Verification Continuum™

# Identify®

# Debugging Environment for Microchip Reference Manual

October 2020

SYNOPSYS®

Synopsys Confidential Information

# Copyright Notice and Proprietary Information

© 2020 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

# Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

# Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

# Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

# Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

# Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 East Middlefield Road
Mountain View, CA 94043
www.synopsys.com

October 2020

This section contains the following topics:

Introduction

Command Description

User Interface Commands

User Interface Overview

GUI Popup Menu Commands


© 2020 Synopsys, Inc.

© 2020 Synopsys, Inc.

# Contents

## Chapter 1: Introduction


Overview of the Commands . . . . . 10
About Tcl Commands . . . . . 10
About the GUI Commands . . . . . 11

Syntax Conventions . . . . . 13

Tool Conventions . . . . . 14
File System Conventions . . . . . 14
Design Hierarchy Conventions . . . . . 15

Symbol Conventions . . . . . 17


## Chapter 2: Command Description


activation . . . . . 20
breakpoints . . . . . 21
cd . . . . . 23
chain . . . . . 24
clear . . . . . 26
com . . . . . 26
condition info . . . . . 28
device . . . . . 31
encryption . . . . . 35
exit . . . . . 36
fpga . . . . . 36
haps . . . . . 37
help . . . . . 40
hierarchy . . . . . 41
idcode . . . . . 46
instrumentation . . . . . 59
jtag_server . . . . . 61
licenseinfo . . . . . 63
logicanalyzer . . . . . 63
log . . . . . 65




© 2020 Synopsys, Inc.

project . . . 66
pwd . . . 67
remote_trigger . . . 67
run . . . 69
searchpath . . . 71
setsys . . . 72
show . . . 73
signals . . . 74
source . . . 79
statemachine . . . 80
stop . . . 88
transcript . . . 90
verdi . . . 91
watch . . . 92
waveform . . . 95
write fsdb . . . 98
write instrumentation . . . 99
write samples . . . 101
write vcd . . . 103
write vhdlmodel . . . 104

## Chapter 3: User Interface Commands

File Menu . . . 106
Open Debugger Project . . . 106
Create Image Command . . . 107

Edit Menu . . . 110

View Menu . . . 111
Toolbar Command . . . 111
Debugger Panels . . . 112

Instrumentor . . . 113
IICE . . . 114
Find in Instrumentor Source . . . 123
Instrmentor Preferences . . . 124

Debugger . . . 129
Waveform Viewer . . . 130
Find in debugger source . . . 130
Trigger Position . . . 131


© 2020 Synopsys, Inc.

Configure State Machine . . . . . 132
Configure Confpro . . . . . 133
Setup debugger . . . . . 133
Debugger information . . . . . 146

Options Menu . . . . . 148
Editor Options Command . . . . . 148

Window . . . . . 150

Help Menu . . . . . 151
Preferred License Selection Command . . . . . 151

## Chapter 4: User Interface Overview

Instrumentor GUI . . . . . 154
Control Panel . . . . . 154
Search Panel . . . . . 155
Hierarchy Browser . . . . . 156
View Panel . . . . . 157

Debugger GUI . . . . . 158
Run Panel and Run Button . . . . . 159
Search Panel . . . . . 160
Hierarchy Browser . . . . . 162
View Panel . . . . . 162
TCL Window . . . . . 168

Toolbars . . . . . 170
Text Editor Toolbar . . . . . 170
Edit Toolbar . . . . . 171
File Toolbar in Instrumentor . . . . . 172
File Toolbar in Debugger . . . . . 172

Keyboard Shortcuts . . . . . 174
Instrumentor Keyboard Shortcuts . . . . . 174
Debugger Keyboard Shortcuts . . . . . 175

## Chapter 5: GUI Popup Menu Commands

Popup Menus . . . . . 178
Hierarchy Browser Popup Menu . . . . . 178
Tcl Window Popup Menu . . . . . 179

RTL View Popup Menus . . . . . 180
RTL View Options in Instrumentor . . . . . 180
RTL View Options in Debugger . . . . . 182


© 2020 Synopsys, Inc.
Synopsys Confidential Information

# : Index

SYNOPSYS®
Silicon to Software™

CHAPTER 1

# Introduction

This document is part of a set of documents for the Synopsys® FPGA instrumentor and debugger. These tools allow you to debug your design at the VHDL/Verilog source level, in the target system, and at the target speed.This chapter introduces the command interface, the GUI, and various conventions used in the rest of the book.


- Overview of the Commands, on page 10
- Syntax Conventions, on page 13
- Tool Conventions, on page 14
- Symbol Conventions, on page 17



© 2020 Synopsys, Inc.

# Overview of the Commands

This document is part of a set that includes reference and procedural information for the tools. This document describes the commands available for the synthesis tool, which usually includes a graphical user interface (GUI) as well as command line access. Most commands have both GUI and command line versions, so you can use either mode to specify commands.

The following sections provide an overview of the commands in the tool:


- About Tcl Commands, on page 10
- About the GUI Commands, on page 11


## About Tcl Commands

Tcl (Tool Command Language) is a popular scripting language for controlling software applications. The Tcl command set has been extended with additional commands that you can use to run the instrumentor and debugger.

Tcl scripts are text files that have a .tcl file extension and contain a set of Tcl commands designed to complete a task or set of tasks. You can also run Tcl scripts through the Tcl window (see *TCL Window*, on page 168).

The Tcl commands are described here. For information on the standard Tcl commands, syntax, language, and conventions, refer to the Tcl online help (Help > TCL).

### Tcl Conventions

This is a list of conventions to respect when entering Tcl commands and/or creating Tcl scripts.

- Tcl is case sensitive.
- Comments begin with a hash mark or pound sign (#).
- Enclose all path names and filenames in double quotes (").
- Use a forward slash (/) as the separator between directory and path names on all operating systems (including the Microsoft® Windows® operating system). For example:

```
designs/big_design/test.v
```

## About the GUI Commands

Access the GUI versions of the commands from the graphical interface. Most commands open dialog boxes where you can specify parameters for the command.

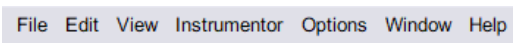
The GUI provides a few ways to access commands:


- Menus, on page 11
- Context-sensitive Popup Menus, on page 11
- Toolbars, on page 12
- Keyboard Shortcuts, on page 12
- Buttons and Options, on page 12

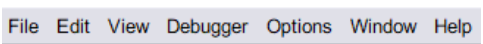

### Menus

The set of commands on the pull-down menus in the menu bar varies depending on the view, design status, task to perform, and selected object(s). For example, the File menu commands in the Project view differ slightly from those in the RTL view. Menu commands that are not available for the current context are dimmed out. The menu bar in the Project view is shown below:

Instrumentor menu bar




Debugger menu bar




For details on individual menus, their commands, and the associated dialog boxes, see Chapter 3, *User Interface Commands*.

### Context-sensitive Popup Menus

Popup menus, available by right-clicking, offer access to commonly used commands that are specific to the current context. See Chapter 5, *GUI Popup Menu Commands*, for information on individual popup menus.


© 2020 Synopsys, Inc.

### Toolbars

Toolbars contain icons associated with commonly used commands. For more information about toolbars, see *Toolbars*, on page 170.

### Keyboard Shortcuts

Keyboard shortcuts are available for commonly used commands. The shortcut appears next to the command in the menu. See *Keyboard Shortcuts*, on page 174 for details.

### Buttons and Options

The Project view has buttons for quick access to commonly used commands and options.

# Syntax Conventions

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

| Text Convention | Represents... |
|---|---|
| **bold** | Commands and literal arguments that are entered as shown. |
| *italics* | User-defined arguments or example command information. |
| [ ] | Optional information or arguments. Do not include the brackets with the command within the command line. |
| ... | Items that can be repeated any number of times. |
| \| | Alternative choices. The choices are located on either side of this symbol. |
| # | Comments about the code, or other non-command information. |
| { } | Escape characters for search strings; also entered in bold font as a literal in some commands |


© 2020 Synopsys, Inc.

# Tool Conventions

There are tool concepts you must familiarize yourself with when using the 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 uses specific conventions.

### Path Separator "/"

All file system commands that contain a directory name use only forward slashes, whether the underlying operating system is Microsoft® Windows® or Linux:

```
/usr/data.dat
```

```
c:/Synopsys/data.dat
```

### Wildcards for File Systems

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 tool set 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 | Verilog |
|---|---|
| • The top-level entity<br>• Architectures<br>• Component instantiation statements<br>• Process statements<br>• Control flow statements: if-then-else, and case<br>• Subprogram statements<br>Block statements | • The top-level module<br>• Module instantiation statements<br>• Always statements<br>• Control flow statements: if-then-else, and case<br>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 for Design Hierarchy

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:

| Syntax | Description |
|---|---|
| [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.

# Symbol Conventions

The commands in this manual use symbols to designate the tool that uses these commands. The symbols are adjacent to the command name.

| This convention... | Organizes this information... |
|---|---|
|  | Any command that is used in the instrumentor only. This symbol follows instrumentor commands in the command listing chapter. |
|  | Any command that is used in the debugger only. This symbol follows debugger commands in the command listing chapter. |
|  | Any command that is used in both the instrumentor and debugger tools. This symbol follows commands that have applications in both tools. |


© 2020 Synopsys, Inc.

© 2020 Synopsys, Inc.

SYNOPSYS®
Silicon to Software™

CHAPTER 2

# Command Description

All commands are listed alphabetically in this chapter. Each command contains syntax, argument return values, default values, and examples. The table below is linked to the individual command descriptions.

| | | | |
|---|---|---|---|
| activation | breakpoints | cd | chain |
| clear | com | condition info | device |
| encryption | exit | fpga | haps |
| help | hierarchy | idcode | iice |
| instrumentation | jtag_server | licenseinfo | log |
| project | pwd | remote_trigger | run |
| searchpath | setsys | show | signals |
| source | statemachine | stop | transcript |
| verdi | watch | waveform | write fsdb |
| write instrumentation | write samples | write vcd | write vhdlmodel |

The commands are divided into several specific categories. These categories separate the commands in terms of which tool (instrumentor or debugger) utilizes the command. These symbols are:




Command available only in the instrumentor

Command available only in the debugger

Command available in both the instrumentor and the debugger


© 2020 Synopsys, Inc.

# 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, te named activation is loaded or saved; if *activation-Name* 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 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* ...]

**breakpoints map** *breakpointName MictorPinName*

**breakpoints preconfigure** [**-iice** *iiceID*|**all**] *breakpointName* [*breakpointName* ...]
**-condition {***integer***}**|**all -state** [**0**|**1**]

## Arguments and Options

For the add and delete options, one or more breakpoints can be added or deleted at the same time.

**add** *breakpointName* [*breakpointName* ...]

**delete** *breakpointName* [*breakpointName* ...]

The map option is used exclusively with the real-time debugging feature to assign a breakpoint to a Mictor connector pin. In the above syntax, *MictorPinName* is the concatenation of the Mictor board HapsTrak® connector location, the Mictor connector name, and the Mictor pin name separated with periods. For example, 3.M1.D3e is the D3e pin of Mictor connector M1 on the Mictor board installed in HapsTrak connector J3.

The preconfigure option specifies the trigger conditions to enable for the specified breakpoints. The -condition argument is a Tcl list of conditions or all, and the -state argument specifies the state (0 or 1).

Breakpoint names 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.


© 2020 Synopsys, Inc.

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

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

## Command Example

```
breakpoints add /beh/arb_inst/beh/process_83/case_88/arb.vhd:90

breakpoints delete -iice trap2
   /beh/blk_xfer_inst/beh/process_85/case_97/xfer.vhd:107

breakpoints map /beh/process_50/case_88/if_90/alu.v:72 3.M1.D5e

breakpoints preconfigure /beh/process_10/case_88/if_90/alu.v:72
   -condition {1 3} -state 0
```

## See Also

- *stop*, on page 88

© 2020 Synopsys, Inc.

# 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 irrespective of the operating system. 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 67


© 2020 Synopsys, Inc.

# chain 

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

## Syntax

**chain add** *deviceName instructionRegisterWidth*

**chain add** *deviceName instructionRegisterLengthchipID*

**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 chain 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.

**add** *deviceName instructionRegisterLengthchipID*

Use this chain command to set up a chain representation in the Identify Debugger. The chain add command is only supported on Microchip devices supported by Identify.

**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
```

**Note:** Do not use these commands to program the Identify hardware.

## See Also

- *device*, on page 31
- *com*, on page 26


© 2020 Synopsys, Inc.

## clear 

Removes all the console output in the graphical user interface. This command is only supported in the graphical modes.

### Syntax

**clear**

### Arguments and Options

none

## com 

Sets up and manipulates communication settings between the 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 byteblaster, Microchip_BuiltinJTAG, JTAGTech3710, Catapult_EJ1, Digilent_JTAG_HS1, and demo. A umrbus selection in also available to indicate that the UMRBus is to be used as the communication interface between the hardware and the host machine running the Identify debugger.

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

Specifies or reports cable-specific option settings:

**catapult_ip_address** [*address*] – specifies the IP address/hostname for the Catapult EJ-1 cable.

**Microchip_builtinJTAG_port** [*portID* | auto] – specifies which port to use when multiple cables are connected (use auto for single cable configuration).

**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.

**flashPro5_trst** [*string*] – specifies the setting of the TRST (tristate) pin. Accepted values (*string*) are off, toggle, low, and hi; the default is off.

**JTAGTech_type** [**PCI**|**USB**] – specifies the type of computer interface connection (parallel or USB); the default is PCI (parallel).

**JTAGTech_port** [*integer*] – specifies the interface card address (0 to 255). The default address is 0.

**JTAGTech_tapnum** [*integer*] – specifies the active tap port on the JTAG Technologies tap pod. Values (*integer*) range from 1 to 4; the default is 1.

**byteblaster_port** [*integer*] – specifies the parallel port number; default is 1 (lpt1).

**Digilent_JTAG_HS1_speed** [*commSpeed*] – specifies the communication speed for the Digilent JTAG HS1/HS3 cable.


© 2020 Synopsys, Inc.

**check**

Performs a connectivity check on the JTAG cable connection.

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

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

### Command Example

```
com cabletype byteblaster
com cableoptions byteblaster_port 2
com port lpt1
```

### See Also

- *chain*, on page 24

## condition info 

Lists the information about the conditions defined in state-machine triggering for the instrumented signals along with the trigger values.

### Syntax

condition info [-iice <value>]

condition info [-condition | -cond <value>]

condition info [-raw]

condition info [-help]

### Arguments and options

[-iice <value>]

Specifies the IICE Name for which the signals' condition values are defined.

[-condition | -cond <value>]

State machine conditions to be specified in integer format.

[-raw]

Creates machine readable format.

[-help]

Provides the command help information.

**Note:** If no arguments and options are provided, the command results all conditions by default.

## Command Examples

condition info -iice IICE_0 -cond 0

```
------------------------------------------------------------------------

IICE 0       "1"             /reset_n

========================================================================
```

condition info -iice IICE_1 -condition 1

```
------------------------------------------------------------------------

IICE 1          "000100"
/SRS/gen_add\.6\.count6bit\.counter_inst/count[5:0]

========================================================================
```


© 2020 Synopsys, Inc.

condition info

```
IICE Condition From Value To Value Signal Name
-----------------------------------------------------------------------
IICE 0       "1"            /reset_n
IICE 1       '1'            /reset_n
IICE 2       '1'            /reset_n
=======================================================================
```

# device 

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

## Instrumentor Syntax

**device estimate** [**-iice all**|*iiceName*] [**-resources** |**-noresources** |**-raw**]

**device jtagport** [**builtin**|**soft** |**umrbus**]

**device prepare_incremental** [**0**|**1**]

**device skewfree** [**0**|**1**]

**device technologydefinitions** [**0**|**1**]

**device capimbaseaddr** *integer*

## Arguments and Options

**estimate** or **estimate -iice all**

Reports total number of instrumented signals and estimated resource utilization for the current implementation.

**estimate -iice** *iiceName*

Reports total number of instrumented signals and estimated resource utilization for the named IICE (*iiceName*) for the current implementation.

**estimate -resources** or **estimate -resources -iice all**

Reports only the estimated resource utilization for current implementation.


© 2020 Synopsys, Inc.

**estimate -resources -iice** *iiceName*

Reports only estimated resource utilization for the named IICE (*iiceName*) for the current implementation.

**estimate -noresources** or **estimate -noresources -iice all**

Reports only the number of instrumented signals for current implementation.

**estimate -resources -iice** *iiceName*

Reports only the number of instrumented signals for the named IICE (*iiceName*) for the current implementation.

**estimate -raw**

Displays the instrumented signal information and estimated resource utilization for the current implementation in a machine-readable format.

**jtagport** [**builtin**|**soft**|**umrbus**]

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 instrumentor. With no argument specified, the current setting is displayed. The following selections are available:

Specifies that the UMRBus is to be used as the communication interface between the hardware and the host machine running the debugger (the JTAG port is not used).

**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 instrumentor. The Synopsys JTAG port requires four additional user pins.

**umrbus**

Specifies that the UMRBus is to be used as the communication interface between the hardware and the host machine running the debugger (the JTAG port is not used).

**prepare_incremental**

Sets up the instrumentor to support incremental changes to the instrumented signals.

**skewfree** [**0**|**1**]

Causes the IICE to be built using skew-resistant hardware when no global clock resources are available for the JTAG clock. When this option is enabled (1), master-slave flip-flops are used on the JTAG chain to prevent clock skew from affecting the logic. This setting also causes the instrumentor to NOT explicitly define the JTAG clock as requiring global clock resources. The skewfree option is disabled (0) by default.

**technologydefinitions** [**0**|**1**]

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

**capimbaseaddr** *baseAddress*

Specifies or reports the base address of a CAPIM inserted for UMRBus communication. In a multi-FPGA debugging environment, different FPGAs on the same UMRBus require unique CAPIM addresses when using post partitioned instrumentation (by default, FPGAs on the same board share the same CAPIM address which would disrupt communications with the debugger). The capinbaseaddr argument is used to assign different CAPIMs when two or more FPGAs have independent instrumentations. The CAPIM base address value ranges up to 63 with a default base address of 57. This value is decremented with each CAPIM added (user CAPIM addresses are incremented beginning with 1).

## Debugger Syntax

The following device options report settings available from the debugger.

**family** – Shows the device family in use

**jtagport** – Shows the device JTAG interface selected

**skewfree** – Shows the device skewfree option setting


© 2020 Synopsys, Inc.

## Command Example

```
device estimate -iice IICE -noresources
device jtagport builtin
device skewfree 1
```

## See Also

- *chain*, on page 24
- *com*, on page 26

# encryption 

Sets the current password to use before encrypting or decrypting a file. In the instrumentor, this command sets the password to be used when writing out an encrypted file with the write instrumentation command. In the 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 instrumentor or debugger is shut down.

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

## Command Example

```
encryption set_passwd xyzzy
```

## See Also

- *write instrumentation*, on page 99
- *project*, on page 66


© 2020 Synopsys, Inc.

## exit 

Exits the program and closes the window.

### Syntax

**exit**

### Arguments and Options

None

### Command Example

```
exit
```

## fpga 

Adds an FPGA for distributed instrumentation.

### Syntax

**fpga add** [*fpga* [*fpga* ...]] [**-iice** *iiceID*] **-type** *fpgaType* **-master_dtd2_reset**

*fpga* [*fpga* ...]

A list of the FPGAs to be added.

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

Used when more than one IICE is defined to specify which IICE (*iiceID*) to use for distributed instrumentation. If the argument **all** is specified, the FPGA type applies to each IICE unit.

**-type** *fpgaType*

The type of FPGA to use for distributed instrumentation.

**-master_dtd2_reset**

Identifies the FPGA as the source of the master-reset signal.

# haps 

Queries the hardware to generate the requisite Tcl file for board generation and performs the verification tests.

## Syntax

**haps**
**boardstatus** [*boardID*]
**settings {***setting value* [*setting value* **...**]**}**
**prog** *binFile devID*
**setvcc** *voltage* [*region*]
**setclk** *clockName frequency*
**clear**
**vbgen** *tclFile*
**hstdm_report** [**{***hstdmReportList***}** |**ALLFPGAS** |**poll** [**{***interval***}**]]
[**verbose**] [**device {***index***}**]
**clock_check**
**con_check** [*connectivityFileName*] [*logFileName*]
**con_speed** *speed* {**fast**|**sweep**} [*connectivityFileName*] [*logFileName*]
**umr_check** *fpgaID*

## Arguments and Options

**boardstatus** [*boardID*]

Displays the board status to the console window. Status includes clock and voltage settings, reset condition, daughter card connections, firmware version, and board serial number.

**settings {***setting value* [*setting value* **...**]**}**

Specifies the HAPS port (PORT_NAME), device (DEV_ID), and bus (BUS_NUM) settings. With no arguments, reports the current settings. The curly braces enclosing the arguments are required.

**prog** *binFile devID*

Programs the FPGA identified by *devID* with the specified bin file. The *devID* value ranges from 1 to 32 with 1 corresponding to the first FPGA on the board.

**setvcc** *voltage* [*region*]


© 2020 Synopsys, Inc.

Sets the I/O voltage for the board regions. The voltage value and region are selected from the corresponding drop-down menus and differ with the board/system selected. Multiple regions can be selected using the Ctrl key. If *region* is omitted, all regions are set to *voltage*.

**setclk** *clockName frequency*

Sets the frequency for the global input clock identified by *clockName* to the specified frequency. The *frequency* value is in kHz unless specified otherwise. For example, the command haps setclk GCLK1 150MHz specifies a clock frequency of 150 MHz for GCLK1.

**restart**

Clears the entire board/system configuration including the FPGA configuration, voltage, and clock settings.

**confscr** *scriptFileName*

Runs confprosh tcl scripts. For example, the confscr option can be used to source a HAPS clock and voltage-region configuration script; the user could then run clock checks to verify the on-board clock configuration.

**tssgen** *tclFile*

Queries the HAPS system and generates a Tcl file that describes the hardware setup. Use this file to check the hardware setup.

**hstdm_report** [**{***hstdmReportList***}**|**ALLFPGAS**|**poll** [**{***interval***}**]]
[**verbose**] [**device {***index***}**]

Queries and reports the current training status of the HSTDM blocks running in the user design.

*hstdmReportList* – A Tcl list of training report files to query.

**ALLFPGAS** – Query the training report files from all FPGAs.

**poll** [*interval*] – Poll the FPGAs for the specified interval in seconds; the poll argument is not supported from the GUI.

**verbose** – Include additional content in the training report.

**device** *index* – List training status for the FPGAs on the indexed board device.

**clock_check** – Reports the clock frequency of each GCLK output to allow of all of the GCLK frequencies to be verified. When the All argument is used, runs all local tests with the individual test parameter defaults.

**con_check** [*connectivityFileName*] [*logFileName*] – Verifies that the cabling between HapsTrak connectors is consistent with the defined connectivity file.

*connectivityFileName* – Specifies the Tcl script that describes the connections between HapsTrak connectors in your current system set-up. The default script in the current working directory is named connectivity.tcl.

*logFileName* – Specifies the name of the log file. The default file in the current working directory is named hapstest.log. Note that if you use a non-default log file, you must also explicitly specify the connectivity file even if you intend to use the default; for example:

```
haps run con_check {} ./logfiles/hapstest2.log
```

**con_speed** *speed* {**fast**|**sweep**} [*connectivityFileName*] [*logFileName*] – Verifies the connectivity between HapsTrak connectors as well as the hypothetical speed at which HSTDM can run. In the syntax:

*speed* – The raw data transfer speed in Mbps; the acceptable values are 840, 960, 1080, or ALL (the ALL selection scans a range of speeds to determine the highest rate possible).

**fast**|**sweep** – Sets the run mode. The default is fast mode. When mode is set to sweep, the test sweeps every channel of the connection which can require up to four hours to complete.

*connectivityFileName* – Specifies the Tcl script that describes the connections between HapsTrak connectors in your current system set-up. The default script in the current working directory is named connectivity.tcl.

*logFileName* – Specifies the name of the log file. The default file in the current working directory is named hapstest.log. If you use a non-default log file, you must also explicitly specify the connectivity file even if you intend to use the default.

**umr_check** *fpgaID* – Verifies the basic functionality of the UMRBus. In the syntax:

*fpgaID* – Indicates which FPGA device is to be tested. The default is 1, which is the first FPGA device on the first board.


© 2020 Synopsys, Inc.

### Command Example

```
haps run umr_check 2 180
haps setclk GCLK1 150MHz
haps setvcc 1v8
haps help con_speed
haps settings {PORT_NAME emu:1 DEV_ID 4}
haps hstdm_report mb74_uD_hstdmreport.txt verbose
```

## 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, help descriptions for all commands are printed to the screen.

# hierarchy 

Navigates through the design hierarchy and shows design and hierarchy elements in the HDL design. These design elements include the following types, depending on the HDL language used to describe the design:

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

**hierarchy pwd**

**hierarchy toplevel**

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

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

**hierarchy load** [**-bg**] [**-ls**] [*path* [*path* ...]]

## 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 instrumentor.


© 2020 Synopsys, Inc.

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.

**-recursive**

Allow hierarchies to be traversed when a wildcard is included in the *element* argument.

---

**Note:** The -sample, -trigger, -breakpoint, and -recursive 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.

**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.

**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.


© 2020 Synopsys, Inc.

**-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 the debugger).
- **enabled** – Limits search to enabled watchpoints, breakpoints, and other enabled HDL design units (available only in the debugger).
- **instrumented** – Limits search to the sampling clock, and watchpoints and breakpoints that have been marked as instrumented (available only in the instrumentor).
- **not-instrumented** – Limits search to watchpoints and breakpoints that have not been instrumented (available only in the instrumentor).
- **sample_only** – Limits search to sample-only watchpoints (available only in the instrumentor).
- **trigger_only** – Limits search to trigger-only watchpoints (available only in the 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 debugger, breakpoints that were not instrumented. By, default, HDL elements with enabled status are searched.

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

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

---

**load** [**-bg**] [**-ls**] [*path* [*path* ...]]

Loads sub-hierarchy of all listed paths. The -bg argument runs the command in the background. The load argument applies only to the instrumentor.

## Command Example

```
hierarchy cd ..

hierarchy cd /top/u1/arui

hierarchy ls -recursive

hierarchy find -type breakpoint -stat instrumented
```

## See Also

- *show*, on page 73


© 2020 Synopsys, Inc.

# 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 spaces.

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 00100000001110001000100010000 device_type 8

idcode add -quiet 00100000001110001000100010000 "device type" 8

idcode clear
```

## See Also

- *device*, on page 31
- *chain*, on page 24


© 2020 Synopsys, Inc.

