

# Identify<sup>®</sup> Actel Edition Reference

---

March 2011

<http://solvnet.synopsys.com>

**SYNOPSYS<sup>®</sup>**

## Disclaimer of Warranty

Synopsys, Inc. makes no representations or warranties, either expressed or implied, by or with respect to anything in this manual, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose of for any indirect, special or consequential damages.

## Copyright Notice

Copyright © 2011 Synopsys, Inc. All Rights Reserved.

Synopsys software products contain certain confidential information of Synopsys, Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means without the prior written permission of Synopsys, Inc. While every precaution has been taken in the preparation of this book, Synopsys, Inc. assumes no responsibility for errors or omissions. This publication and the features described herein are subject to change without notice.

## Trademarks

### Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CoMET, Design Compiler, DesignWare, Formality, Galaxy Custom Designer, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, METeor, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, the Synplicity logo, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

### Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC,

Galaxy, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM, HSiMP<sup>plus</sup>, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

### **Service Marks (SM)**

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license. ARM and AMBA are registered trademarks of ARM Limited. Saber is a registered trademark of SabreMark Limited Partnership and is used under license. All other product or company names may be trademarks of their respective owners.

## **Restricted Rights Legend**

Government Users: Use, reproduction, release, modification, or disclosure of this commercial computer software, or of any related documentation of any kind, is restricted in accordance with FAR 12.212 and DFARS 227.7202, and further restricted by the Synopsys Software License and Maintenance Agreement. Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043, U. S. A.

Printed in the U.S.A  
March 2011



# Contents

---

## **Chapter 1: Command Reference Introduction**

Manual Conventions .....	8
Text Conventions .....	8
Syntax Conventions .....	8
Symbol Conventions .....	9
Tool Conventions .....	10
File System Conventions .....	10
Design Hierarchy Conventions .....	11

## **Chapter 2: Startup Modes**

Synthesis Tool Pointers .....	16
Configuring the Synthesis Tool .....	16
License Types .....	18
Custom Initialization Script .....	19
Script Locations .....	19
Scripting Priority .....	19
Sample Initialization File .....	20

## **Chapter 3: Command Concepts**

General Commands .....	23
File System Commands .....	24
Design Hierarchy Commands .....	24
Design Instrumentation Commands .....	25
Design Debugging Commands .....	26

**Chapter 4: Alphabetical Command Reference**

activation	29
breakpoints	30
cd	31
chain	32
clear	34
com	34
compile	36
device	37
encryption	39
exit	40
help	40
hierarchy	41
idcode	46
iice	48
instrumentation	55
licenseinfo	56
log	56
project	58
pwd	59
remote_trigger	60
run	61
searchpath	63
setsys	64
show	65
signals	66
source	68
statemachine	69
stop	73
transcript	75
watch	76
waveform	79
write instrumentation	81
write samples	82
write vcd	84
write vhdlmodel	85

## CHAPTER 1

# Command Reference Introduction

---

The Identify<sup>®</sup> Actel Edition tool set consists of the Identify instrumentor and the Identify debugger. These two tools allow you to debug your HDL design:

- In the target system
- At the target speed
- At the VHDL/Verilog RTL Source level

The Identify tool set increases your debugging capabilities of high-end FPGA designs, FPGA-based prototypes, and system-on-a-chip designs. For the first time you will be able to debug live hardware with the internal design visibility you need while using intuitive debugging techniques.

To efficiently use the system and its underlying tools, this manual provides you with a comprehensive listing of all the commands the Identify software accepts. You can access this command information by concept listing or alphabetically.

In addition to easy look-up access to the commands, this manual also contains conventions that help display the command information easily and quickly. These manual conventions organize each command so that the information can be attained easily.

The remainder of this chapter describes:

- [Manual Conventions](#)
- [Tool Conventions](#)

---

# Manual Conventions

There are several conventions this manual uses in order to organize command information effectively. These conventions are:

<b>This convention...</b>	<b>Organizes this information...</b>
Text convention	Text containing paths, directories, command names, and menu selections. All system specific command and path information has its own text convention to make is decipherable throughout the manual.
Syntax convention	Symbols used to separate command examples and other important system text. Syntax conventions include any type of brackets and parentheses that separate commands and functions from the rest of the text.

---

## Text Conventions

There are several text conventions this manual uses to organize command, path and directory information. These conventions or text styles are:

<b>This convention...</b>	<b>Organizes this information...</b>
<b>Bold</b>	command titles
Monospacing type	command, path name, and directory examples
Sans-serif type	commands, literals, and keywords
<i>Italics</i>	variable arguments

---




## Syntax Conventions

There are several conventions this manual uses to convey command syntax. These conventions are:

<b>This convention...</b>	<b>Organizes this information...</b>
<b>bold</b>	Commands and literal arguments entered as shown.
<i>italics</i>	User-defined arguments or example command information.
[ ]	Optional information or arguments for command use. Do not use these brackets with the command within the command line.
...	Items that can be repeated any number of times.
	Choices you can make between two items or commands. The items are located on either side of this symbol.
#	Comments concerning the code or information within the command line.
{ }	Escape characters for search strings; also entered in bold font as a literal in some commands

## Symbol Conventions

This manual contains symbol conventions detailing the tools that use these commands. These symbols are located adjacent to the command name in any of the command listing chapters. These symbols are:

<b>This convention...</b>	<b>Organizes this information...</b>
	Any command that is used in the Identify instrumentor only. This symbol is located underneath any Identify instrumentor command in the command listing chapters.
	Any command that is used in the Identify debugger only. This symbol is located underneath any Identify debugger command in the command listing chapters.
	Any command that is used in both the Identify instrumentor and Identify debugger tools. This symbol is located underneath the commands that have applications in both tools in the command listing chapters.

# Tool Conventions

There are tool concepts you must familiarize yourself with when using the Identify tool set. These concepts help you to decipher structural and HDL related information.

## File System Conventions

The term file system refers to any command that uses file, directory, or path name information in its argument. A file system command must contain specific conventions.

### Path Separator “/”

All file system commands that contain a directory name use only forward slashes, regardless of the underlying operating system:

```
/usr/data.dat
c:/Synplicity/data.dat
```

### Wildcards

A wildcard is a command element you can use to search for specific file information. You can use these wildcards in combination with the file system commands. Conventions for wildcards are as follows:

Syntax	Description
*	Matches any sequence of characters
?	Matches any single character

Square brackets are used in pattern matching as follows:

Syntax	Description
[abcd]	Matches any character in the specified set.
[a-d]	Matches any character in a specified range.

To use square brackets in wildcard specifications, you must delimit the entire name with curly braces {}. For example

```
{ [a-d] 1 }
```

matches any character in the specified range (a-d) preceding the character 1.

## Design Hierarchy Conventions

Design hierarchy refers to the structure of your design. Design hierarchy conventions define a way to refer to objects within the design hierarchy.

The Identify software supports VHDL and Verilog. These languages vary in their hierarchy conventions. The VHDL and Verilog languages contain design units and hierarchies of these design units. In VHDL, these design units are entity/architecture pairs, in Verilog they are modules. VHDL and Verilog design units are organized hierarchically. Each of the following HDL design units creates a new level in the hierarchy:

### VHDL

- The top-level entity
- Architectures
- Component instantiation statements
- Process statements
- Control flow statements: if-then-else, and case
- Subprogram statements
- Block statements

### Verilog

- The top-level module
- Module instantiation statements
- Always statements
- Control flow statements: if-then-else, and case
- Functions and tasks

## Design Hierarchy References

A reference to an element in the design hierarchy consists of a path made up of references to design units (similar to a file reference described earlier). Regardless of the underlying HDL (VHDL or Verilog) the path separator character is always “/”:

```
/inst/reset_n
```

Absolute path names begin with a path separator character. The top-level design unit is represented by the initial “/”. Thus, a port on the top-level design unit would be represented:

```
/port_name
```

The architecture of the top-level VHDL design unit is represented:

```
/arch
```

Relative path names do not start with the path separator, and are relative to the current location in the design hierarchy. Initially, the current location is the top-level design unit, but commands exist that allow you to change the location.

---

**Note:** Design unit and hierarchy information can be case sensitive depending on the HDL language. VHDL names are not case sensitive. In contrast, all Verilog names are case sensitive.

---

## Wildcards

A wildcard is a command element you can use to search for specific design hierarchy information. You can use these wildcards in combination with the design hierarchy commands. Conventions for wildcards are as follows:

Syntax	Description
*	Matches any sequence of characters
?	Matches any single character

Square brackets are used in hierarchy pattern matching as follows:

<b>Syntax</b>	<b>Description</b>
[abcd]	Matches any character in the specified set.
[a-d]	Matches any character in a specified range.

To use square brackets in pattern matching, you must delimit the entire name with curly braces {}. For example

```
{ [a-d] 1 }
```

matches any character in the specified range (a-d) preceding the character 1.



## CHAPTER 2

# Startup Modes

---

The Identify instrumentor and the Identify debugger can each be started in any of three execution modes as outlined below:

- `identify_instrumentor`  
Opens the Identify instrumentor in the graphical interface.
- `identify_instrumentor -f fileName.tcl`  
Runs a Tcl startup file and then opens the Identify instrumentor in the graphical user interface.
- `identify_instrumentor_shell`  
Opens the Identify instrumentor in the shell and/or script mode. If the optional `-version` argument is included, reports the software version without opening the Identify instrumentor.
- `identify_debugger`  
Opens the Identify debugger in the graphical interface.
- `identify_debugger -f fileName.tcl`  
Runs a Tcl startup file and then opens the Identify debugger in the graphical user interface.
- `identify_debugger_shell`  
Opens the Identify debugger in the shell and/or script mode. If the optional `-version` argument is included, reports the software version without opening the Identify debugger.

