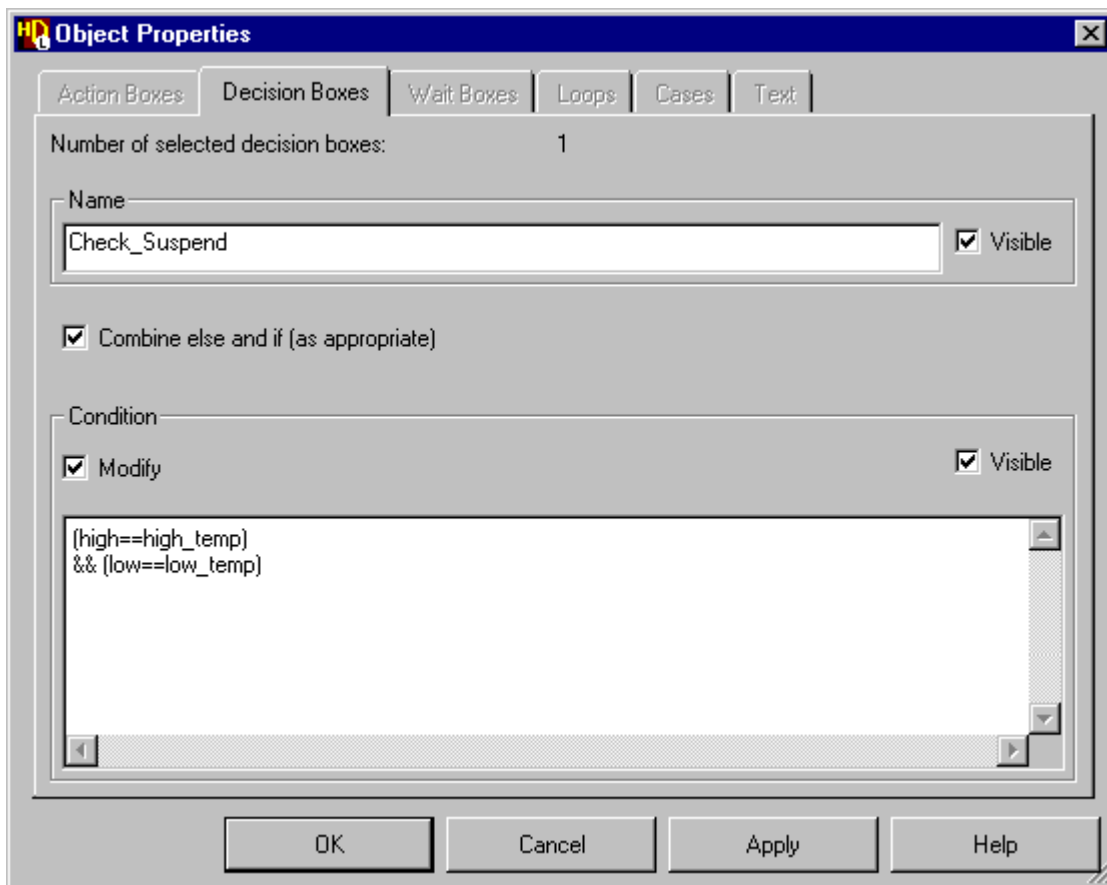
Select the flows connected to the start point and end point in the child flow chart and use the  key to delete them.



You can use the  key with the left mouse button to select both flows.

Use the  button to add a **decision box** below the start point in the *Check\_Output* flow chart and use the **Decision Boxes** tab of the Flow Chart Object Properties dialog box to change the name of the decision box to *Check\_Suspend* and enter the following condition:


```
(high==high_temp)
&& (low==low_temp)
```



Select the existing action box. Use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to change its name to *Pass* and enter the following actions:


### Pass

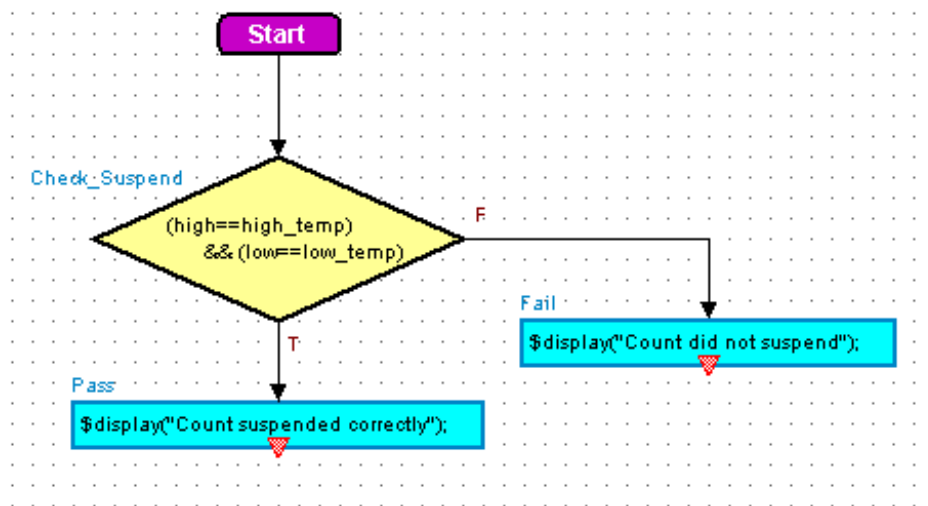
```
$display("Count suspended correctly");
```

Use the  button to add an action box for the False (F) condition. Use the **Action Boxes** tab to change its name to *Fail* and enter the following actions:

### Fail

```
$display("Count did not suspend");
```


Reposition the action boxes by dragging with the mouse and use the  button to connect flows between the start point, decision box and action boxes.





When a decision box object is selected, the popup menu includes an option to swap the True and False outputs.