# iice

Duplicates the functionality of the IICE Configuration dialog box.

## Syntax

**iice new**|**rename**|**list**|**delete**|**current**|**info**|**clock**|**controller**|**preconfigure**|**sampler**|**assignmentsreport** [*option*]

## Arguments and Options

**iice new** [*iiceID*] [**-type rtd**|**regular**]

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 -type option indicates if the IICE is to be configured for real-time debugging (rtd) or normal debugging (regular). For more information on the real-time debugging feature, see the *Debugger User Guide*. The iice new command is only available in the instrumentor.

**iice rename** *iiceID*

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

**iice list**

Lists the IDs (names) of each defined IICE.

**iice delete** *iiceID*

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

**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 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 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 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 state machine. 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 -counterval option (applies only to the 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**


© 2020 Synopsys, Inc.

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 -counterval.

**-counterval** *unsignedInteger*

Sets a value for the complex counter and loads that value into the complex counter (applies only to the debugger). The value must fit into the complex counter width as defined in the instrumentor. The default value for the complex counter is 16.

**-counterwidth** *integer*

Instruments a versatile counter of variable size for triggering (applies only to the instrumentor). An integer parameter in the range between 1 and 32 specifies a new counter width, and 0 suppresses the creation of a state-machine counter (a -counterwidth value of 0 is not recognized for complex-counter triggering). All other values are invalid. The default value for the counter width is 16 for both complex and state-machine counter triggering.

**-triggerconditions** *integer*

Used when instrumenting a design for state-machine triggering (applies only to the instrumentor). This command specifies the number of trigger conditions available for state-machine triggering. The range is from 1 to 16. The default value is 4. If no argument is given, the command shows the current pattern-tree setting.

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 the instrumentor). This option specifies the maximum number of states instrumented in the state-machine triggering. The range is 2 to 16; powers of 2 are preferable as other integers limit

functionality and do not provide any cost savings. The default is 4. If no argument is given, the option shows the current triggerstates setting.

**-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 the instrumentor). If the -iice argument is omitted, the command applies to the current IICE.

**-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 the instrumentor). Enables (1) or disables 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 the 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.

---

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

Determines the trigger conditions in the 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**


© 2020 Synopsys, Inc.

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).

**iice preconfigure** [*options*]

Pre-configures a set of common instrumentor IICE options.

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

Used when more than one IICE is defined to identify the IICE unit; if all is specified, effective for all IICE units.

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

Pre-configures the mode for the counter.

**-counterval** *integer*

Pre-configures the value for the counter.

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

Pre-configures the cross-triggering mode.

**-crosstriggeriice** *iiceID*

IICE to be used when crosstriggermode is set to after.

**-triggertime early|middle|late**

Sets/gets trigger time for next run.

**-samplemode normal|qualified_fill|qualified_intr|always_armed**

Sets/gets the iice sample mode.

**-datacompression 0|1|off|on|false|true**

Sets data compression.

**-group 1|2|3|4|5|6|7|8**

Sets the specified sample group for sampling.

**-notification 0|1|off|on|false|true**

Disables/enables notification of a trigger event via an LED. If enabled, a top-level port identify_trig_notification_LED_n_*iiceID* (active low) will be added. It is then the user's responsibility to set up an appropriate location constraint for this port.

**iice sampler** [*options*]

The following iice sampler options are supported in the instrumentor:

**-iice** {*iiceID*|**all**} **internal_memory|haps_DTD**
**-compression 0|1**
**-ram** *option* [*value*]
**-rtd {mictorlocs {***location* [*location ...*]**}|board** *boardType***}**
**-depth** *depthValue*
**-qualified_sampling 0|1**
**-always_armed 0|1**

The following iice sampler options are supported in the debugger:

**-triggertime early|middle|late**
**-samplemode normal|qualified_fill|qualified_intr|always_armed**
**-runselftest 0|1**
**-datacompression 0|1**
**-enablemask 0|1** [**-msb** *integer* **-lsb** *integer*] *signalName*
**-group** *interger*

## Instrumentor iice sampler Options

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

Used when more than one regular 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 qualified IICE.

**internal_memory|hapsDTD**

The internal_memory/haps_DTD argument specifies the type of RAM used to capture the sample data; internal_memory uses local FPGA memory (the default) and haps_DTD uses an external RAM daughter board.

**-compression 0|1**

The -compression option determines if data compression is to be applied when the sample data is unchanged between cycles (the data is


© 2020 Synopsys, Inc.

automatically decompressed when viewed). A value of 1 enables data compression. An internal default is set to force an update after 64 cycles of unchanging data. The -compression option applies only to regular IICE units and is not supported by real-time debug IICE.

**-ram {***option value***}**

The -ram (or -sram) option applies only when the *bufferType* is set to hapssram and an external RAM daughter board is installed. The -ram options are described below. The -ram option applies only to regular IICE units and is not supported by real-time debug IICE.

**{board** *type***}**

Specifies the board type for HAPS-60 Series systems; recognized values are HAPS-61, HAPS-62, and HAPS-64.

**{ramlocations** *location***}**

The connector location or locations where the daughter board is physically installed. For a HAPS-60 Series system, *location* is one or more integers between 1 and 6 that identify the J1 through J6 HapsTrak II connectors on the mother board. For a HAPS-70 Series system, *location* is a set of three adjacent HapsTrak 3 connectors on the mother board where the daughter board is installed. Multiple daughter boards can be installed, but only one daughter board can be dedicated to deep trace debug (additional daughter boards are allocated to user data memory).

**{numberboardstack** *integer***}**

The number of SRAM daughter cards stacked at the specified RAM location. *Integer* is 1 or 2 for an SRAM installed on a HAPS-70 Series system and 1, 2, or 4 for an SRAM installed on a HAPS-60 Series system. The integer value specified applies to all connector locations (the stack depth must be the same at all locations).

**{type** *value***}**

The RAM daughter card type according to the following table:

| Type Value | Memory Daughter Board | System Type |
|---|---|---|
| 6 | SRAM_1x_HTII | HAPS-60 Series |
| 7 | SRAM_HT3 | HAPS-70 Series |

**{clockfreq** *value***}**

Specifies the frequency of the clock source in MHz for the SRAM. For more information on SRAM clocks, see the *Debugger User Guide*.

**{clocktype external|internal** *clockName***}**

Specifies the source of the SRAM clock. When internal is specified, the clock source must be identified.

**-rtd** *arguments*

The -rtd option applies only when the IICE type is set to rtd. The -rtd arguments for the real-time debugging feature are described below. The -rtd option applies only to real-time debug IICE and is not supported by regular IICE units.

**mictorlocs** *location* [*location* **...**]

Specifies the location of the Mictor board (or boards) installed in the HapsTrak connectors. Values range from 1 through 6 and more than one location can be specified by separating the values with spaces.

**board** *boardType*

Specifies the HAPS board type. The *boardType* entered must be in all caps.

**-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 and is only supported by regular IICE units. The default setting for *depthValue* is 128.

**-qualified_sampling 0|1**

Enables/disables qualified sampling. When enabled (1), a single sample of all sampled signals is collected each time the trigger condition is true. When a trigger condition occurs, instead of filling the entire buffer, the IICE collects the single sample and then waits for the next trigger to acquire the next sample to allow design operation to be monitored over an extended period of time.

Qualified sampling is impacted by the -samplemode setting; selecting the qualified_fill mode allows single samples to be acquired at each trigger event until the sample buffer is full, and selecting the qualified_intr mode


© 2020 Synopsys, Inc.

allows samples to continue to be acquired (and possibly overwritting existing samples) until interrupted.

The -qualified_sampling option applies only to regular IICE units and is not supported by real-time debug IICE. Using qualified sampling includes a minimal area and clock-speed penalty.

You can also perform qualified sampling using equivalent debugger Tcl commands. The following debugger example command sequence samples the data every *N* cycles beginning with the first trigger event.

```
iice sampler -samplemode qualified_fill
statemachine clear -iice IICE -all
statemachine addtrans -iice IICE -from 0 -to 1
-cond "true" -cntval 0
statemachine addtrans -iice IICE -from 1 -to 2
-cond "c0" -cntval 15 -trigger
statemachine addtrans -iice IICE -from 2 -to 2
-cond "! cntnull" -cnten
statemachine addtrans -iice IICE -from 2 -to 2
-cond "cntnull" -cntval 15 -trigger
```

**-always_armed 0|1**

Enables/disables always-armed sampling. When enabled (1), the 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. The -always_armed option applies only to regular IICE units and is not supported by real-time debug IICE. Using always-armed sampling includes a minimal area and clock-speed penalty.

## Debugger iice sampler Options

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

Controls how a detected trigger affects data sampling (applies only to the 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.

**-samplemode** [**normal**|**qualified_fill**|**qualified_intr**|**always_armed**]

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

**qualified_fill**

Performs qualified sampling until the buffer is full.

**qualified_intr**

Performs qualified sampling until interrupted.

**always_armed**

Always-on triggering.

**-runselftest 0**|**1**

Runs self-test to verify the deep trace debug hardware configuration. The self-test writes data patterns to the external memory and reads back the data pattern written to detect configuration errors, connectivity problems, and frequency mismatches.

**-datacompression 0**|**1**

Compresses debugger data when the sample data is unchanged between cycles (the data is automatically decompressed when viewed). A value of 1 enables data compression. An internal default is set to force an update after 64 cycles of unchanging data.

**-enablemask 0**|**1** [**-msb** *interger* **-lsb** *integer*] *signalName*

Used when -datacompression option is enabled to selectively mask individual bits or buses from being considered as changing values within the sample data.

**-group** *integer*


© 2020 Synopsys, Inc.

Selects multiplexed group of instrumented signals defined in the instrumentor for activation in the debugger. *Integer* is the number of the multiplexed group which ranges from 1 to 8.

**iice assignmentsreport** [**-filename** *fileName*]

Prints the real-time debugging IICE assignments report showing signal, breakpoint, and connector assignment information. Executing this command prior to assigning logic analyzer pods to Mictor pin groups lists only the signals/breakpoints assignments information. Executing the command after assigning logic analyzer pods to Mictor pin groups additional lists the pod assignments. Including the optional -filename argument writes the report to the specified file instead of to standard out.

## Command Example

```
iice clock -edge falling clk2

iice controller -counterwidth 8 statemachine

iice current IICE_2

iice sampler -triggertime late

iice sampler -datacompression 1

iice sampler -enablemask 1 -msb 3 -lsb 0 ctrlbus1a

iice sampler -iice IICE_2 -rtd {mictorloc {1 3 5}}

iice sampler -ram {ramlocations {7_8_9}}
```

# instrumentation 

Manipulates incremental instrumentations.

## Syntax

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

**instrumentation list**

**instrumentation new -instr {***baseName***}**
**-ncdfile {***pathtoFilename***.ncd** [*fpgaName*]**}** |
**-dcpfile {***pathtoFilename***.dcp** [*fpgaName*]**}**

**instrumentation current**

**instrumentation load** *name*

**instrumentation save**

## Arguments and Options

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

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

**list**

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

**new -instr {***baseName***}**
**-ncdfile {***pathtoFilename***.ncd** [*fpgaName*]**}** |
**-dcpfile {***pathtoFilename***.dcp** [*fpgaName*]**}**

**current**

Returns a key value-pair TCL list with information of current instrumentation.

**load** *name*

Loads an existing instrumentation into the instrumentor.


© 2020 Synopsys, Inc.

**save**

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

## Command Example

```
instrumentation load instr_2

instrumentation new –instr {rev_1} –dcpfile
   {./proto/pr_1/post_route.dcp}
```

# jtag_server 

Configures the JTAG server.

## Syntax

**jtag_server set -addr {***hostName|IP_address***} -port {***serverPort***} -logf {***logFfileName***} -usecs 1|0**

**jtag_server get**

**jtag_server stop -forced 0|1**

## Arguments and Options

**set**

Configures the JTAG server

**-addr {***hostName|IP_address***}**

The IP address or the name of the server.

**-port {***serverPort***}**

The port number over which the client and server communicate.

**-logf {***logFfileName***}**

The name of the log file.

**-usecs 1|0**

Enables or disables the client-server configuration for the USB-based UMRBus.

**get**

Returns the server host name or IP address, port number, and log file name.


© 2020 Synopsys, Inc.

**start**

Selects the server startup mode.

**-standalone 0|1**

Selects the server startup mode. If set to 1, the debugger application is closed, and the JTAG server runs in the background.

Selects the cable type.

**stop**

Stops the server.

**-forced 0|1**

A value of 1 immediately stops all communications.

## Command Example

```
jtag_server set -addr myhost -port 58015 -logf servercom.log

jtag_server get
   INFO: addr 127.0.0.1 port 57015 logf ipc_tcp_microchip.log

jtag_server start -standalone 1

jtag_server stop
```

# licenseinfo 

Displays information about the product version and license status.

## Syntax

**licenseinfo**

# logicanalyzer 

Configures the logic analyzer for real-time debugging (RTD). The scan options define the target logic analyzer, the assignpod option describes the analyzer interface, and the submit option sends the data to the logic analyzer. Additional options display the most recently used logic analyzer scan settings (lastscansettings option) and show the logic analyzer's presently scanned pod and module information (pods option).

## Syntax

**logicanalyzer scan -latype tla -hostname** *hostName* **-username** *userName* **-script** *scriptName* **-assignpodsauto yes**|**no**

**logicanalyzer scan -latype la16700**|**la16900 -hostname** *hostName* **-assignpodsauto yes**|**no**

**logicanalyzer assignpod -micconpingrp** *groupName* **-module** *moduleNumber* **-pod** *podIdentifier*

**logicanalyzer submit**

**logicanalyzer lastscansettings**

**logicanalyzer pods**

## Arguments and Options

**-latype**

The type of logic analyzer interfaced to the Mictor connector. Recognized types are tla, la16700, and la16900.


© 2020 Synopsys, Inc.

**-hostname**

The name or IP address (*hostName*) for the debugger host.

**-username**

The user name (*userName*) on the logic analyzer (Tektronix only).

**-script**

The name of the script (*scriptName*) to run to set up logic analyzer (Tektronix only).

**-assignpodsauto**

Determines if pods are automatically assigned to the Mictor connectors.

**-micconpingrp**

The Mictor connector pin group (*groupName*). The connector pin group is identified by the concatenation of the Mictor board HapsTrak connector location, the Mictor connector name, and the Mictor odd/even pin bank separated with periods. For example, 3.M1.e addresses the even bank of Mictor connector M1 on the Mictor board installed in HapsTrak connector 3.

**-module**

The module name.

**-pod**

The connector pod (slot) on the logic analyzer.

## Command Examples

```
logicanalyzer scan -latype la16900 -hostname sisyphus
   -assignpodsauto yes

logicanalyzer assignpod -micconpingrp 2.M1.e -module 1
   -pod A2A3CK0
```

# 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 96

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


© 2020 Synopsys, Inc.

# project 

Opens existing projects and displays project information.

## Syntax

**project import** *projectFile*

**project open** [**-password** *password*] [*fileName*]

**project name** [**-path**]

## Arguments and Options

**import** *projectFile*

Imports the specified project file.

**open** [**-password** *password*] *SynopsysFPGAprojectFile*

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

-**password** *password*

Specifies the password to use to decrypt an encrypted source file (applies only to the 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.

**name** [**-path**]

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

## Command Example

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

### See Also

- *encryption*, on page 35

## pwd 

Displays the current working directory.

### Syntax

**pwd**

### See Also

- *cd*, on page 23

## 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*


© 2020 Synopsys, Inc.

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 -skip_config**

**run -wait_for_trigger** *integer*

## 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 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.

**-skip_config**

Used with a pre-configured trigger to bypass the normal IICE trigger setup configuration.

**-wait_for_trigger** *integer*

Initializes the clock control module and waits *integer* seconds for interrupt.


© 2020 Synopsys, Inc.

**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 debugger menu bar. There is no stop command in the command shell.

## Command Example

```
run -wait_for_trigger 15
```

# 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}
```


© 2020 Synopsys, Inc.

# 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, returns the current value.

## Command Example

```
setsys set lpt_address 4095
```

© 2020 Synopsys, Inc.

# show 

Displays an HDL source code context on the command line.

## Syntax

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

## 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*.

## Command Example

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


© 2020 Synopsys, Inc.

# signals 

Instructs the instrumentor to create special debug logic for the IIICE to sample a signal from your HDL design or to delete the debug logic and return the signal to its "not instrumented" status. The group options assign and report signals in multiplexed groups.

## 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*

**signals group {***groupNumber***}** *sigName* [*sigName* ...]
**signals group -show all**|*sigName* [*sigName* ...]
**signals group -show_tab all**|*sigName* [*sigName* ...]

**signals map {***sigName***}** *connectorPin*

**signals map_fpga** [*options*] *sigName* [*sigName* ...] [**-show_tab** *value*]
[**-fpga** *value*]

**signals preconfigure** [*options*]

## 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 that 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.

**-silent**

Suppresses resource estimation display when a signal is added (by default, adding a signal automatically updates the total instrumentation requirements in the console window).

**-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.

---


© 2020 Synopsys, Inc.

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).

**group {***groupNumber* [*groupNumber*]**}** *sigName* [*sigName* ...]
**group -show all**|*sigName* [*sigName* ...]
**group -show_tab all**|*sigName* [*sigName* ...]

In the first syntax statement, *groupNumber* is an integer value specifying the assigned multiplexed group number from 1 through 8, and *sigName* is the full hierarchical path name of the instrumented signal to be assigned to that group. Multiple signals can be assigned to a group by separating the signal names with spaces, and signals can be assigned to more than one group by including additional group numbers separated by spaces and enclosed in curly braces.

The following options are available with the group argument:

**-show all**|*sigName* [*sigName* ...]

Lists the group or groups assigned to *sigName*. If the all argument is included, lists all of the signals that have been assigned to groups and their group numbers.

**-show_tab all**|*sigName* [*sigName* ...]

Lists the group or groups assigned to *sigName* in tabular format. If the all argument is included, lists all of the signals that have been assigned to groups and their group numbers.

**map {***sigName***}** *connectorPin*

Assigns *sigName* to the specified Mictor connector pin location. In the above syntax, *sigName* is the full hierarchical path name to the signal or bus and *MictorPinName* is the concatenation of the Mictor board HapsTrak connector location, the Mictor connector name, and the Mictor pin name separated with periods. For example, 3.M1.D3e is the D3e pin of Mictor connector M1 on the Mictor board installed in HapsTrak connector 3.

**signals map_fpga** [*options*] *sigName* [*sigName* ...] [**-show_tab** *value*|**all**] [**-fpga** *fpga*]

Maps signals to the specified FPGA. The following options are available with the map_fpga argument:

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

The IICE unit to use or all IICE units.

**-show_tab** [**all**]

Shows the FPGA mapping for the specified the signals listed in tabular form; specifying all shows the mapping for all signals.

**-fpga** *fpga*

The target FPGA for the mapping.

**preconfigure** [*options*] *signalName* [**-msb** *integer* **-lsb** *integer*] [**-field** *argument*]

Sets/gets preconfigured trigger conditions for the specified signal.

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

The IICE unit to use or all IICE units.

**-condition {***triggerCondition***}**|**all**

Specifies the trigger conditions to enable. Accepts a Tcl list of conditions (e.g., {1 2 5}) or all for all conditions.

**-msb** *integer*

The msb of a bus slice to instrument.

**-lsb** *integer*

The lsb of a bus slice to instrument.


© 2020 Synopsys, Inc.

**-field** arg

field of a record to instrument

## Command Example

```
signals add /top/u1/reset_n
signals add -iice IICE_2 -trigger /top/u1/clken
signals add -sample -field iport_mem {/Struc_P_Signed_LDDT_iport}
signals delete -msb 63 -lsb 32 /top/data_in
signals group {2 3} /top/data_in top_data_out
signals group -show_tab all
signals map /beh/blk_xfer_cntrl/req_o 4.M1.D13o
signals map /beh/blk_xfer_inst/beh/{slave_bus[0]} 4.M1.D13o
```

## See Also

- *breakpoints*, on page 21
- 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 debugger.

## Command Example

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


© 2020 Synopsys, Inc.

# statemachine 

Configures the state machine with the desired behavior.

## Syntax

**statemachine addtrans** [**-iice** *iiceID*|**all**] **-from** *state* [**-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 -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 -from** *state*

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

**addtrans** [**-to** *state*]

Specifies the state to which the transition goes. If the -to option is not given, the state 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” (that is, 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, ... c***n*: 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.
- **ti***triggerInID*: 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.


© 2020 Synopsys, Inc.

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

The debugger statemachine clear command deletes all transitions from the states given in the argument, or from all states if the argument -all is specified.

**-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* ...]

The debugger statemachine info command prints the current state machine settings for the states given in the argument, or for the entire state machine, if the option -all is specified. If the option -raw is given, the information is returned in a machine-processible form.

**-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.

## Convenience Functions

Move to ref. There are a number of convenience functions to set up complex triggers available in the file *InstallDir*/share/contrib/syn_trigger_utils.tcl which is loaded into the debugger at startup:

- **st_events** *condition integer* – Sets up the state machine to mimic counter mode events of the simple triggering mode as described above. The argument *condition* is a boolean equation setting up the condition, and *integer* is the counter value.
- **st_watchdog** *condition integer* – Same as st_events for watchdog mode.
- **st_cycles** *condition integer* – Same as above for cycles mode.
- **st_pulsewidth** *condition integer* – Same as above for pulsewidth mode.
- **st_B_after_A** *conditionA conditionB* [*integer***:=1**] – Sets up a trigger mode to trigger if *conditionB* becomes true anytime after *conditionA* became true. The optional *integer* argument defaults to 1 and denotes how many times *conditionB* must become true in order to trigger.
- **st_B_after_A_before_C** *conditionA conditionB conditionC* [*integer***:=1**] – Sets up a trigger mode to trigger if *conditionB* becomes true after *conditionA* becomes true, but without an intervening *conditionC* becoming true (same as the second example above). The optional *integer* argument defaults to 1 and denotes how many times *conditionB* must become true without seeing *conditionC* in order to trigger.
- **st_snapshot_fill** *condition* [*integer*] – Uses qualified sampling to sample data until sample buffer is full. The argument *condition* is a boolean equation defining the trigger condition, and *integer* is the number of samples to take with each occurrence of the trigger (default 1).
- **st_snapshot_intr** *condition* [*integer*] – Uses qualified sampling to sample data until manually interrupted by an debugger stop command. The argument *condition* is a boolean equation defining the trigger condition and *integer* is the number of samples to take with each occurrence of the trigger (default 1).

  Refer to the file syn_trigger_utils.tcl mentioned above for the implementation of these trigger modes using the debugger statemachine command. Users can add their own convenience functions by following the examples in this file.

## 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
```


© 2020 Synopsys, Inc.

```
statemachine addtrans -from 2 -cond "c2 && cntnull" -trigger

statemachine addtrans -from 0 -cond "IICE_1 and IICE_2" -trigger

statemachine addtrans -from 1 -cond "c1 or c2" -cnten

statemachine addtrans -from 0 -to 1 -cond c1 -cntval 9
statemachine addtrans -from 1 -cond "c2 & cntnull" -trigger
statemachine addtrans -from 1 -to 0 -cond c3
statemachine addtrans -from 1 -cond "c2" -cnten

statemachine clear 1

statemachine info -all
```

## Cross Triggering with State Machines

Cross triggering allows a specific IICE unit to be triggered by one or more IICE units in combination with its own internal trigger conditions. The IICE being triggered is referred to as the “destination” IICE; the other IICE units are referred to as the “source” IICE units.