---

**Note:** Depending on the command shell, you may be required to provide the full path name to the program executable as well as the full path name to the files specified in any of the filename arguments.

---

## Synthesis Tool Pointers

When the Identify instrumentor or the Identify debugger is started from the command line, two additional arguments can be passed to the tool to identify the location and type of the synthesis tool associated with the project.

---

**Note:** The path to the synthesis tool and the tool type can also be defined after the Identify instrumentor or Identify debugger has been started as described in [Command Line Configuration, on page 18](#).

---

The syntax of the two command-line arguments is:

**-synplify\_install** *synthesisToolPath*

**-synplify\_tool** *synplify* | *synplify\_pro* | *synplify\_premier* | *synplify\_premier\_dp*

The arguments are entered on the same command line in any order.

## Configuring the Synthesis Tool

When the Identify instrumentor or Identify debugger is launched from the synthesis tool GUI, the path to the synthesis tool and its tool type are defined. When the Identify instrumentor or Identify debugger is started independently and then opens a synthesis project file, the path to the synthesis tool executable and the tool type must be defined either when starting the Identify instrumentor or Identify debugger (see [Synthesis Tool Pointers](#) above) or after the tool is started as defined in the following subsections.

---

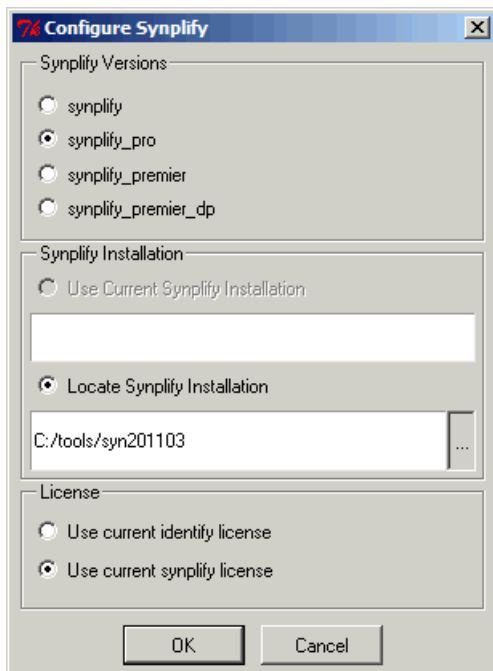
**Note:** The Identify instrumentor uses the synthesis tool compiler to compile the design.

---

## GUI Configuration

To set the path to the synthesis tool from the Identify GUI:

1. Select Options->Configure Synplify from the menu to display the Configure Synplify dialog box.



2. In the dialog box:
  - Select the corresponding synthesis tool radio button in the Synplify Versions section.
  - Select the Locate Synplify Installation radio button and enter the path to the synthesis tool installation in the Synplify Installation section.
  - Select the appropriate license radio button in the License section.
3. Click the OK button to accept the values and close the dialog box.

## Command Line Configuration

The path to the synthesis tool can also be set from the command line with a `set_synplify_configuration` command. The syntax for the command is:

```
set_synplify_configuration -type synToolVersion {-locate installPath | -current}  
-license identify|synplify [-help]
```

In the syntax, *synToolVersion* is `synplify`, `synplify_pro`, `synplify_premier`, or `synplify_premier_dp` and *installPath* is the path to the synthesis tool installation directory.

## License Types

All of the above commands accept an optional `-licensetype` argument to specify a license type other than the default. The license type can be either a vendor-specific license or a full license.

---

**Note:** The `-licensetype` argument is required when initially starting the Identify instrumentor or Identify debugger in the shell mode.

---

The following values are accepted by the `-licensetype` argument:

<code>identinstrumentor</code>	requests a full Identify instrumentor license
<code>identinstrumentor_act</code>	requests an Actel-only Identify instrumentor license
<code>identdebugger</code>	requests a full Identify debugger license
<code>identdebugger_act</code>	requests an Actel-only Identify debugger license

The following examples illustrate using the `-licensetype` argument. The first example opens the Identify instrumentor in the graphical interface with an Actel-only license, and the second example opens the Identify debugger in the shell mode with a full license.

```
idnetify_instrumentor -licensetype identinstrumentor_act  
identify_debugger_shell -licensetype identdebugger
```

To verify the license currently being used by the Identify instrumentor or Identify debugger, enter the command `licenseinfo` in the console window or at the shell prompt.

---

**Note:** Changing the license type with the `-licensetype` argument is valid only for the current session and does not change the default license type defined in the Select available license dialog box.

---

## Custom Initialization Script

The Identify tool set can be customized using a TCL-style initialization script. This script is sourced on tool startup and allows you to define custom procedures and variables, or to program start-up behavior.

### Script Locations

The Identify software first looks for initialization scripts, titled `synrc.tcl`, in the `/etc` directory of the Identify installation path and then in the user's home directory defined by the value of the environment value `USERPROFILE`. On a standard Windows installation, this path is generally represented as:

```
c:/Documents and Settings/userName/synrc.tcl
```

### Scripting Priority

The script is first sourced from the installation directory and then from the home directory. Since the format is TCL, the second user-specific script overrides TCL procedures and variables previously defined.

## Sample Initialization File

A sample `synrc` file is distributed with the software. The file resides in the `/etc` directory of the installation path. The file is named `synrc.template.tcl` and must be renamed to `synrc.tcl` to enable its functionality. This file contains some sample functions and can be used as an example of how to provide a custom waveform viewer for use in the Identify debugger. If you are interested in interfacing and using your own waveform viewer with the Identify debugger, refer to the application note “Interfacing Your Waveform Viewer” available on the Synopsys SolvNet web site.

## CHAPTER 3

# Command Concepts

All commands that the Identify tool set uses are divided into several specific categories. These categories, or concepts, separate the commands in terms of how these commands impact the system and also in terms of the tools within the system that utilize them. These concepts are defined in their respective sections below.

Each section in this chapter also contains a table that lists the commands alphabetically. The tables separate the commands into three different columns of information. These columns are:

- Tool Usage – describes in symbol form the tools that can utilize the specific commands. The symbols are:



Command available only in Identify instrumentor



Command available only in Identify debugger



Command available in both Identify instrumentor and Identify debugger












- Use this command... – lists the command name or command example.
- To do this... – defines the uses of the command and lists command conventions and standards.

This chapter contains:

- [General Commands, on page 23](#)
- [File System Commands, on page 24](#)
- [Design Hierarchy Commands, on page 24](#)
- [Design Instrumentation Commands, on page 25](#)
- [Design Debugging Commands, on page 26](#)



# General Commands

A general command is any command that allows you to control and access aspects of both the Identify instrumentor and Identify debugger. These commands access aspects of these tools, such as accessing the online command help and exiting the program.

	Use this command...	To do this...
	<code>clear</code>	Remove all the console output in the graphical user interface.
	<code>exit</code>	Exit the program and close its window.
	<code>help</code>	Display the online help system and a help topic about a command, if given.
	<code>instrumentation</code>	Manipulate incremental instrumentations.
	<code>licenseinfo</code>	Display product version and license status.
	<code>log</code>	Record commands and their output into a log file.
	<code>project</code>	Create, open, and save projects.
	<code>searchpath</code>	Set a search path to find HDL design files.
	<code>setsys</code>	Manipulate internal customization variables.
	<code>source</code>	Run a TCL script.
	<code>transcript</code>	Record commands into a transcript file.



# File System Commands

File system commands allow you to navigate through the file system on your computer.

	<b>Use this command...</b>	<b>To do this...</b>
	<code>cd</code>	Change the working directory.
	<code>pwd</code>	Display the present working directory








# Design Hierarchy Commands

Design hierarchy handling commands allow you to navigate through the design hierarchy of your HDL design.

	<b>Use this command...</b>	<b>To do this...</b>
	<code>hierarchy</code>	Navigate through the design hierarchy of your HDL design. A find option allows you to search for specific HDL design units within an HDL design.
	<code>show</code>	Display the HDL source code in the source window.















# Design Instrumentation Commands






Design entry and instrumentation commands allow you to manipulate and instrument your HDL design and to insert the IICE.

	Use this command...	To do this...
	<a href="#">breakpoints</a>	Instrument breakpoints as HDL source level trigger conditions of the IICE.
	<a href="#">compile</a>	Import and compile HDL design files.
	<a href="#">device</a>	Describe the type of target device(s) used to implement your design.
	<a href="#">encryption</a>	Set the current password to use before encrypting or decrypting a file.
	<a href="#">iice</a>	Duplicate IICE Configuration dialog box functions including defining sample clock, setting counter width for complex triggering, defining trigger conditions and states, and sampling method.
	<a href="#">signals</a>	Instrument signals for sampling and/or triggering in the IICE.
	<a href="#">write instrumentation</a>	Write the instrumented HDL design files to a specified directory.

# Design Debugging Commands

Design debugging commands allow you to debug your HDL design, interacting with the IICE and analyzing sample data at the RTL HDL source level.