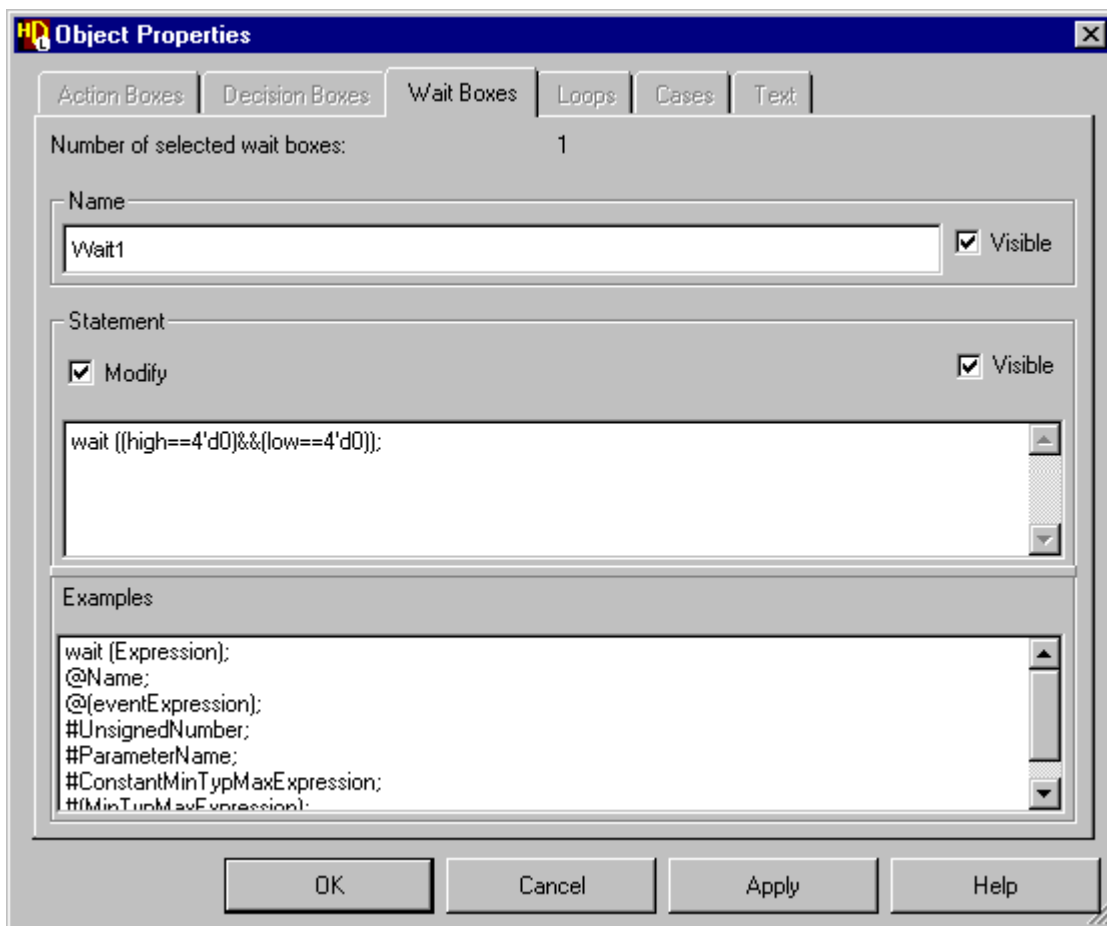


## Add a Wait Box

Use the  button to add an action box below the *Pass* action box and use the **Action Boxes** tab of the Flow Chart Object Properties dialog box to enter the following actions:

```
wait_clock(4);
stop=0;
```

Change the name of this action box to *Continue* and use the  button to add a [wait box](#) below it. Click the mouse over the wait box (or use the  button) to display the **Wait Boxes** tab of the Flow Chart Object Properties dialog box:



Use the **Wait Box** tab of the Flow Chart Object Properties dialog box to rename the wait box *Wait1* and specify the following wait condition:


```
wait ((high==4'd0)&&(low==4'd0));
```



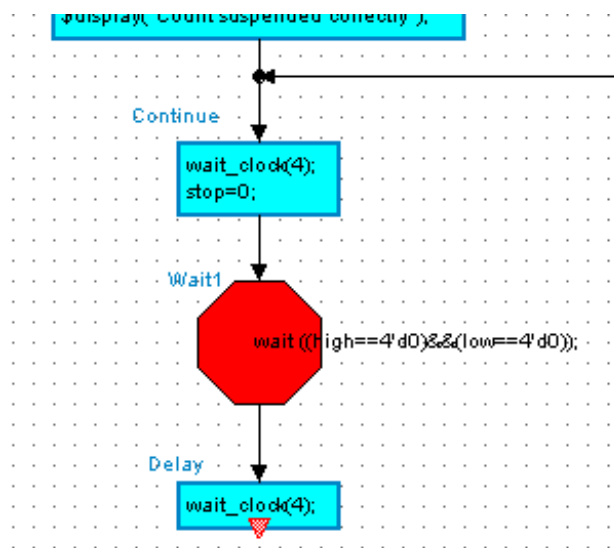
You can click on one of the examples shown in the dialog box to use it as a template. Any valid expression can be used in the wait statement which must be terminated by a semi-colon.

Use the  button to connect flows between the action boxes and wait box.

Notice that a flow join is automatically created when you connect the two flows from the *Pass* and *Fail* action boxes together.


Use the  button to add an action box below the wait box. Use the Flow Chart Object Properties dialog box to change the name of this action box to *Delay* and enter the following action:



```
wait_clock(4);
```



## Copy the Decision Tree

The child hierarchical flow chart can be completed by using similar procedures to those described in the last two topics. However, the next section can be done more quickly by copying the decision tree objects which have already been created.

Drag select the *Check\_Suspend* decision box, plus the *Pass* and *Fail* action boxes and use the  button (or choose **Copy** from the popup or **Edit** menu)

Scroll the window down and click the  mouse button below the *Delay* action box to de-select the objects. Use the  button (or choose **Paste** from the popup or **Edit** menu) to paste a copy of the objects from the clipboard in the middle of the window. Use the mouse to drag the objects to the required position. Notice that each of the pasted objects is automatically given a unique name.