Multiple IICE designs allow triggering and sampling of signals from different clock domains. With an asynchronous design, a separate IICE unit can be assigned to each clock domain, triggers can be set on signals within each IICE unit, and then the IICE units scheduled to trigger each other on a user-defined sequence using cross triggering. In this configuration, each IICE unit is independent and can have unique IICE parameter settings including sample depth, sample/trigger options, and sample clock and clock edges.

Cross triggering is supported in all three IICE controller configurations (simple, complex counter, and state-machine triggering) and all three configurations make use of state machines.

Cross triggering is enabled in the instrumentor (cross triggering can be selectively disabled in the debugger). To enable a destination IICE unit to accept a trigger from a source IICE unit, enter the following command in the instrumentor console window (by default, cross triggering is disabled):

```
iice controller -crosstrigger 1
```

For cross triggering to function correctly, the destination and the contributing source IICE units must be instrumented by selecting breakpoints and watchpoints. Concurrently run these units either by selecting the individual IICE units and clicking the RUN button in the debugger project view or by entering one of the following commands in the debugger console window:

```
run -iice all
run -iice {iiceID1 iiceID2 ... iiceIDn}
```

When simple- or complex-counter triggering is selected in the destination IICE controller, the following debugger cross-trigger commands are available:

- The following debugger command causes the destination IICE to trigger normally (the triggers from source IICE units are ignored).

  ```
  iice controller -crosstriggermode DISABLED
  ```

- The following debugger command causes the destination IICE to trigger when any source IICE triggers or on its own internal trigger.

  ```
  iice controller -crosstriggermode ANY
  ```

- The following debugger command causes the destination IICE to trigger when all source IICE units and the destination IICE unit have triggered in any order.

  ```
  iice controller -crosstriggermode ALL
  ```

- The following debugger commands cause the destination IICE to trigger after the source IICE unit triggers coincident with the next destination IICE internal trigger.

  ```
  iice controller -crosstriggermode after -crosstriggeriice iiceID
  iice controller -crosstriggermode after -crosstriggeriice all
  ```

  The first debugger command uses a single source IICE unit (*iiceID*), and the second debugger command requires all source IICE units to trigger.

When state-machine triggering is selected, the state machine must be specified with at least three states (three states are required for certain triggering conditions, for example, when the destination IICE is in Cycles mode and you want to configure the destination IICE to trigger after another (source) IICE.

With state-machine triggering, the following debugger statemachine command sequences are available in the debugger console window:

- The following debugger command sequence is equivalent to disabling cross triggering. The destination IICE triggers on its own internal trigger condition (c0).

  ```
  statemachine clear -all
  statemachine addtrans -from 0 -cond "c0" -trigger
  ```


© 2020 Synopsys, Inc.

- In the following debugger command sequence, the destination IICE waits for *iiceID* to trigger and then triggers on its own internal trigger condition (c0). This sequence implements the “after *iiceID*” functionality of the simple- and complex-counter triggering modes.

```
statemachine clear -all
statemachine addtrans -from 0 -to 1 -cond "iiceID"
statemachine addtrans -from 1 -to 0 -cond "c0" -trigger
```

- In the following debugger command sequence, the destination IICE triggers when the last running IICE triggers.

```
statemachine clear -all
statemachine addtrans -from 0 -cond "c0 and iiceID and iiceID1
   and iiceID2" -trigger
statemachine addtrans -from 0 -to 1 -cond "c0"
statemachine addtrans -from 1 -to 0 -cond "iiceID and iiceID1
   and iiceID2" -trigger
```

- In the following debugger command sequence, the destination IICE waits for all the other running source IICE units to trigger and then triggers on its own internal trigger condition (c0).

```
statemachine clear -all
statemachine addtrans -from 0 -to 1 -cond "iiceID and iiceID1
   and iiceID2"
statemachine addtrans -from 1 -cond "c0" -trigger"
```

The incorporation of a counter in the state-machine configuration is similar to the use of a counter in non-cross trigger mode for a state machine.

## State-machine Triggering with Tcl Commands

The IICE can be configured using Tcl commands entered from both the instrumentor and debugger console windows. Some of the example commands are as follows:

- To delete the state transitions from each IICE, use the following debugger command:

```
statemachine clear -iice all
```

- To enable complex counter triggering, use the following instrumentor command:

```
iice controller complex
```

- To set the counter width, use the following instrumentor command:

```
iice controller -counterwidth 8
```

- To configure an IICE for state-machine triggering, use the following instrumentor command sequence:

```
iice controller -iice IICE statemachine
iice controller -iice IICE -counterwidth 4
iice controller -iice IICE -triggerconditions 2
iice controller -iice IICE -triggerstates 2
```

  In addition to state-machine triggering, the above instrumentor commands set the number of trigger conditions to 2 and the number of trigger states to 2.

- To enable cross triggering, use the following instrumentor command:

```
iice controller -crosstrigger 1
```

- Similarly, to configure the sample depth, use the following instrumentor command:

```
iice sampler -depth 2048
```

Note that the only option for buffer type is internal_memory.

## 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
- *condition info*, on page 28
- *Cross Triggering with State Machines*, on page 84
- *State-machine Triggering with Tcl Commands*, on page 86

**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.


© 2020 Synopsys, Inc.

# stop 

Activates/deactivates an HDL source-level breakpoint that has been added by the 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 or breakpoints. If only one trigger condition exists in the current design, this option can be omitted, otherwise it is required. The identifier all disables all breakpoints 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 breakpoints. If only one trigger condition exists in the current design, this option can be omitted, otherwise it is required. The identifier all enables the breakpoints 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 21
- *iice*, on page 48


© 2020 Synopsys, Inc.

# 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 instrumentor and syn_hhd.scr for the debugger.

## Default Value

By default, command recording is off.

## Command Example

```
transcript on
```

## See Also

- *log*, on page 65

# verdi 

Imports or instruments signals from the Verdi essential signal database.

## Syntax

**verdi getsignals** *ESDBpath*
**verdi instrument**

## Arguments and Options

*ESDBpath* is the location where es.esdb++ is installed.


© 2020 Synopsys, Inc.

# 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 an HDL source-level watchpoint. 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 that the LSB value.

© 2020 Synopsys, Inc.

**-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 that 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 one or more watchpoints. If only one trigger condition exists in the current design, this option can be omitted, otherwise it is required. The identifier all enables the watchpoints from all trigger conditions.**watch disable -condition** (*triggerCondition*|**all**) *signalName*


© 2020 Synopsys, Inc.

- **watch enable -condition** (*triggerCondition*|**all**) *signalName value1* [*value2* ...]
- **watch info** [**-raw**] *signalName*

The parameter *triggerCondition* is a list value conforming to the Tcl language. Examples are: `1`, `"1 2 3"`, `{2 3}`, or `[list 1 2 3]`, quotes, braces, and brackets included, respectively. Alternatively, the keyword all can be specified to apply the setting to all trigger conditions.

The debugger watch info command reports status information about the signal. This information is returned in machine-processible form if the optional parameter -raw is specified.

Similarly for the debugger stop command:

- **stop enable -condition** (*triggerCondition*|**all**) *breakpoint*
- **stop disable -condition** (*triggerCondition*|**all**) *breakpoint*
- **stop info** [**-raw**] *breakpoint*

The semantics of the parameters are identical to the above descriptions.

**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 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

```
watch enable /top/u2/current_state {red}
watch enable -condition {1 2} /top/u1/count {"0X01"} {"0010"}
watch radix current_state hex
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 74
- controller -triggerconditions option – *iice*, on page 48

## waveform 

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

### Syntax

**waveform custom** [*userProcedure*]

**waveform period** [*period_in_ns*]

**waveform show** [*options*]

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

### Arguments and Options

**custom** [*userProcedure*]


© 2020 Synopsys, Inc.

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
- *toplevel* – 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**|**verdi**|**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. An asterisk preceding the waveform viewer name in the list indicates the currently selected viewer.

## Command Example

```
waveform viewer -list

waveform show -showequiv
```


© 2020 Synopsys, Inc.

# write fsdb 

Writes the sample data of each specified signal in FSDB format for analysis and display in Verdi nWave.

## Syntax

**write fsdb** [*options*] *fsdbFilename*

## Arguments and Options

**-iice** *iiceID*

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

**-showequiv**

Includes the sample data for all equivalent signals.

**-range {***start stop***}**

Writes a specified range of sample data to the FSDB file.

*fsdbFilename*

Writes the sample data to the specified fast signal database output file.

## Command Example

```
write fsdb D:/tmp/b.fsdb
```

# write instrumentation 

Writes the instrumented design files to the project directory.

## Syntax

**write instrumentation** *options*

## Options

**-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.

**-idc_loc** *directory*

Save Identify constraints to the specified directory.

**-idc_only**

Save only the idc file.

**-cdc_only**

Save only the CDC file.

## Command Example

```
write instrumentation -encrypt_orig_scr
write instrumentation -save_orig_src
write instrumentation -idc_only
```


© 2020 Synopsys, Inc.

## See Also

- *encryption*, on page 35

# 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 that 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.


© 2020 Synopsys, Inc.

**-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 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 the specified output file.

## Command Example

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


© 2020 Synopsys, Inc.

# 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.vhd
```

CHAPTER 3

# User Interface Commands

This chapter describes the graphical user interface (GUI) commands available from the menus:


- File Menu, on page 106
- Edit Menu, on page 110
- View Menu, on page 111
- Instrumentor, on page 113
- Debugger, on page 129
- Options Menu, on page 148
- Window, on page 150
- Help Menu, on page 151


For information about context-sensitive commands accessed from right-click popup menus, see *GUI Popup Menu Commands*, on page 177.


Synopsys Confidential Information

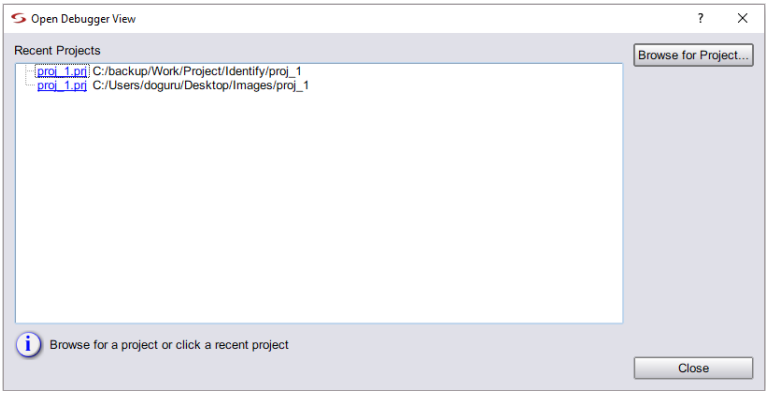
# File Menu

Use the File menu for opening, loading trigger settings, saving, and closing projects and files. The following table describes the File menu commands.

| Feature | Description |
|---|---|
| Open Debugger Project | Opens debugger view. It also has shortcuts to Board Bring-Up, Confpro and Configure Confpro utilities. See Open Debugger Project, on page 106. |
| Open | Opens a file browser to allow you to locate an existing file to open. |
| Close | Closes the currently opened project. |
| Load Activation | Allows you to reload a set of trigger settings. |
| Save Activation | Allows you to save a set of trigger settings. |
| Save All | Save all the unsaved changes. |
| Print | Opens the dialog to select a printer from a list of printers for printing the files. |
| Print Setup | Opens a dialog to specify the paper type, page orientation, and page margins for printing the files. |
| Create Image | This command is available in the following views:<br>• HDL Analyst Views<br>• FSM Viewer<br>Displays a camera pointer ( ). Drag a selection rectangle around the region for which you want to create an image, then release the mouse button. Clicking the current view opens the Create Image dialog box. See Create Image Command, on page 107. |
| Recent Files | Opens a submenu containing the list of recently used files for opening. |
| Exit | Closes the application. |

## Open Debugger Project

Click File > Open Debugger Project to open a debugger file.




| Feature | Description |
|---|---|
| Recent Projects | Displays a list of recently opened projects, if any. Click a debug.prj link to open the project. |
| Browse for Project | Opens a browser to select the location of a different debug.prj file. |
| Close | Navigates to the Debugger window. |

When you open a project from either the tool or system, the working directory is automatically set from the corresponding project file. If the project was saved with encrypted original sources, you are prompted to enter the original password used to encrypt the files. This password is then used to read any encrypted files.

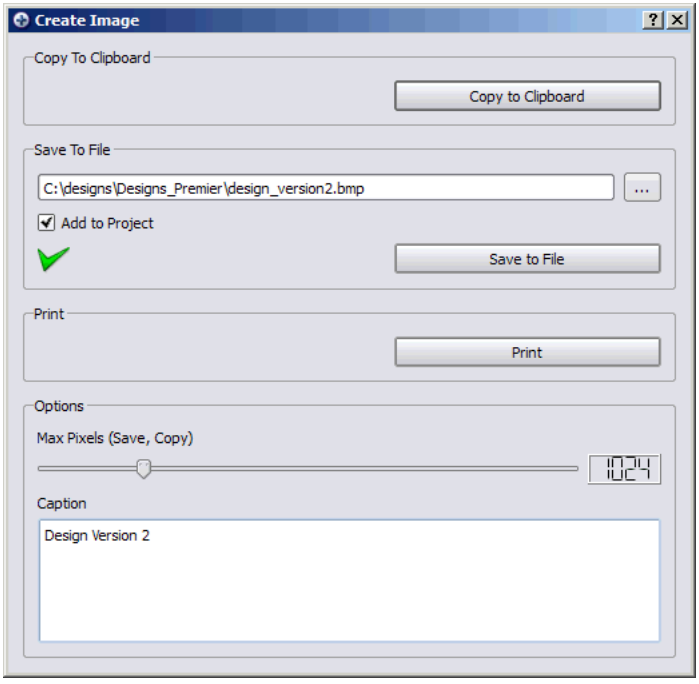
## Create Image Command

Select File > Create Image to capture an image from any of the following views:

- HDL Analyst Views
- FSM Viewer

Drag the camera cursor to define the area for the image. When you release the cursor, the Create Image dialog box appears. Use the dialog box to copy the image, save the image to a file, or to print the image.


© 2020 Synopsys, Inc.




| Feature | Description |
|---|---|
| Copy to Clipboard | Copies the image to the clipboard so you can paste it into a selected application (for example, a Microsoft Word file). When you copy an image to the clipboard, a green check mark appears in the Copy To Clipboard field. |
| Save to File | Saves the image to the specified file. You can save the file in various formats (platform dependent) including bmp, jpg, png, ppm, tif, xbm, and xpm. |
| Add to Project | Adds the saved image file to the Images folder in the Project view. This option is enabled by default. |
| Save to File button | You must click this button to save an image to the specified file. When you save the image, a green check mark appears in the Save To File field. |

© 2020 Synopsys, Inc.

| Feature | Description |
|---|---|
| Print | Prints the image. When you print the image, a green check mark appears in the Print field. |
| Options | Allows you to select the resolution of the image saved to a file or copied to the clipboard. Use the Max Pixels slider to change the image resolution. |
| Caption | Allows you to enter a caption for a saved or copied image. The overlay for the caption is at the top-left corner of the image. |


© 2020 Synopsys, Inc.

# Edit Menu

Use the Edit menu to edit text files (such as HDL source files) in your project. This includes cutting, copying, pasting, finding, replacing text, and toggle between bookmarks.

| Feature | Description |
|---|---|
| Undo | Cancels the last action. |
| Redo | Performs the action undone by Undo. |
| Cut | Removes the selected text and makes it available to Paste. |
| Copy | Duplicates the selected text and makes it available to Paste. |
| Paste | Pastes text that was cut (Cut) or copied (Copy). |
| Select All | Selects the entire text of the script. |
| Find | Searches the file for text matching a given search string. This option will be enabled while using the HDL Analyst to find certain instances, ports, pins, nets in the schematic. |
| Goto | Goes to a specific line number in the RTL script. |
| Toggle bookmarks | Toggles between inserting and removing a bookmark on the line that contains the text cursor. |
| Next bookmarks | Takes you to the next bookmark. |
| Previous bookmarks | Takes you to the previous bookmark. |
| Delete all bookmarks | Removes all bookmarks from the Text Editor window. |

© 2020 Synopsys, Inc.

# View Menu

Use the View menu to set the display and viewing options, choose toolbars, and enable or disable the debugger panels.

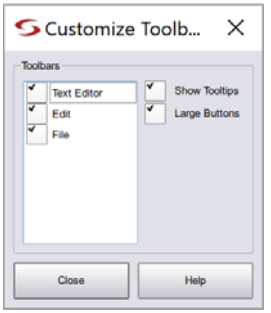
| Feature | Description |
|---|---|
| Font Size | Changes the font size in the tool. You can select one of the following options:<br>• Increase Font Size<br>• Decrease Font Size<br>• Reset Font Size (default size) |
| Toolbars | Displays the Toolbars dialog box, where you specify the toolbars to display. See Toolbar Command, on page 111. |
| Status Bar | When enabled, displays context-sensitive information in the lower-left corner of the main window as you move the mouse pointer over design elements. This information includes element identification. |
| Tcl Window | When enabled, displays the Tcl Script and Messages panels. All commands you execute in the Project view appear in the Tcl window. You can enter or paste Tcl commands and scripts in the Tcl window. Check for notes, warnings, and errors in the Messages window. See Debugger Panels, on page 112. |
| Debugger Panels | You can toggle between the panels by selecting from the sub-menu. See Debugger Panels, on page 112. |

## Toolbar Command

Select View > Toolbars to display the Toolbars dialog box, where you can:

- Choose the toolbars to display
- Customize the appearance of tool


© 2020 Synopsys, Inc.




| Feature | Description |
|---|---|
| Toolbars | Lists the available toolbars. Select the toolbars that you want to display. |
| Show Tooltips | When selected, a descriptive tooltip appears whenever you position the pointer over an icon. |
| Large Buttons | When selected, large icons are displayed. |

## Debugger Panels

You can toggle between the control/search, browser, and view panels.

| Feature | Description |
|---|---|
| Toggle Control/Search Panel | Click to view or hide the Search panel. |
| Toggle Browser Panel | Click to view or hide the Browser (hierarchy) panel. |
| Toggle View Panel | Click to view or hide the View (RTL, Watchpoints, Breakpoints, and Sampled Signals tabs) panel. |
| Show All Panels | Click to view all panels. |

# Instrumentor

Use the Instrumentor menu to work with In-Circuit Emulator (IICE™), search instrumentor source, and define instrumentor preferences.

| Feature | Description |
|---|---|
| IICE | Lists the IICE related activities in a sub-menu. See IICE, on page 114.<br>• Add IICE - To add an IICE. See Add IICE, on page 114.<br>• Delete IICE - To delete an existing IICE. See Delete IICE, on page 114.<br>• Rename IICE - To rename the name of the IICE. See Rename IICE, on page 114.<br>• Edit IICE - To edit or define the IICE parameters. See Edit IICE, on page 115.<br>• Edit RTD IICE Mappings - This feature can be used to change the RTD pin mappings. RTD uses a daughter card (Mictor/LAI). |
| SRS Instrumentation View | Displays the instrumentation schematics. |
| Find in Instrumentor Source | Finds target expression in instrumentor source. Can set additional parameters to filter results. See Find in Instrumentor Source, on page 123. |
| Find Next in Instrumentor Source | Finds the next instance of target expression set in Find in instrumentor source. See Find in Instrumentor Source, on page 123. |
| Make Incremental | Makes incremental instrumentation. This can improve the runtime for large designs. |


© 2020 Synopsys, Inc.

| Feature | Description |
|---|---|
| Estimate Resources | Estimates the required resources for each design module and creates an estimation (est) file and log (_est.srr). |
| Instrumentor Preferences | Click to define the instrumentor preferences. See Instrmentor Preferences, on page 124. |
| Windows | Click and select the panel from the sub-menu.<br>• Control/Search Panel — To view or hide the Control and Search panels.<br>• Browser — To view or hide the Browser (hierarchy) panel.<br>• View panel — To view or hide the View (RTL, Instrumentation tabs) panel.<br>• Show All — To view all panels. |

# IICE

An important part preparing a design for debugging is setting the parameters for the IICE in the instrumentor. The IICE parameters determine the implementation of one or more IICE units and configure the units to establish the proper communication with the debugger.

## Add IICE

Click to add an IICE to the design. For information on adding IICE, see the *Debugger User Guide*.

## Delete IICE

Click to delete an existing IICE in the design. For information on deleting IICE, see the *Debugger User Guide*.

## Rename IICE

Click to rename the existing IICE name. A dialog is displayed to enter the new IICE name.

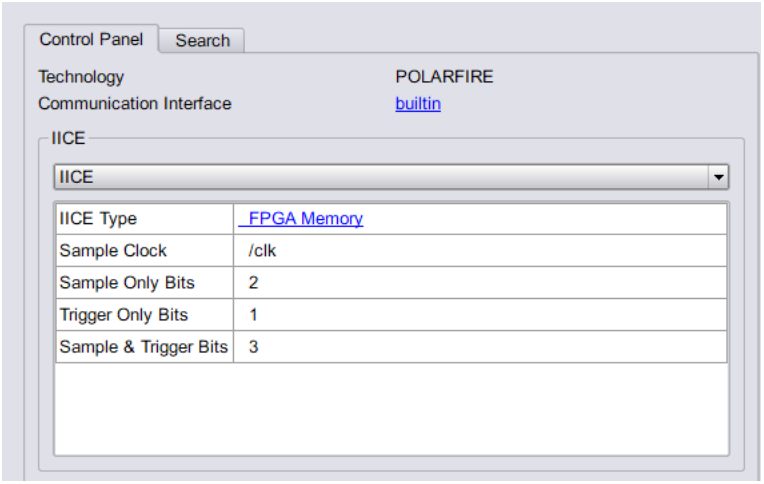
## Edit IICE

The IICE parameters unique to each IICE definition in a multi-IICE configuration are interactively set on the Edit IICE dialog box tabs. For information on steps to define and edit IICE parameters, see the *Debugger User Guide*.

**Note:** IICE configurations set in the instrumentor impact the operations available in the debugger.



The individual parameters for each IICE are defined using a series of tabs of the Edit IICE Settings dialog box. Select the target IICE unit in the Control Panel tab before you open this dialog box.




With the target IICE selected, select Instrumentor > IICE > Edit IICE or click the IICE Type field in the Control Panel to display the Edit IICE Settings dialog box.

The Edit IICE dialog has the following tabs.

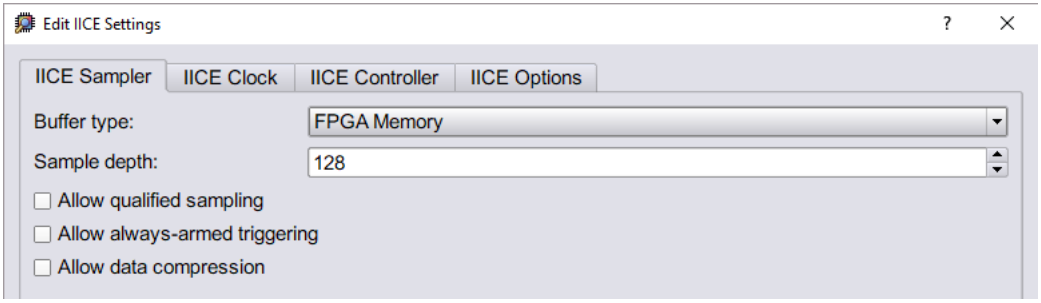
- IICE Sampler Tab, on page 116
- IICE Clock Tab, on page 117
- IICE Controller Tab, on page 119
- IICE Options Tab, on page 122


© 2020 Synopsys, Inc.