	Use this command...	To do this...
	<a href="#">activation</a>	Save or reload a set of trigger settings
	<a href="#">chain</a>	Set up the JTAG chain of devices to be debugged.
	<a href="#">com</a>	Set up the communication to the on-chip hardware, including cable and port settings.
 	<a href="#">device</a>	Query the type of target device(s) used to implement your design.
 	<a href="#">encryption</a>	Set the current password to use before encrypting or decrypting a file.
	<a href="#">idcode</a>	Set up and maintain a table of device ID codes.
 	<a href="#">iice</a>	Query the IICE Configuration dialog box settings selected during the instrumentation phase including sample clock, counter width for complex triggering, trigger conditions and states, and selected sampling method.
	<a href="#">remote_trigger</a>	Sets/resets events on a specified debugger instantiation, on all debuggers in a multiple debugger configuration, or on specific IICE units in a multi-IICE configuration.
	<a href="#">run</a>	Arm the IICE with currently activated trigger conditions and wait for a trigger. When the trigger occurs, acquire and display the sample data.
	<a href="#">statemachine</a>	Define the behavior of the state machine triggering hardware.
	<a href="#">stop</a>	Activate or deactivate a HDL source level breakpoint.

	<b>Use this command...</b>	<b>To do this...</b>
	<code>watch</code>	Activate or deactivate a HDL source level watchpoint.
	<code>waveform</code>	Implement a waveform viewer
	<code>write samples</code>	Write the sample data in text format.
	<code>write vcd</code>	Write the sample data to a Verilog Change Dump (vcd) format.
	<code>write vhdlmodel</code>	Write the sample data in a VHDL model format.



## CHAPTER 4

# Alphabetical Command Reference

All commands are listed alphabetically in this chapter. Each command contains syntax, argument return values, default values, and examples.

## activation

Allows you to save or reload a set of trigger settings (enabled watchpoints and breakpoints). Including the `-sample` option causes the sample data to be loaded or saved with the trigger settings. If the optional *activationName* argument is included, the named activation is loaded or saved; if *activationName* is omitted, `last_run.adb` is used as the default activation name. The `activation clear` and `activation list` commands clear the current trigger settings and list all of the saved activations for the current instrumentation, respectively.

### Syntax

```
activation load|save [-sample] [activationName]
```

```
activation clear|list
```

### Command Example

```
activation load -sample instr_trial1
```

## breakpoints

Instructs the Identify instrumentor to add or delete special debug logic to or from the specified IICE. This debug logic implements breakpoint-style RTL source level trigger conditions.

### Syntax

```
breakpoints add|delete [-iice iiceID|all] breakpointName [breakpointName ...]
```

### Arguments and Options:

One or more breakpoints can be added/deleted at the same time.

```
add breakpointName [breakpointName ...]
```

```
delete breakpointName [breakpointName ...]
```

A breakpoint name consists of two components:

- The full hierarchical path of the HDL design unit that denotes the underlying control statement of the breakpoint.
- The HDL source code location given by the filename and the line number of the breakpoint.

These two components together ensure that each breakpoint has a unique name for identification purposes.

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the IICE (*iiceID*) where the breakpoint is to be added or deleted. If the argument **all** is specified, the corresponding breakpoint is added to or deleted from each IICE.

### Command Example

```
breakpoints add /struct/u1/case_128/cpu.vhd:120
breakpoints delete -iice trap2 /home/designs/cpu/alu.v:58
```

### See Also

- [stop, on page 73](#)

**cd** 

Changes the present working directory in the file system to a different designated directory.

**Syntax**

```
cd directory
```

**Arguments and Options**

*directory*

Specifies the designated directory name. You must use forward slashes to describe relative and absolute path names. On a Windows-based platform, the directory may include a drive letter followed by a colon.

**Command Example**

```
cd c:/temp  
cd ../homedirs/adam
```

**See Also**

- [pwd, on page 59](#)

## chain

Sets up and manipulates the JTAG chain of devices. Because more than one device can be connected in a JTAG chain, the commands allows you to setup the JTAG chain representation in the Identify debugger to select the particular device to be debugged.

### Syntax

**chain add** *deviceName instructionRegisterWidth*

**chain clear**

**chain info** [-raw|-active]

**chain replace** *position chipID instructionRegisterLength*

**chain select** *chipID*

### Arguments and Options

**add** *deviceName instructionRegisterWidth*

Creates and labels a device and assigns that device with an instruction register width. Every device attached to the JTAG must be identified by a unique name. This device name can include any alpha-numeric characters. Spaces and other characters cannot be used.

The instruction register is an N-bit register that holds the OPCODE for the JTAG controller. Every device has a specific instruction register width, which can be found in the device's Data Book.

#### **clear**

Deletes the current chain description.

#### **info**

Displays the chain description.

**info -raw**

Returns a machine readable JTAG chain description. The chain is represented by a Tcl list of chain elements where each element is a two item Tcl list specifying the device name and instruction register width. Example:

```
{{device_a 8} {device_b 10}}
```

**info -active**

Returns the name of the device that is currently selected for debugging.

**chain replace *position chipID instructionRegisterLength***

Changes the name or register length of a device that has been previously defined using the chain add command. In the command syntax, *position* is the value shown by the chain info command for the device to be replaced.

**select *deviceName***

Selects a device for system debugging. Only devices added and labeled using chain add can be selected.

**Command Example**

```
chain add fpga 5
chain select fpga
chain info -active
chain replace 1 new_fpga 8
```

**See Also**

- [device](#), on page 37
- [com](#), on page 34

## clear

Removes all the console output in the graphical user interface.

### Syntax

```
clear
```

### Arguments and Options

none

---

**Note:** This command is only supported in the graphical modes.

---

## com

Sets up and manipulates communication settings between the Identify debugger and the Intelligent In-Circuit Emulator (IICE).

### Syntax

```
com cabletype [type]
```

```
com cableoptions option [value]
```

```
com check
```

```
com port [lpt1|lpt2|lpt3|lpt4]
```

### Arguments and Options

**cabletype** [type]

Describes the type of cable connecting the system to the hardware being analyzed. The supported cable types are `Actel_BuiltinJTAG`, which is compatible with the `flashPro`, `flashProLite`, and `flashPro3` cables, and `demo`.

**cableoptions** *option* [*value*]

Specifies or reports cable-specific option settings:

**flashPro\_trst** [*string*] – specifies the setting of the TRST (tristate) pin. Accepted values (*string*) are off, toggle, low, and hi; the default is off.

**flashProLite\_trst** [*string*] – specifies the setting of the TRST (tristate) pin. Accepted values (*string*) are off, toggle, low, and hi; the default is off.

**flashPro3\_trst** [*string*] – specifies the setting of the TRST (tristate) pin. Accepted values (*string*) are off, toggle, low, and hi; the default is off.

**check**

Performs a connectivity check on the JTAG cable connection.

**port** [*lpt1|lpt2|lpt3|lpt4*]

Specifies the host computer parallel port to which the JTAG cable is connected. The supported ports are lpt1, lpt2, lpt3, and lpt4.

**Command Example**

```
com cabletype flashPro
com cableoptions flashPro_trst toggle
com check
com port lpt1
```

**See Also**

- [chain](#), on page 32

## compile

Prints a list of the design files and the respective order in which they are read.

### Syntax

`compile list`

### Arguments and Options

`list [-vhdl|-verilog]`

Prints a list of the design files and the respective order in which they are read. If the `-vhdl` or `-verilog` option is included, limits the list to only the specified file type.

### Command Example

```
compile list -vhdl
```

### See Also

- [searchpath](#), on page 63

## device

Defines device-specific parameters used to implement the instrumented HDL design.

### Syntax

**device estimate**

**device jtagport [builtin|soft]**

**device technologydefinitions [0|1]**

### Arguments and Options

**estimate [-raw]**

Reports total estimated area requirements for all IICE configurations (available only in Identify instrumentor). The `-raw` option generates the report in a machine-readable format.

**jtagport [builtin|soft]**

Determines if the built-in JTAG port of the target device is used for the IICE connection or if the Synopsys test port is used. Selection can only be set in the Identify instrumentor. With no argument specified, the current setting is displayed. The following selections are available:

#### **builtin**

Specifies that the JTAG port built into the target device is the port used. No extra user pin is required. This is the default value when the device family specified is other than generic.

#### **soft**

Specifies that the IICE communicates through a JTAG TAP controller that is automatically inserted by the Identify instrumentor. The Synopsys FPGA JTAG port requires four extra user pins.

**technologydefinitions [0|1]**

Disables/enables the generation of black boxes for undefined module definitions. This option is available only in Identify instrumentor and is enabled by default.

---

**Note:** Valid values must be set for these options before you instrument your design.

---

### **Command Example**

```
device jtagport builtin
```

### **See Also**

- [chain](#), on page 32
- [com](#), on page 34

## encryption

Sets the current password to use before encrypting or decrypting a file. In the Identify instrumentor, this command sets the password to be used when writing out an encrypted file with the `write instrumentation` command. In the Identify debugger, this command is used to set the password to enable encrypted files to be displayed.

---

**Note:** Setting the password with this command displays the password on the screen and in any log files that you create. If this is a concern, use only the graphical interface when instrumenting and debugging designs that use the encryption feature.