Use the Flow Chart Object Properties dialog box to change the names, conditions and actions where required.

### Check\_Alarm


```
alarm==1
```

### Alarm\_Pass

```
$display("Alarm asserted correctly");
```

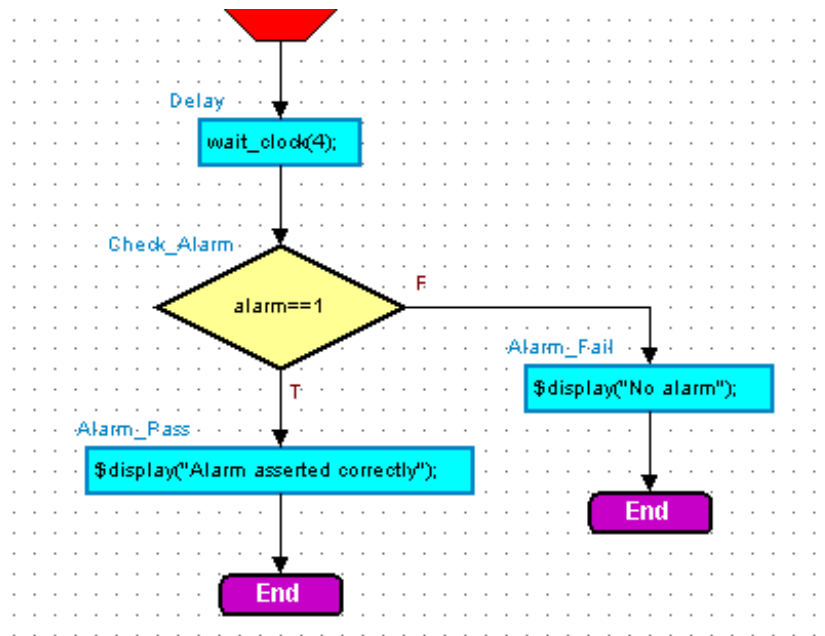
### Alarm\_Fail

```
$display("No alarm");
```



Drag the end point beneath the other objects and use the  button to connect the remaining flows on the child flow chart.



Multiple end points can be used. For example, in the following picture, a separate end point has been attached to the *Alarm\_Pass* and *Alarm\_Fail* action boxes.



## Completing the Flow Chart

Use the  button (or choose **Open Up** from the **File** menu) to display the parent flow chart and the  button to add an action box below the hierarchical action box. Use the Flow Chart Object Properties dialog box to change the name of this action box to *Clear* and enter the following actions:



```

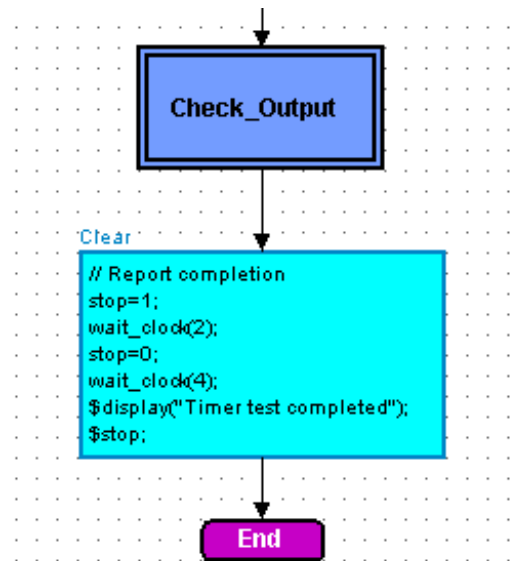
stop=1;
wait_clock(2);
stop=0;
wait_clock(4);
$display("Timer test completed");
$stop;


```



These actions are executed when the timer test has completed successfully and the \$stop system task is used to stop the simulator.

Use the  button to add an end point and use the  button to connect any remaining flows.



Use the  button to save the flow chart.

Return to [“Generate HDL for the Hierarchy”](#) on page 2-62 in the main VHDL tutorial.