## IICE Sampler Tab

The IICE Sampler tab is the default tab and defines:

- Buffer type
- Sample depth
- Sampling/triggering options




### Buffer Type

The Buffer type parameter specifies the type of RAM to be used to capture the on-chip signal data. The default value is FPGA Memory.

### Sample Depth

The Sample depth parameter specifies the amount of data captured for each sampled signal. Sample depth is limited by the capacity of the FPGAs implementing the design, but must be at least 8 due to the pipelined architecture of the IICE.

Sample depth can be maximized by taking into account the amount of RAM available on the FPGA. As an example, if only a small amount of block RAM is used in the design, then a large amount of signal data can be captured into block RAM. If most of the block RAM is used for the design, then only a small amount is available to be used for signal data. In this case, it may be more advantageous to use logic RAM. The sample depth increases significantly with the deep-trace debug feature.

### Allow Qualified Sampling

The Allow qualified sampling check box, when checked, causes the instrumentor to build an IICE block that is capable of performing qualified sampling. When qualified sampling is enabled, one data value is sampled each time the trigger condition is true. With qualified sampling, 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. For more information on qualified sampling, see *-qualified_sampling* under the iice command description.

### Allow Always-Armed Triggering

The Allow always-armed triggering check box, when checked, saves the sample buffer for the most recent trigger and waits for the next trigger or until interrupted. When always-armed sampling is enabled, a snapshot is taken each time the trigger condition becomes true.

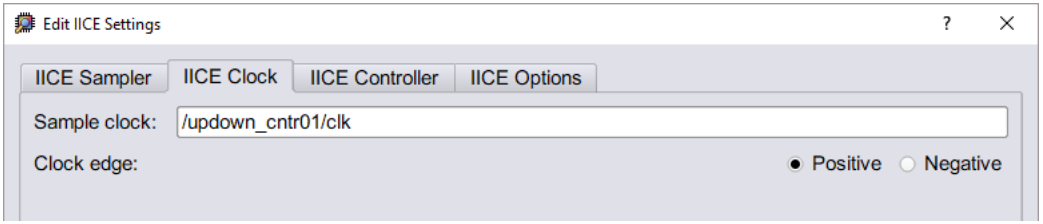
With always-armed triggering, you always acquire the data associated with the last trigger condition prior to the interrupt. This mode is helpful when analyzing a design that uses a repeated pattern as a trigger (for example, bus cycles) and then randomly freezes. You can retrieve the data corresponding to the last time the repeated pattern occurred prior to freezing. Using always-armed sampling includes a minimal area and clock-speed penalty.

### Allow Data Compression

The Allow data compression check box, when checked, adds compression logic to the IICE to support sample data compression in the debugger (see the *Debugger User Guide*). When unchecked (the default), compression logic is excluded from the IICE, and data compression in the debugger is unavailable. Note that there is a logic data overhead associated with data compression and that the check box should be left unchecked when sample data compression is not to be used.

## IICE Clock Tab

The IICE Clock tab defines the sample clock.

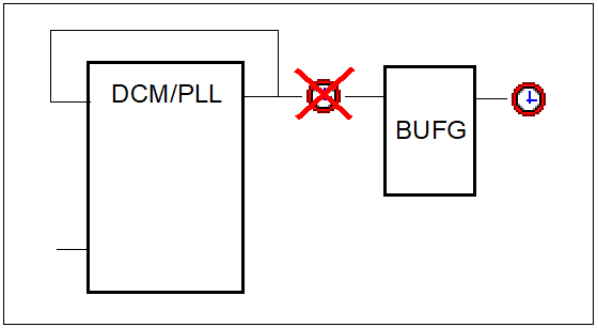



### Sample Clock

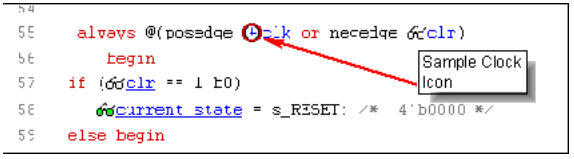

© 2020 Synopsys, Inc.

The Sample clock parameter determines when signal data is captured by the IICE. The sample clock can be any signal in the design that is a single-bit scalar type. Enter the complete hierarchical path of the signal as the parameter value.

Care must be taken when selecting a sample clock because signals are sampled on an edge of the clock. For the sample values to be valid, the signals being sampled must be stable when the specified edge of the sample clock occurs. Usually, the sample clock is either the same clock with which the sampled signals are synchronous or a multiple of that clock; an asynchronous clock can also be selected as sampling clock. The sample clock must use a scalar, global clock resource of the chip and should be the highest clock frequency available in the design. The source of the clock must be the output from a BUFG/IBUFG device.




You can also select the sample clock from the RTL window by right-clicking on the watchpoint icon in the source code display and selecting Sample Clock from the popup menu. The icon for the selected (single-bit) signal changes to a clock face as shown in the following figure.

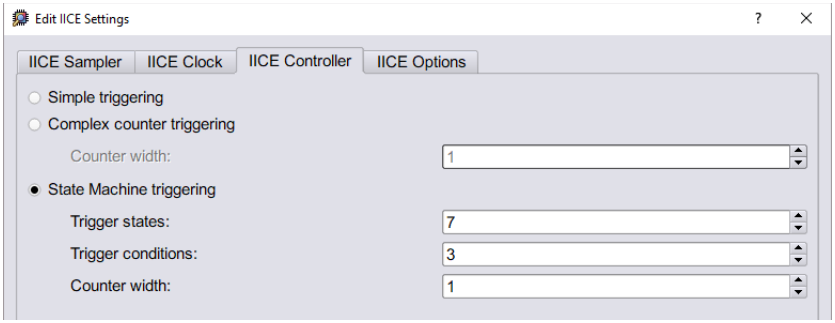



### Clock Edge

The Clock edge radio buttons determine if samples are taken on the rising (positive) or falling (negative) edge of the sample clock. The default is the positive edge.

## IICE Controller Tab

The IICE Controller tab selects the IICE controller's triggering mode. All of these instrumentation choices have a corresponding effect on the area cost of the IICE.




### Simple Triggering

Simple triggering allows you to combine breakpoints and watchpoints to create a trigger condition for capturing the sample data.

### Complex-Counter Triggering

Complex-counter triggering augments the simple triggering by instrumenting a variable-width counter that can be used to create a more complex trigger function. Use the width setting to control the desired width of the counter.

### State-Machine Triggering

State-machine triggering allows you to pre-instrument a variable-sized state machine that can be used to specify an ultimately flexible trigger condition. Use Trigger states to customize how many states are available in the state machine. Use Trigger condition to control how many independent trigger conditions can be defined in the state machine. Use Counter width to control the desired width of the counter. For more information on state-machine triggering, see the *Debugger User Guide*.

Setting up an instrumented design to enable advanced triggering is extremely easy. There are two iice controller command options available in the instrumentor that control the extent and cost of the instrumentation:

- **-triggerconditions** *integer* – The *integer* argument to this option defines how many trigger conditions are created. The range is from 1 to 16. All these trigger conditions are identical in terms of signals and breakpoints


© 2020 Synopsys, Inc.

connected to them, but they can be programmed separately in the debugger.

- **-triggerstates** *integer* – The *integer* argument to this option defines how many states the trigger state machine will have. The range is 2 to 10; powers of 2 are preferable as other numbers limit functionality and do not provide any cost savings.

Similar to the simple-triggering mode, a counter can be instrumented to augment the functionality of the state machine. To instrument a counter, enter an iice controller -counterwidth option with an argument greater than 0 in the instrumentor console window.

Refer to the following text to determine cost and consequences of these settings in the instrumentor.

Structural Implementation of State Machine Triggering

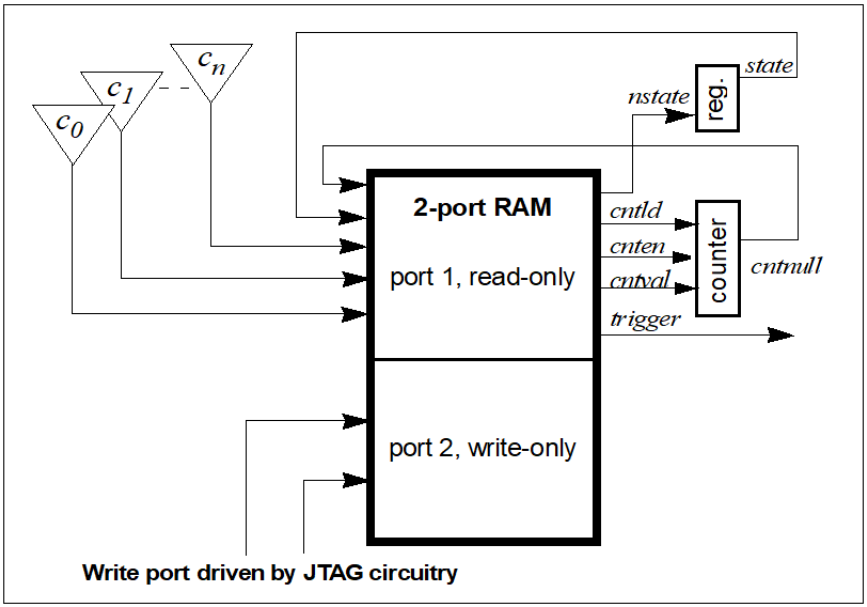
For each trigger condition $c_i$, a logic cone is implemented which evaluates the signals and the breakpoints connected to the trigger logic and culminates in a 1-bit result identical to the trigger condition in simple mode. All these 1-bit results are connected to the address inputs of a RAM table.

If a counter has been added to the instrumentation, the counter output is compared to constant 0, and the single-bit output of that comparison is also connected to the address inputs of the same RAM table.

The other address inputs are provided by the state register. The outputs of the RAM table are:

- The next-state value nstate
- The trigger signal trigger (causes the sample buffer to take a snapshot if high)
- The counter-enable signal cnten (if '1', counter is decremented by 1)
- The counter-load signal cntld (if '1', counter is loaded with cntval)
- The counter value cntval (only useful in conjunction with cntld)

The last three outputs are only present if a counter is instrumented. Also refer to the figure below.




The implementation of the RAM table is identical to the implementation of the sample buffer (that is, the device buffertype setting selects the implementation of both the sample buffer and the state-machine RAM table).

### Cost Estimation

The most critical setting with respect to cost is the number of trigger conditions, as each trigger condition results in an additional address bit on the RAM, and thus doubles the size of the RAM table with each bit. Next in importance is the counter width as this factor contributes directly to RAM table width and is especially significant in the context of FPGA RAM primitives that allow a trade-off of width for depth.

The block RAM on Microchip SmartFusion2, IGLOO2, and RTG4 devices includes 18k and 1k bits per block and PolarFire devices includes 20k and 768 bits per block. The following table provides some hints for good, trigger state-machine settings for the smaller, 20k-bit, 18k-bit, 1k-bit, and 768-bit devices when using only a single block for the trigger-state machine.


© 2020 Synopsys, Inc.

| RAM size | | | With counter | | | Without counter | |
|---|---|---|---|---|---|---|---|
| **Address** | **Depth** | **Data** | **Conditions** | **States** | **Counter** | **Conditions** | **States** |
| 10-bit | 1024 | 18-bit | 6 | 8 | 12-bit | No useful setting | |
| 6-bit | 1024 | 18-bit | 7 | 4 | 13-bit | No useful setting | |
| 6-bit | 64 | 18-bit | 3 | 4 | 13-bit | No useful setting | |
| 6-bit | 64 | 18-bit | 2 | 8 | 12-bit | No useful setting | |

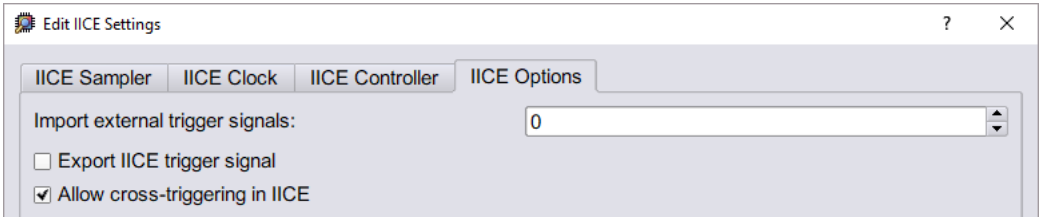
| RAM size | | | With counter | | | Without counter | |
|---|---|---|---|---|---|---|---|
| **Address** | **Depth** | **Data** | **Conditions** | **States** | **Counter** | **Conditions** | **States** |
| 10-bit | 1024 | 20-bit | 6 | 8 | 14-bit | No useful setting | |
| 6-bit | 1024 | 20-bit | 7 | 4 | 15-bit | No useful setting | |
| 6-bit | 64 | 12-bit | 3 | 4 | 7-bit | No useful setting | |
| 6-bit | 64 | 12-bit | 2 | 8 | 6-bit | No useful setting | |

The actual instrumentation, however, is not limited to the values provided, nor is it limited to the use of a single block RAM (for example, it may be advantageous in a particular situation to trade away states for additional trigger conditions or for additional counter width). Any configuration can be automatically implemented, as long as it fits on the device with the remainder of the design.

Although RAM parameters are automatically determined by the instrumentor, this information should be monitored to make sure that no resources are wasted.

## IICE Options Tab

The IICE Options tab configures external triggering to allow a trigger from an external source to be imported and configured as a trigger condition for the active IICE. The external source can be a second IICE located on a different device or external logic on the board rather than the result of an instrumentation.




Import External Trigger Signals

The imported trigger signal includes the same triggering capabilities as the internal trigger sources used with state machines. The adjacent field selects the number of external trigger sources with 0, the default, disabling recognition of any external trigger. Selecting one or more external triggers automatically enables state-machine triggering.

**Note:** When using external triggers, the pin assignments for the corresponding input ports must be defined in the synthesis or place and route tool.

Export IICE Trigger Signal

The Export IICE trigger signal check box, when checked, causes the master trigger signal of the IICE hardware to be exported to the top-level of the instrumented design.

Allow cross-triggering in IICE

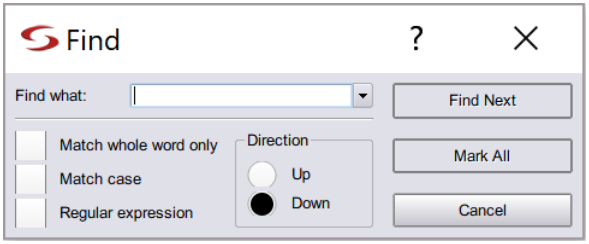
The Allow cross-triggering in IICE check box, when checked, allows this IICE unit to accept a cross-trigger from another IICE unit. For more information on cross-triggering, see the *Debugger User Guide*.

Enable using the following command in the instrumentor console window:
`iice controller -crosstrigger 1`

## Find in Instrumentor Source

Select Instrumentor > Find in instrumentor Source to search and find the required information in the text editor.


© 2020 Synopsys, Inc.

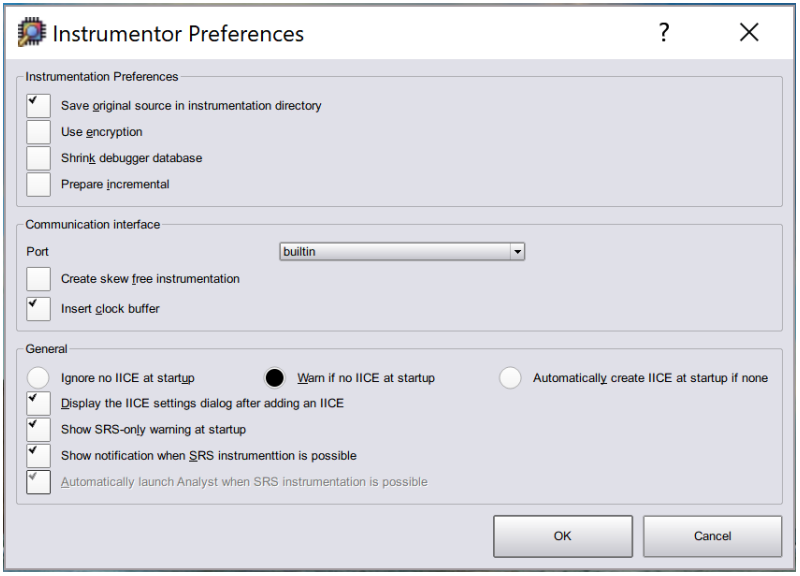



| Field/Option | Description |
|---|---|
| Find What/Search for | Search string matching the text to find. In the text editor, you can use the pull-down list to view and reuse search strings used previously in the current session. |
| Match whole word only | When enabled, matches the entire word rather than a portion of the word. |
| Match Case | When enabled, searching is case sensitive. |
| Regular expression | When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including an empty string. |
| Direction | Changes search direction to Up or Down. |
| Find Next | Initiates a search for the search string (see Find What/Search for). In the text editor, searching starts again after reaching the end (Down) or beginning (Up) of the file. |
| Mark All | Highlights the line numbers of the text matching the search string and closes the Find dialog box. |

## Instrmentor Preferences

The IICE parameters common to all IICE units are set in the Instrumentor Preferences dialog box.

Select Instrumentor > Instrumentor Preferences to define the IICE parameters in the Instrumentor Preferences dialog box.




The IICE parameters unique to each IICE definition in a multi-IICE configuration are interactively set on the Edit IICE dialog box tabs.

| Field | Description |
|---|---|
| Save original source in instrumentation directory | Before you save and instrument the design, select this option to include the original HDL source in the design. It simplifies the design transfer when instrumentation and debugging are performed on separate machines and is especially useful when a design is being debugged on a system that does not have access to the original sources. |
| Use encryption | Select if the original sources are encrypted when they are written. See User Encryption, on page 126. |
| Shrink debugger database | Select to reduce the size of the database while exporting without loosing the data. |
| Prepare incremental | Select to prepare the design for incremental instrumentation. |
| Port | Select the required communication port. See Port, on page 127. |


© 2020 Synopsys, Inc.

| Field | Description |
|---|---|
| Create skew free instrumentation | When enabled, you can avoid an additional global clock buffer to prevent clock skew for JTAG clock. |
| Insert clock buffer | Inserts the clock buffer in the design. |
| Ignore no IICE at startup | You can choose to ignore and run the instrumentation without any IICE. |
| Warn if no IICE at startup | Displays a warning if no IICE is defined. |
| Automatically create IICE at startup if none | Creates an IICE automatically if no IICE is defined at startup. |
| Display the IICE settings dialog after adding an IICE | Displays the Edit IICE dialog to define the IICE parameters when you add a new IICE. |
| Show SRS-only warning at startup | Display the SRS-only warning while launching the Instrumentor. |
| Show notification when SRS instrumentation is possible | Displays a pop-up message while launching Identify Instrumentor. |
| Automatically launch Analyst when SRS instrumentation is possible | The compiled schematic (SRS view) in HDL Analyst is displayed in a separate tab. |

## User Encryption

The encryption is based on a password that is requested when you write out the instrumented design. Encryption allows you to debug on a machine that you feel would not be sufficiently secure to store your sources. After you export the files to the unsecure machine, you are prompted to reenter the encryption password when you open the design in the debugger. The decrypted files are never written to the unsecure machine's hard disk.

For maximum security when selecting an encryption password:

- Use spaces to create phrases of four or more words (multiple words defeat dictionary-type matching)
- Include numbers, punctuation marks, and spaces
- Make passwords greater than 16 characters in length

- Passwords are the user's responsibility; Synopsys cannot recover a lost or forgotten password.

## Port

Specifies the type of connection to be used to communicate with the on-chip IICE. The connection types available from the drop-down menu are:

- builtin – indicates that the IICE is connected to the JTAG Tap controller available on the target device.

**UJTAG Wrapper Support for Microchip FPGA Devices**

To debug a design which contains user JTAG interface module, you must use the UJTAG_WRAPPER module and not the UJTAG module. This is because Identify debugger also uses JTAG ports, so using the UJTAG module for the user JTAG interface module will result in conflicts.

To use the UJTAG_WRAPPER module and ensure seamless connection between the user JTAG and the Identify debugger JTAG:

- Instantiate UJTAG_WRAPPER in the design.
- Make necessary RTL interface connections for TDI, CLK, SHIFT_EN, CAP, RESET, IR_REG and TDO ports of user's JTAG interface module with UTDI, UDRCK, UDRSH, UDRCAP, URSTB, UIREG and UTDO ports of UJTAG_WRAPPER respectively.

There are some restrictions for the UREG connection. Two UJTAG OPCODEs, UIREG[5:1] = 5'b00001 and UIREG[5:1] = 5'b00010 are reserved for use by Identify. Also, UIREG[6] is used by Identify as an enable signal.

- The ujtag_wrapper.v file is available in the installation folder <synplify_dir>/lib/di/. Add the file to the Synplify Pro project.
- If you want to instrument the design using Identify, then you must add `define IDENTIFY_DEBUG_IMPL in the Verilog source file or specify the set_option in Synplify Pro:

```
set_option -hdl_define -set IDENTIFY_DEBUG_IMPL
```

If you do not perform the steps above, the following message is displayed while saving the instrumentation:

```
Error: The design contains 1 UJTAG instance (UJTAG_inst), which would clash with the UJTAG instance in the debug IP core!
```

To resolve the error:

- Replace UJTAG with UJTAG_WRAPPER.
- Add the file <synplify_dir>/lib/di/ujtag_wrapper.v to your project.
- Type set_option -hdl_define -set IDENTIFY_DEBUG_IMPL at the Tcl prompt if you are going to debug your design using Identify.

- soft – indicates that the Synopsys Tap controller is to be used. The Synopsys FPGA Tap controller is more costly in terms of resources

because it is implemented in user logic and requires four user I/O pins to connect to the communication cable.

- umrbus – indicates that the IICE is connected to the target device through the UMRBus.

# Debugger

Use the Debugger menu to run the debugger, set the trigger position, configure state machine, define debugger preferences, etc.

| Feature | Description |
|---|---|
| Run | Click to debug the active design according to the watchpoints and breakpoints set. See Run Panel and Run Button, on page 159. |
| Waveform viewer | Opens the default selected Waveform viewer. See Waveform Viewer, on page 130. |
| Find in debugger source | Finds target expression in debugger source. Can set additional parameters to filter results. See Find in debugger source, on page 130. |
| Find next in debugger source | Finds the next instance of target expression set in Find in debugger source. See Find in debugger source, on page 130. |
| Trigger Position | Sets the position of trigger in sample buffer. See Trigger Position, on page 131. |
| Configure state machine | Opens the State Machine editor to configure the state machine. See Configure State Machine, on page 132. |
| IICE | Brings up a submenu to select the IICE configuration.<br>• Configure IICE Settings - Brings up the Enhanced IICE Settings dialog box to enable data compression, multiplexed group, and to run a DTD self test.<br>• RTD Report - Displays the signal/breakpoint interface assignments to the Mictor connector.<br>• |
| Client/Server | Brings up the Configure Client/Server dialog box to set up the connection between the debugger and the instrumented device through the cable. See the *Debugger User Guide*. |

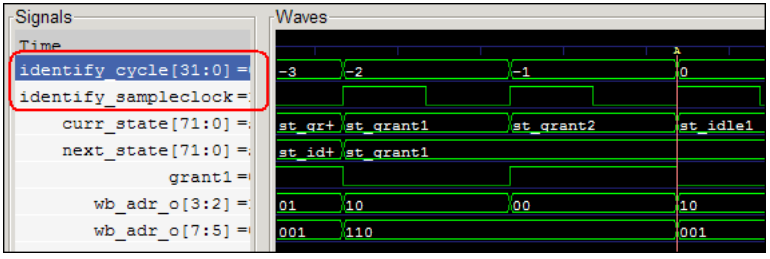

© 2020 Synopsys, Inc.

| Feature | Description |
|---|---|
| Verdi Debug | |
| Setup debugger | Sets the general debugger and waveform display settings. See Setup debugger, on page 133. |
| Debugger information | Displays the debugger details. See Debugger information, on page 146. |

## Waveform Viewer

All sampled signals in the design are included in the waveform display. Invoke the waveform viewer after running the debugger, by selecting Debugger > Waveform viewer.

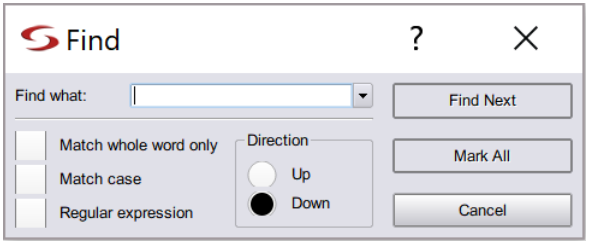
The following figure shows a typical waveform view with the identify_cycle and identify_sampleclock signals enabled (highlighted in the figure).