---

### Syntax

```
encryption set_passwd password
```

### Arguments and Options

**set\_passwd** *password*

The `set_passwd` argument requires a single string (*password*) entry. The new password is stored for decrypting/encrypting until it is changed or until the Identify instrumentor or Identify debugger is shut down.

---

**Note:** Passwords are the user's responsibility; Synopsys cannot recreate a lost or forgotten password.

---

### Command Example

```
encryption set_passwd xyzyz
```

### See Also

- [write instrumentation, on page 81](#)
- [project, on page 58](#)

## exit

Exits the program and closes the window.

### Syntax

`exit`

### Arguments and Options

None

### Command Example

```
exit
```

## help

Displays the online help system and a help topic about a command.

### Syntax

`help [commandName]`

### Arguments and Options

*commandName*

Displays help text about the specified command. If the *commandName* argument is omitted, the help texts for all commands are printed to the screen.

## hierarchy

Navigates through the design hierarchy and shows design and hierarchy element in the HDL design. These design elements include the following types, depending on the HDL language the HDL design has been written in:

- Entity – VHDL design unit type.
- Module – Verilog design unit type.
- Instance – VHDL or Verilog design unit type.

### Syntax

**hierarchy add** [*options*] *element* [*element* ...]

**hierarchy cd** *hierarchyPath*

**hierarchy delete** [*options*] *element* [*element* ...]

**hierarchy find** [*options*] [*hierarchyPath*]

**hierarchy ls** [-long] [-recursive] [-all] [*hierarchyPath*]

**hierarchy pwd**

**hierarchy toplevel**

### Arguments and Options

**add** [*options*] *element* [*element* ...]

Connects all signals or breakpoints in the specified hierarchical element to the IICE. The add argument applies only to the Identify instrumentor.

The following **add** argument options are available:

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify which IICE (*iiceID*) to connect. If the argument **all** is specified, the signals or breakpoints are connected to each IICE.

**-sample**

Connects all signals in the specified hierarchical element to the IICE sample buffer.

**-trigger**

Connects all signals in the specified hierarchical element to the IICE trigger logic.

**-breakpoint**

Connects all breakpoints in the specified hierarchical element to the IICE.

---

**Note:** The **-sample**, **-trigger**, and **-breakpoint** options can be combined in a single **add** argument.

---

**cd *hierarchyPath***

Changes the current design hierarchy to the one specified by *hierarchyPath*. Either a relative or an absolute hierarchical path name can be used.

cd /

Changes the current design hierarchy to the top level of the hierarchy.

cd ..

Changes the current design hierarchy to next higher level.

**delete [*options*] *element* [*element* ...]**

Disconnects all signals or breakpoints in the specified hierarchical element from the IICE. The **delete** argument applies only to the Identify instrumentor.

The following **delete** argument options are available:

**-iice** *iiceID|all*

Used when more than one IICE is defined to specify which IICE (*iiceID*) to disconnect. If the argument **all** is specified, the signals or breakpoints are disconnected from each IICE.

**-signal**

Disconnects all sample and trigger signals in the specified hierarchical element from the IICE .

**-breakpoint**

Disconnects all breakpoints in the specified hierarchical element from the IICE.

---

**Note:** The **-signal** and **-breakpoint** options can be combined in a single **add** argument.

---

**find** [*options*] [*hierarchyPath*]

Searches for specific HDL design units and lists those elements. Use this command to locate specified design units in the compiled HDL design file. The search is started from the specified hierarchical path. If you do not provide *hierarchyPath*, the search starts from the current working hierarchy.

The following **find** options are available:

**-iice** *iiceID|all*

Used when more than one IICE is defined to specify the IICE (*iiceID*) to be searched. If the argument **all** is specified, each IICE is searched.

**-name** *elementName*

The HDL element name to be located.

**-noequiv**

Limits the search to named path only and does not search equivalent paths.

**-type instance|breakpoint|signal|\***

The type of HDL element for the target search. If \* is entered, search includes all elements.

**-ls**

Prints verbose information for each HDL element found.

**-stat status|\***

Serves as a filter to search for an HDL element with a specific instrumentation status. If \* is entered, any instrumentation status is included in the search. The *status* argument takes the following options:

- **disabled** – limits search to disabled watchpoints, breakpoints and other disabled HDL design units (available only in Identify debugger).
- **enabled** – limits search to enabled watchpoints, breakpoints, and other enabled HDL design units (available only in Identify debugger).
- **instrumented** – limits search to the sampling clock, and watchpoints and breakpoints that have been marked as instrumented (available only in Identify instrumentor).
- **not-instrumented** – limits search to watchpoints and breakpoints that have not been instrumented (available only in Identify instrumentor).
- **sample\_only** – limits search to sample-only watchpoints (available only in Identify instrumentor).
- **trigger\_only** – limits search to trigger-only watchpoints (available only in Identify instrumentor).

**-maxdepth integer**

Limits search to a maximum depth within the hierarchy tree.

**-all**

Lists “hidden” HDL design units, such as signals/breakpoints within dead code or, in the Identify debugger, breakpoints that were not instrumented. By default, HDL elements with enabled status are searched.

**ls** [-long] [-recursive] [-all] [*hierarchyPath*]

Displays all information about the HDL design units within the current design hierarchy. You can display this design unit information in a long listing using the `-long` option or you can display this information recursively using the `-recursive` option. The `-all` option shows all HDL elements including hidden elements.

**pwd**

Lists the current HDL design hierarchy.

**toplevel**

Shows top-level hierarchy name.

### Command Example

```
hierarchy cd ..
hierarchy cd /top/u1/arui
hierarchy ls -recursive
hierarchy find -type breakpoint -stat instrumented
```

### See Also

- [show](#), on page 65

## idcode

Sets up and maintains a table of device ID codes. The ID code information is used for auto-detection of the devices on the JTAG chain during debugging. If the chain can be successfully detected, you do not need to manually specify the chain using the chain command.

### Syntax

```
idcode add [-quiet] idcode deviceName instructionRegisterWidth
```

```
idcode clear
```

```
idcode info -raw
```

### Arguments and Options

```
add [-quiet] idcode deviceName instructionRegisterWidth
```

Creates an entry in the device table for a given device.

The *idcode* argument should be a binary representation of a 32-bit number in the form of a string. The string can contain 'x' entries for bits that are irrelevant.

The *deviceName* argument can be any descriptive string. The string must be quoted if it includes blanks.

The *instructionRegisterWidth* argument takes an integer value. Every device has a specific instruction register width, which can be found in the device's Data Book.

The -quiet option adds the device, but does not display a user notification.

#### **clear**

Deletes the entire ID code table.

**info [-raw]**

Returns a description of the device table. The table is represented by a Tcl list of device elements where each element is a three item Tcl list specifying the ID code, device name, and instruction register width. Example:

```
{11001100110011001100110011001100 device_a 8}  
{00001100110011001100110011001111 device_b 10}
```

The optional `-raw` option generates the description in a machine-readable format.

**Command Example**

```
idcode add 0010000000111000100010001000 device_type 8  
idcode add -quiet 0010000000111000100010001000 "device type" 8  
idcode clear
```

**See Also**

- [device](#), on page 37
- [chain](#), on page 32

**iice** 

Duplicates the functionality of the IICE Configuration dialog box.

**Syntax**

```
iice clock|controller|current|delete|info|list|new|rename|sampler [option]
```

**Arguments and Options**

```
iice clock [options] [signalName]
```

Defines the signal to be used for the IICE sample clock. The *signalName* is the full hierarchical path name to the signal. You can select any signal within the HDL design as the sample clock. However, this signal cannot be sampled itself while used as the sample clock. This option can only be used during instrumentation. If *signalName* is not specified, the option returns the name of the IICE clock.

**-edge positive|negative**

Specifies the active edge of the clock (positive or negative) when an IICE sample clock is specified. The **-edge** option is only available in the Identify instrumentor; the default edge is rising (positive).

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify/report the controller parameters for the specified IICE (*iiceID*). If the argument **all** is specified, the controller parameters apply to each IICE.

```
iice controller [options] [none|counter|statemachine]
```

Specifies IICE controller configuration; simple triggering (**none**), complex triggering (**counter**), or **statemachine**. The following options are supported:

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify/report the controller parameters for the specified IICE (*iiceID*). If the argument **all** is specified, the controller parameters apply to each IICE.

**-countermode [events|cycles|watchdog|pulsewidth]**

Selects the complex counter mode. The value  $n$  referenced below is the value set by the `countvalue` option (applies only to Identify debugger).

**events**

Stops sampling after the trigger condition occurs for the  $n+1$ 'th time. This is the default value for `-countermode`.

**cycles**

Stops sampling  $n$  cycles after the trigger condition occurs.

**watchdog**

Stops sampling if the trigger condition does not occur for  $n$  consecutive cycles.

**pulsewidth**

Stops sampling when the trigger condition has met  $n$  consecutive cycles. The number  $n$  is controlled by the current setting of `countvalue`.

If no argument is given, the current mode is returned.

**-counterval *unsignedInteger***

Sets a value for the complex counter (applies only to Identify debugger).

*unsignedInteger*

Load the given value into the complex counter. This value must fit into the complex counter width as defined in the Identify instrumentor. The default value for the complex counter is 0 which disables the counter.

If no value is given, the current setting is returned.