Two additional signals are added to the top of the display when enabled by their corresponding check boxes. The first signal, identify_cycle, reflects the trigger location in the sample buffer. The second signal, identify_sampleclock, is a reference that shows every clock edge. For information about setting up the waveform, see the *Debugger User Guide*.

## Find in debugger source

Select Debugger > Find in debugger source to search and find the required information in the text editor.




| Field/Option | Description |
|---|---|
| Find What/Search for | Search string matching the text to find. In the text editor, you can use the pull-down list to view and reuse search strings used previously in the current session. |
| Match whole word only | When enabled, matches the entire word rather than a portion of the word. |
| Match Case | When enabled, searching is case sensitive. |
| Regular expression | When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including an empty string. |
| Direction | Changes search direction to Up or Down. |
| Find Next | Initiates a search for the search string (see Find What/Search for). In the text editor, searching starts again after reaching the end (Down) or beginning (Up) of the file. |
| Mark All | Highlights the line numbers of the text matching the search string and closes the Find dialog box. |

## Trigger Position

The trigger position can be changed without requiring the design to be re-instrumented or recompiled. A new trigger position setting takes effect the next time the Run command is executed.


© 2020 Synopsys, Inc.

### Early Position



The sample buffer trigger position can be set to early so that the majority of the samples occurs after the trigger event. To set the trigger position to early, select Debugger > Trigger Position > Set trigger early in sample buffer from the menu or click on the Set trigger position early in the sample buffer icon.

### Middle Position



The sample buffer trigger position defaults to middle so that there is an equal number of samples before and after the trigger event. To set the trigger position to middle, select Debugger > Trigger Position > Set trigger to middle of sample buffer from the menu or click on the Set trigger position to middle of the sample buffer icon.

### Late Position



The sample buffer trigger position can be set to late so that the majority of the samples occurs before the trigger event. To set the trigger position to late, select Debugger > Trigger Position > Set trigger late in sample buffer from the menu or click on the Set trigger late in sample buffer icon.

## Configure State Machine

During instrumentation, the number of states was previously defined using the -triggerstates option of the instrumentor iice controller command. Now, at debug time, you can define what happens in each state and transition depending on the pattern matches computed by the trigger conditions.

The debugger statemachine command is used to configure the trigger state machine with the desired behavior. This is very similar to the “advanced” trigger mode offered by many logic analyzers. As it is very easy to introduce errors in the process of specifying the state machine, special caution is appropriate.

Also, a state-machine editor is available in the debugger user interface to facilitate state-machine development and understanding. It is also important to note that the initial state for each run is always state 0 and that not all of the available states need to be defined. See the *Debugger User Guide*.

## Configure Confpro

Sets the path to the Confpro installation directory. See the *Confpro User Guide*, for more information.

## Setup debugger

All parts of the debugging system must be configured correctly to make a successful connection between the debugger and the instrumented device through the cable.

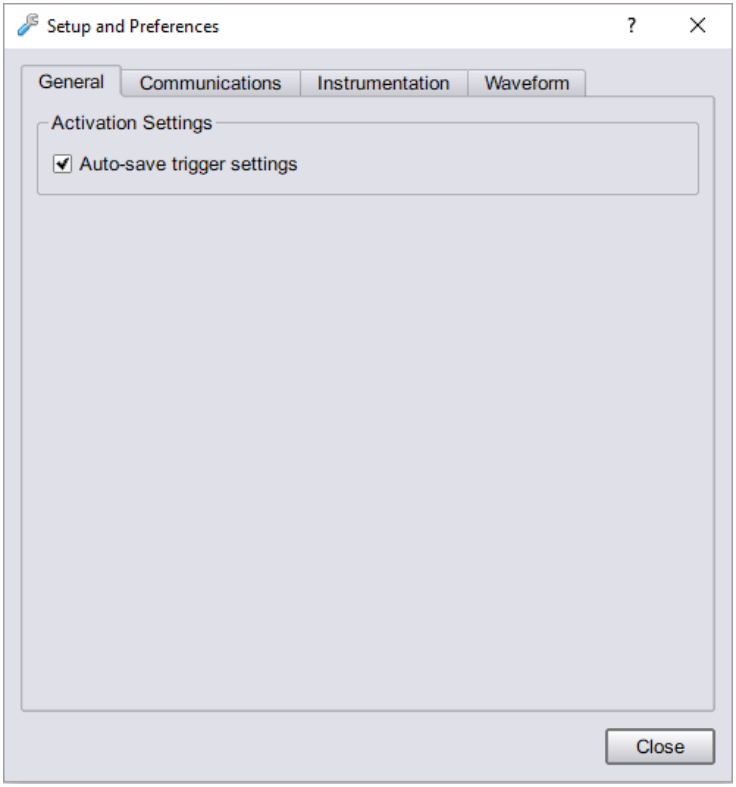
This feature is used to define the general debugger and waveform display settings. The Setup and Preferences window is displayed. See the *Debugger User Guide*.

**Note:** Make sure that the Device Family setting is correct; an incorrect setting requires the entire instrumentation flow to be repeated before the design can be debugged.

### General

By default, when you exit the debugger tool without explicitly saving an activation, the active activation is automatically saved to the last_run.adc file if the Auto-save trigger settings check box is selected. This file is automatically loaded the next time when the same project is loaded. By default, this checkbox is selected. See the *Debugger User Guide*.


© 2020 Synopsys, Inc.

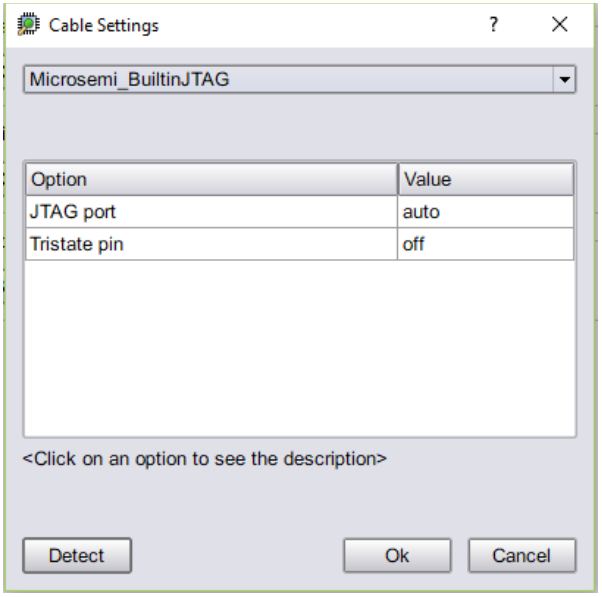

Setup and Preferences
?
General
Communications
Instrumentation
Waveform
Activation Settings
Auto-save trigger settings
Close

## Communications

In the Communications tab, set the cable type, client server, and communication chains.

### Cable Settings

The cable type and port specification communication settings can be set or changed. Click the Cable Settings button to open the Cable Settings dialog box.




There is a list of possible vendor cable-type settings available from the drop-down menu.

1. Set the Cable type value according to the type of cable you are using to connect to the programmable device (see the *Debugger User Guide*).
2. Adjust the port setting based on the port where the communication cable is connected. Most often, lpt1 is the correct setting for parallel ports.

The following table lists the correspondence between cable-type setting and the available vendor cables supported by the Identify debugger.

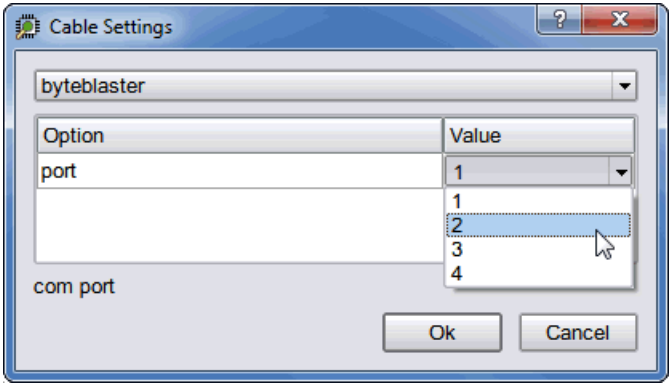

© 2020 Synopsys, Inc.

| Cable Type Settings | Compatible Hardware Cables |
|---|---|
| **Microchip_BuiltinJTAG** | Microchip FlashPro, FlashProLite, or FlashPro3 |
| **JTAGTech3710** | JTAGTech3710 |
| **Catapult_EJ1** | Standard Ethernet cable in an IP network |

If you are using the command interface, set the com command's cabletype option to Microchip_BuiltinJTAG, JTAGTech3710, Catapult_EJ1, or demo according to the cable being used.

Byteblaster Cable Setting

To configure a ByteBlaster cable, click the Cable Settings button to display the Cable Settings dialog box and select the appropriate port from the Value drop-down menu (see following figure).



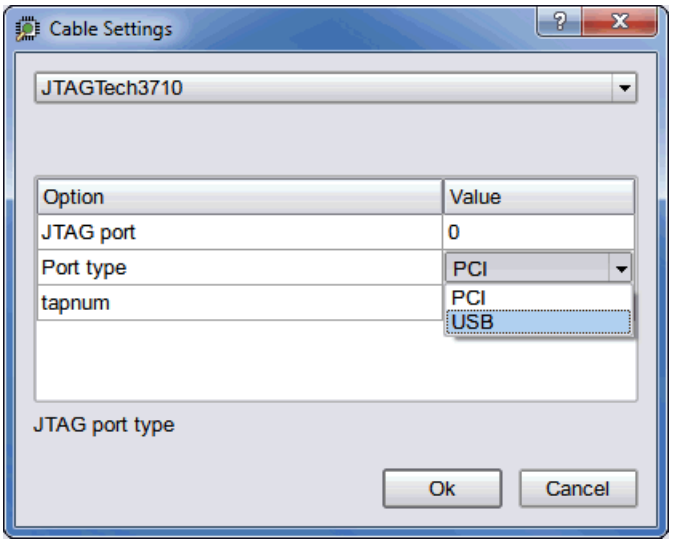

If you are using the command interface, set the com command's cableoptions byteblaster_port option to 1 (lpt1), 2 (lpt2), 3 (lpt3), or 4 (lpt4). Different computers have their lpt ports defined for different address ranges so the port you use depends on how your computer is configured.

The Identify debugger uses the "standard" I/O port definitions: lpt1: 0x378-0x37B, lpt2: 0x278-0x27B, lpt3: 0x3BC-0x3BF, and lpt4: 0x288-0x28B if it cannot determine the proper definitions from the operating system. If the hardware address for your parallel port does not match the addresses for lpt1 through lpt4, you can use the setsys set command variable lpt_address to set the hardware port address (for example, setsys set lpt_address 0x0378 defines port lpt1).

© 2020 Synopsys, Inc.

JTAGTech3710 Cable Settings

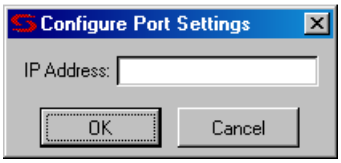
To configure a JTAGTech3710 cable, click the Port Settings button to display the Configure Port Settings dialog box (see following figure) and enter the corresponding parameters (type, port, and tap number). If you are using the command interface, use the com command's cableoptions option to set the cable-specific parameters – JTAGTech_type (takes values PCI and USB; default is PCI), JTAGTech_port (takes values 0, 1, 2, ...; default value is 0), and JTAGTech_tapnum (takes values 1, 2, 3, or 4; default is 1).




Microchip Actel_BuiltinJTAG cable Settings

To configure a Microchip FlashPro, FlashProLite, or FlashPro3 cable, simply select the Microchip_BuiltinJTAG setting from the Cable type drop-down menu. If you are using the command interface, you can additionally use the com command's cableoptions option to set the tristate pin parameter (see the com command cableoptions option in the *Command Description*, on page 19 for the parameter syntax).


© 2020 Synopsys, Inc.

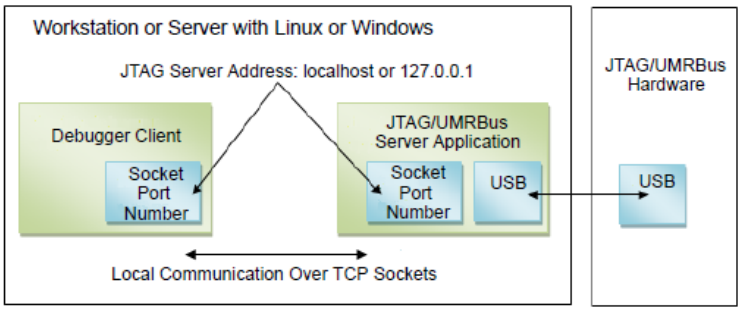



Catapult EJ-1 Settings

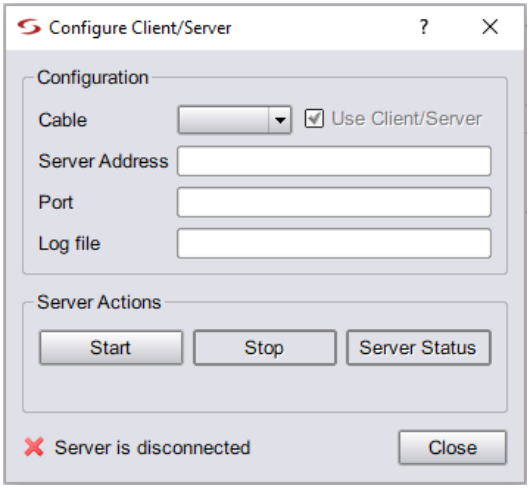
To configure a Catapult EJ-1 cable, select the Catapult_EJ1 setting from the Cable type drop-down menu. Click the Port Settings button to display the Configure Port Settings dialog box and enter the host IP address.

## Configure Client Server

Set up the connection between the debugger and the instrumented device through the cable by clicking the Configure Client Server button.




The default settings are usually correct for most configurations and require changing only when the default server port address is already in use or when the debugger is being run from a remote machine that is not the same machine connected to the FPGA board/device. See the *Debugger User Guide*.




The available configure client-server settings in the dialog box are defined in the following table:

| Setting | Function |
|---|---|
| Cable | The type of interface cable (see *Cable Settings*, on page 135). |
| Use Client/Server | Check box for enabling client-server communications when the cable type is USB-based UMRBus (limited to HAPS-70 systems). |
| Server Address | The address of the server. The address localhost (or 127.0.0.1) is used when the debugger is run on the same machine connected to the FPGA device. The server address is set to the name or tcp/ip4 address of the machine connected to the FPGA device/board when the debugger is run from a different machine. |
| Port | The port number of the server. For all Microchip cable types, the default port number is 58015. Change the server port setting when there is a conflict with another tool on the machine. |
| Log file | The name of the log file. |

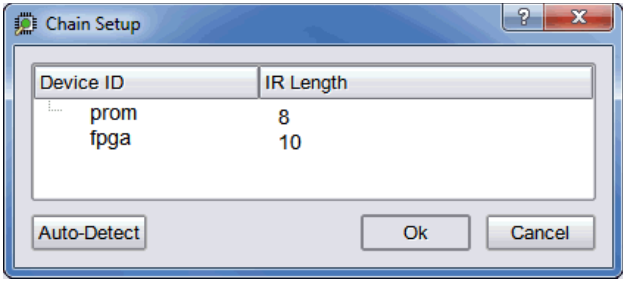

© 2020 Synopsys, Inc.

| Setting | Function |
|---|---|
| Start | Server control button for starting the server in stand-alone mode. Activating the button adds a start entry to the log file. |
| Stop | Server control button for stopping the server in stand-alone mode. Activating the button adds a stop entry to the log file. |
| Server Status | Adds a start/stop entry to the log file. |

Check the Cable type setting in the main page of the debugger after loading the project.

### Show Chain

Click the Show Chain button to view the JTAG chain settings. Normally, the JTAG chain settings for the devices are automatically extracted from the devices available in the JTAG chain. When the chain settings cannot be determined, they must be created and/or edited using the chain command in the console window. See the *Debugger User Guide*. The settings shown below are for a 2-device chain that has JTAG identification register lengths of 8 and 10 bits. In addition, the device named fpga has been enabled for debugging. By clicking Auto-Detect, the tool automatically identifies the programmable devices that are daisy chained on the hardware.

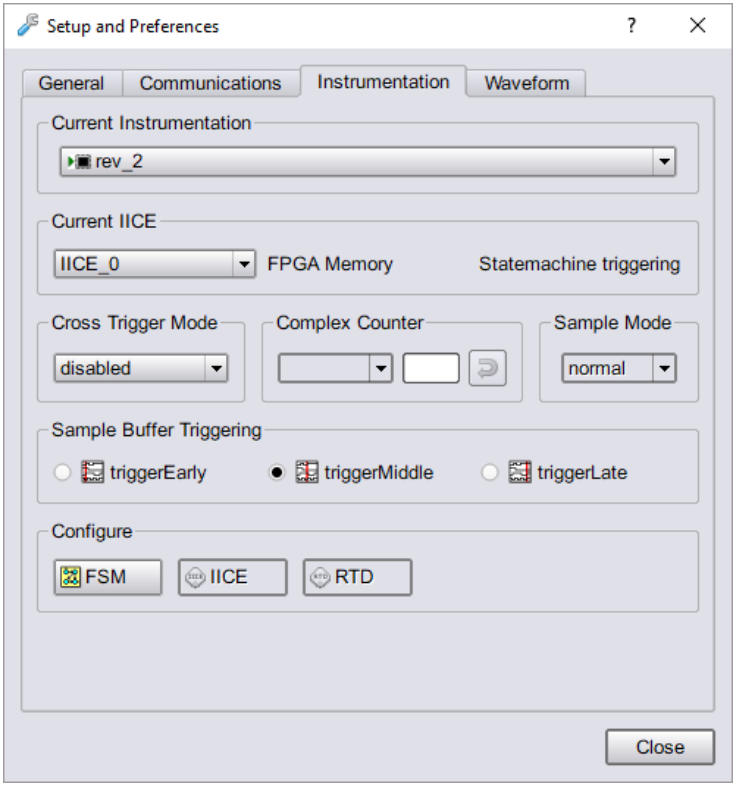



### Comm Check

Runs a hardware check to verify board and cable integrity. The results of the check are displayed in the Tcl Script window.

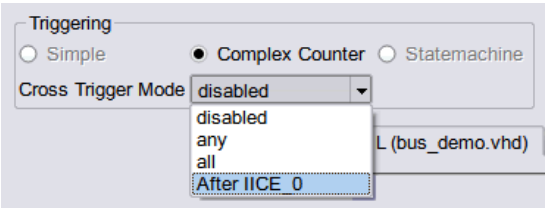
## Instrumentation

As per the IICE settings in the Instrumentation tool, you can perform the instrumentation in the Debugger tool.




1. Select the required implementation to instrument from the Current Instrumentation drop-down list in case of multiple implementations.
2. Select the required IICE from the Current IICE drop-down list. Based on the selected IICE settings in the Instrumentation tool, you can perform the triggering here.
3. Select the Allow cross-triggering in IICE check box in the IICE Options tab for the local IICE unit to enable cross triggering for an IICE unit. The cross-trigger mode is selected from the drop-down menu in the debugger as shown below.


© 2020 Synopsys, Inc.

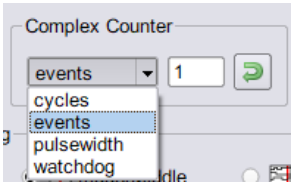



The drop-down menu selections are as follows:

| Menu Selection | Function |
|---|---|
| disabled | No triggers accepted from external IICE units (event trigger can only originate from local IICE unit). |
| any | Event trigger from local IICE unit occurs when an event at any IICE unit, including the local IICE unit, occurs. |
| all | Event trigger from local IICE unit occurs when all events, irrespective of order, occur at all IICE units including the local IICE unit. |
| *After IICE* | Event trigger from local IICE unit occurs only after the event at selected external IICE unit has occurred (external IICE units are individually listed). |

These drop-down menu options are available only when the Allow cross-triggering in IICE option is enabled on the IICE Options tab of the instrumentor and the design have multiple IICE units defined.

4. The counter sends a trigger event to the sample block when a termination condition occurs. The form of the termination condition depends on the mode of operation of the counter and on the target value of the counter:
   - The counter target value can be set to any value in the counter's range.
   - The counter has four modes: events, cycles, watchdog, and pulsewidth.
   - The counter target value and the counter mode can be set directly from the debugger GUI.




The following table provides a general description of the trigger behavior for the various complex counter modes. Each mode is described in more detail in individual subsections, and examples are included showing how the modes are used. In both the table and subsection descriptions, the counter target value setting is represented by the symbol *n*.

| Counter mode | Target value = 0 | Target value *n* > 0 |
|---|---|---|
| events | Illegal | Stop sampling on the *n*th trigger event. See events Mode, on page 143. |
| cycles | Stop sampling on 1st trigger event | Stop sampling *n* cycles after the first trigger event. See cycles Mode, on page 143. |
| watchdog | Illegal | Stop sampling if the trigger condition is not met for *n* consecutive cycles. See watchdog Mode, on page 144. |
| pulsewidth | Illegal | Stop sampling the first time the trigger condition is met for *n* consecutive cycles. See pulsewidth Mode, on page 144. |

### events Mode

In the events mode, the number of times the Master Trigger Signal logic produces an event is counted. When the *n*th Master Trigger Signal event occurs, the complex counter sends a trigger event to the sample block. For example, this mode could be used to trigger on the 12278th time a collision was detected in a bus arbiter.

### cycles Mode

In the cycles mode, the complex counter sends a trigger event to the sample block on the *n*th cycle after the first Master Trigger Signal event is received. The clock cycles counted are from the clock defined for sampling. For example, this mode could be used to observe the behavior of a design 2,000,000 cycles after it is reset.


© 2020 Synopsys, Inc.

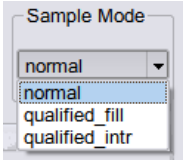
### watchdog Mode

In the watchdog mode, the counter sends a trigger event to the sample block if no Master Trigger Signal events have been received for *n* cycles. For example, if an event is expected to occur regularly, such as a memory refresh cycle, this mode triggers when the expected event fails to occur.

### pulsewidth Mode

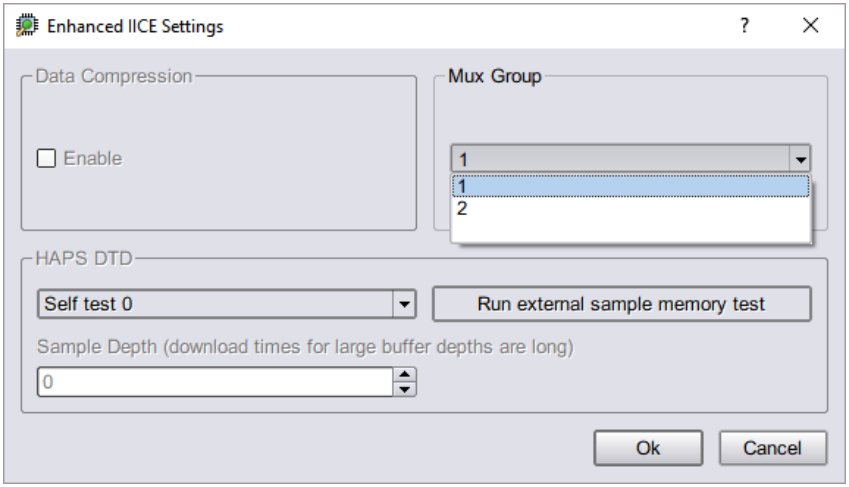
In the pulsewidth mode, the complex counter sends a trigger event to the sample block if the Master Trigger Signal logic has produced an event during each of the most recent *n* consecutive cycles. For example, this mode can be used to detect when a request signal is held high for more than *n* cycles thereby detecting when the request has not been serviced within a specified interval.

5. Select qualified_fill or qualified_intr from the Sample Mode drop-down list. For more information, see *-qualified_sampling* under the *iice*, on page 48 command description.




6. FSM- Clicking the FSM button displays the Configure State Machine dialog box for the selected IICE. For information on state machine, see the *Debugger User Guide*.

7. IICE - Clicking the IICE displays Enhanced IICE Settings dialog box.



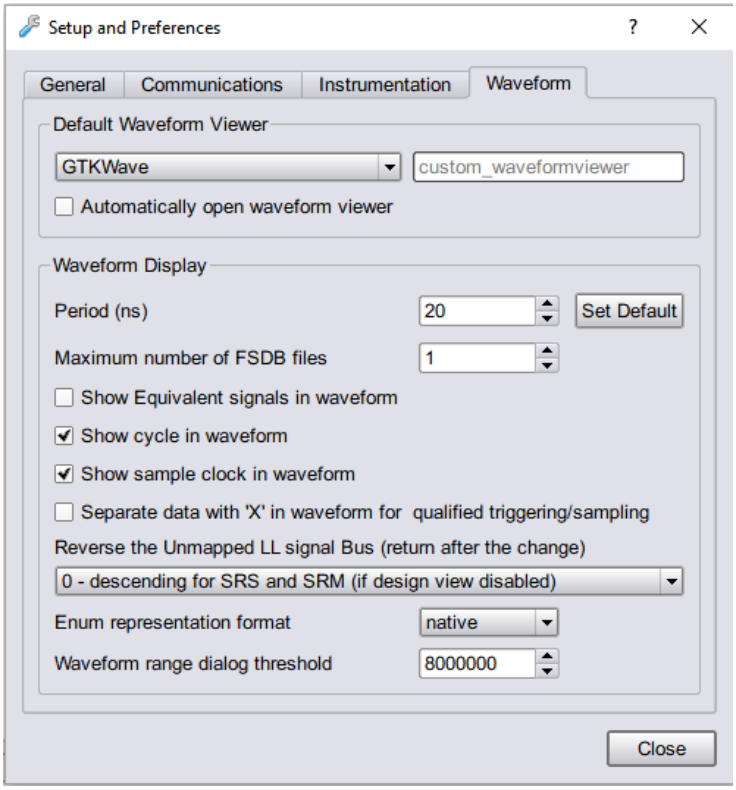

– From the Mux Group drop-down menu select the group number to be active for the debug session.

– The signals group command can be used to assign groups from the console window (see *signals* in the *Reference Manual*).

## Waveform

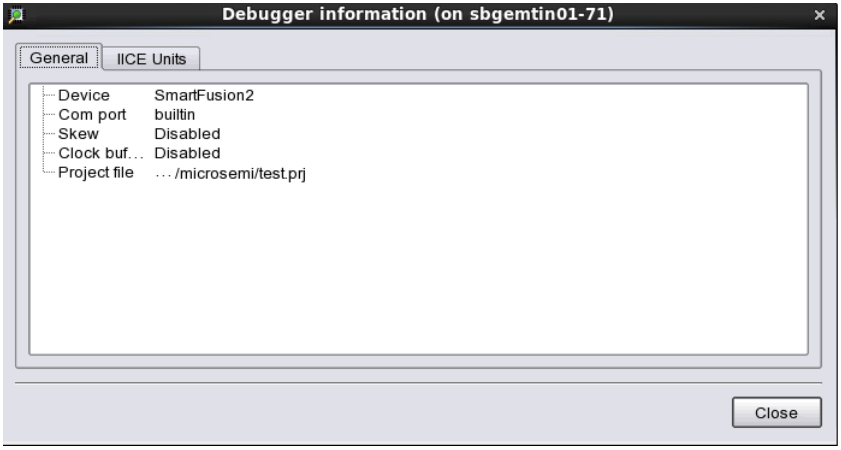
Define the required settings for waveform. For setting information, see the *Debugger User Guide*.


© 2020 Synopsys, Inc.




## Debugger information

The instrumentation settings are displayed in the Debugger Information window. Because these configuration settings are inherited from the instrumentor and used to construct the IICE, you cannot change these settings in the debugger. To view the debugger information, select Debugger > Debugger Information. The Debugger Information window pops up.





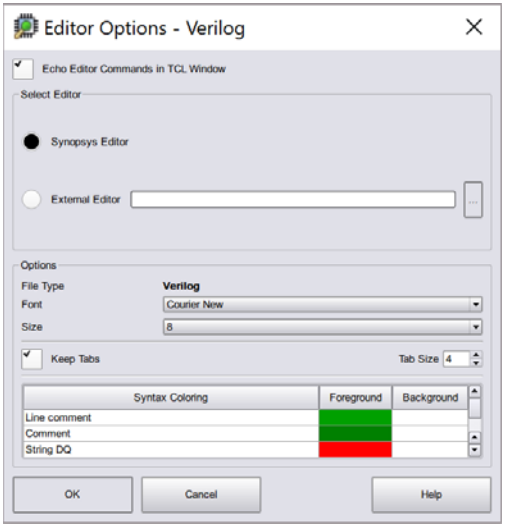
© 2020 Synopsys, Inc.

# Options Menu

The options menu sets the editor options for the tool. Sets your Text Editor syntax coloring, font, and tabs.

## Editor Options Command

Select Options > Editor Options to define the internal text editor or an external text editor.



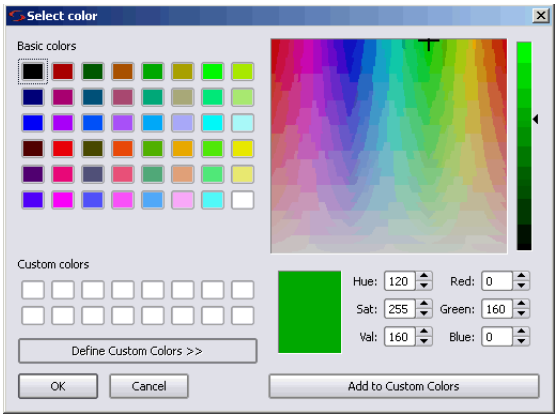

The following table describes the Editor Options dialog box features.

| Feature | Description |
|---|---|
| Synopsys Editor | Sets the Synopsys text editor as the default text editor. |
| External Editor | Uses the specified external text editor program to view text files from within the Synopsys tool. The executable specified must open its own window for text editing. See the *Debugger User Guide* for a procedure.<br>*Note:* Files opened with an external editor cannot be crossprobed. |
| File Type | You can define text editor preferences for the following file types: project files, HDL files, log files, constraint files, and default files. |

| Feature | Description |
|---|---|
| Font | Lets you define fonts to use with the text editor. |
| Font Size | Lets you define font size to use with the text editor. |
| Keep Tabs<br>Tab Size | Lets you choose whether to use tab in the text editor and specify the tab size. |
| Syntax Coloring | Lets you define foreground or background syntax coloring to use with the text editor. See Color Options, on page 149. |

## Color Options

Click in the Foreground or Background field for the corresponding object in the Syntax Coloring field to display the color palette.




You can set syntax colors for some common syntax options listed in the following table.

| Syntax | Description |
|---|---|
| Line Comment | Line comments contained in the HDL source, C, C++, and log files. |
| Comment | Comment strings contained in all file types. |
| String DQ | String values within double quotes contained in the project, HDL source, constraint, C, C++, and log files. |


© 2020 Synopsys, Inc.

# Window

The window menu controls window alignment and navigation options. It also displays a list of all open windows and allows you to activate a window.

| Option | Description |
|---|---|
| Tile Horizontally | Positions the open windows horizontally. |
| Tile Vertically | Positions the open windows vertically. |
| Cascade | Allows you to cascade the views with multiple windows. |
| Customize for | Customize for debugging<br>Customize for board bring-up |
| Close All | Closes all open windows in the tool. |
| Next / Previous | Navigate through open windows. |

# Help Menu

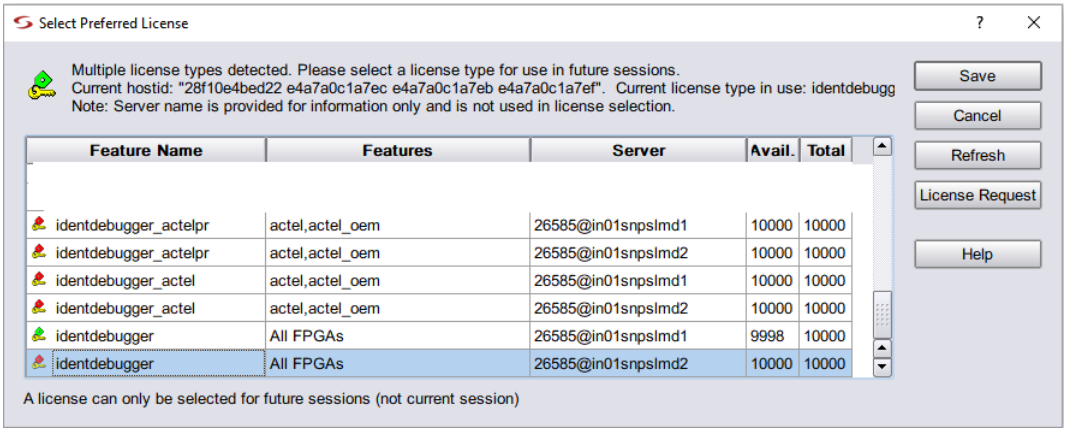
The Help menu provides a level of user help from within the graphical window. The following table describes the individual Help menu selections and their functions.

| Command | Description |
|---|---|
| Help Topics | Displays hyperlinked online help for the product. |
| Additional Products | Displays information about any additional products (if available). |
| How to Use Help | Displays help on how to use Synopsys FPGA online help. |
| PDF Documents | Displays an Open dialog box with hyperlinked PDF documentation for the product including user guide and reference manuals. You need Adobe Acrobat Reader® to view the PDF files. |
| Error Messages | Displays help on the message viewer. |
| TCL | Displays help for Tcl commands. |
| License Agreement | Displays the Synopsys software license agreement. |
| Preferred License Selection | Displays the floating licenses that are available for your selection. See Preferred License Selection Command, on page 151. |
| About this program | Displays the About dialog box, showing the tool product name, license expiration date, customer identification number, version number, and copyright.<br>Clicking the Versions button in the About dialog box displays the Version Information dialog box, listing the installation directory and the versions of all the compiler and mapper programs for the tool. |

## Preferred License Selection Command

Select Help > Preferred License to display the Select Preferred License dialog box, listing the available licenses for you to choose from. Select a license from the License Type column and click Save. Close and restart the Synopsys tool, then the new session uses the preferred license you selected.


© 2020 Synopsys, Inc.


Select Preferred License
Multiple license types detected. Please select a license type for use in future sessions.
Current hostid: "28f10e4bed22 e4a7a0c1a7ec e4a7a0c1a7eb e4a7a0c1a7ef". Current license type in use: identdebugg
Note: Server name is provided for information only and is not used in license selection.
Save
Cancel
Refresh
License Request
Help
Feature Name
Features
Server
Avail.
Total
identdebugger_actelpr actel,actel_oem 26585@in01snpslmd1 10000 10000
identdebugger_actelpr actel,actel_oem 26585@in01snpslmd2 10000 10000
identdebugger_actel actel,actel_oem 26585@in01snpslmd1 10000 10000
identdebugger_actel actel,actel_oem 26585@in01snpslmd2 10000 10000
identdebugger All FPGAs 26585@in01snpslmd1 9998 10000
identdebugger All FPGAs 26585@in01snpslmd2 10000 10000
A license can only be selected for future sessions (not current session)

CHAPTER 4

# User Interface Overview

This chapter describes the graphical user interface (GUI) of instrumentor and debugger tools.


- Instrumentor GUI, on page 154
- Debugger GUI, on page 158



© 2020 Synopsys, Inc.

# Instrumentor GUI

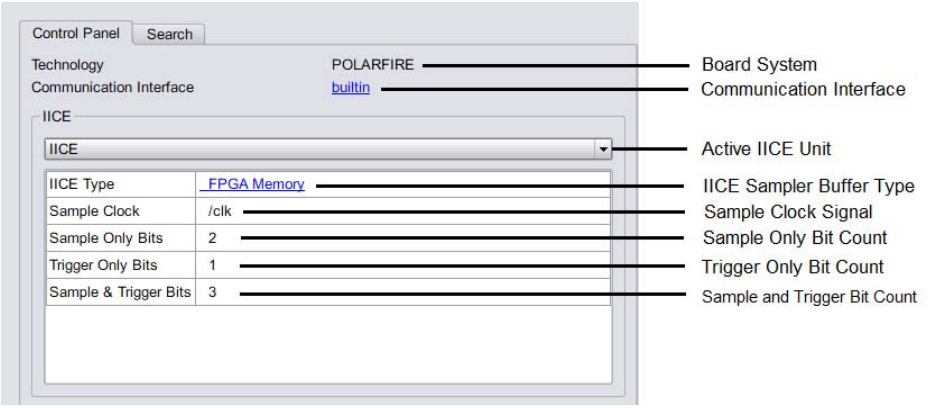
The GUI for the instrumentor includes all of the required user controls, a hierarchy browser in the center, and the following panels for instrumentor-specific functions:

- Control Panel, on page 154
- Search Panel, on page 155
- Hierarchy Browser, on page 156
- View Panel, on page 157
- TCL Window, on page 168

You can toggle between the control/search, browser, and view panels from Instrumentor > Windows. See *Instrumentor*, on page 113.

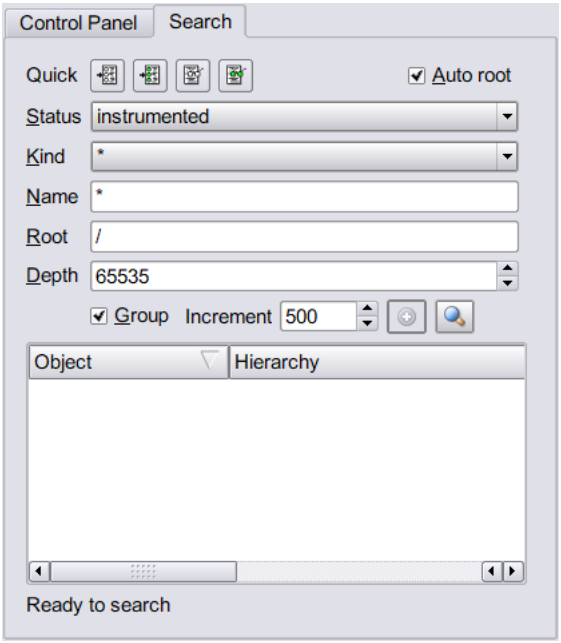
## Control Panel

The Control Panel tab describes the current status of the instrumented design. Note that some entries are dependent on the IICE sampler buffer type selected.



# Search Panel

The Search panel is a general utility to search for signals, breakpoints, and/or instances. The panel includes an area for specifying the objects to find and an area for displaying the results of the search. The instrumentor tool includes a set of menu commands and, in most cases, icons for listing signals and breakpoint conditions. For detailed information on using the Search panel, see the *Debugger User Guide*.




## List Breakpoints Available for Instrumentation

To list all of the breakpoints that are available for instrumentation, click the Search non-instrumented breakpoints icon.

## List Instrumented Breakpoints

To list all of the breakpoints that have been instrumented, click the Search instrumented breakpoints icon.


© 2020 Synopsys, Inc.

### List Signals Available for Instrumentation



To see only the signals in the design available for instrumentation, click the Search non-instrumented signals icon.

### List Instrumented Signals

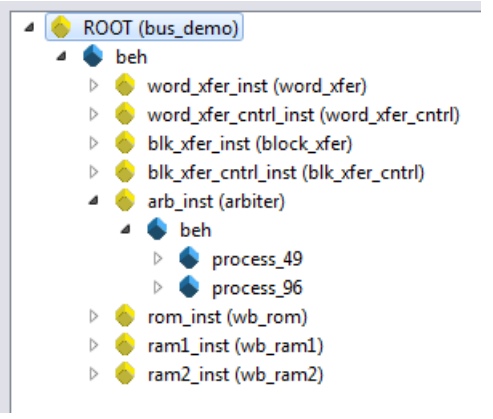


To view all of the signals currently instrumented in the entire design, click the Search for instrumented signals icon in the Quick Search area in the Search panel.

## Hierarchy Browser

The hierarchy browser shows a graphical representation of the design hierarchy. At the top of the browser is the ROOT node. The ROOT node represents the top-level entity or module of your design. For VHDL designs, the first level below the ROOT is the architecture of the top-level entity. The level below the top-level architecture for VHDL designs, or below the ROOT for Verilog designs, shows the entities or modules instantiated at the top level.

Double-clicking on an entry opens the entity/module instance so that the hierarchy below that instance can be viewed. Lower levels of the browser represent instantiations, case statements, if statements, functional operators, and other statements.




Clicking on any element in the hierarchy browser causes the associated HDL code to be visible in the RTL tab display.

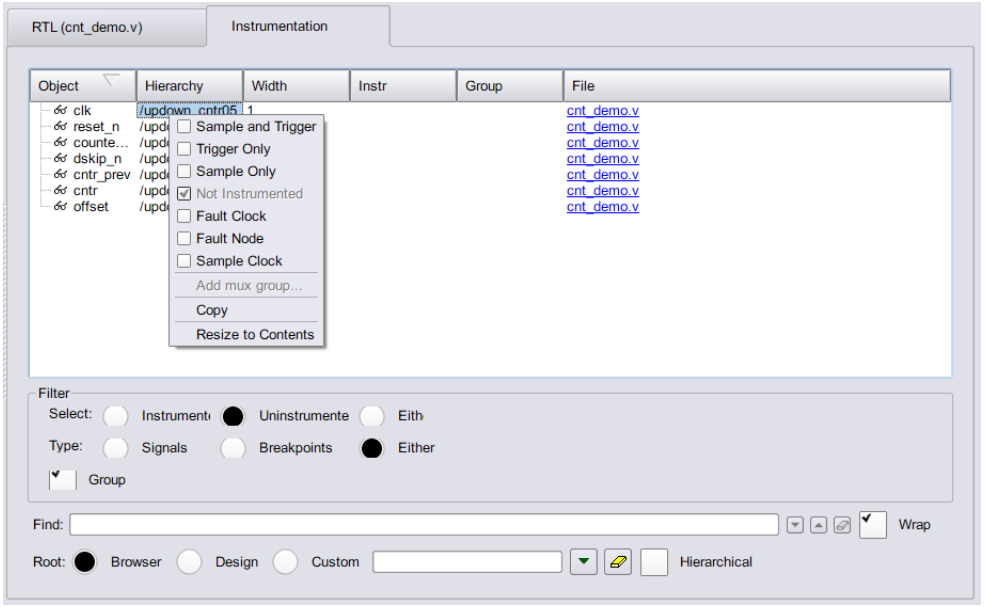
# View Panel

The following list of panels are grouped as view panels:

- Instrumentation Tab, on page 157

## Instrumentation Tab

The Instrumentation tab lists the active watchpoint and breakpoint entries that have been set within the active module or entity. The entries can be modified by selecting the entry and assigning a new value from the popup menu.



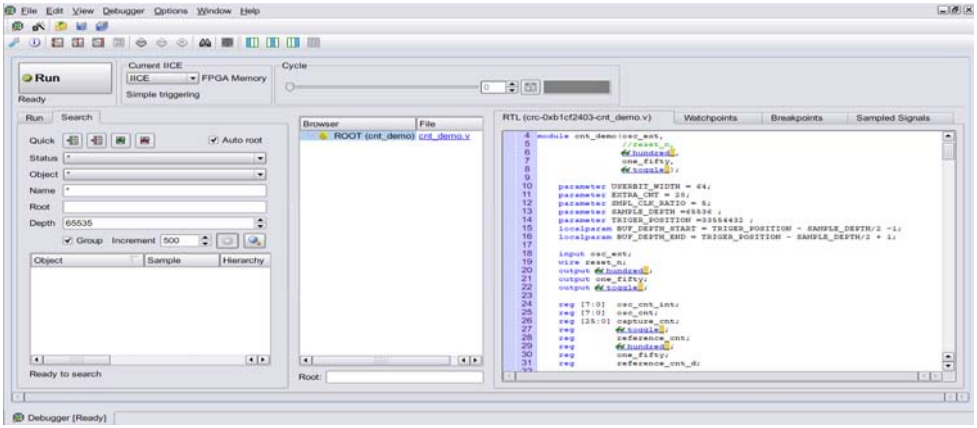


© 2020 Synopsys, Inc.

# Debugger GUI

The GUI for the debugger includes all of the required user controls, a hierarchy browser in the center, and the following panels for debugger-specific functions:

- Run Panel and Run Button, on page 159
- Search Panel, on page 160
- Hierarchy Browser, on page 162
- View Panel, on page 162
- TCL Window, on page 168




In this section, each of these areas and their uses are described. The following discussions assume that:

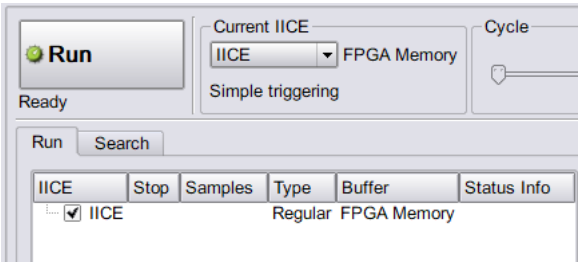
- An HDL design has been loaded into the instrumentor and instrumented
- The design has been synthesized in the synthesis tool
- The synthesized output netlist has been placed and routed by the place and route tool
- The resultant bit file has been used to program the FPGA with the instrumented design
- The board containing the programmed FPGA is cabled to the host for analysis by the debugger

© 2020 Synopsys, Inc.

You can toggle between the control/search, browser, and view panels. See *Debugger Panels*, on page 112.

## Run Panel and Run Button

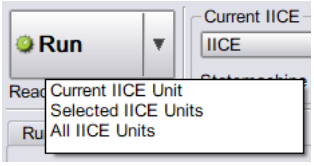
The Run panel reports the results of a run according to the watchpoints and breakpoints set.




The Run panel includes six columns:

- IICE – Displays the IICE unit or units selected for the run. Clicking an adjacent checkbox enables the IICE unit.
- Stop – Displays a Stop sign icon while the run is in progress; clicking the icon prematurely stops the run.
- Samples – Displays the number of samples collected when the run is completed.
- Type – Displays type of the IICE.
- Buffer – Displays the memory selected in the Instrumentor.
- Status Info – Indicates the current IICE status.

A run is initiated by either clicking the Run button directly above the Run panel or by selecting Debugger > Run from the menu bar. The Run button drop-down menu determines the unit or units selected for the run. In case of multiple IICEs, you can run a single or multiple IICEs.





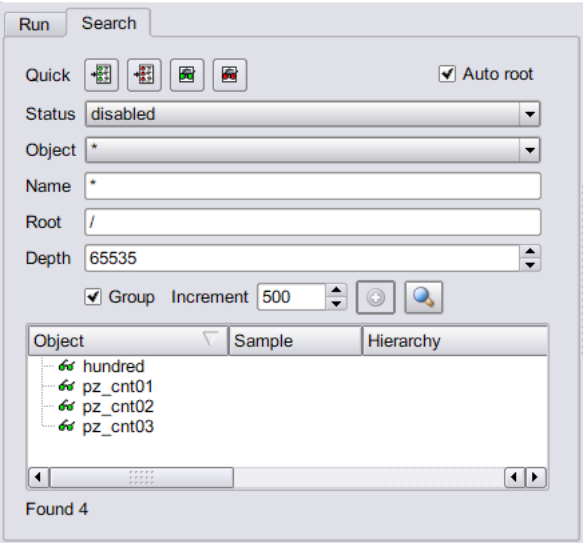
© 2020 Synopsys, Inc.

When initiated, the command sends watchpoint and breakpoint activations to the IICE, waits for the trigger event to occur, receives the captured sampled data back from the IICE when the trigger occurs, and then updates the watchpoint/breakpoint data in the RTL panel.

If data compression is to be used on the sample data, see the *Debugger User Guide*. After the Run command is executed, the sample of signal values at the trigger position is annotated to the HDL code in the RTL panel. This data can be displayed in a waveform viewer or written out to a file. See the debugger waveform and write vcd command descriptions, for more information.

## Search Panel

The Search panel is a general utility to search for signals, breakpoints, conditions and/or instances. The panel includes an area for specifying the objects to find and an area for displaying the results of the search.




The search criteria in the upper section of the panel includes the following options:

- Quick – icons that preset the conditions for instrumented or non-instrumented watchpoints or breakpoints.
  - Search for disabled breakpoints


–To display the disabled (inactive) breakpoints, click the Search for disabled breakpoints icon.

– Search for enabled breakpoints


–To display the enabled (active) breakpoints, click the Search for enabled breakpoints icon.

– Search for disabled watchpoints


–To display the disabled (inactive) watchpoints, click the Search for disabled watchpoints icon.

– Search for enabled watchpoints


–To display the enabled (active) watchpoints, click the Search for enabled watchpoints icon.