**-counterwidth *integer***

Instruments a versatile counter of variable size for complex triggering (applies only to Identify instrumentor). If no argument is given, the command shows the current counter width. An integer parameter in the range between 1 and 32 specifies a new counterwidth. 0 suppresses the creation of any counter. All other values are invalid. The default value for the counterwidth is 16 (the IICE contains a 16-bit complex counter).

**-triggerconditions** *integer*

Used when instrumenting a design for state-machine triggering (applies only to Identify instrumentor). This command specifies the number of trigger conditions to be available for state-machine triggering.

If no argument is given, the command shows the current patterntree setting. An integer parameter greater than zero specifies a new number of trigger conditions. The default value is 1.

This option is a critical setting with respect to instrumentation cost. Choosing a trigger setup with the minimum amount of trigger conditions is recommended to reduce resource usage in the instrumentation. Choosing a trigger-condition value greater than 1 requires that multiple trigger states be created. Use the `triggerstates` option to specify the desired number of states.

**-triggerstates** [*integer*]

Used when instrumenting a design to use state-machine triggering (applies only to Identify instrumentor). This option specifies the maximum number of states instrumented in the state machine.

If no argument is given, the option shows the current `triggerstates` setting. An integer parameter greater than zero specifies a new number of states. The default value is "1".

**-exporttrigger** 0|1

Determines if the master trigger signal of the IICE hardware is exported to the top-level of the instrumented design (applies only to Identify instrumentor). Enables (1) or disables (0) the creation of a trigger port. Export trigger port creation is disabled by default.

**-importtrigger** *integer*

Determines if the master trigger signal of the active IICE hardware includes any triggers received from external sources (applies only to Identify instrumentor). Specifying a value between 1 and 8 creates a corresponding number of input ports.

---

**Note:** When using an external trigger, the pin assignment for the corresponding input port must be defined in the synthesis or place and route tool.

---

**-crosstrigger 0|1 [-iice *iiceID*]**

Enables (1) or disables an IICE to include trigger signals from other IICE units when determining its trigger condition (applies only to Identify instrumentor). If the *-iice* argument is omitted, the command applies to the current IICE.

**-crosstriggermode disabled|any|all|after -crosstriggeriice *iiceID* |all**

Determines the trigger conditions in the Identify debugger when the IICE controller is set to simple or complex-counter triggering. The following options are supported:

**disabled**

Destination IICE triggers normally (triggers from source IICE units are ignored).

**any**

Destination IICE triggers when any source IICE triggers or on its own internal trigger.

**all**

Trigger occurs when all events, irrespective of order, occur at all IICE units including local IICE unit.

**after -crosstriggeriice *iiceID* |all**

Trigger occurs after source IICE triggers coincident with next destination IICE trigger. The *-crosstriggeriice* argument specifies a specific source IICE unit (*iiceID*) or all source IICE units (*all*).

**-triggerinmode disabled|any|all|after -triggerin {*triggerInID* |all}****disabled**

Causes the Identify debugger to ignore all external trigger-in sources.

**any**

Triggering occurs coincident with any of the external trigger-in sources or the internal IICE trigger becoming active.

**all**

Triggering occurs coincident with all of the external trigger-in sources and the internal IICE trigger becoming active.

### **after -triggerin**

Triggering occurs coincident with the following conditions according to the -triggerin setting: when *triggerInID* is specified, triggering occurs after the specified external trigger coincident with the internal trigger; when all is specified, triggering occurs after all external triggers occur coincident with the internal trigger.

### **iice current [*iiceID*]**

Used when more than one IICE is defined to select the active IICE (*iiceID*). If the *iiceID* argument is omitted, reports the ID of the currently active IICE. Note that *iiceID* is case sensitive.

### **iice delete *iiceID***

Deletes the specified IICE (*iiceID*). The *iice delete* command is only available in the Identify instrumentor.

### **iice info [*iiceID*]**

Reports the status of the specified IICE (*iiceID*). If the *iiceID* argument is omitted, reports the status of the currently active IICE.

### **iice list**

Lists the IDs (names) of each defined IICE.

### **iice new [*iiceID*]**

Creates a new IICE with the name *iiceID*. If the *iiceID* argument is omitted, the new IICE is named IICE\_ *n* where *n* is the next sequential integer. The *iice new* command is only available in the Identify instrumentor

### **iice rename *iiceID***

Renames the currently active IICE to the name specified (*iiceID*). The *iice rename* command is only available in the Identify instrumentor

**iice sampler** [*options*]

The following options are supported:

**-iice** *iiceID*|all

Used when more than one IICE is defined to specify/report the IICE sampler parameters for the specified IICE (*iiceID*). If the argument all is specified, the IICE sampler parameters apply to each IICE.

**-always\_armed** 0|1

Enables/disables always-armed sampling. When enabled, the Identify instrumentor saves the sample buffer for the most recent trigger and waits for the next trigger or until interrupted. With always-armed sampling, a snapshot is taken each time the trigger condition becomes true so that you always acquire the data associated with the last trigger condition prior to the interrupt. Using always-armed sampling includes a minimal area and clock-speed penalty.

**-depth** *depthValue*

Changes the default sample depth of the IICE sample buffer to an assigned value *depthValue*. This option can only be used during instrumentation. The default setting for *depthValue* is 128.

**-qualified\_sampling** [0|1]

Enables/disables qualified sampling. When enabled (1), causes the Identify instrumentor to build an IICE block that is capable of performing qualified sampling. With qualified sampling, one data value is sampled each time the trigger condition is true so that you can follow the operation of the design over a longer period of time (for example, you can observe the addresses in a number of bus cycles by sampling only one value for each bus cycle instead of a full trace). Using qualified sampling includes a minimal area and clock-speed penalty.

**-samplemode** [normal|qualified\_fill|qualified\_intr|always\_armed]

Selects the trigger mode (applies only to Identify debugger).

**qualified\_fill**

Performs qualified sampling until the buffer is full.

**qualified\_intr**

Performs qualified sampling until interrupted.

**always\_armed**

Always-on triggering.

**-triggertime [early|middle|late]**

Controls how a detected trigger affects data sampling (applies only to Identify debugger).

**early**

Approximately 10 percent of the sample data is pre-trigger and approximately 90 percent is post-trigger.

**middle**

Approximately 50 percent of the sample data is pre-trigger and approximately 50 percent is post-trigger. This is the default sample trigger.

**late**

Approximately 90 percent of the sample data is pre-trigger and approximately 10 percent is post-trigger.

**Command Example**

```
iice clock -edge falling clk2  
iice controller -counterwidth 8 statemachine  
iice current IICE_2  
iice sampler -triggertime late
```

## instrumentation

Manipulates incremental instrumentations.

### Syntax

**instrumentation info** [-raw] *name*

**instrumentation list**

**instrumentation load** *name*

**instrumentation save**

### Arguments and Options

**info** [*options*] *name*

Shows information about the specified instrumentation(s). If the -raw option is included, returns information in a machine readable format

**list**

Lists the existing instrumentations (applies only to Identify instrumentor).

**load** *name*

Loads an existing instrumentation into the instrumentor.

**save**

Saves the current instrumentation settings (applies only to Identify instrumentor).

### Command Example

```
instrumentation load instr_2
```

## licenseinfo

Displays information about the product version and license status.

### Syntax

```
licenseinfo
```

## log

Allows logging the console output in the graphical user interface to a file.

### Syntax

```
log fileName|on|off
```

### Arguments and Options

*fileName*

Starts logging to the specified file

**on**

Starts logging to the last specified file or to the default files `syn_di.log` or `syn_hhd.log`.

**off**

Stops logging.

### Command Example

```
log on  
log off  
log mylog.log
```

### See Also

- [transcript](#), on page 75

---

**Note:** This command is not supported in the command line tools. Use the operating system capability to pipe the console input into a file.

---

## project

Displays project information and opens existing projects.

### Syntax

```
project import SynopsysFPGAprojectFile
```

```
project name [-path]
```

```
project open [-password password] [fileName]
```

### Arguments and Options

```
import SynopsysFPGAprojectFile
```

Performs a simple import of a Synopsys FPGA synthesis project (*prj*) file by extracting the design files, the device technology, and the design top level. This data is used to create an Identify instrumentor project (applies only to Identify instrumentor). After extracting the files, the design is automatically compiled.

```
name [-path]
```

Returns the name of the current project. If the *-path* option is specified, includes the full path to the project.

```
open [-password password] [fileName]
```

Opens a specified project.

```
-password password
```

Specifies the password to use to decrypt an encrypted source file (applies only to Identify debugger). Note that setting the password with this command displays the password on the screen and in any log files that you create. If this is a concern, use only the graphical interface when instrumenting and debugging designs that use the encryption feature.

## Command Example

```
project import C:/space/designs/mydesign.prj
project open -reapply demo_design.prj
project open -password xyzzy demo_design.prj
```

## See Also

- [encryption, on page 39](#)

## pwd

Displays the present working directory.

## Syntax

```
pwd
```

## See Also

- [cd, on page 31](#)

## remote\_trigger

Triggers the event (stops data collection and downloads data).

### Syntax

```
remote_trigger [-all | -info | -pid processID | -iice iiceID ]
```

```
remote_trigger -set | -reset [ -pid processID | -iice iiceID ]
```

### Arguments and Options

#### **-all**

Triggers the event for every IICE in all debugger instantiations on the corresponding machine.

#### **-info**

Lists the names of the triggers in the current debugger instantiation.

#### **-pid *processID***

Triggers every IICE on the debugger instantiation identified by *processID*. To identify the process ID of the active debugger instantiation, enter *pid* at the command prompt. The default is to trigger every IICE in all debugger instantiations (-all).

#### **-iice *iiceID***

Triggers the event only on the specified IICE in the current debugger instantiation. The default is to trigger every IICE in all debugger instantiations (-all).

#### **-set [-pid *processID* | -iice *iiceID*]**

Sets the trigger. If the -pid argument is specified, sets the trigger on every IICE on the debugger instantiation identified by *processID*; if the -iice argument is specified, sets the trigger only on the IICE unit specified by *iiceID*.

**-reset** [-pid *processID* | -iice *iiceID*]

Clears the trigger. If the -pid argument is specified, resets the trigger on every IICE on the debugger instantiation identified by *processID*; if the -iice argument is specified, resets the trigger only on the IICE unit specified by *iiceID*.

## Command Example

```
remote_trigger
remote_trigger -info
remote_trigger -set -pid 12
remote_trigger -reset -pid 12
remote_trigger -set -iice IICE0
```

## See also

- triggermode option – [iice](#), on page 48
- triggertime option – [iice](#), on page 48

## run

Arms the IICE with the current trigger settings and waits until the trigger condition has occurred and has been detected by the IICE. Once the trigger condition has occurred, the sample data is downloaded from the IICE and is displayed on the screen.

## Syntax

```
run -iice iiceID|all
run -timeout integer
run -wait
run -remote_trigger pid|0
```

## Arguments and Options

### **-iice** *iiceID|all*

Used when more than one IICE is defined to specify the active IICE (*iiceID*) for triggering. If the argument *all* is specified, triggering applies to each IICE.

### **-timeout** *integer*

Specifies the number of seconds that the Identify debugger waits for a trigger before stopping. Whenever a time-out occurs, the data buffer is automatically updated. A value of 0 disables the time-out feature.

### **-wait**

Causes the IICE to wait for the hardware to stop running before returning.

### **-remote\_trigger** *pid|0*

Used when running multiple debugger instantiations to send a trigger to either the debugger instantiation identified by *pid* or to all debugger instantiations if 0 is specified when a local trigger condition is detected. To identify the process ID of the active debugger instantiation, enter *pid* at the command prompt.

---

**Note:** The `run` command does not stop running until the trigger occurs. If the trigger does not occur, the `run` command does not stop. To cancel the `run` command, you must click the **Stop** button in the Identify debugger menu bar. There is no `stop` command in the command shell.

---

## Command Example

```
run -remote_trigger 1336
```

## searchpath

Sets a search path to find HDL design files during instrumentation or debugging.

### Syntax

```
searchpath [directoryList]
```

### Arguments and Options

Without an argument, the current search path is displayed.

[*directoryList*]

Searches the specified directories, in order, for design files. *DirectoryList* can take the form of the following:

- On a Windows platform: a semicolon-separated list of valid directories. Note that the Windows “\” separator is not allowed in path names.
- On a Linux platform: a colon-separated list of valid directories.

### Default Value

By default, the search path is the current working directory.

### Command Example

```
searchpath {C:/temp;D:/user/joe}  
searchpath {/home/john:/home/designs}
```

### See Also

- add option – [compile](#), on page 36

## setsys

Sets and queries user customization variables.

### Syntax

**setsys list**

**setsys variables**

**setsys set lpt\_address** [*value*]

### Arguments and Options

#### list

Lists all available variables with their respective values.

#### variables

Lists all available variables with a short description explaining their function.

#### set lpt\_address

 [*value*]

Specifies the device address for the parallel port. This setting overrides the operating system defaults. *Value* ranges from 0 to 65535; the default value is “0”. If no value is supplied, then returns the current value.

### Command Example

```
setsys set lpt_address 4095
```

## show

Displays an HDL source code context on the command line.

### Syntax

```
show [-integer] [+integer] fileName:lineNumber
```

```
show hierarchyPath
```

### Arguments and Options

*-integer*

Displays a designated number (*integer*) of lines before the *lineNumber* listed. The default number of lines displayed is 5.

*+integer*

Displays a designated number (*integer*) of lines after the *lineNumber* listed. The default number of lines displayed is 5.

*fileName:lineNumber*

Displays an HDL design file at the selected *lineNumber*.

*hierarchyPath*

Displays the HDL design unit described by the given hierarchical path.

### Command Example

```
show -8 +16 cpu.vhd:29
```

```
show /top/u1/reset_n
```

## signals

Instructs the Identify instrumentor to create special debug logic for the IICE to sample a signal from your HDL design or to delete the debug logic associated with the specified signal by returning the instrumentation status to “not instrumented.”

### Syntax

```
signals add [options] sigName [sigName ... ]
signals add [options] -msb value [-lsb value] sigName

signals delete [options] sigName [sigName ... ]
signals delete [options] -msb value [-lsb value] sigName
```

### Arguments and Options

```
add sigName [sigName ...]
add -msb value [-lsb value] sigName
```

*SigName* is the full hierarchical path name of the signal. In the first syntax statement, more than one signal can be specified for sampling or triggering by including additional signal names separated by spaces. In the second syntax statement, the `-msb` and `-lsb` arguments specify a bit or bit range of a bus.

Note that when specifying partial buses:

- Use the `-msb` argument (without an `-lsb` argument) to specify a single bit
- Observe the index order of the bus. For example, when defining a partial bus range for bus [63:0] (or “63 downto 0”), the MSB value specified must be greater than the LSB value. Similarly, for bus [0:63] (or “0 upto 63”), the MSB value specified must be less than the LSB value.

The following options are available with the `add` argument:

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) for signal sampling/triggering. If the argument `all` is specified, signal sampling/triggering applies to each IICE.

**-sample**

Connects the specified signal or signals to the IICE sample buffer.

**-field** *fieldName*

Instruments the named field or record for the specified signal (partial instrumentation).

**-trigger**

Connects the specified signal or signals to the IICE trigger logic.

---

**Note:** The **-sample** and **-trigger** options can be combined or both options can be omitted to specify a signal for both sampling and triggering.

---

**delete** *sigName* [*sigName* ...]

**delete -msb** *value* [**-lsb** *value*] *sigName*

*SigName* is the full hierarchical path name of the signal. In the first syntax statement, more than one signal can be specified for deletion by including additional signal names. In the second syntax statement, the **-msb** and **-lsb** arguments identify a previously specified bit or bit range of a bus.

---

**Note:** When a partial bus is defined, you must explicitly delete the individual bus segments to return their status to non-instrumented.

---

The following options are available with the **delete** argument:

**-iice** *iiceID*|**all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) for sample signal deletion. If the argument **all** is specified, sample signal deletion applies to each IICE.

**-field** *fieldName*

Removes the instrumentation from the named field or record for the specified signal (partial instrumentation).

## Command Example

```
signals add /top/u1/reset_n
signals add -iice IICE_2 -trigger /top/u1/clken
signals add -sample -msb 63 -lsb 32 /top/data_in
signals add -sample -field iport_mem {/Struc_P_Signed_LDDT_iport}
signals delete -msb 63 -lsb 32 /top/data_in
```

## See Also

- [breakpoints, on page 30](#)
- clock option – [iice, on page 48](#)

## source

Runs a TCL script of commands.

## Syntax

```
source fileName
```

## Arguments and Options

*FileName* contains a script of TCL commands plus commands for the Identify debugger.

## Command Example

```
source /home/joe/syn.tcl
source E:/counter/load.tcl
```

## statemachine

Configures the state machine with desired behavior.

### Syntax

```
statemachine addtrans -from state [-iice iiceID|all] [-to state]
[-cond "equation|ti triggerInID"] [-cntval integer] [-cnten] [-trigger]
```

```
statemachine clear [-iice iiceID|all] -all |state [state ...]
```

```
statemachine info [-iice iiceID|all] [-raw] -all |state [state ...]
```

### Arguments and Options

#### addtrans -from state

Specifies the state the transition is exiting from. This option is required to add a transition to the state machine.

#### addtrans -iice *iiceID*|all

Used when more than one IICE is defined to specify the active IICE (*iiceID*) for state-machine configuration. If the argument `all` is specified, state-machine configuration applies to each IICE.

#### addtrans [-to state]

Specifies the state the transition goes to. If this is not given, it defaults to the state given by the `-from` option, thus creating a transition back to the `-from` state.

#### addtrans [-cond "*equation*|*ti triggerInID*"]

Specifies the condition or external trigger under which the transition is to be taken. The default is "true" (i.e., the transition is taken regardless of any input data).

The conditions are specified using boolean expressions comprised of variables and operators. The available variables are:

- **c0, ... cn**: where *n* is the number of trigger conditions instrumented. These variables represent the trigger output of the respective trigger condition.

- **cntnull**: true whenever the counter is equal to '0' (only available if a counter has been instrumented using the `-counterwidth` option of the `iice controller` command).
- **iiceID**: this variable is used with cross triggering to define the source IICE units to be included in the equation for the destination IICE trigger.
- **titriggerInID**: the ID (0 thru 7) of an external trigger input.