- Auto root – When checked, fills in the root hierarchy.
- Status – Specifies the status of the object to be found from the drop-down menu. The values can be enabled, disabled, sample_clock, sample_only, or “*” (any). The default is “*” (any). Status will have readback as drop down if the loaded project is of readback type.
- Object – Specifies the type of object to search for from the drop-down menu: breakpoint, instance, signal or “*” (any). The default is “*” (any).
- Name – Specifies a name, or partial name to search for in the design. Wild cards are allowed in the name. The default is “*” (any).
- Root – Specifies the location in the design hierarchy to begin the recursive search. Root (“/”) is the default setting.
- Depth – Specifies the hierarchical depth of the instrumented signals to be searched.
- Increment – Specifies the maximum number of signals to be listed upon successful search. Click the icon  to add more results to the list.
- Group – Indicates to expand search to all groups when checked.

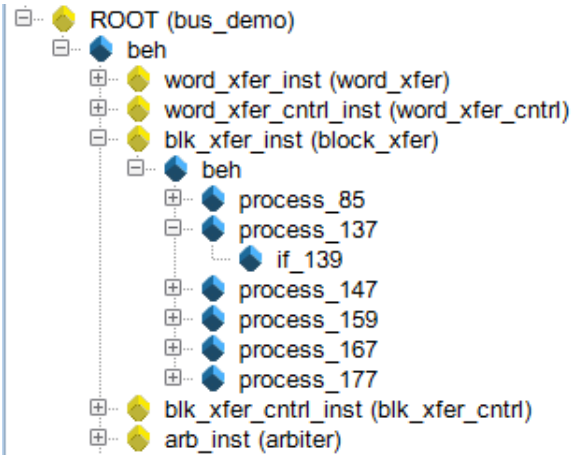
The search results in the lower section of the panel show each object found along with its hierarchical location. In addition, for breakpoints and watchpoints, the results section includes the corresponding icon (watchpoint or breakpoint) that indicates the instrumentation status of the qualified signal or breakpoint.


© 2020 Synopsys, Inc.

## Hierarchy Browser

The hierarchy browser shows a graphical representation of the design hierarchy. At the top of the browser is the ROOT node. The ROOT node represents the top-level entity or module of your design. For VHDL designs, the first-level below the ROOT is the architecture of the top-level entity. The level below the top-level architecture for VHDL designs, or below the ROOT for Verilog designs, shows the entities or modules instantiated at the top-level.

Clicking on a + sign opens the entity/module instance so that the hierarchy below that instance can be viewed. Lower levels of the browser represent instantiations, case statements, if statements, functional operators, and other statements.




Single clicking on any element in the hierarchy browser causes the associated HDL code to be displayed in the RTL panel.

## View Panel

The following list of panels are grouped as view panels:

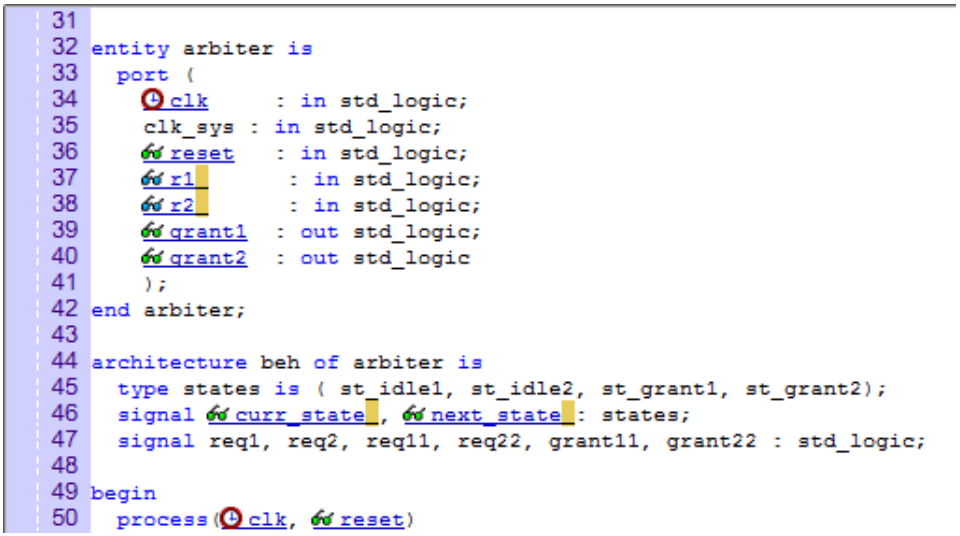
- RTL Panel, on page 163
- Watchpoints Panel, on page 164
- Breakpoints Panel, on page 164
- Sampled Signals Panel, on page 167

## RTL Panel

The RTL panel displays the HDL source code annotated with signals and breakpoints that were previously instrumented.

**Note:** Signals and breakpoints that were not enabled in the instrumentor are not displayed in the debugger.

Signals that can be selected for setting watchpoints are underlined, colored in blue text, and have goggles or icons next to them. Breakpoints that can be activated have small green circular icons in the left margin to the left of the line number.



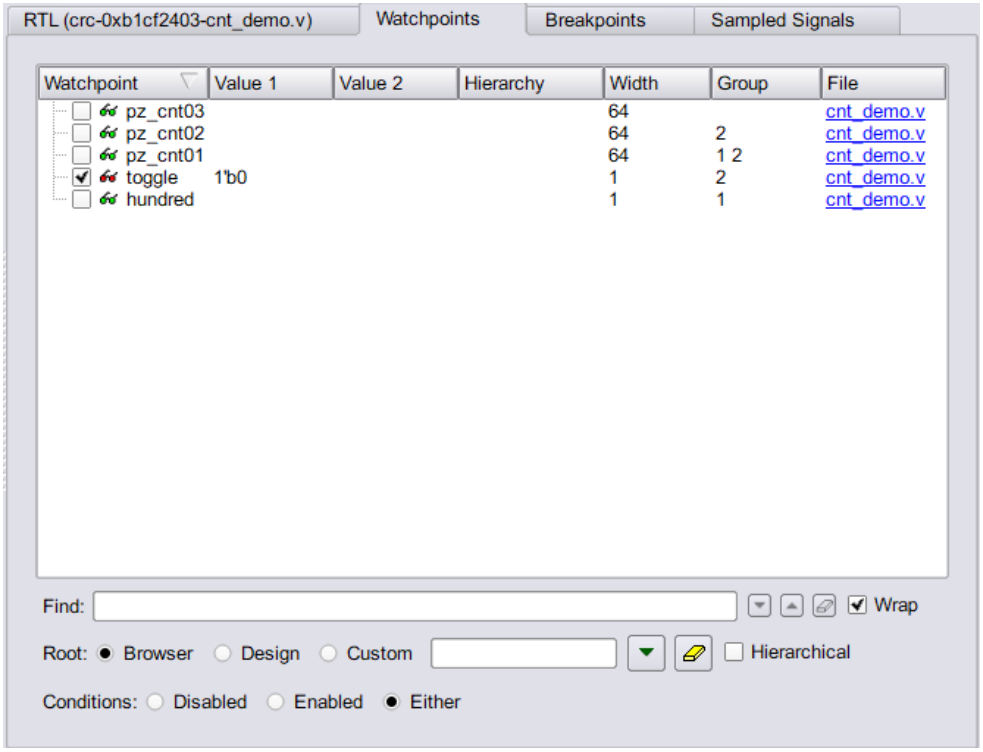

Selecting a watchpoint or icon next to a signal (or the signal itself) allows you to select the Watchpoint Setup dialog box from the popup menu. This dialog box specifies a watchpoint expression for the signal. See *Setting a Watchpoint Expression*, in the *Debugger User Guide*.

Selecting the green breakpoint icon on the left-side of the source line number causes that breakpoint to become armed when the run command is executed. See *Run Command*, in the *Debugger User Guide*.


© 2020 Synopsys, Inc.

## Watchpoints Panel

The watchpoint panel lists all of the active and inactive watchpoints in the design. The icons of active watchpoints are highlighted in red with an adjacent marked check box. Inactive watch points are green regardless of their type. Values for the active watchpoints are listed. You can list all the watchpoints hierarchically based on a specific Root option.

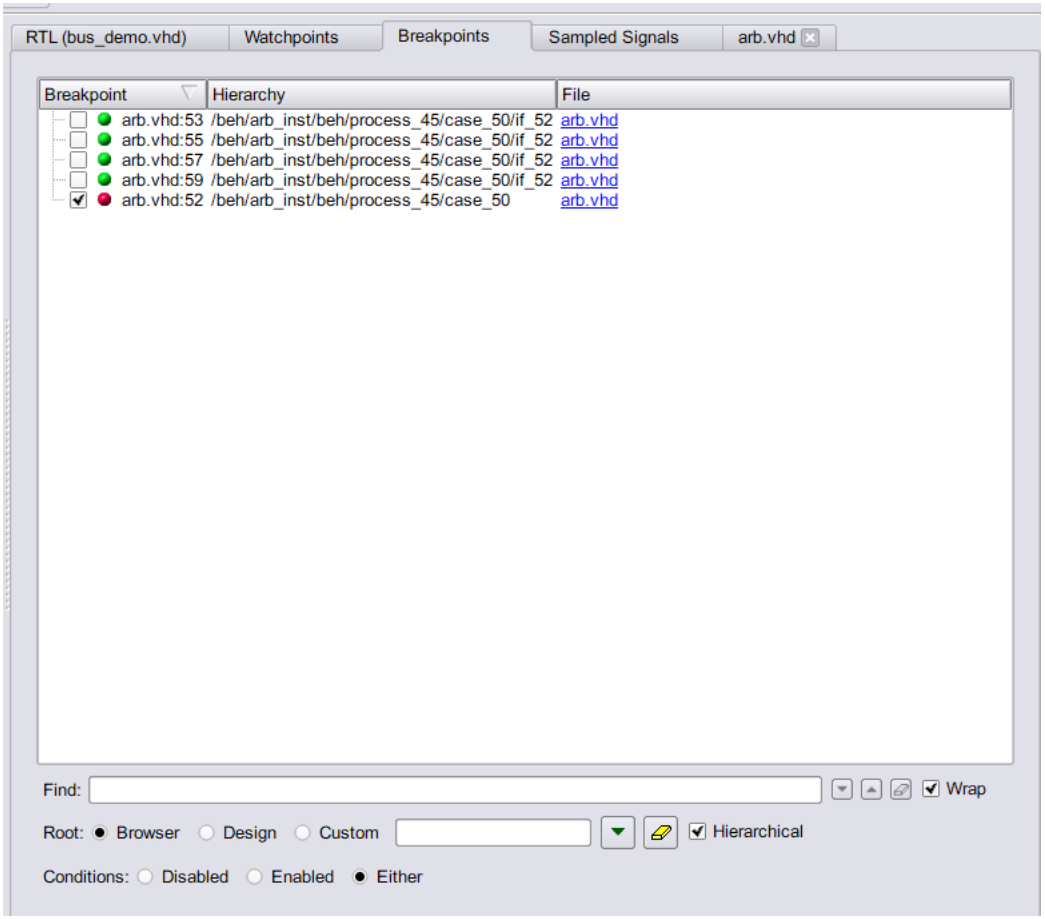



A watchpoint creates a trigger that is determined by the state of a signal in the design. The watchpoint can trigger either on the value of a signal or on a transition of a signal from one value to another.

## Breakpoints Panel

The breakpoints panel lists all of the active and inactive breakpoints in the design. The icons of active breakpoints are highlighted in red with an adjacent marked check box. Inactive watch points are green regardless of their type. Values for the active watchpoints are listed. You can list all the breakpoints hierarchically based on a specific Root option.

© 2020 Synopsys, Inc.

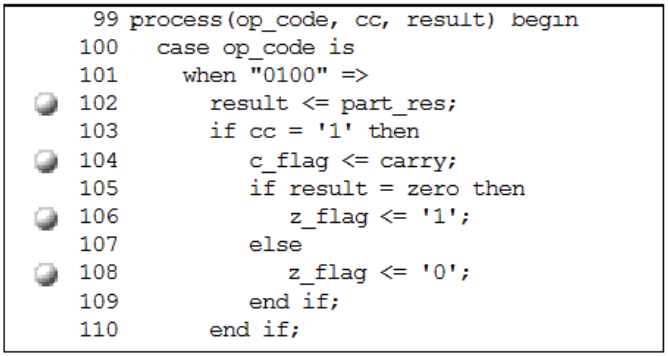



Breakpoints are a way to easily create a trigger that is determined by the flow of control in the design.

In both Verilog and VHDL, the flow of control in a design is primarily determined by if, else, and case statements. The control state of these statements is determined by their controlling HDL conditional expressions. Breakpoints provide a simple way to trigger when the conditional expressions of one or more if, else, or case statements have particular values.

The example below shows a VHDL code fragment and its associated breakpoints.


© 2020 Synopsys, Inc.




The four breakpoints correspond to these control flow equations:

- Breakpoint at line number 102:

  ```
  (op_code = "0100")
  ```

- Breakpoint at line number 104:

  ```
  (op_code = "0100") and (cc = '1')
  ```

- Breakpoint at line number 106:

  ```
  (op_code = "0100") and (cc = '1') and (result = zero)
  ```

- Breakpoint at line number 108:

  ```
  (op_code = "0100") and (cc = '1') and (result != zero)
  ```

## Multiple Activated Breakpoints and Watchpoints

How breakpoints and watchpoints operate individually is described in the *Debugger User Guide*. Activated breakpoints and watchpoints also interact with each other in a very specific way.

## Multiple Activated Breakpoints

Each breakpoint is implemented as logic that watches for a particular event in the design. When an instrumented design has more than one activated breakpoint, the breakpoint events are ORed together. This effectively allows the breakpoints to operate independently – only one activated breakpoint must trigger in order to cause the sampling buffer to acquire its sample.

### *Multiple Activated Watchpoints*

Each watchpoint is implemented as logic that watches for a specific event consisting of a bit pattern or transition on a specific set of signals. When an instrumented design has more than one activated watchpoint, the watchpoint events are ANDed together. This effectively causes the watchpoints to be dependent on each other – all activated watchpoint events must occur concurrently to cause the sampling buffer to acquire its sample.

For example, if watchpoint 1 implements (count == 23) and watchpoint 2 implements (ack == '1'), then activating these watchpoints together effectively creates a new watchpoint: (count == 23) && (ack == '1').

*Combining Activated Breakpoints and Activated Watchpoints*

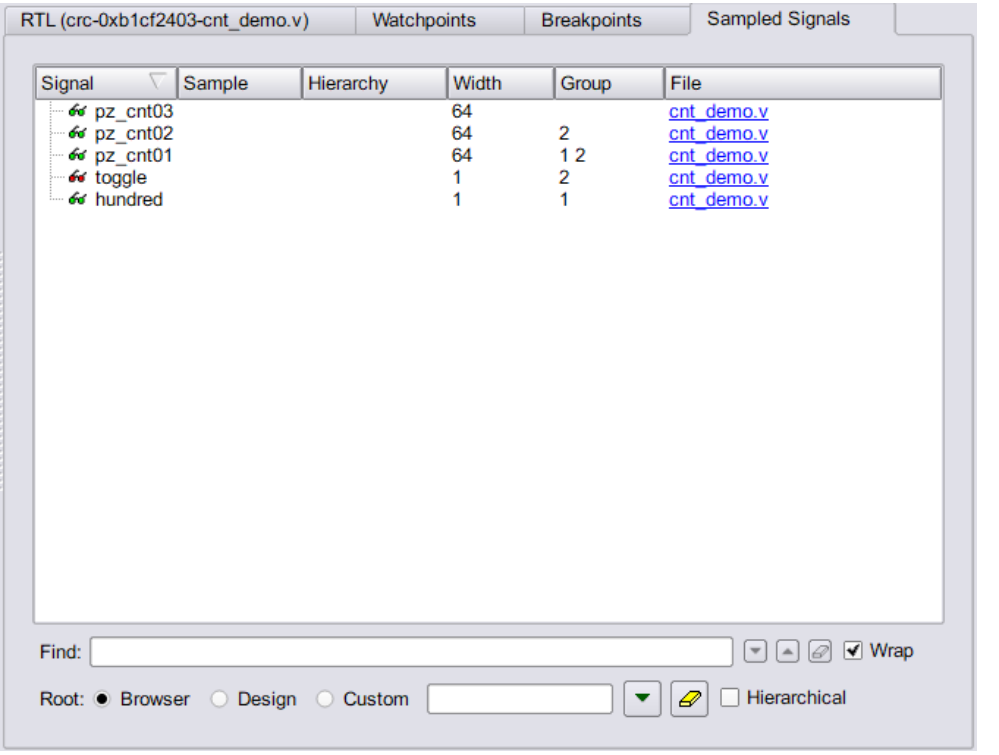
When an instrumented design has one or more activated breakpoints and one or more activated watchpoints, the result of the OR of the breakpoint events and the result of the AND of the watchpoint events is ANDed together. The result of this AND operation is called the Master Trigger Signal. This ANDing effectively causes the breakpoints and watchpoints to be dependent on each other – one activated breakpoint and all activated watchpoint events must occur concurrently to cause the sampling buffer to acquire its sample.

As a result, a Master Trigger Signal event can be constructed that operates like a conditional breakpoint. For example, activating a breakpoint and the two watchpoints from the previous example produces a conditional breakpoint: (breakpoint event) && (count== 23) && (ack == '1').

## Sampled Signals Panel

The Sampled Signals panel lists all of the sampled signals and their values when the sample trigger occurred. You can list all the sampled signals hierarchically based on a specific Root option.


© 2020 Synopsys, Inc.




The sampling block is basically a large memory used to store all the sampled signals. During an active debugging session, the sampled signals are continually being stored in the sample block. When the sample block receives an event from the Master Trigger Signal event logic or the complex counter logic, the sampling block stops writing new data to the buffer and holds its contents. Eventually, the contents of the sample block are uploaded to the debugger for display and formatting.

## TCL Window

TCL window is divided into Tcl console (Information) and Tcl shell windows. The Tcl console window displays status of the commands that have been executed, including those executed by menu selections and button clicks. The Tcl shell window allows you to type debugger commands and to view the results of command execution in the console window.

To capture all text written to the Tcl shell window, use the log console command, and to capture all commands executed in the Tcl Script window, use the transcript command. To clear the text from the console window, use the clear command (see the *Debug Environment Reference Manual* for descriptions and syntax.

The Tcl script window also displays each command executed in the course of running the tool, regardless of whether it was initiated from a menu, button, or keyboard shortcut. Right-clicking inside the Tcl window displays a popup menu with the Copy, Paste, Hide, and Help commands.

You can type or paste Tcl commands at the prompt ("% "). For a list of the available commands, type "help *" (without the quotes) at the prompt. For general information about Tcl syntax, choose Help > TCL.


© 2020 Synopsys, Inc.

# Toolbars

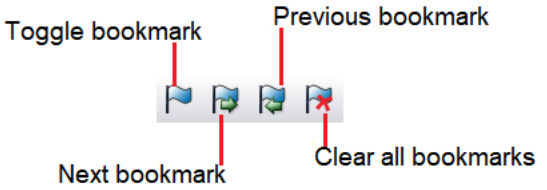
Toolbars provide a quick way to access common menu commands by clicking their icons. The following standard toolbars are available the debugger tool:

- Text Editor Toolbar, on page 170
- Edit Toolbar, on page 171
- File Toolbar in Instrumentor, on page 172
- File Toolbar in Debugger, on page 172

You can enable or disable the display of individual toolbars - see *Toolbar Command*, on page 111.

## Text Editor Toolbar

The Edit toolbar is active whenever the Text Editor is active. You use it to edit *bookmarks* in the file. The Edit toolbar provides the following icons, by default:



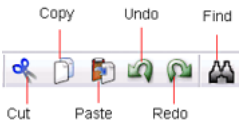

The following table describes the default Edit icons. See *Edit Menu*, on page 110, for more information.

| Icon | Description |
|---|---|
| Toggle Bookmark | Alternately inserts and removes a bookmark at the line that contains the text cursor.<br>Same as Edit ->Toggle bookmark. |

| Icon | Description |
|---|---|
| Next Bookmark | Takes you to the next bookmark.<br>Same as Edit ->Next bookmark. |
| Previous Bookmark | Takes you to the previous bookmark.<br>Same as Edit ->Previous bookmark. |
| Clear All Bookmarks | Removes all bookmarks from the Text Editor window.<br>Same as Edit ->Delete all bookmarks. |

## Edit Toolbar

The edit toolbar provides the following icons, by default:



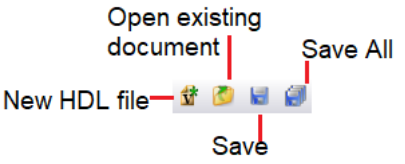

The following table describes the default edit icons. Each is equivalent to a Edit Menu command; for more information, see *Edit Menu*, on page 110.

| Icon | Description |
|---|---|
| Cut | Cuts text or graphics from the active view, making it available to Paste.<br>Same as Edit > Cut. |
| Copy | Copy the selected text.<br>Same as File ->Open. |
| Paste | Pastes previously cut or copied text or graphics to the active view.<br>Same as Edit > Paste. |
| Undo | Undoes the last action taken.<br>Same as Edit > Undo. |
| Redo | Performs the action undone by Undo.<br>Same as Edit > Redo. |
| Find | Finds text in the Text Editor or objects in an RTL view or Technology view.<br>Same as Edit > Find. |


© 2020 Synopsys, Inc.

## File Toolbar in Instrumentor

The file toolbar in instrumentor provides the following icons, by default:

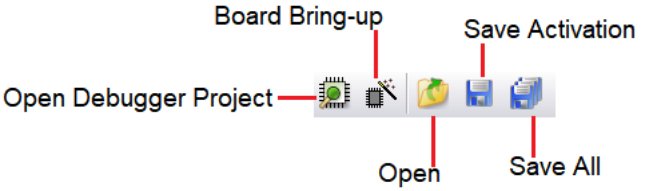



The following table describes the default File icons. Each is equivalent to a File Menu command; for more information, see *File Menu*, on page 106.

| Icon | Description |
|---|---|
| New HDL file | Opens the Text Editor window with a new, empty source file. |
| Open | Displays the Open File dialog box, to open a file.<br>Same as File ->Open. |
| Save | Saves the current file.<br>Same as File ->Save. |
| Save All | Saves all the unsaved changes.<br>Same as File ->Save All. |

## File Toolbar in Debugger

The file toolbar in debugger provides the following icons, by default:




The following table describes the default File icons. Each is equivalent to a File Menu command; for more information, see *File Menu*, on page 106.

| Icon | Description |
|---|---|
| Open Debugger Project | Open an additional debugger view. It also has shortcuts to Board Bring-Up, Confpro and Configure Confpro utilities. See *Edit Menu* , on page 110.<br>Same as File ->Open Debugger Project. |
| Open | Displays the Open File dialog box, to open a file.<br>Same as File ->Open. |
| Save Activation | Allows you to save a set of trigger settings.<br>Same as File ->Save Activation. |
| Save All | Saves all the unsaved changes.<br>Same as File ->Save All. |


© 2020 Synopsys, Inc.

# Keyboard Shortcuts

Keyboard shortcuts are key sequences that you type in order to run a command. Menus list keyboard shortcuts next to the corresponding commands. This section describes the keyboard shortcuts applicable in the instrumentor and debugger tools.