Operators are:

- Negation: `not`, `!`, `~`
- AND operators: `and`, `&&`, `&`
- OR operators: `or`, `||`, `|`
- XOR operators: `xor`, `^`
- NOR operators: `nor`, `~|`
- NAND operators: `nand`, `~&`
- XNOR operators: `xnor`, `~^`
- Equivalence operators: `==`, `.`, `=`
- Constants: `0`, `false`, `1`, `true`

Parentheses ('(', ')') are recommended whenever the operator precedence is in question. Use the `state info` command to verify the conditions specified.

#### **addtrans [-cntval *integer*]**

Specifies that in the case when the transition is taken, the counter must be loaded with the given value. This option is only valid if a counter was instrumented using the `iice controller -counterwidth` option.

#### **addtrans [-cnten]**

If this flag is given, the counter is decremented by '1' during this transition. This flag is only valid if a counter was instrumented using the `iice controller -counterwidth` option.

#### **addtrans [-trigger]**

If this flag is given, the trigger occurs during this transition.

**clear** [-iice *iiceID*|all] -all|state [*state* ...]

Deletes state transitions.

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) for state-machine transition deletion. If the argument **all** is specified, transition deletion applies to each IICE.

**-all|state [*state* ...]**

Deletes the state transitions from the states given in the argument, or from all states if the argument **-all** is specified.

**info** [-iice *iiceID*|all] [-raw] -all|state [*state* ...]

Prints the current state-machine settings.

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) reporting the state-machine settings. If the argument **all** is specified, the settings for each IICE are reported.

**-all|state [*state* ...]**

Reports the settings for the states given in the argument or, if the option **-all** is specified, for the entire state machine.

**-raw**

Reports the settings in a machine-processible form.

## Command Example

```
statemachine addtrans -from 0 -to 1 -cntval 9
statemachine addtrans -from 0 -cond "(c1 | c2)" -trigger
statemachine addtrans -from 1 -cond "c1 && c2" -cnten
statemachine addtrans -from 2 -cond "c2 && cntnull" -trigger
statemachine addtrans -from 0 -cond "IICE_1 and IICE_2" -trigger
statemachine clear 1
statemachine info -all
```

## See Also

- iice controller -counterwidth option – [iice, on page 48](#)
- iice controller -triggerconditions option – [iice, on page 48](#)
- iice controller -crosstrigger option – [iice, on page 48](#)

---

**Note:** The order in which the transitions are added is important. In each state, the first transition condition that matches the current data, is taken. There may be other transitions later in the list that also match the current data, but they are ignored.

---

**stop** 

Activates/deactivates an HDL source-level breakpoint that has been added by the Identify instrumentor. All activated breakpoints are used to form the trigger condition of the IICE. Only breakpoints that have been instrumented using the breakpoints add command can be activated. One or more breakpoints can be activated/deactivated at the same time. A breakpoint name consists of two components:

- The fully hierarchical path of the HDL design unit that denotes the underlying control statement of the breakpoint.
- The HDL source code location given by the file name and the line number of the breakpoint.

The combination of these two components ensures that each breakpoint has a unique name.

**Syntax**

**stop disable** [*options*] *breakpointName* [*breakpointName* ...]

**stop enable** [*options*] *breakpointName* [*breakpointName* ...]

**stop info** [-raw] *breakpointName*

**Arguments and Options**

**disable** [*options*] *breakpointName* [*breakpointName* ...]

Deactivates one or more HDL source-level breakpoints.

**-iice** *iiceID*|all

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the breakpoint to be disabled. If the argument all is specified, disabling the breakpoint applies to each IICE.

**-condition all** | {*conditionlist*}

Specifies a list of trigger conditions in which to disable the breakpoint(s). If only one trigger condition exists in the current design, this option can be omitted, otherwise it is required. The identifier all disables the breakpoint(s) from all trigger conditions.

**enable** [*options*] *breakpointName* [*breakpointName* ...]

Activates one or more HDL source-level breakpoints.

**-iice** *iiceID*|**all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the breakpoint to be enabled. If the argument **all** is specified, enabling the breakpoint applies to each IICE.

**-condition** **all**|{*conditionlist*}

Specifies a list of trigger conditions in which to enable the breakpoint(s). If only one trigger condition exists in the current design, this option can be omitted, otherwise it is required. The identifier **all** enables the breakpoint(s) from all trigger conditions.

**info** [**-raw**] *breakpointName* [*breakpointName* ...]

Displays information about the settings for the given HDL breakpoint. The **-raw** option provides the information in a machine-readable format.

## Command Example

```
stop disable -condition 1 /top/u1/case_128/cpu.vhd:29
```

## See also

- [breakpoints, on page 30](#)
- [iice, on page 48](#)

## transcript

Controls recording of all typed commands into a transcript file.

### Syntax

```
transcript [fileName]
```

```
transcript [off]
```

```
transcript [on]
```

### Arguments and Options

**transcript *fileName***

Saves all typed commands to the file specified by *fileName*.

**transcript off**

Commands system to stop recording commands.

**transcript on**

Commands system to start recording all typed commands and to store them to the default transcript file. The default file is `syn_di.scr` for the Identify instrumentor and `syn_hhd.scr` for the Identify debugger.

### Default Value

By default, command recording is off.

### Command Example

```
transcript on
```

## watch

Activates/deactivates a watchpoint as a trigger condition for the IICE. A watchpoint triggers when the sample value of the watched signal matches the watch value. Only signals that have been instrumented using the `signals add` command can be used for watchpoints.

### Syntax

```
watch disable [options] signalName [signalName ...]  
watch disable [options] -msb value [-lsb value] signalName
```

```
watch enable [options] signalName {value}|{valueFrom} {valueTo}  
watch enable [options] -msb value [-lsb value]  
    signalName {value}|{valueFrom} {valueTo}
```

```
watch info [-raw] signalName
```

```
watch radix [options] signalName [default|binary|octal|integer|unsigned|hex]
```

```
watch width signalName
```

### Arguments and Options

Deactivates one or more HDL source-level watchpoints. One or more watchpoints can be deactivated at the same time.

```
disable [options] signalName [signalName ...]  
disable [options] -msb value [-lsb value] signalName
```

*SigName* is the full hierarchical path name of the signal. In the first syntax statement, more than one signal can be deactivated for sampling or triggering by including additional signal names separated by spaces. In the second syntax statement, the `-msb` and `-lsb` arguments specify a bit or bit range of a bus. Note that when specifying partial buses:

- Use the `-msb` argument (without an `-lsb` argument) to specify a single bit
- Observe the index order of the bus. For example, when defining a partial bus range for bus [63:0] (or “63 downto 0”), the MSB value specified must be greater than the LSB value. Similarly, for bus [0:63] (or “0 upto 63”), the MSB value specified must be less than the LSB value.

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the watchpoint to be disabled. If the argument **all** is specified, disabling the watchpoint applies to each IICE.

**-condition all|{*conditionlist*}**

Specifies a list of trigger conditions in which to disable the watchpoint. If only one trigger condition exists in the current design, then this option can be omitted, otherwise it is required. The identifier **all** can be used to disable the watchpoint from all trigger conditions.

**enable** [*options*] *signalName* {*value*}|{*valueFrom*} {*valueTo*}

**enable** [*options*] **-msb** *value* [**-lsb** *value*] *signalName* {*value*}|{*valueFrom*} {*valueTo*}

When only *value* is specified for *signalName*, gives the watchpoint signal an exact value that the system watches for, and enables that watchpoint for triggering. When *valueFrom*/*valueTo* is specified, gives the watchpoint signal two values that the system watches for, and enables the watchpoint for triggering. These formats allow you to specify a trigger condition on the value transition of a signal. In the second syntax statement, the **-msb** and **-lsb** arguments specify a bit or bit range of a bus. Note that when specifying partial buses:

- Use the **-msb** argument (without an **-lsb** argument) to specify a single bit
- Observe the index order of the bus. For example, when defining a partial bus range for bus [63:0] (or “63 downto 0”), the MSB value specified must be greater than the LSB value. Similarly, for bus [0:63] (or “0 upto 63”), the MSB value specified must be less than the LSB value.

**-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the watchpoint to be enabled. If the argument **all** is specified, enabling the watchpoint applies to each IICE.

**-condition all|{*conditionlist*}**

Specifies a list of trigger conditions in which to enable the watchpoint(s). If only one trigger condition exists in the current design, this option can be left out, otherwise it is required. The identifier **all** enables the watchpoint(s) from all trigger conditions.

**info** [-raw] *breakpointName* [*breakpointName* ...]

Displays information about the settings for the given HDL watchpoint. The -raw option provides the information in a machine readable format.

**radix** [*options*] *signalName* [default|binary|octal|integer|unsigned|hex]

Displays or changes the radix of the specified watchpoint signal for the sampled data. Specifying default resets the radix to its initial intended value. Note that the radix value is maintained in the “activation database” and that this information will be lost if you fail to save or reload your activation. Also, the radix set on a signal is local to the Identify debugger and is not propagated to any of the waveform viewers. Note that with partial buses, the radix applies to the entire bus

**-iice** *iiceID*|all

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing *signalName*. If the argument all is specified, the radix is reported/changed for each IICE.

**width** *signalName*

Reports the width of a vectored (bused) signal. Note that with partial buses, the width reported always applies to the entire bus.