- Instrumentor Keyboard Shortcuts, on page 174
- Debugger Keyboard Shortcuts, on page 175


## Instrumentor Keyboard Shortcuts

The following table describes the keyboard shortcuts in the instrumentor tool.

| Keyboard Shortcut | Description |
|---|---|
| Ctrl-+ | To increase the font size. |
| Ctrl-- | To decrease the font size. |
| Ctrl-0 | To reset the font size to default. |
| Ctrl-a | In the RTL View, selects the entire data. |
| Ctrl-c | Copies the selected object. Same as Edit > Copy. This shortcut is sometimes available even when Edit > Copy is not. |
| Ctrl-D | To delete the selected IICE. Same as Instrumentor > Delete IICE. |
| Ctrl-E | To modify the selected IICE. Same as Instrumentor > Edit IICE. |
| Ctrl-f | Finds the selected object in the instrumentor source. Same as Edit > Find. |
| Ctrl-F2 | Alternately inserts and removes a bookmark to the line that contains the text cursor.<br>Same as Edit > Toggle bookmark (see Edit Menu, on page 110). |
| Ctrl-F4 | Closes the current window. Same as File > Close. |
| Ctrl-g | In the Text Editor, jumps to the specified line. Same as Edit > Goto (see Edit Menu, on page 110). |
| Ctrl-I | To add a new IICE. Same as Instrumentor > Add IICE. |

| Keyboard Shortcut | Description |
|---|---|
| Ctrl-o | Opens an existing file. Same as File > Open. |
| Ctrl-p | Prints the current view. Same as File > Print. |
| Ctrl-s | To save the file. Same as File > Save File. |
| Ctrl-v | Pastes the last object copied or cut. Same as Edit > Paste. |
| Ctrl-x | Cuts the selected object(s), making it available to Paste. Same as Edit > Cut. |
| Ctrl-y | Performs the action undone by Undo. Same as Edit > Redo. |
| Ctrl-z | Undoes the last action. Same as Edit > Undo. |
| Ctrl-Shift-F2 | Removes all bookmarks from the Text Editor window. Same as Edit > Delete all bookmarks (see Edit Menu, on page 110). |
| F1 | Provides context-sensitive help. Same as Help > Help. |
| F2 | In the Text Editor, takes you to the next bookmark. Same as Edit > Next bookmark (see Edit Menu, on page 110). |
| F3 | Finds the next instance of target expression set in Find in instrumentor source. See Find in Instrumentor Source, on page 123. Same as Instrumentor > Find next in instrumentor source. |
| F8 | Synthesizes (compiles and maps) your design. |
| Shift-F2 | In the Text Editor, takes you to the previous bookmark. |

## Debugger Keyboard Shortcuts

The following table describes the keyboard shortcuts in the debugger tool.

| Keyboard Shortcut | Description |
|---|---|
| Ctrl-+ | To increase the font size. |
| Ctrl-- | To decrease the font size. |
| Ctrl-0 | To reset the font size to default. |
| Ctrl-a | In the RTL View, selects the entire data. |


© 2020 Synopsys, Inc.

| Keyboard Shortcut | Description |
|---|---|
| Ctrl-b | To open the board bring-up settings. Same as Debugger > Board Bring-up. |
| Ctrl-c | Copies the selected object. Same as Edit > Copy. This shortcut is sometimes available even when Edit > Copy is not. |
| Ctrl-f | Finds the selected object in the debugger source. Same as Edit > Find. |
| Ctrl-F2 | Alternately inserts and removes a bookmark to the line that contains the text cursor.<br>Same as Edit > Toggle bookmark (see Edit Menu, on page 110). |
| Ctrl-F4 | Closes the current window. Same as File > Close. |
| Ctrl-g | In the Text Editor, jumps to the specified line. Same as Edit > Goto (see Edit Menu, on page 110). |
| Ctrl-o | Opens an existing file. Same as File > Open. |
| Ctrl-p | Prints the current view. Same as File > Print. |
| Ctrl-r | To debug the design. Same as Debugger > Run. |
| Ctrl-s | To save the file. Same as File > Save Activation. |
| Ctrl-v | Pastes the last object copied or cut. Same as Edit > Paste. |
| Ctrl-x | Cuts the selected object(s), making it available to Paste. Same as Edit > Cut. |
| Ctrl-y | Performs the action undone by Undo. Same as Edit > Redo. |
| Ctrl-z | Undoes the last action. Same as Edit > Undo. |
| Ctrl-Shift-F2 | Removes all bookmarks from the Text Editor window. Same as Edit > Delete all bookmarks (see Edit Menu, on page 110). |
| F1 | Provides context-sensitive help. Same as Help > Help. |
| F2 | In the Text Editor, takes you to the next bookmark. Same as Edit > Next bookmark (see Edit Menu, on page 110). |
| F3 | Finds the next instance of target expression set in Find in debugger source. See Find in debugger source, on page 130. Same as Debugger > Find next in debugger source. |
| Shift-F2 | In the Text Editor, takes you to the previous bookmark. |

CHAPTER 5

# GUI Popup Menu Commands

In addition to the GUI menu commands described in Chapter 3, *User Interface Commands*, the instrumentor and debugger tools also have context-sensitive commands that are accessed from popup or right-click menus in different parts of the interface. Most of these commands have an equivalent menu command. This chapter only describes the unique commands that are not documented in the previous chapter.

See the following sections for details:


- Popup Menus, on page 178
- RTL View Popup Menus, on page 180



© 2020 Synopsys, Inc.

# Popup Menus

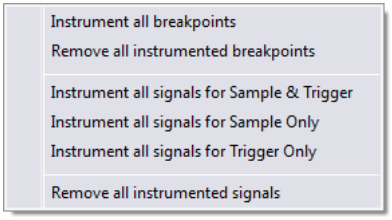
Popup menus, available by clicking the right mouse button, offer quick ways to access commonly used menu commands that are specific to the view where you click. Commands shown greyed out (dimmed) are currently inaccessible. Popup menu commands generally duplicate commands available from the regular menus, but sometimes have commands that are only available from the popup menu.

The following popup menus are available.

- Hierarchy Browser Popup Menu, on page 178
- Tcl Window Popup Menu, on page 179

## Hierarchy Browser Popup Menu

In the instrumentor tool, a popup menu is available in the hierarchy browser to set or clear breakpoints or watchpoints at any level of the hierarchy. Positioning the cursor over an element and clicking the right mouse button displays the following menu.




The selected operation is applied to all breakpoints or signal watchpoints at the selected level of hierarchy. You cannot instrument signals when a sample clock is included in the defined group.

Black-box modules are represented by a black icon, and their contents can not be instrumented. Also, certain modules cannot be instrumented.

## Tcl Window Popup Menu

The Tcl window popup menu contains the Copy, Paste, and Find commands from the Edit menu, as well as the Delete command, which empties the Tcl window. These popup menu options are the same in the debugger tool as well. For information on the Edit menu commands available in the Tcl window, see *Edit Menu*, on page 110.


© 2020 Synopsys, Inc.

# RTL View Popup Menus

This section describes the available popup menus of the RTL view in the instrumentor and debugger tools separately.


- RTL View Options in Instrumentor, on page 180
- RTL View Options in Debugger, on page 182


The commands on the popup menu are context-sensitive, and vary depending on the object selected, the kind of view, and where you click. In general, if you have a selected object and you right-click in the background, the menu includes global commands as well as selection-specific commands for the objects.

## RTL View Options in Instrumentor

Some commands are only available from the popup menus in the RTL and Technology views, but most of the commands are duplicates of commands from the Edit, View, and Instrumentor menus. The popup menus in the RTL and Technology views are nearly identical. See the following:

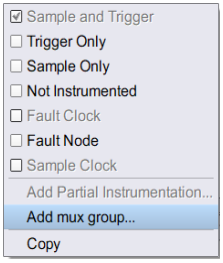
| Command | Description |
|---|---|
| Sample and Trigger | To set the selected signal for sampling and triggering. See the *Debugger User Guide*. |
| Trigger Only | To set the selected signal for triggering only. See the *Debugger User Guide*. |
| Sample Only | To set the selected signal for sampling only. See the *Debugger User Guide*. |
| Not Instrumented | To remove the set instrumentation of the selected signal. See the *Debugger User Guide*. |
| Fault Clock | To set the selected signal as fault clock signal. See the *Debugger User Guide*. |
| Fault Node | To set the selected signal as fault node signal. See the *Debugger User Guide*. |

| Command | Description |
|---|---|
| Sample Clock | To set the selected clock as sample clock. See the *Debugger User Guide*. |
| Add mux group | To add the selected signal to the multiplex group. See Add Mux Group, on page 181. |
| Copy | To copy the selected signal. |

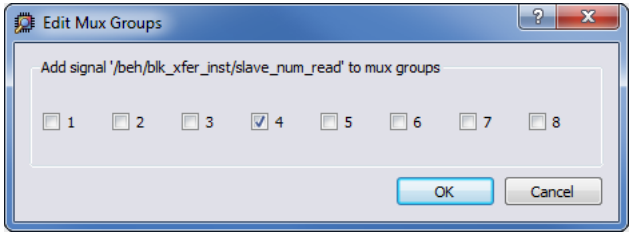
## Add Mux Group

Multiplexed groups allow signals to be assigned to logical groups. Using multiplexed groups can substantially reduce the amount of pattern memory required during subsequent debugging when all of instrumented signals are not required to be loaded into memory at the same time.

Only signals or buses that are instrumented as either Sample and Trigger or Sample only can be added to a multiplexed group. To create multiplexed groups, right click on each individual instrumented signal or bus and select Add mux group from the popup menu.




In the Add mux group dialog box displayed, select a corresponding group by checking the group number and then click OK to assign to the signal or bus to that group. A signal can be included in more than one group by checking additional group numbers before clicking OK.





© 2020 Synopsys, Inc.

When assigning instrumented signals to groups:

- A maximum of eight groups can be defined; signals can be included in more than one group, but only one group can be active in the debugger at any one time.
- Signals instrumented as Sample Clock or Trigger only cannot be included in multiplexed groups.
- Partial buses cannot be assigned to multiplexed groups.
- The signals group command can be used to assign groups from the console window (see *signals* in the *Reference Manual*). Command options allow more than one instrumented signal to be assigned in a single operation and allow the resultant group assignments to be displayed.

For information on using multiplexed groups in the debugger, see the *Debugger User Guide*.

## RTL View Options in Debugger

Some commands are only available from the popup menus in the RTL and Technology views, but most of the commands are duplicates of commands from the Edit, View, and Debugger menus. The popup menus in the RTL and Technology views are nearly identical.

The following RTL view popup options are available in the debugger tool. These options are nearly identical in the RTL, Watchpoints, Breakpoints, Samples Signals tabs.

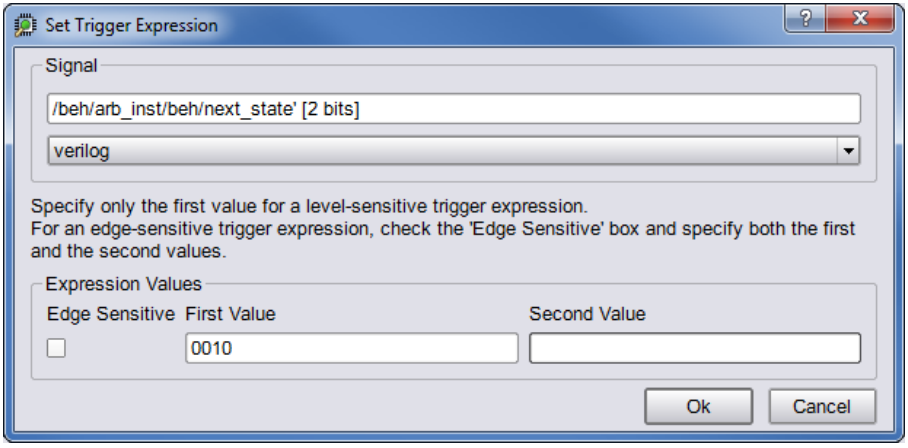
- Trigger Conditions, on page 183
- Change Signal Radix, on page 185
- Example of Sampling Signals in a Folded Hierarchy, on page 186
- Example of Selecting Breakpoints in Folded Hierarchies, on page 187

## Trigger Conditions

Use the following pre-defined trigger options available by selecting Watchpoint > Conditions.

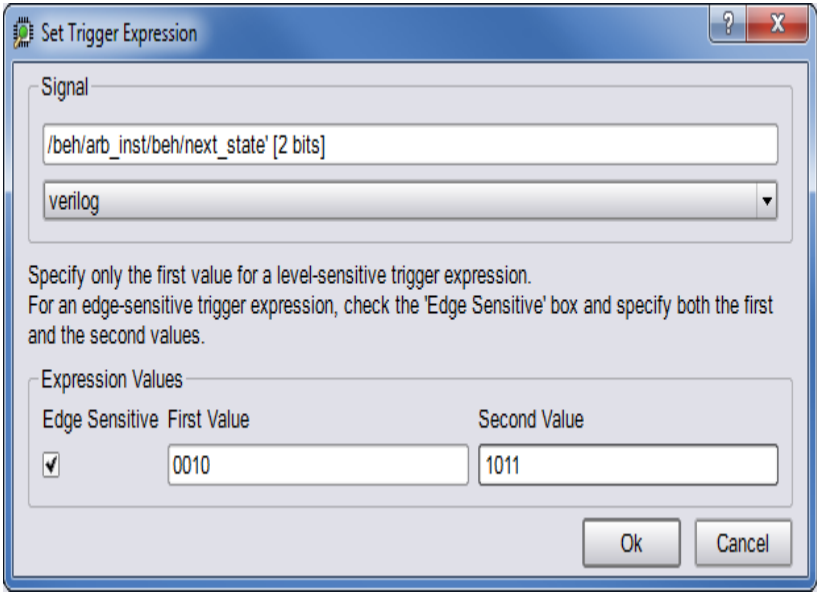
- Trigger low - Triggers when the signal is low.
- Trigger high - Triggers when the signal is high.
- Trigger don't care - Triggers when the signal is unknown.
- Trigger 0->1 - Triggers when the signal is 0 to 1.
- Trigger 1->0 - Triggers when the signal is 1 to 0.
- Custom trigger - To set the specific trigger values.

In the example below, the signal is a 4-bit signal, and the watchpoint expression is set to "0010" (binary). Any legal VHDL or Verilog (as appropriate) constant expression is accepted.




To create a transition watchpoint, assign a second constant expression to the watchpoint. A transition watchpoint triggers when the watched signal value is equal to the first expression during a clock period and the value is equal to the second expression during the next clock period. In the example below, the transition being defined is a transition from "0010" to "1011."


© 2020 Synopsys, Inc.




The VHDL or Verilog expressions that are entered in the Watchpoint Setup dialog box can also contain “X” values. The “X” values allow the value of some bits of the watched signal to be ignored (effectively, “X” values are don’t-care values). For example, the above value watchpoint expression can be specified as “X010” which causes the watchpoint to trigger only on the values of the three right-most bits.

Hexadecimal values can additionally be entered as watchpoint values using the following syntax:

**x"***hexValue***"**

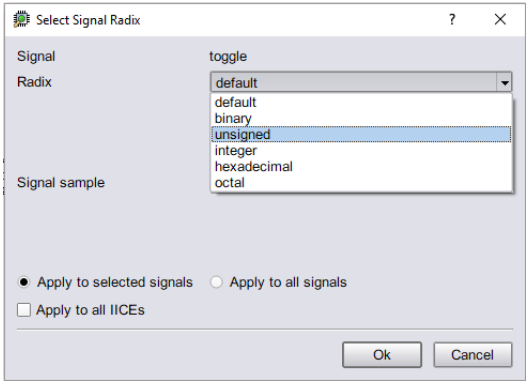
As shown, a hexadecimal value is introduced with an x character and the value must be enclosed in quotation marks.

Clicking OK on the Watchpoint Setup dialog box activates the watchpoint (the watchpoint or “P” icon changes to red) which is then armed in the hardware the next time the Run button is pressed.

## Change Signal Radix

The radix of the sampled data displayed can be set to any of a number of different number bases. To change the radix of a sampled signal:

1. Right-click on the signal name or the watchpoint or “P” icon and select Change signal radix to display the following dialog box.




2. Select the desired radix from the Radix drop-down menu.
3. Click OK.

---

**Note:** You can change the radix before the data is sampled. The watchpoint signal value will appear in the specified radix when the sampled data is displayed.

---

Selecting 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 debugger and is not propagated to any of the waveform viewers.

---

**Note:** Changing the radix of a partial bus changes the radix for all bus segments.

---

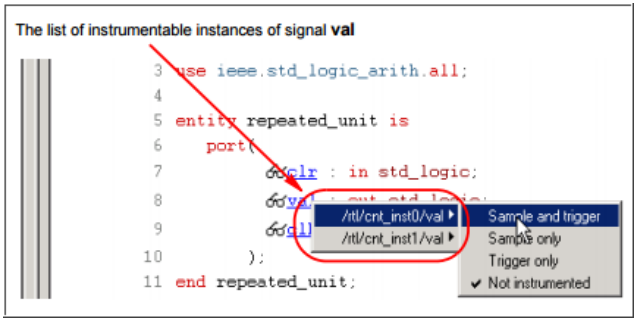

© 2020 Synopsys, Inc.

## Example of Sampling Signals in a Folded Hierarchy

The example below consists of a top-level entity called two-level and two instances of the repeated_unit entity. The source code of repeated_unit is displayed, and the list of instances of the val signal is displayed by clicking on the watchpoint icon or the signal name. Two instances of the signal val are available for sampling:

```
/rtl/cnt_inst0/val
/rtl/cnt_inst1/val
```

Either, or both, of these instances can be selected for sampling by selecting the signal instance and then sliding the cursor over to select the type of sampling to be instrumented for that signal instance.




Alternately, any of the instances of a folded signal can be selected or deselected at the instrumentor console window prompt by using the absolute path name of the instance. For example,

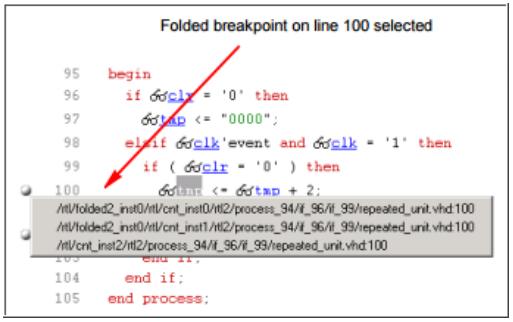
```
signals add /rtl/cnt_inst1/val
```

See the *Debugger User Guide* for more information.

## Example of Selecting Breakpoints in Folded Hierarchies

The example below consists of a top-level entity called top and two instances of the repeated_unit entity. The source code of repeated_unit is displayed; the list of instances of the breakpoint on line 100 is displayed by clicking on the breakpoint icon next to the line number. As shown in the following figure, three instances of the breakpoint are available for sampling.

Any or all of these breakpoints can be selected by clicking on the corresponding line entry in the list displayed.




Alternatively, any of the instances of a folded breakpoint can be selected or deselected at the instrumentor console window prompt by using the absolute path name of the instance. For example,

```
breakpoints add
   /rtl/inst0/rtl/process_18/if_20/if_23/repeated_unit.vhd:24
```

See the *Debugger User Guide* for more information.

The lines in the list of breakpoint instances act to toggle the selection of an instance of the breakpoint. To disable an instance of a breakpoint that has been previously selected, simply select the appropriate line in the list box.


© 2020 Synopsys, Inc.

© 2020 Synopsys, Inc.

# Index


## A

About this program command 151
activation command 20
Additional Products command 151
always-armed triggering 117
asynchronous clocks 84

## B

black boxes 178
blocks
  sampling 168
board file
  generation 37
board query 37
breakpoints
  activating/deactivating 88
  combined with watchpoints 167
  listing available 155
  listing instrumented 155
  multiple 166
  searching 44
breakpoints command 21
buffer
  sample depth 55
Byteblaster cable settings 136

## C

cable compatibility 135
cable option settings 27
cable type settings
  Byteblaster 136
  JTAGTech3710 137
  Microchip 137
cable types 26
camera mouse pointer 106
CAPIM 33
cd command 23
chain command 24
clear command 26
clock
  sampling 49
clock option 49
clocks
  asynchronous 84
  edge selection 118
  sample 118
Close command 106
com command 26
command history 90
commands
  recording 90
communication cable
  settings 135
complex counter
  cycles mode 143
  events mode 143
  pulsewidth mode 144
  watchdog mode 144
complex triggering 50, 119
configuration
  IICE 48
Configure IICE dialog box
  IICE Controller tab 119
  IICE Sampler tab 116
console output
  logging 65
console window 168
  operations 169
context-sensitive popup menus
  *See* popup menus
convenience functions 82
conventions
  design hierarchy 15
  file system 14



© 2020 Synopsys, Inc.

symbol 17
syntax 13
tool 14
Copy command 110
copying
for pasting 174, 176
copying image
Create Image command 106
counterwidth option 50
Create Image command 106
cross triggering 84
commands 85
enabling 84
state machine commands 85
Cut command 110
cycles mode
complex counter 143


## D


debugger
process ID 67
depth option 55
design files
writing 99
design hierarchy 41
design hierarchy conventions 15
device command 31
device ID codes 46
directories
changing 23
displaying working 67
distributed instrumentation 36


## E


Edit menu 110
encrypting source files 126
encryption command 35
event trigger 67
events mode
complex counter 143
exit command 36


## F


file system conventions 14
files
encrypting source 126
searching 71
syn_trigger_utils.tcl 82
Verdi fast signal database 98
Verilog Change Dump 103
writing design 99
find option 43
fpga command 36
FSM toolbar 174


## G


graphical user interface (GUI), overview 153
GUI
clearing 26
logging console output 65
GUI (graphical user interface), overview 153


## H


haps command 37
HDL files
searching 71
Help command 151
help command 40
Help menu 151
hierarchy browser
popup menu 178
hierarchy command 41
hierarchy separator 15
How to Use Help command 151


## I


idcode command 46
IICE
arming 69
communicating with 26
cross triggering 84
selecting multiple 48

iice command 48
IICE Controller tab 119
IICE parameters
  buffer type 116
IICE Sampler tab 116
IICE settings
  sample clock 118
  sample depth 116
importing projects 66
incremental implementations 59
instrumentation command 59

## J

JTAG chain
  settings 140
JTAG chains 24
jtag_server command 61
JTAGTech3710 cable settings 137

## K

keyboard shortcuts 174

## L

license
  saving 151
License Agreement command 151
log command 65

## M

menubar 11
menus
  context-sensitive
    *See* popup menus
  Edit 110
  Help 151
  Options 148
  popup
    *See* popup menus
  View 111
Messages
  Tcl Window command 111
Microchip
  cable type settings 137
models
  VHDL 104
modes
  cross triggering 142
multi-IICE selection 48
multiple debuggers 67
multiple implementations 59
multiplexed groups
  assigning 181

## N

New command 106

## O

online help 40
Open command
  File menu 106
operators
  state machine 81
Options menu 148

## P

parallel port
  defining 28
passwords
  encryption 35
  encryption/decryption 126
Paste command 110
path names 15
path separator 14
popup menus
  RTL view 180
  Tcl window 179
  Technology view 180
Preferred License Selection command 151
Print command 106
Print Setup command 106
printing image
  Create Image command 106
process ID


© 2020 Synopsys, Inc.

debugger 67
project command 66
projects
creating new 66
importing 66
opening 66
pulsewidth mode
complex counter 144
pwd command 67

## Q

qualified sampling 116

## R

RAM resources 116
Redo command 110
remote_trigger command 67
RTL view
popup menu 180
run command 69, 159

## S

sample clock 118
sample data 101
sampling
qualified 116
sampling block 168
sampling signals 155, 156, 163
Save All command 106
searching 43
searchpath command 71
separator
hierarchy 15
path 14
server configuration 61
setsys command 72
settings
cable 135
JTAG chain 140
sample clock 118
sample depth 116
shortcuts

keyboard
*See* keyboard shortcuts
show command 73
signals
exporting trigger 123
listing available 156, 163
listing instrumented 155, 156, 163
sampling selection 155, 156, 163
status 94
signals command
debug logic 74
simple triggering 119
source code
displaying 73
source command 79
source files
encrypting 126
state machines
configuring 80
operators 81
triggering 50, 120
statemachine command 80, 132
state-machine triggering 119
Status Bar command 111
status reporting 94
stop command 88, 94
symbol conventions 17
syn_trigger_utils.tcl file 82
syntax conventions 13
system variables 72

## T

tables
idcode 46
target device 31
Tcl (Tool Command Language) 10
Tcl conventions 10
TCL Help command 151
Tcl Script
Tcl Window command 111
Tcl scripts 79
Tcl window
popup menu 179


© 2020 Synopsys, Inc.

Tcl Window command 111
Technology view
  popup menu 180
text
  copying, cutting and pasting 110
tool conventions 14
toolbars 170
  FSM 174
Toolbars command 111
tooltips
  displaying 112
transcript command 90
transition watchpoint 183
trigger conditions 21
trigger settings
  reloading 20
  saving 20
trigger signal
  exporting 123
triggering
  advance mode 119
  always-armed 117
  between IICEs 84
  complex 50, 119
  simple 119
  state machine 119, 120
  state machines 50
triggermode option 57
triggers 92
triggerstates option 50
triggertime option 56

## U

UMRBus communication 33
Undo command 110
user interface, overview 153

## V

variables 72
verdi command 91
version information 151
VHDL models 104

View menu 111

## W

watch command 92
watchdog mode
  complex counter 144
watchpoints 92, 164
  combined with breakpoints 167
  hexadecimal values 184
  multiple 167
  searching 44
  transition 183
waveform command 95
waveform display 130, 145
wildcards
  in hierarchies 16
  in path names 14
  text Find 124, 131
windows
  closing 174, 176
  console 168
working directory
  displaying 67
write fsdb command 98
write instrumentation command 99
write samples command 101
write vcd command 103
write vhdlmodel command 104


© 2020 Synopsys, Inc.