## Command Example

VHDL examples:

```
watch enable /top/u2/current_state {red}
watch enable -condition {1 2} /top/u1/count {"0X01"} {"0010"}
watch radix current_state hex
```

Verilog examples:

```
watch enable /top/bx {4'b0010}
watch enable -msb 3 -lsb 0 /top/u2/data_sel {4'h0}
watch enable -condition all /top/done {1'b0} {1'b1}
```

## See also

- [signals, on page 66](#)
- controller-triggerconditions option – [iice, on page 48](#)

## waveform

Configures the waveform preferences and launches the desired waveform viewer once the Identify debugger has uploaded data from the instrumented design.

### Syntax

**waveform custom** [*userProcedure*]

**waveform period** [*period\_in\_ns*]

**waveform show** [*options*]

**waveform viewer** [*options*] **aldec|debussy|dve|gtkwave|modelsim|custom**

### Arguments and Options

**custom** [*userProcedure*]

Sets/gets user-defined TCL procedure (*userProcedure*) that is used to launch a custom waveform viewer. This procedure must be defined in the TCL window or sourced through a startup script prior to launching the waveform viewer. The default value is `custom_waveform`. This procedure is called by `waveform show` with the following five arguments:

- *lang* – the language the design is written in -- Verilog or VHDL
- *oplevel* – the name of the top-level module or entity
- *firstcycle* – the cycle number of the first cycle
- *sampledepth* – the total number of samples
- *period* – the period for the waveform display independent of the design speed

**period** [*period\_in\_ns*]

Sets/gets the period with which to display the debug data in the waveform viewer. Since the debugger has no information about the timing of the user design, this setting is merely used for customizing the display.

## **show**

Launches the waveform viewer that is currently selected with the current set of sample data.

### **-iice *iiceID*|all**

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the sample data to be displayed. If the argument **all** is specified, the sample data is displayed for each IICE.

### **-showequiv**

Includes all equivalent signals in the sample data.

### **viewer [*options*] **aldec|debussy|dve|gtkwave|modelsim|custom****

Selects the user preference for the waveform viewer. The selection **custom** causes the waveform show command to call the procedure specified by the waveform custom command.

### **-list**

Lists the available waveform viewer choices.

## **Command Example**

```
waveform viewer -list  
waveform show -equiv
```

## write instrumentation

Writes the instrumented design files to the project directory.

### Syntax

```
write instrumentation [-pretty | -save_orig_src | -encrypt_orig_src]
```

### Options

#### **-pretty**

Instrument HDL with human-readable formatting

#### **-save\_orig\_src**

Create an “orig\_sources” directory in the project directory and copy the user's original sources into this directory.

#### **-encrypt\_orig\_src**

Encrypt the original sources in the “orig\_sources” directory. The encryption is based on a password which must previously be set with the `encryption set_passwd` command. Attempting to use this flag without a valid password set results in an error. Note that the `-encrypt_orig_src` flag implies and overrides the `-save_orig_src` flag. When neither flag is set, no “orig\_sources” directory is created in the project directory.

### Command Example

```
write instrumentation -encrypt_orig_src
write instrumentation -save_orig_src
write instrumentation
```

### See Also

- [encryption, on page 39](#)

## write samples

Writes the sample data of each specified signal.

### Syntax

```
write samples [options] signalName [signalName ...]  
write samples [options] -msb value [-lsb value] signalName
```

In the above syntax statements, *sigName* is the full hierarchical path name of the signal. In the first syntax statement, sample data can be written for more than one signal by including additional signal names separated by spaces. In the second syntax statement, the **-msb** and **-lsb** arguments specify a bit or bit range of a bus. Note that when specifying partial buses:

- Use the **-msb** argument (without an **-lsb** argument) to specify a single bit
- Observe the index order of the bus. For example, when defining a partial bus range for bus [63:0] (or “63 downto 0”), the MSB value specified must be greater than the LSB value. Similarly, for bus [0:63] (or “0 upto 63”), the MSB value specified must be less than the LSB value.

### Arguments and Options

**-iice** *iiceID*

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the sample data for the specified signal.

**-cycle** {*cycleFirst cycleLast*}

Specifies the range of sample data displayed. You can view the data at different points of the trigger event. Enter a negative cycle value to view data sampled before the triggered event. Enter a positive cycle value to view data samples after the trigger event. Enter a zero cycle value to view data sampled during the trigger event.

**-file** *fileName*

Writes the sample data to a specified output file. If no file is given, the data is displayed on the screen.

**-force**

Overwrite *fileName* if it exists

**-raw**

Return machine-readable samples. For each signal specified, the command returns a Tcl list formatted as shown:

```
{{signalName cycleFirst cycleLast} {sampleValuesList}}
```

**Command Example**

```
write samples -file D:/tmp/samples.txt /top/u1/count
write samples -cycle { -10 10 } /top/u2/current_state
write samples -msb 31 -lsb 0 /top/u3/data_outA
```

## write vcd

Writes the sample data of each specified signal to a Verilog Change Dump (vcd) format.

### Syntax

```
write vcd [options] fileName
```

### Arguments and Options

**-iice** *iiceID*

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the sample data for the specified signal.

**-comment** *commentText*

Inserts a text comment into a file. Use curly braces ‘{}’ to group together a multi-word comment.

**-gtkwave**

Creates a GTKWave control file for the VCD output file.

**-showequiv**

Includes the sample data for all equivalent signals.

*fileName*

Writes the sample data to a specified output file.

### Command Example

```
write vcd -gtkwave D:/tmp/b.vcd
```

## write vhdlmodel

Creates a VHDL model from sample data. This command is not supported in Verilog-based designs or in mixed-language designs when the top-level is a Verilog module.

### Syntax

```
write vhdlmodel [options] fileName
```

### Arguments and Options

**-iice** *iiceID*

Used when more than one IICE is defined to specify the active IICE (*iiceID*) containing the sample data for the VHDL model.

**-showequiv**

Includes the sample data for all equivalent signals.

*fileName*

Writes the VHDL model to a specified output file.

### Command Example

```
write vhdlmodel D:/tmp/b.v
```



# Index

---

## A

activation command 29

## B

breakpoints

    activating/deactivating 73

    searching 44

breakpoints command 30

buffer

    sample depth 53

## C

cable option settings 35

cable types 34

cd command 31

chain command 32

clear command 34

clock

    sampling 48

clock option 48

com command 34

command history 75

command summary

    design debugging commands 26

    design hierarchy commands 24

    design instrumentation commands 25

    file system commands 24

    general commands 23

commands

    recording 75

compile command 36

complex triggering 49

configuration

    IICE 48

    synthesis tool 16

console output

    logging 56

conventions

    design hierarchy 11

    file system 10

    symbol 9

    syntax 8

    text 8

    tool 10

counterwidth option 49

## D

debugger

    process ID 60

depth option 53

design debugging

    command summary 26

design files

    listing 36

    writing 81

design hierarchy 41

    command summary 24

design hierarchy conventions 11

design instrumentation  
    command summary 25  
device command 37  
device ID codes 46  
directories  
    changing 31  
    displaying working 59

## E

encryption command 39  
event trigger 60  
exit command 40

## F

file system  
    command summary 24  
file system conventions 10  
files  
    initialization 20  
    listing design 36  
    searching 63  
    synrc 20  
    Verilog Change Dump 84  
    writing design 81  
find option 43

## G

GUI  
    clearing 34  
    logging console output 56

## H

HDL files  
    searching 63  
help command 40  
hierarchy command 41

hierarchy separator 12

## I

idcode command 46  
IICE  
    arming 61  
    communicating with 34  
    selecting multiple 52  
iice command 48  
importing projects 58  
incremental implementations 55  
instrumentation command 55

## J

JTAG chains 32

## L

license types 18  
log command 56

## M

models  
    VHDL 85  
multi-IICE selection 52  
multiple debuggers 60  
multiple implementations 55

## O

online help 40  
operators  
    state machine 70

**P**

- parallel port
  - defining 35
- passwords
  - encryption 39
- path names 12
- path separator 10
- process ID
  - debugger 60
- project command 58
- projects
  - creating new 58
  - importing 58
  - opening 58
- pwd command 59

**R**

- remote\_trigger command 60
- run command 61

**S**

- sample data 82
- searching 43
- searchpath command 63
- separator
  - hierarchy 12
  - path 10
- set\_synplify\_configuration command 18
- setsys command 64
- show command 65
- signals command
  - debug logic 66

- source code
  - displaying 65
- source command 68
- startup 16
- state machines
  - configuring 69
  - operators 70
  - triggering 50
- statemachine command 69
- stop command 73
- symbol conventions 9
- synrc file 20
- syntax conventions 8
- synthesis tool
  - location 16
- system variables 64

**T**

- tables
  - idcode 26, 46
- target device 37
- Tcl scripts 68
- text conventions 8
- tool conventions 10
- transcript command 75
- trigger conditions 30
- trigger settings
  - reloading 29
  - saving 29
- triggering
  - complex 49
  - state machines 50
- triggermode option 53
- triggers 76
- triggerstates option 50
- triggertime option 54

## **V**

variables 64

vendor-specific licenses 18

Verilog

    hierarchy 11

VHDL

    hierarchy 11

VHDL models 85

## **W**

watch command 76

watchpoints 76

    searching 44

waveform command 79

wildcards

    in hierarchies 12

    in path names 10

working directory

    displaying 59

write instrumentation command 81

write samples command 82

write vcd command 84

write vhdlmodel command 85